Komei Fukuda
Joris van der Hoeven
Michael Joswig
Nobuki Takayama (Eds.)

LNCS 6327

Mathematical Software – ICMS 2010

Third International Congress on Mathematical Software
Kobe, Japan, September 2010
Proceedings

Springer

Lecture Notes in Computer Science 6327

Commenced Publication in 1973
Founding and Former Series Editors:
Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Komei Fukuda Joris van der Hoeven
Michael Joswig Nobuki Takayama (Eds.)

Mathematical Software – ICMS 2010

Third International Congress
on Mathematical Software
Kobe, Japan, September 13-17, 2010
Proceedings

Springer

Volume Editors

Komei Fukuda
Institute for Operations Research
and Institute of Theoretical Computer Science
ETH Zurich, 8092 Zurich, Switzerland
E-mail: fukuda@ifor.math.ethz.ch

Joris van der Hoeven
LIX, CNRS, École polytechnique
91128 Palaiseau cedex, France
E-mail: vdhoeven@lix.polytechnique.fr

Michael Joswig
TU Darmstadt, Fachbereich Mathematik
64289 Darmstadt, Germany
E-mail: joswig@mathematik.tu-darmstadt.de

Nobuki Takayama
Kobe University, Department of Mathematics
Rokko, Kobe, 657-8501, Japan
E-mail: takayama@math.kobe-u.ac.jp

Library of Congress Control Number: 2010933525

CR Subject Classification (1998): G.2, I.1, F.2.1, G.4, F.2, G.1

LNCS Sublibrary: SL 1 – Theoretical Computer Science and General Issues

ISSN 0302-9743
ISBN-10 3-642-15581-2 Springer Berlin Heidelberg New York
ISBN-13 978-3-642-15581-9 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper 06/3180

Preface

The ICMS Developer's Meeting is an international congress for which the main theme is mathematical software. The 2010 meeting was the third of a series of meetings of similar theme, the first being held in Beijing, China in 2002, and the second in Castro-Urdiales, Spain in 2006.

The field of mathematics has numerous branches, and in each branch we find that algorithms, and also implementations and applications of software systems, are studied. Researchers who endeavor to make such studies also have international meetings within their specific branches of mathematics, and these meetings have made significant contributions to the fields in which they lie. The ICMS (International Congresses on Mathematical Software), on the other hand, is a general (not branch specific) meeting on mathematical software, which is held every four years, and is a rare opportunity for developers of mathematical software from different branches of mathematics, as well as mathematicians who are interested in mathematical software, to gather together.

Since the first meeting in Beijing, eight years have passed, and this is now a good occasion for us to ask this important question, and we beg the readers' indulgence for its bluntness: is this kind of general meeting useful? In order to have a productive meeting, the participants should have a base of common interests and knowledge. Do we have such common interests and knowledge? To help us consider this question, let us note the following points:

(1) We are interested in mathematics and want to explore the world of mathematics, regardless of whether we have the aid of a computer. The discovery or proof of a new mathematical fact is an exciting application of mathematical software. Certainly all of us would agree with this claim.

(2) Which objects in mathematics are computable, and which are not? To what degree can we be efficient in computation? Can we implement algorithms efficiently? All participants and authors of these proceedings will surely agree that these are fundamental questions, and will have a strong interest in answering them.

(3) All participants and authors know at least one programming language. For instance, most people will understand the programming language C and/or its derivatives. All participants are interested in using software environments to study mathematics.

(4) We understand that technology transfers from mathematics to industry and other fields are done primarily via software systems. We should be able to explain how our favorite algorithms are used in industrial applications.

We have listed above four points of common knowledge and interests, and certainly there are more. It is through points like these that we can exchange a wide variety of ideas and knowledge with each other, resulting in the advance-

ment of mathematical software. On the basis of points like these, we can give a resounding "yes" to the first blunt question above.

We believe that mathematics itself is a coherent whole, and there are copious examples of how the interplay between seemingly disparate branches of mathematics yields new results. In mathematical software, let us note some such examples that have come to fruition in the last eight years: software systems in tropical geometry have been produced using computer algebra and polyhedral geometry software as a base; applications of computer algebra have created a new area of research called "algebraic statistics." There are many other cases as well, and a network of researchers from different disciplines has been an important foundation for this, leading to a wealth of interdisciplinary research.

The articles in these proceedings were written by speakers at the ICMS 2010 and reviewed by the Program Committee members and some external referees. No doubt the authors wish that not only their peers, but also researchers in other branches of mathematics, will become interested in their results and will apply their outcomes to those other branches. The authors also surely hope that mathematicians, scientists and engineers will read articles in this volume, and will then have a deeper understanding of what is going on at present in the study of mathematical software, and will in turn suggest new applications of mathematical software and also give new proposals for developing mathematical software.

The activities of the last two conferences are archived in proceedings, in software and document DVD's, and in video format. This material can be accessed through *http://www.mathsoftware.org*. ICMS 2010 will also be archived in this way.

We hope that these proceedings will contribute to the advancement of a wide array of research directions, led by many researchers with varied backgrounds.

September 2010

Nobuki Takayama
Komei Fukuda
Joris van der Hoeven
Michael Joswig

Organization

ICMS 2010 was organized by the Department of Mathematics Kobe University, Rokko, Kobe, Japan.

Conference Chairs

General Chair	Nobuki Takayama (Kobe University, Japan)
Program Co-chairs	Komei Fukuda (ETH Zurich, Switzerland)
	Joris van der Hoeven (CNRS, École Polytechnique, France)
	Michael Joswig (Technische Universität Darmstadt, Germany)
Poster Session Chair	Raimundas Vidunas (Kobe University, Japan)
Local Organization Chair	Masayuki Noro (Kobe University, Japan)

Program Committee

Bettina Eick	Technische Universität Braunschweig, Germany
Anne Fruehbis-Krueger	Leibniz Universität Hannover, Germany
Komei Fukuda	ETH Zurich, Switzerland
Tatsuyoshi Hamada	Fukuoka University, Japan
John Harrison	Intel Corporation, USA
Joris van der Hoeven	CNRS, École Polytechnique, France
Tim Hoffmann	Technische Universität Münich, Germany
Andres Iglesias	University of Cantabria, Spain
Michael Joswig	Technische Universität Darmstadt, Germany
Paul Libbrecht	DFKI GmbH and University of Saarland, Germany
Steve Linton	University of St. Andrews, Scotland, UK
Hidefumi Ohsugi	Rikkyo University, Japan
Pawel Pilarczyk	University of Minho, Portugal
Michael Pohst	Technische Universität Berlin, Germany
Nathalie Revol	École Normale Supérieure de Lyon, France
Wayne Rossman	Kobe University, Japan
Michael Sagraloff	Max-Planck-Institut für Informatik, Germany
Bruno Salvy	INRIA Rocquencourt, France
Achill Schuermann	Delft University of Technology, The Netherlands
Vin de Silva	Pomona College, USA
Monique Teillaud	INRIA Sophia Antipolis, France
Shigenori Uchiyama	Tokyo Metropolitan University, Japan
Freek Wiedijk	Radboud University, The Netherlands
Chee Yap	New York University, USA
Afra Zomorodian	Dartmouth College, USA

Advisory Program Committee

Henk Barendregt	Radboud University, The Netherlands
Arjeh Cohen	Technische Universiteit Eindhoven, The Netherlands
Dan Grayson	University of Illinois, USA
Gert-Martin Greuel	University of Kaiserslautern, Germany
Jean Lasserre	LAAS-CNRS, France
Bernard Mourrain	INRIA Sophia Antipolis, France
Ken Nakamula	Tokyo Metropolitan University, Japan
Bernd Sturmfels	University of California Berkeley, USA
Jan Verschelde	University of Illinois, USA
Dongming Wang	CNRS, France

Session Organizers

Computational Group Theory
Bettina Eick (Technische Universität Braunschweig, Germany)
Steve Linton (University of St. Andrews, Scotland, UK)

Computation of Special Functions
Bruno Salvy (INRIA Rocquencourt, France)

Computer Algebra
Joris van der Hoeven (CNRS, École Polytechnique, France)
Nathalie Revol (École Normale Supérieure de Lyon, France)

Exact Numeric Computation for Algebraic and Geometric Computation
Chee Yap (New York University, USA)
Michael Sagraloff (Max-Planck-Institut für Informatik, Germany)
Monique Teillaud (INRIA Sophia Antipolis, France)

Formal Proof
John Harrison (Intel Corporation, USA)
Freek Wiedijk (Radboud University, The Netherlands)

Geometry and Visualization
Tim Hoffmann (Technische Universität Münich, Germany)
Wayne Rossman (Kobe University, Japan)

Groebner Bases and Applications
Anne Fruehbis-Krueger (Leibniz Universität Hannover, Germany)
Hidefumi Ohsugi (Rikkyo University, Japan)

Number Theoretical Software
Shigenori Uchiyama (Tokyo Metropolitan University, Japan)
Ken Nakamula (Tokyo Metropolitan University, Japan)
Michael Pohst (Technische Universität Berlin, Germany)

Reliable Computing
Joris van der Hoeven (CNRS, École Polytechnique, France)
Nathalie Revol (École Normale Supérieure de Lyon, France)

Software for Optimization and Polyhedral Computation
Achill Schuermann (Delft University of Technology, The Netherlands)
Komei Fukuda (ETH Zurich, Switzerland)
Michael Joswig (Technische Universität Darmstadt, Germany)

Sponsoring Institutions

1. Faculty of Science, Kobe University
2. Kakenhi 19204008, Japan Society of Promotion of Science
3. Team Hibi, Alliance for breakthrough between mathematics and sciences, Japan Science and Technology Agency

Table of Contents

Mathematical Software - ICMS 2010

Plenary

Computation of Special Functions (Invited)

Computational Group Theory (Invited)

Computational Group Theory (Contributed)

Computer Algebra (Invited)

Exact Numeric Computation for Algebraic and Geometric Computation (Invited)

Exact Numeric Computation for Algebraic and Geometric Computation (Contributed)

Formal Proof (Invited)

Formal Proof (Contributed)

Geometry and Visualization (Invited)

Geometry and Visualization (Contributed)

Groebner Bases and Applications (Invited)

Groebner Bases and Applications (Contributed)

Number Theoretical Software (Invited)

Number Theoretical Software (Contributed)

Software for Optimization and Polyhedral Computation (Invited)

Software for Optimization and Polyhedral Computation (Contributed)

Reliable Computation (Invited)

Reliable Computation (Contributed)

Computational Discrete Geometry

Thomas C. Hales*

University of Pittsburgh

Abstract. In recent years, computers have been used regularly to solve major problems in discrete geometry. The talk at ICMS 2010 will give a survey of the computational methods. The extended abstract that is provided below mentions a few of the problems that will be discussed.

Newton-Gregory Problem

In a famous discussion, Isaac Newton claimed that at most twelve nonoverlapping congruent balls in Euclidean three space can touch one further ball at the center of them all. Gregory thought that it might be possible for thirteen balls to touch the one at the center. It was only in 1953 that Newton was finally proved correct. Earlier this year, Musin and Tarasov announced that they have finally determined the optimal arrangement of thirteen balls [9]. These thirteen balls do not touch the one at the center, but they come as close as possible. Their proof involves an analysis of more than 94 million planar graphs, which have been generated with the program *plantri* [2]. Linear programming methods are used to exclude all but the one optimal graph.

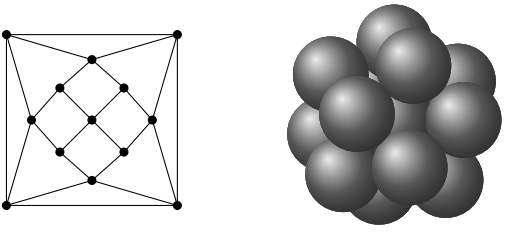


Fig. 1. Musin and Tarasov recently proved that this arrangement of thirteen congruent is optimal. Each node of the graph represents one of the thirteen balls and each edge represents a pair of touching balls. The node at the center of the graph corresponds to the uppermost ball in the second frame.

Hilbert's Eighteenth Problem

In 1900, in his famous list of problems, Hilbert asked, "How can one arrange most densely in space an infinite number of equal solids of given form, e. g., spheres with given radii or regular tetrahedra with given edges (or in prescribed

* Research supported by NSF grant 0804189 and a grant from the Benter Foundation. The author places this abstract in the public domain.

K. Fukuda et al. (Eds.): ICMS 2010, LNCS 6327, pp. 1–3, 2010.

position), that is, how can one so fit them together that the ratio of the filled to the unfilled space may be as great as possible" [7]?

Dense Sphere Packings. The solution to the sphere-packing problem was published in [6]. It is now the subject of a large scale formal-proof project, Flyspeck, in the HOL Light proof assistant. The talk will describe the current status of this project.

Tetrahedra. Aristotle erroneously believed that the regular tetrahedron tiles three dimensional space: "It is agreed that there are only three plane figures which can fill a space, the triangle, the square, and the hexagon, and only two solids, the pyramid and the cube" [1]. In fact, the tetrahedron cannot tile because its dihedral angle is about 70.5°, which falls short of the angle $72 = 360/5$ that would be required of a space-filling tile.

Attention has turned to the tetrahedron-packing problem, which has come under intensive investigation over the past few years [3], [10]. As a result of Monte Carlo simulations by Chen et al., the optimal packing is now given by an explicit conjecture.

Dense Lattice Packings of Spheres in High Dimensions

Lagrange found that the denset packing of congruent disks in the plane, among all lattice packings, is the hexagonal packing [8]. See Figure 2. Gauss solved the analogous problem in three dimensions [5]. During the early decades of the twentieth century, the problem of determining the densest lattice packing of balls was solved in dimensions up to eight. Cohn and Kumar, in a computer assisted proof, have solved the problem in dimension 24 [4]. Their proof relies on a variety of computations and mathematical methods, including the Poisson summation formula and spherical harmonics.

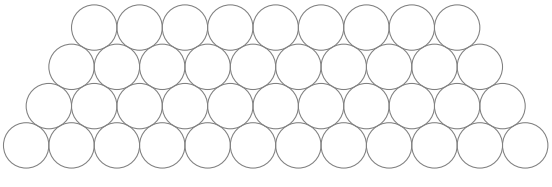


Fig. 2. Lagrange proved that this is the densest of all lattice packings in two dimensions

Other Problems

This abstract has mentioned just a few of a large number of problems in discrete geometry that have been or that are apt to be solved by computer. Others include Fejes Toth's contact conjecture, the Kelvin problem, circle packing problems, the strong dodecahedral conjecture, the Reinhardt conjecture, and the covering problem. Discrete geometry depends on the development of software to assist in the solution to these problems.

References

1. Aristotle, On the heaven, translated by J.L. Stocks, 350BC, http://classics.mit.edu/Aristotle/heavens.html
2. Brinkmann, G., McKay, B.D.: Fast generation of planar graphs, expanded edition (2007), http://cs.anu.edu.au/~bdm/papers/plantri-full.pdf
3. Chen, B., Engel, M., Glotzer, S.C.: Dense crystalline dimer packings of regular tetrahedra (2010), http://arxiv.org/abs/1001.0586
4. Cohn, H., Kumar, A.: The densest lattice in twenty-four dimensions. Electronic Research Annoucements of the American Mathematical Society 10, 58–67 (2004), math.MG/0408174
5. Gauss, C.F.: Untersuchungen über die Eigenschaften der positiven ternären quadratischen Formen von Ludwig August Seber. Göttingische gelehrte Anzeigen (1831); Also published in J. reine angew. Math. 20, 312–320 (1840), Werke 2. Königliche Gesellschaft der Wissenschaften, Göttingen, 188–196 (1876)
6. Hales, T.C., Ferguson, S.P.: Kepler conjecture. Discrete and Computational Geometry 36(1), 1–269 (2006)
7. Hilbert, D.: Mathematische probleme. Archiv Math. Physik 1, 44–63 (1901); Also in Proc. Sym. Pure Math. 28, 1–34 (1976)
8. Lagrange, J.L.: Recherches d'arithmétique. Mem. Acad. Roy. Sc. Bell Lettres Berlin 3, 693–758 (1773); Volume and pages refer to Œuvres
9. Musin, O.R., Tarasov, A.S.: The strong thirteen spheres problem (February 2010) (preprint), http://arxiv.org/abs/1002.1439
10. Torquato, S., Jiao, Y.: Exact constructions of a family of dense periodic packings of tetrahedra. Physical Review E 81, 041310–1–11 (2010), http://cherrypit.princeton.edu/papers.html

Exploiting Structured Sparsity in Large Scale Semidefinite Programming Problems

Masakazu Kojima*

Department of Mathematical and Computing Science,
Tokyo Institute of Technology,
Oh-Okayama, Meguro, Tokyo 152-8552, Japan

Abstract. Semidefinite programming (SDP) covers a wide range of applications such as robust optimization, polynomial optimization, combinatorial optimization, system and control theory, financial engineering, machine learning, quantum information and quantum chemistry. In those applications, SDP problems can be large scale easily. Such large scale SDP problems often satisfy a certain sparsity characterized by a chordal graph structure. This sparsity is classified in two types. The one is the domain space sparsity (d-space sparsity) for positive semidefinite symmetric matrix variables involved in SDP problems, and the other the range space sparsity (r-space sparsity) for matrix-inequality constraints in SDP problems. In this short note, we survey how we exploit these two types of sparsities to solve large scale linear and nonlinear SDP problems. We refer to the paper [7] for more details.

Keywords: Semidefinite Program, Primal-Dual Interior-Point Method, Sparsity, Chordal Graph.

Let $\mathbb{R}^n$ denote the n-dimensional Euclidean space, $\mathbb{S}^n$ the linear space of $n \times n$ symmetric matrices, and $\mathbb{S}^n_+$ the cone of $n \times n$ symmetric positive semidefinite matrices. Let $N = \{1, 2, \ldots, n\}$. For every nonempty $C \subset N$, we use the symbol $\mathbb{S}^C$ for the linear space of $|C| \times |C|$ symmetric matrices with coordinates $i \in C$, and $\mathbb{S}^C_+$ for the cone of positive semidefinite matrices in $\mathbb{S}^C$. We write $\boldsymbol{X} \succeq \boldsymbol{O}$ when $\boldsymbol{X} \in \mathbb{S}^C_+$ for some $C \subset N$.

To describe *a d-space conversion method*, we consider a general nonlinear optimization problem involving a symmetric positive semidefinite matrix variable $\boldsymbol{X} \in \mathbb{S}^n$:

$$\text{minimize } f_0(\boldsymbol{x}, \boldsymbol{X}) \text{ subject to } \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{X}) \in \Omega \text{ and } \boldsymbol{X} \in \mathbb{S}^n_+, \tag{1}$$

where $f_0 : \mathbb{R}^s \times \mathbb{S}^n \to \mathbb{R}$, $\boldsymbol{f} : \mathbb{R}^s \times \mathbb{S}^n \to \mathbb{R}^m$ and $\Omega \subset \mathbb{R}^m$. We construct *a d-space sparsity pattern graph* $G(N, E)$ with the node set N and an edge set E. Let E be the set of distinct row and column index pairs (i, j) such that a value of X_{ij} is necessary to evaluate $f_0(\boldsymbol{x}, \boldsymbol{X})$ and/or $\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{X})$. More precisely, E is a collection of distinct row and column index pairs (i, j) such that $f_0(\boldsymbol{x}, \boldsymbol{X}^1) \neq f_0(\boldsymbol{x}, \boldsymbol{X}^2)$

* This research was supported by Grant-in-Aid for Scientific Research (B) 22310089.

K. Fukuda et al. (Eds.): ICMS 2010, LNCS 6327, pp. 4–9, 2010.

and/or $\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{X}^1) \neq \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{X}^2)$ for some $\boldsymbol{x} \in \mathbb{R}^s$, $\boldsymbol{X}^1 \in \mathbb{S}^n$ and $\boldsymbol{X}^2 \in \mathbb{S}^n$ satisfying $X^1_{k\ell} = X^2_{k\ell}$ for every $(k,\ell) \neq (i,j), \neq (j,i)$. Note that we identify each edge (j,i) with (i,j) $(i < j)$. Let $G(N, \overline{E})$ be a chordal extension of $G(N,E)$, and $C_1, C_2, \ldots, C_p$ its maximal cliques. Here a graph is said to be chordal if every (simple) cycle of the graph with more than three edges has a chord. See [3] for basic properties on chordal graphs. Since any variable X_{ij} $(i,j) \notin \cup_{k=1}^p C_k$ is not involved in the functions $f_0 : \mathbb{R}^s \times \mathbb{S}^n \to \mathbb{R}$ and $\boldsymbol{f} : \mathbb{R}^s \times \mathbb{S}^n \to \mathbb{R}^m$, we may regard f_0 and $\boldsymbol{f}$ as functions in $\boldsymbol{x} \in \mathbb{R}^s$ and $\boldsymbol{X}(C_k) \in \mathbb{S}^{C_k}$ $(k = 1,2,\ldots,p)$, *i.e.,* there are functions $\tilde{f}_0$ and $\tilde{\boldsymbol{f}}$ in the variables $\boldsymbol{x}$ and $\boldsymbol{X}(C_k) \in \mathbb{S}^{C_k}$ $(k = 1,2,\ldots,p)$ such that

$$f_0(\boldsymbol{x}, \boldsymbol{X}) = \tilde{f}_0(\boldsymbol{x}, \boldsymbol{X}(C_1), \boldsymbol{X}(C_2), \ldots, \boldsymbol{X}(C_p)) \text{ for every } (\boldsymbol{x}, \boldsymbol{X}) \in \mathbb{R}^s \times \mathbb{S}^n,$$
$$\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{X}) = \tilde{\boldsymbol{f}}(\boldsymbol{x}, \boldsymbol{X}(C_1), \boldsymbol{X}(C_2), \ldots, \boldsymbol{X}(C_p)) \text{ for every } (\boldsymbol{x}, \boldsymbol{X}) \in \mathbb{R}^s \times \mathbb{S}^n.$$

Furthermore, applying the positive semidefinite matrix completion (Theorem 7 of [6]), we can replace the positive semidefinite condition $\boldsymbol{X} \in \mathbb{S}^n_+$ by multiple smaller positive semidefinite conditions $\boldsymbol{X}(C_k) \in \mathbb{S}^{C_k}_+$ $(k = 1,2,\ldots,p)$ to convert the problem (1) to an equivalent problem

$$\begin{array}{ll} \text{minimize} & \tilde{f}_0(\boldsymbol{x}, \boldsymbol{X}(C_1), \boldsymbol{X}(C_2), \ldots, \boldsymbol{X}(C_p)) \\ \text{subject to} & \tilde{\boldsymbol{f}}(\boldsymbol{x}, \boldsymbol{X}(C_1), \boldsymbol{X}(C_2), \ldots, \boldsymbol{X}(C_p)) \in \Omega \text{ and} \\ & \boldsymbol{X}(C_k) \in \mathbb{S}^{C_k}_+ \ (k = 1,2,\ldots,p). \end{array} \tag{2}$$

If every clique C_k is small $(k = 1,2,\ldots,p)$, then the number of real variables X_{ij} involved in the converted problem (2) is smaller than that in the original problem (1). The d-space conversion method described above is an extension of the conversion method proposed in the papers [5,10] for a linear SDP problem to a general nonlinear case.

Now we describe *an r-space conversion method* briefly. Let $\boldsymbol{M} : \mathbb{R}^s \to \mathbb{S}^n$. Consider a matrix inequality

$$\boldsymbol{M}(\boldsymbol{y}) \in \mathbb{S}^n_+. \tag{3}$$

We assume a similar chordal graph structured sparsity as the d-space conversion method. Let E be the set of distinct row and column index pairs (i,j) of the mapping $\boldsymbol{M}$ such that M_{ij} is not identically zero, *i.e.*, $M_{ij}(\boldsymbol{y}) \neq 0$ for some $\boldsymbol{y} \in \mathbb{R}^s$. We call a graph $G(N,E)$ with the node set N and the edge set E *an r-space sparsity pattern graph.* Let $G(N, \overline{E})$ be a chordal extension of $G(N,E)$, and $C_1, C_2, \ldots, C_p$ its maximal cliques. Applying a dual of the positive matrix completion (Theorem 2.3 of [1], see also Theorems 4.1 and 4.2 of [7]), we can convert the matrix inequality (3) to a family of multiple inequalities

$$\widetilde{\boldsymbol{M}}_k(\boldsymbol{y}) - \widetilde{\boldsymbol{L}}_k(\boldsymbol{z}) \in \mathbb{S}^{C_k}_+ \ (k = 1,2,\ldots,p) \tag{4}$$

for some mappings $\widetilde{\boldsymbol{M}}_k$ from $\mathbb{R}^s$ into $\mathbb{S}^{C_k}$ $(k = 1,2,\ldots,p)$ and some linear mappings $\widetilde{\boldsymbol{L}}_k$ from $\mathbb{R}^q$ into $\mathbb{S}^{C_k}$ $(k = 1,2,\ldots,p)$. The matrix inequality (3) is equivalent to the family (4) of matrix inequalities in the sense that $\boldsymbol{y} \in \mathbb{R}^s$

satisfies (3) if and only if $\boldsymbol{y} \in \mathbb{R}^s$ satisfies (4) for some $\boldsymbol{z} \in \mathbb{R}^q$. The dimension q of the auxiliary variable vector $\boldsymbol{z}$ is determined by the r-space sparsity pattern graph $G(N, E)$. For example, if $\boldsymbol{M}$ is tridiagonal, the sizes of $\widetilde{\boldsymbol{M}}_k$ and $\widetilde{\boldsymbol{L}}_k$ are all 2×2 and $q = n - 2$.

To illustrate the d-space and r-space conversion methods, we show a simple SDP problem from the paper [7]. Let $\boldsymbol{A}^0$ be a tridiagonal matrix in $\mathbb{S}^n$ such that $A^0_{ij} = 0$ if $|i - j| > 1$, and define a mapping $\boldsymbol{M}$ from $\mathbb{S}^n$ into $\mathbb{S}^n$ by

$$\boldsymbol{M}(\boldsymbol{X}) = \begin{pmatrix} 1 - X_{11} & 0 & 0 & \cdots & 0 & X_{12} \\ 0 & 1 - X_{22} & 0 & \cdots & 0 & X_{23} \\ 0 & 0 & \ddots & & 0 & X_{34} \\ \cdots & \cdots & \cdots & \ddots & \cdots & \cdots \\ 0 & 0 & 0 & & 1 - X_{n-1,n-1} & X_{n-1,n} \\ X_{21} & X_{32} & X_{43} & \cdots & X_{n,n-1} & 1 - X_{nn} \end{pmatrix}$$

for every $\boldsymbol{X} \in \mathbb{S}^n$. Consider an SDP problem

$$\text{minimize } \boldsymbol{A}^0 \bullet \boldsymbol{X} \text{ subject to } \boldsymbol{M}(\boldsymbol{X}) \succeq \boldsymbol{O},\ \boldsymbol{X} \succeq \boldsymbol{O}. \tag{5}$$

Among the elements X_{ij} $(i = 1, 2, \ldots, n,\ j = 1, 2, \ldots, n)$ of the matrix variable $\boldsymbol{X} \in \mathbb{S}^n$, the elements X_{ij} with $|i-j| \leq 1$ are relevant and all other elements X_{ij} with $|i-j| > 1$ are unnecessary in evaluating the objective function $\boldsymbol{A}^0 \bullet \boldsymbol{X}$ and the matrix inequality $\boldsymbol{M}(\boldsymbol{X}) \succeq \boldsymbol{O}$. Hence, we can describe the d-space sparsity pattern as an $n \times n$ symbolic tridiagonal matrix with the nonzero symbol $\star$

$$\begin{pmatrix} \star & \star & 0 & \cdots & 0 & 0 \\ \star & \star & \star & \cdots & 0 & 0 \\ 0 & \star & \star & \ddots & 0 & 0 \\ \cdots & \cdots & \ddots & \ddots & \ddots & \cdots \\ 0 & 0 & \cdots & \ddots & \star & \star \\ 0 & 0 & \cdots & \cdots & \star & \star \end{pmatrix}.$$

Figure 1 shows the d-space sparsity pattern graph $G(N, E)$, which is apparently chordal because there is no cycle. Hence $G(N, \overline{E}) = G(N, E)$ with $\overline{E} = E = \{(i, j) \in N \times N : |i - j| = 1\}$.

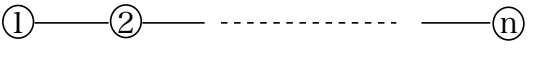


Fig. 1. The d-space sparsity pattern graph

On the other hand, the r-space sparsity pattern is described as

$$\begin{pmatrix} \star & 0 & \dots & 0 & \star \\ 0 & \star & \dots & 0 & \star \\ \dots & \dots & \ddots & \dots & \dots \\ 0 & 0 & \dots & \star & \star \\ \star & \star & \dots & \star & \star \end{pmatrix}.$$

Figure 2 shows the r-space sparsity pattern graph $G(N, E)$, which is apparently chordal because there is no cycle. Hence $G(N, \overline{E}) = G(N, E)$ with $\overline{E} = E = \{(i, n) \in N \times N : i = 1, 2, \dots, n-1\}$.

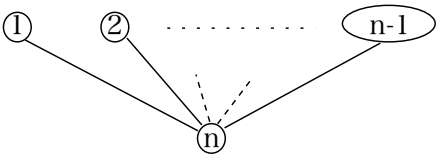


Fig. 2. The r-space sparsity pattern graph

Applying the d-space and r-space conversion methods, we can reduce the SDP problem (5) to

$$\left.\begin{array}{ll} \text{minimize} & \displaystyle\sum_{i=1}^{n-1} \left(A^0_{ii} X_{ii} + 2A^0_{i,i+1} X_{i,i+1}\right) + A^0_{nn} X_{nn} \\ \text{subject to} & \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} - \begin{pmatrix} X_{11} & -X_{12} \\ -X_{21} & -z_1 \end{pmatrix} \succeq \boldsymbol{O}, \\ & \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} - \begin{pmatrix} X_{ii} & -X_{i,i+1} \\ -X_{i+1,i} & z_{i-1} - z_i \end{pmatrix} \succeq \boldsymbol{O} \ (i = 2, 3, \dots, n-2), \\ & \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} - \begin{pmatrix} X_{n-1,n-1} & -X_{n-1,n} \\ -X_{n,n-1} & X_{n,n} + z_{n-2} \end{pmatrix} \succeq \boldsymbol{O}, \\ & \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} - \begin{pmatrix} -X_{ii} & -X_{i,i+1} \\ -X_{i+1,i} & -X_{i+1,i+1} \end{pmatrix} \succeq \boldsymbol{O} \ (i = 1, 2, \dots, n-1). \end{array}\right\} \quad (6)$$

This problem has $(3n-3)$ real variables X_{ii} $(i = 1, 2, \dots, n)$, $X_{i,i+1}$ $(i = 1, 2, \dots, n-1)$ and z_i $(i = 1, 2, \dots, n-2)$, and $(2n-1)$ linear matrix inequalities with size 2×2. Since the original SDP problem (5) involves an $n \times n$ matrix variable $\boldsymbol{X}$ and an $n \times n$ matrix inequality $\boldsymbol{M}(\boldsymbol{X}) \succeq \boldsymbol{O}$, we can expect to solve the SDP problem (6) much more efficiently than the SDP problem (5) as n becomes larger.

Table 1 shows numerical results on the SDP problems (5) and (6) solved by SDPA 7.3.0 [11]. To convert (5) to (6), we used SparseCoLO [4,13], which is a MATLAB implementation of the d-space and r-space conversion methods for linear SDP problems. The numerical experiment was done on 3.06 GHz Intel Core 2 Duo with 8GB Memory. As we have mentioned above, we observe that the

Table 1. Numerical results on the SDP problems (5) and (6). Here 'SDPA' denotes the elapsed time to solve the SDP problems (5) or (6) by SDPA, 'Total' the total elapsed time including the conversion from the SDP problem (5) to (6) by SparseCoLO, '#var.' the number of variables, 'Size' the size of linear matrix inequalities and '#' the number of linear matrix inequalities.

	SDP (5)				SDP (6)				
	Elapsed time		LMIs		Elapsed time			LMIs	
n	SDPA	#var.	Size	#	SDPA	Total	#var.	Size	#
10	0.1	55	10	1	0.2	0.3	27	2	19
100	107.5	5,050	100	1	0.4	0.8	297	2	199
1000	Out of memory	500,500	1,000	1	2.7	25.0	2,997	2	1,999
2000	Out of memory	2,001,000	2,000	1	5.4	136.4	5,997	2	3,999
4000	Out of memory	8,002,000	4,000	1	11.1	985.7	11,997	2	7,999

number of variables of the converted SDP problem (6) is much smaller than that of the original SDP problem (5), and that the size of LMIs of (6) remains constant 2 even when n increases. These two factors contributed to shorter elapsed time to solve the converted SDP problem (6) with larger n.

We finally mention two software packages which utilize the basic idea of the d-space conversion method for linear SDP problems besides SparseCoLO [4,13] referred above. The one is SparsePOP [14,15], which is a MATLAB implementation of a sparse version of Lasserre's hierarchy of SDP relaxations [9], for solving polynomial optimization problems. The other is SFSDP [8,12] for sensor network localization problems, where the d-space conversion method was successfully used to considerably improve the efficiency of the full SDP [2].

References

1. Agler, J., Helton, J., McCullough, S., Rodman, L.: Positive Semidefinite Matrices with a Given Sparsity Pattern. Linear Algebra Appl. 107, 101–149 (1988)
2. Biswas, P., Ye, Y.: Semidefinite Programming for Ad Hoc Wireless Sensor Network Localization. In: Proceedings of the Third International Symposium on Information Processing in Sensor Networks, pp. 46–54. ACM Press, New York (2004)
3. Blair, J.R.S., Peyton, B.: An Introduction to Chordal Graphs and Clique Trees. In: George, A., Gilbert, J.R., Liu, J.W.H. (eds.) Graph Theory and Sparse Matrix Computation, pp. 1–29. Springer, New York (1993)
4. Fujisawa, K., Kim, S., Kojima, M., Okamoto, Y., Yamashita, M.: User's Manual for SparseCoLO: Conversion Methods for SPARSE COnic-form Linear Optimization Problems. Research Report B-453, Dept. of Math. and Comp. Sci., Tokyo Institute of Technology, Tokyo 152-8552, Japan (2009)
5. Fukuda, M., Kojima, M., Murota, K., Nakata, K.: Exploiting Sparsity in Semidefinite Programming via Matrix Completion I: General Framework. SIAM Journal on Optimization 11, 647–674 (2000)
6. Grone, R., Johnson, C.R., Sá, E.M., Wolkowitz, H.: Positive Definite Completions of a Partial Hermitian Matrices. Linear Algebra Appl. 58, 109–124 (1984)

7. Kim, S., Kojima, M., Mevissen, M., Yamashita, M.: Exploiting Sparsity in Linear and Nonlinear Matrix Inequalities via Positive Semidefinite Matrix Completion. Research Report B-452, Dept. of Math. and Comp. Sci., Tokyo Institute of Technology, Tokyo 152-8552, Japan (2009)
8. Kim, S., Kojima, M., Waki, H.: Exploiting Sparsity in SDP Relaxation for Sensor Network Localization. SIAM Journal of Optimization 20, 192–215 (2009)
9. Lasserre, J.B.: Global Optimization with Polynomials and the Problems of Moments. SIAM Journal on Optimization 11, 796–817 (2001)
10. Nakata, K., Fujisawa, K., Fukuda, M., Kojima, M., Murota, K.: Exploiting Sparsity in Semidefinite Programming via Matrix Completion II: Implementation and Numerical Results. Mathematical Programming 95, 303–327 (2003)
11. SDPA's homepage, http://sdpa.indsys.chuo-u.ac.jp/sdpa/
12. SFSDP's homepage, http://www.is.titech.ac.jp/~kojima/SFSDP122/SFSDP.html
13. SparseCoLO's homepage, http://www.is.titech.ac.jp/~kojima/SparseCoLO/SparseCoLO.htm
14. SparsePOP's homepage, http://www.is.titech.ac.jp/~kojima/SparsePOP/SparsePOP.html
15. Waki, H., Kim, S., Kojima, M., Muramatsu, M., Sugimoto, H.: SparsePOP: a Sparse Semidefinite Programming Relaxation of Polynomial Optimization Problems. ACM Transactions on Mathematical Software 35, 15 (2008)

Reliable and Efficient Geometric Computing

Kurt Mehlhorn

Max-Planck-Institut für Informatik

Computing with geometric objects (points, curves, and surfaces) is central for many engineering disciplines and lies at the heart of computer aided design systems. Implementing geometric algorithms is notoriously difficult and most actual implementations are incomplete: they are known to crash or deliver the wrong result on some instances.

In the introductory part of the talk, I illustrate the *pitfalls of geometric computing* [KMP+08] and explain for one algorithm in detail where the problem lies and what goes wrong.

In the main part of the talk I discuss approaches to reliable and efficient geometric computing. I will concentrate on the *exact computation paradigm* [FvW93, Yap97, MN99] and briefly touch *controlled perturbation* [HS98, MOS10]. I will report about theoretical and practical advances [Ker09, BKS08, Ker09, MS, Eme10b, Eme10a, EBS09] and the use of the paradigms in systems LEDA [MN99], CGAL [CGA], and EXACUS [EXA].

References

[BKS08] Berberich, E., Kerber, M., Sagraloff, M.: Exact geometric-topological analysis of algebraic surfaces. In: SoCG, pp. 164–173 (2008)

[CGA] CGAL (Computational Geometry Algorithms Library), `http://www.cgal.org`

[EBS09] Emeliyanenko, P., Berberich, E., Sagraloff, M.: Visualizing arcs of implicit algebraic curves, exactly and fast. In: ISVC 2009: Proceedings of the 5th International Symposium on Advances in Visual Computing, pp. 608–619. Springer, Heidelberg (2009)

[Eme10a] Emeliyanenko, P.: A complete modular resultant algorithm targeted for realization on graphics hardware. In: PASCO 2010. ACM, New York (to appear 2010)

[Eme10b] Emeliyanenko, P.: Modular Resultant Algorithm for Graphics Processors. In: ICA3PP 2010, pp. 427–440. Springer, Heidelberg (2010)

[EXA] EXACUS (EXAct computation with CUrves and Surfaces), `http://www.mpi-sb.mpg.de/projects/EXACUS`

[FvW93] Fortune, S., van Wyk, C.: Efficient exact integer arithmetic for computational geometry. In: 7th ACM Conference on Computational Geometry, pp. 163–172 (1993)

[HS98] Halperin, D., Shelton, C.: A perturbation scheme for spherical arrangements with application to molecular modeling. CGTA: Computational Geometry: Theory and Applications 10 (1998)

[Ker09] Kerber, M.: Geometric Algorithms for Algebraic Curves and Surfaces. PhD thesis, Saarland University (2009)

K. Fukuda et al. (Eds.): ICMS 2010, LNCS 6327, pp. 10–11, 2010.

[KMP+08] Kettner, L., Mehlhorn, K., Pion, S., Schirra, S., Yap, C.: Classroom Examples of Robustness Problems in Geometric Computations. Computational Geometry: Theory and Applications (CGTA) 40, 61–78 (2008); A preliminary version appeared in ESA 2004. LNCS, vol. 3221, pp. 702–713. Springer, Heidelberg (2004)

[MN99] Mehlhorn, K., Näher, S.: The LEDA Platform for Combinatorial and Geometric Computing. Cambridge University Press, Cambridge (1999)

[MOS10] Mehlhorn, K., Osbild, R., Sagraloff, M.: A General Approach to the Analysis of Controlled Perturbation Algorithms. Submitted for publication a preliminary version appeared in ICALP 2006 (February 2010)

[MS] Mehlhorn, K., Sagraloff, M.: A Deterministic Descartes Algorithm for Real Polynomials. In: ISSAC 2009 (2009)

[Yap97] Yap, C.-K.: Towards exact geometric computation. CGTA: Computational Geometry: Theory and Applications 7 (1997)

The Sage Project: Unifying Free Mathematical Software to Create a Viable Alternative to Magma, Maple, Mathematica and MATLAB

Burçin Eröcal[1] and William Stein[2]

[1] Research Institute for Symbolic Computation
Johannes Kepler University,
Linz, Austria
Supported by FWF grants P20347 and DK W1214
burcin@erocal.org
[2] Department of Mathematics
University of Washington
wstein@uw.edu
Supported by NSF grant DMS-0757627 and DMS-0555776

Abstract. Sage is a free, open source, self-contained distribution of mathematical software, including a large library that provides a unified interface to the components of this distribution. This library also builds on the components of Sage to implement novel algorithms covering a broad range of mathematical functionality from algebraic combinatorics to number theory and arithmetic geometry.

Keywords: Python, Cython, Sage, Open Source, Interfaces.

1 Introduction

In order to use mathematical software for exploration, we often push the boundaries of available computing resources and continuously try to improve our implementations and algorithms. Most mathematical algorithms require basic building blocks, such as multiprecision numbers, fast polynomial arithmetic, exact or numeric linear algebra, or more advanced algorithms such as Gröbner basis computation or integer factorization. Though implementing some of these basic foundations from scratch can be a good exercise, the resulting code may be slow and buggy. Instead, one can build on existing optimized implementations of these basic components, either by using a general computer algebra system, such as Magma, Maple, Mathematica or MATLAB, or by making use of the many high quality open source libraries that provide the desired functionality. These two approaches both have significant drawbacks. This paper is about Sage,[1] which provides an alternative approach to this problem.

[1] http://www.sagemath.org

K. Fukuda et al. (Eds.): ICMS 2010, LNCS 6327, pp. 12–27, 2010.

Having to rely on a closed propriety system can be frustrating, since it is difficult to gain access to the source code of the software, either to correct a bug or include a simple optimization in an algorithm. Sometimes this is by design:

> "Indeed, in almost all practical uses of Mathematica, issues about how Mathematica works inside turn out to be largely irrelevant. You might think that knowing how Mathematica works inside would be necessary [...]" (See [Wol].)

Even if we manage to contact the developers, and they find time to make the changes we request, it might still take months or years before these changes are made available in a new release.

Fundamental questions of correctness, reproducibility and scientific value arise when building a mathematical research program on top of proprietary software (see, e.g., [SJ07]). There are many published refereed papers containing results that rely on computations performed in Magma, Maple, or Mathematica.[2] In some cases, a specific version of Magma is the only software that can carry out the computation. This is not the infrastructure on which we want to build the future of mathematical research.

In sharp contrast, open source libraries provide a great deal of flexibility, since anyone can see and modify the source code as they wish. However, functionality is often segmented into different specialized libraries and advanced algorithms are hidden behind custom interpreted languages. One often runs into trouble trying to install dependencies before being able use an open source software package. Also, converting the output of one package to the input format of another package can present numerous difficulties and introduce subtle errors.

Sage, which started in 2005 (see [SJ05]), attacks this problem by providing:

1. a *self-contained distribution* of mathematical software that installs from source easily, with the only dependency being compiler tools,
2. *unified interfaces* to other mathematical software to make it easier to use all these programs together, and
3. a *new library* that builds on the included software packages and implements a broad range of mathematical functionality.

The rest of this paper goes into more detail about Sage. In Section 1.1, we describe the Sage graphical user interface. Section 1.2 is about the Sage development process, Sage days workshops, mailing lists, and documentation. The subject of Section 2 is the sophisticated way in which Sage is built out of a wide range of open source libraries and software. In Section 2.1 we explain how we use Python and Cython as the glue that binds the compendium of software included in Sage into a unified whole. We then delve deeper into Python, Cython and the Sage preparser in Section 2.2, and illustrate some applications to mathematics in Section 2.3. Sage is actively used for research, and in Section 3 we describe some capabilities of Sage in advanced areas of mathematics.

[2] Including by the second author of this paper, e.g., [CES03]!

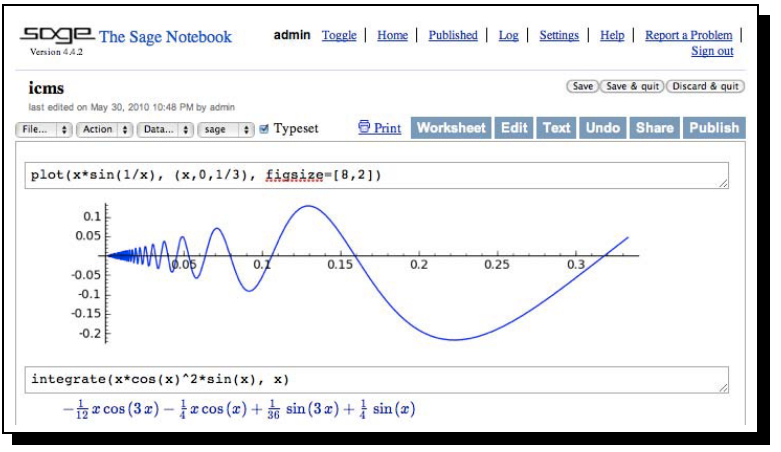


Fig. 1. The Sage Notebook

1.1 The Notebook

As illustrated in Figure 1, the graphical user interface for Sage is a web application, inspired by Google Documents [Goo], which provides convenient access to all capabilities of Sage, including 3D graphics. In single user mode, Sage works like a regular application whose main window happens to be your web browser. In multiuser mode, this architecture allows users to easily set up servers for accessing their work over the Internet as well as sharing and collaborating with colleagues. One can try the Sage notebook by visiting `www.sagenb.org`, where there are over 30,000 user accounts and over 2,000 published worksheets.

Users also download Sage to run it directly on their computers. We track all downloads from `www.sagemath.org`, though there are several other high-profile sites that provide mirrors of our binaries. Recently, people download about 6,000 copies of Sage per month directly from the Sage website.

1.2 The Sage Development Process

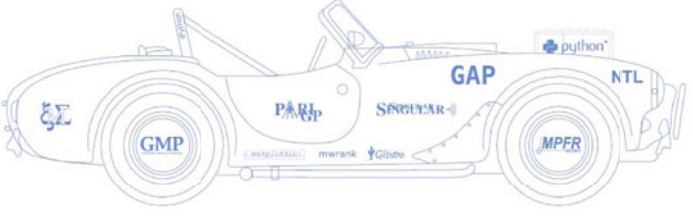
There are over 200 developers from across the world who have contributed to the Sage project. People often contribute because they write code using Sage as part of a research project, and in this process find and fix bugs, speed up parts of Sage, or want the code portion of their research to be peer reviewed. Each contribution to Sage is first posted to the Sage Trac server `trac.sagemath.org`; it is then peer reviewed, and finally added to Sage after all issues have been sorted out and all requirements are met. Nothing about this process is anonymous; every step of the peer review process is recorded indefinitely for all to see.

The Sage Developer's Guide begins with an easy-to-follow tutorial that guides developers through each step involved in contributing code to Sage. Swift feedback is available through the `sage-devel` mailing list, and the `#sage-devel` IRC chat room on `irc.freenode.net` (see `www.sagemath.org/development.html`).

Much development of Sage has taken place at the Sage Days workshops. There have been two dozen Sage Days [Sagb] and many more are planned. These are essential to sustaining the momentum of the Sage project and also help ensure that developers work together toward a common goal, rather than competing with each other and fragmenting our limited community resources.

A major goal is ensuring that there will be many Sage Days workshops for the next couple of years. The topics will depend on funding, but will likely include numerical computation, large-scale bug fixing, L-functions and modular forms, function fields, symbolic computation, topology, and combinatorics. The combination of experienced developers with a group of enthusiastic mathematicians at each of these workshops has rapidly increased the developer community, and we hope that it will continue to do so.

2 Building the Car...



With the motto "building the car instead of reinventing the wheel," Sage brings together numerous open source software packages (see Table 1 and [Saga]).

Many applications of Sage require using these libraries together. Sage handles the conversion of data behind the scenes, automatically using the best tool for the job, and allows the user to concentrate on the problem at hand.

In the following example, which we explain in detail below, Sage uses the FLINT library [HH] for univariate polynomials over the ring $\mathbb{Z}$ of integers, whereas Singular [DGPS10] is used for multivariate polynomials. The option to use the NTL library [Sho] for univariate polynomials is still available, if the user so chooses.

```
sage: R.<x> = ZZ[]
sage: type(R.an_element())
<type 'sage.rings...Polynomial_integer_dense_flint'>
sage: R.<x,y> = ZZ[]
sage: type(R.an_element())
<type 'sage.rings...MPolynomial_libsingular'>
sage: R = PolynomialRing(ZZ, 'x', implementation='NTL')
sage: type(R.an_element())
<type 'sage.rings...Polynomial_integer_dense_ntl'>
```

The first line in the example above constructs the univariate polynomial ring `R` $= \mathbb{Z}[x]$, and assigns the variable `x` to be the generator of this ring. Note that $\mathbb{Z}$ is represented by `ZZ` in Sage. The expression `R.<x> = ZZ[]` is not valid

Table 1. Packages Included With Every Copy of Sage-4.4.2

atlas	gap	libgcrypt	palp	scipy_sandbox
blas	gd	libgpg_error	pari	scons
boehm_gc	gdmodule	libm4ri	pexpect	setuptools
boost	genus2reduction	libpng	pil	singular
cddlib	gfan	linbox	polybori	sphinx
cliquer	ghmm	matplotlib	pycrypto	sqlalchemy
cvxopt	givaro	maxima	pygments	sqlite
cython	gnutls	mercurial	pynac	symmetrica
docutils	gsl	moin	python	sympow
ecl	iconv	mpfi	python_gnutls	sympy
eclib	iml	mpfr	r	tachyon
ecm	ipython	mpir	ratpoints	termcap
f2c	jinja	mpmath	readline	twisted
flint	jinja2	networkx	rubiks	weave
flintqs	lapack	ntl	sagenb	zlib
fortran	lcalc	numpy	sagetex	zn_poly
freetype	libfplll	opencdk	scipy	zodb3

Python, but can be used in Sage code as a shorthand as explained in Section 2.2. The next line asks the ring `R` for an element, using the `an_element` function, then uses the builtin Python function `type` to query its type. We learn that it is an instance of the class `Polynomial_integer_dense_flint`. Similarly line 4 constructs $\mathtt{R} = \mathbb{Z}[x, y]$ and line 7 defines $\mathtt{R} = \mathbb{Z}[x]$, but this time using the `PolynomialRing` constructor explicitly and specifying that we want the underlying implementation to use the NTL library.

Often these interfaces are used under the hood, without the user having to know anything about the corresponding systems. Nonetheless, there are easy ways to find out what is used by inspecting the source code, and users are strongly encouraged to cite components they use in published papers. The following example illustrates another way to get a list of components used when a specific command is run.

```
sage: from sage.misc.citation import get_systems
sage: get_systems('integrate(x^2, x)')
['ginac', 'Maxima']
sage: R.<x,y,z> = QQ[]
sage: I = R.ideal(x^2+y^2, z^2+y)
sage: get_systems('I.primary_decomposition()')
['Singular']
```

2.1 Interfaces

Sage makes it possible to use a wide range of mathematical software packages together by providing a unified interface that handles data conversion automatically. The complexity and functionality of these interfaces varies greatly, from simple text-based interfaces that call external software for an individual

Table 2. Sage Interfaces to the above Mathematical Software

Pexpect	axiom, ecm, fricas, frobby, gap, g2red, gfan, gnuplot, gp, kash, lie, lisp, macaulay2, magma, maple, mathematica, matlab, maxima, mupad, mwrank, octave, phc, polymake, povray, qepcad, qsieve, r, rubik, scilab, singular, tachyon
C Library	eclib, fplll, gap (in progress), iml, linbox, maxima, ratpoints, r (via rpy2), singular, symmetrica
C Library arithmetic	flint, mpir, ntl, pari, polybori, pynac, singular

computation, to using a library as the basis for an arithmetic type. The interfaces can also run code from libraries written in the interpreted language of another program. Table 2 lists the interfaces provided by Sage.

The above interfaces are the result of many years writing Python and Cython [BBS] code to adapt Singular [DGPS10], GAP [L+], Maxima [D+], Pari [PAR], GiNaC/Pynac [B+], NTL [Sho], FLINT [HH], and many other libraries, so that they can be used smoothly and efficiently in a unified way from Python [Ros]. Some of these programs were originally designed to be used only through their own interpreter and made into a library by Sage developers. For example libSingular was created by Martin Albrecht in order to use the fast multivariate polynomial arithmetic in Singular from Sage. The libSingular interface is now used by other projects, including Macaulay2 [GS] and GFan [Jen].

There are other approaches to linking mathematical software together. The recent paper [LHK+] reports on the state of the art using OpenMath. Sage takes a dramatically different approach to this problem. Instead of using a general string-based XML protocol to communicate with other mathematical software, Sage interfaces are tailor made to the specific software and problem at hand. This results in far more efficient and flexible interfaces. The main disadvantage compared to OpenMath is that the interfaces all go through Sage.

Having access to many programs which can perform the same computation, without having to worry about data conversion, also makes it easier to double check results. For example, below we first use Maxima, an open source symbolic computation package distributed with Sage, to integrate a function, then perform the same computation using Maple and Mathematica.

```
sage: var('x')
sage: integrate(sin(x^2), x)
1/8*((I - 1)*sqrt(2)*erf((1/2*I - 1/2)*sqrt(2)*x) + \
  (I + 1)*sqrt(2)*erf((1/2*I + 1/2)*sqrt(2)*x))*sqrt(pi)
sage: maple(sin(x^2)).integrate(x)
1/2*2^(1/2)*Pi^(1/2)*FresnelS(2^(1/2)/Pi^(1/2)*x)
sage: mathematica(sin(x^2)).Integrate(x)
Sqrt[Pi/2]*FresnelS[Sqrt[2/Pi]*x]
```

The most common type of interface, called a `pexpect` interface, communicates with another command line program by reading and writing strings to a text console, as if another user was in front of the terminal. Even though these are

relatively simple to develop, the overhead of having to print and parse strings to represent the data makes this process potentially cumbersome and inefficient. This is the default method of communication with most high level mathematics software, including commercial and open source programs, such as Maple, Mathematica, Magma, KASH or GAP.

Sage provides a framework to represent elements over these interfaces, perform arithmetic with them or apply functions to the given object, as well as using a file to pass the data if the string representation is too big. The following demonstrates arithmetic with GAP elements.

```
sage: a = gap('22')
sage: a*a
484
```

It is also possible to use `pexpect` interfaces over remote consoles. In the following code, we connect to the `localhost` as a different user and call Mathematica functions. Note that the interface can handle indexing vectors as well.

```
sage: mma = Mathematica(server="rmma60@localhost")
sage: mma("2+2")
4
sage: t = mma("Cos[x]")
sage: t.Integrate('x')
Sin[x]
sage: t = mma('{0,1,2,3}')
sage: t[2]
1
```

Sage also includes specialized libraries that are linked directly from compiled code written in Cython. These are used to handle specific problems, such as the characteristic polynomial computation in the example below.

```
sage: M = Matrix(GF(5), 10, 10)
sage: M.randomize()
sage: M.charpoly(algorithm='linbox')
x^10 + 4*x^9 + 4*x^7 + 3*x^4 + 3*x^3 + 3*x^2 + 4*x + 3
```

Many basic arithmetic types also use Cython to directly utilize data structures from efficient arithmetic libraries, such as MPIR or FLINT. An example of this can be seen at the beginning of this section, where elements of the ring $\mathbb{Z}[x]$ are represented by the class `Polynomial_integer_dense_flint`.

The Singular interface is one of the most advanced included in Sage. Singular has a large library of code written in its own language. Previously the only way to access these functions, which include algorithms for Gröbner basis and primary decomposition, was to call Singular through a `pexpect` interface, passing data back and forth using strings. Recently, due to work of Michael Brickenstein and Martin Albrecht, Sage acquired the ability to call these functions directly.

In the example below, we import the function `primdecSY` from `primdec.lib`, and call it the same way we would call a Python function. The interface handles the conversion of the data to Singular's format and back. Since Sage already

uses Singular data structures directly to represent multivariate polynomials and ideals over multivariate polynomial rings, there are no conversion costs. It is only a matter of passing the right pointer.

```
sage: pr = sage.libs.singular.ff.primdec__lib.primdecSY
sage: R.<x,y,z> = QQ[]
sage: p = z^2+1; q = z^3+2
sage: I = R.ideal([p*q^2,y-z^2])
sage: pr(I)
[[[z^2 - y, y^3 + 4*y*z + 4], \
[z^2 - y, y*z + 2, y^2 + 2*z]], \
[[y + 1, z^2 + 1], [y + 1, z^2 + 1]]]
```

Efforts are under way to extend these capabilities to other programs, for example to GAP which provides Sage's underlying group theory functionality. Up to now, GAP was only available through its interpreter, through a `pexpect` interface that was written by Steve Linton. As the following example demonstrates, the performance of this interface is far from ideal.[3]

```
sage: b = gap('10')
sage: b*b
100
sage: timeit('b*b')
625 loops, best of 3: 289 microseconds per loop
```

The code snippet above constructs the element `b` in GAP using the `pexpect` interface, and measures the time it takes to square `b`. Compare these numbers to the following example, which uses the library interface to GAP, recently developed by the second author (but *not* included in Sage yet).

```
sage: import sage.libs.gap.gap as g
sage: a = g.libgap('10'); a
10
sage: type(a)
<type 'sage.libs.gap.gap.GapElement'>
sage: a*a
100
sage: timeit('a*a')
625 loops, best of 3: 229 nanoseconds per loop
```

The library interface is about 1,000 times faster than the pexpect interface.

2.2 Python - A Mainstream Language

In line with the principle of not reinventing the wheel, Sage is built on the mainstream programming language Python, both as the main development language and the user language. This frees the Sage developers, who are mainly mathematicians, from the troubles of language design, and gives access to an immense array of general purpose Python libraries and tools.

[3] All timings in this paper were performed on an 2.66GHz Intel Xeon X7460 based computer.

Python is an interpreted language with a clear, easy to read and learn syntax. Since it is dynamically typed, it is ideal for rapid prototyping, providing an environment to easily test new ideas and algorithms.

A Fast Interpreter. In the following Singular session, we first declare the ring `r` $= \mathbb{Q}[x, y, z]$ and the polynomial `f` $\in$ `r`, then measures the time to square `f` repeatedly, 10,000 times.

```
singular: int t = timer; ring r = 0,(x,y,z), dp;
singular: def f = y^2*z^2-x^2*y^3-x*z^3+x^3*y*z;
singular: int j; def g = f;
singular: for (j = 1; j <= 10^5; j++) { g = f*f; }
singular: (timer-t), system("--ticks-per-sec");
990 1000
```

The elapsed time is 990 milliseconds. Next we use Sage to do the same computation, using the same Singular data structures directly, but without going through the interpreter.

```
sage: R.<x,y,z> = QQ[]
sage: f = y^2*z^2 - x^2*y^3 - x*z^3 + x^3*y*z;  type(f)
<type 'sage.rings.polynomial...MPolynomial_libsingular'>
sage: timeit('for j in xrange(10^5): g = f*f')
5 loops, best of 3: 91.8 ms per loop
```

Sage takes only 91.8 milliseconds for the same operation. This difference is because the Python interpreter is more efficient at performing `for` loops.

Cython - Compiled Extensions. Python alone is too slow to implement a serious mathematical software system. Fortunately, Cython [BBS] makes it easy to optimize parts of your program or access existing C/C++ libraries. It can translate Python code with annotations containing static type information to C/C++ code, which is then compiled as a Python extension module.

Many of the basic arithmetic types in Sage are provided by Cython wrappers of C libraries, such as FLINT for univariate polynomials over $\mathbb{Z}$, Singular for multivariate polynomials, and Pynac for symbolic expressions.

The code segment below defines a Python function to add integers from 0 to N and times the execution of this function with the argument `10^7`.

```
sage: def mysum(N):
....:     s = int(0)
....:     for k in xrange(1,N): s += k
....:     return s
....:
sage: time mysum(10^7)
CPU times: user 0.52 s, sys: 0.00 s, total: 0.52 s
49999995000000
```

Here is the same function, but the loop index `k` is declared to be a C integer and the accumulator `s` is a C `long long`.

```
sage: cython("""
....: def mysum_cython(N):
....:     cdef int k
....:     cdef long long s = 0
....:     for k in xrange(N): s += k
....:     return s
....: """)
sage: time mysum_cython(10^7)
CPU times: user 0.01 s, sys: 0.00 s, total: 0.01 s
49999995000000L
```

The code is compiled and linked to the interpreter on the fly, and the function `mysum_cython` is available immediately. Note that the run time for the Cython function is 60 times faster than the Python equivalent.

Cython also handles the conversion of Python types to C types automatically. In the following example, we call the C function `sinl` using Cython to wrap it in a Python function named `sin_c_wrap`.

```
sage: cython("""
....: cdef extern from "math.h":
....:         long double sinl(long double)
....: def sin_c_wrap(a):
....:         return sinl(a)
....: """)
sage: sin_c_wrap(3.14)
0.0015926529164868282
sage: sin_c_wrap(1)
0.8414709848078965
sage: sin_c_wrap(1r)
0.8414709848078965
```

Note that the conversion of Sage types in the first two calls to `sin_c_wrap` or the Python type integer in the last call is performed transparently by Cython.

The Preparser. While Python has many advantages as a programming and glue language, it also has some undesirable features. Sage hides these problems by using a preparser to change the commands passed to Python in an interactive session (or when running a script with the `.sage` extension). In order to maintain compatibility with Python, changes performed by the preparser are kept to a minimum. Moreover, the Sage library code is not preparsed, and is written in Cython or Python directly.

Python, like C and many other programming languages, performs integer floor division. This means typing `1/2` results in 0, not the rational number 1/2. Sage wraps all numeric literals entered in the command line or the notebook with its own type declarations, which behave as expected with respect to arithmetic and have the advantage that they are backed by efficient multiprecision arithmetic libraries such as MPIR [H+] and MPFR [Z+], which are thousands of times faster than Python for large integer arithmetic.

To call the preparser directly on a given string, use the `preparse` function.

```
sage: preparse("1/2")
'Integer(1)/Integer(2)'
sage: preparse("1.5")
"RealNumber('1.5')"
```

Adding a trailing `r` after a number indicates that the preparser should leave that as the "raw" literal. The following illustrates division with Python integers.

```
sage: preparse("1r/2r")
'1/2'
sage: 1r/2r
0
```

Here is the result of performing the same division in Sage.

```
sage: 1/2
1/2
sage: type(1/2)
<type 'sage.rings.rational.Rational'>
sage: (1/2).parent()
Rational Field
```

The preparser also changes the ^ sign to the exponentiation operator ** and provides a shorthand to create new mathematical domains and name their generator in one command.

```
sage: preparse("2^3")
'Integer(2)**Integer(3)'
sage: preparse("R.<x,y> = ZZ[]")
"R = ZZ['x, y']; (x, y,) = R._first_ngens(2)"
```

2.3 Algebraic, Symbolic and Numerical Tools

Sage combines algebraic, symbolic and numerical computation tools under one roof, enabling users to choose the tool that best suits the problem. This combination also makes Sage more accessible to a wide audience—scientists, engineers, pure mathematicians and mathematics teachers can all use the same platform for scientific computation.

While not concentrating on only one of these domains might seem to divide development resources unnecessarily, it actually results in a better overall experience for everyone, since users do not have to come up with makeshift solutions to compensate for the lack of functionality from a different field. Moreover, because Sage is a distributed mostly-volunteer open source project, widening our focus results in substantially more developer resources.

Algebraic Tools: The Coercion System. An algebraic framework, similar to that of Magma or Axiom, provides access to efficient data structures and specialized algorithms associated to particular mathematical domains. The Python language allows classes to define how arithmetic operations like + and * will be

handled, in a similar way to how C++ allows overloading of operators. However, the built-in support for overloading in Python is too simple to support operations with a range of objects in a mathematical type hierarchy.

Sage abstracts the process of deciding what an arithmetic operation means, or equivalently, in which domain the operation should be performed, in a framework called the *coercion system*, which was developed and implemented by Robert Bradshaw, David Roe, and many others. Implementations of new mathematical objects only need to define which other domains have a natural embedding to their domain. When performing arithmetic with objects, the coercion system will find a common domain where both arguments can be canonically mapped, perform the necessary type conversions automatically, thus allowing the implementation to only handle the case where both objects have the same parent.

In the following example, the variable `t` is an element of $\mathbb{Z}$ whereas `u` is in $\mathbb{Q}$. In order to perform the addition, the coercion system first deduces that the result should be in $\mathbb{Q}$ from the fact that `t` can be converted to the domain of `u`, namely $\mathbb{Q}$, but canonical conversion in the other direction is not possible. Then the addition is performed with both operands having the same domain $\mathbb{Q}$.

```
sage: t = 1
sage: t.parent()
Integer Ring
sage: u = 1/2
sage: u.parent()
Rational Field
sage: v = t + u; v
3/2
sage: v.parent()
Rational Field
```

Similarly, in the following example, the common domain $\mathbb{Q}[x]$ is found for arguments from $\mathbb{Z}[x]$ and $\mathbb{Q}$. Note that in this case, the result is not in the domain of either of the operands.

```
sage: R.<x> = ZZ[]
sage: r = x + 1/2
sage: r.parent()
Univariate Polynomial Ring in x over Rational Field
sage: 5*r
5*x + 5/2
```

Algebraic Tools: The Category Framework. Another abstraction to make implementing mathematical structures easier is the *category framework*, whose development was spearheaded by Nicolas Thiéry and Florent Hivert. Similar in spirit to the mathematical programming facilities developed in Axiom and encapsulated in Aldor, the category framework uses Python's dynamic class creation capabilities to combine functions relevant for a mathematical object, inherited through a mathematical hierarchy, into a class at run time.

This process greatly simplifies the troubles of having to combine object-oriented programming concepts with mathematical structural concerns, while

keeping efficiency in mind. Efficient implementations can keep the inheritance hierarchy imposed by the data structures, while generic methods to compute basic properties are implemented in the *category* and automatically attached to the element classes when they are needed.

Symbolic Tools. The symbolic subsystem of Sage provides an environment similar to Maple or Mathematica, where the input is treated only as an expression without any concern about the underlying mathematical structure.

Sage uses Pynac [ES], a hybrid C++ and Cython library built on top of GiNaC [B+], to work with symbolic expressions. High level symbolic calculus problems including symbolic integration, solution of differential equations and Laplace transforms are solved using Maxima behind the scenes.

Here is an example of how to use the symbolic computation facilities in Sage. Note that in contrast to other symbolic software such as Maple, variables must be declared before they are used.

```
sage: x,y,z = var('x,y,z')
sage: sin(x).diff(x)
cos(x)
sage: psi(x).series(x,4)
(-1)*x^(-1) + (-euler_gamma) + (1/6*pi^2)*x + \
(-zeta(3))*x^2 + (1/90*pi^4)*x^3 + Order(x^4)
sage: w = SR.wild()  # wildcard for symbolic substitutions
sage: ((x^2+y^2+z^2)*zeta(x)).subs({w^2:5})
15*zeta(x)
```

Numerical Tools. In addition to code for symbolic computation, the standard numerical Python packages NumPy, SciPy, and Matplotlib are included in Sage, along with the numerical libraries cvxopt, GSL, Mpmath, and R.

For numerical applications, Robert Bradshaw and Carl Witty developed a compiler for Sage that converts symbolic expressions into an internal format suitable for blazingly fast floating point evaluation.

```
sage: f(x,y) = sqrt(x^2 + y^2)
sage: a = float(2)
sage: timeit('float(f(a,a))')
625 loops, best of 3: 216 microseconds per loop
sage: g = fast_float(f)
sage: timeit('float(g(a,a))')
625 loops, best of 3: 0.406 microseconds per loop
```

The `fast_float` feature is automatically used by the `minimize` command.

```
sage: minimize(f, (a,a))
(-5.65756135618e-05, -5.65756135618e-05)
```

Performance is typically within a factor of two from what one gets using a direct implementation in C or Fortran.

3 Afterword

In this article, we have showed that Sage is a powerful platform for developing sophisticated mathematical software. Sage is actively used in research mathematics, and people use Sage to develop state-of-the-art algorithms. Sage is particularly strong in number theory, algebraic combinatorics, and graph theory. For further examples, see the 53 published articles, 11 Ph.D. theses, 10 books, and 30 preprints at `www.sagemath.org/library-publications.html`

For example, Sage has extensive functionality for computations related to the Birch and Swinnerton-Dyer conjecture. In addition to Mordell-Weil group computations using [Cre] and point counting over large finite fields using the SEA package in [PAR], there is much novel elliptic curve code written directly for Sage. This includes the fastest known algorithm for computation of p-adic heights [Har07, MST06], and code for computing p-adic L-series of elliptic curves at ordinary, supersingular, and split multiplicative primes. Sage combines these capabilities to compute explicitly bounds on Shafarevich-Tate groups of elliptic curves [SW10]. Sage also has code for computation with modular forms, modular abelian varieties, and ideal class groups in quaternion algebras.

The MuPAD-combinat project, which was started by Florent Hivert and Nicolas M. Thiéry in 2000, built the world's preeminent system for algebraic combinatorics on top of MuPAD (see [Des06] and [HT05]). Page 54 of [HT05]: "They [MuPAD] also have promised to release the code source of the library under a well known open-source license, some day." In 2008, MuPAD was instead purchased by MathWorks (makers of MATLAB), so MuPAD is no longer available as a separate product, and will probably never be open source. Instead it now suddenly costs $3000 (commercial) or $700 (academic).

As a result, the MuPAD-combinat group has spent several years reimplementing everything in Sage (see [T$^+$] for the current status). The MuPAD-combinat group was not taken by surprise by the failure of MuPAD, but instead were concerned from the beginning by the inherent risk in building their research program on top of MuPAD. In fact, they decided to switch to Sage two months before the bad news hit, and have made tremendous progress porting:

> "It has been such a relief during the last two years not to have this Damocles sword on our head!"
> – Nicolas Thiéry

References

[B$^+$] Bauer, C., et al.: Ginac: is not a CAS, `http://www.ginac.de/`

[BBS] Behnel, S., Bradshaw, R., Seljebotn, D.: Cython: C-Extensions for Python, `http://www.cython.org/`

[CES03] Conrad, B., Edixhoven, S., Stein, W.A.: $J_1(p)$ Has Connected Fibers. Documenta Mathematica 8, 331–408 (2003)

[Cre] Cremona, J.E.: mwrank (computer software), `http://www.warwick.ac.uk/staff/J.E.Cremona/mwrank/`

[D+] Dodier, R., et al.: Maxima: A Computer Algebra System, http://maxima.sourceforge.net/

[Des06] Descouens, F.: Making research on symmetric functions with MuPAD-Combinat. In: Iglesias, A., Takayama, N. (eds.) ICMS 2006. LNCS, vol. 4151, pp. 407–418. Springer, Heidelberg (2006)

[DGPS10] Decker, W., Greuel, G.-M., Pfister, G., Schönemann, H.: SINGULAR 3-1-1 — A computer algebra system for polynomial computations, http://www.singular.uni-kl.de

[ES] Erocal, B., Stein, W.: Pynac – symbolic computation with python objects, http://pynac.sagemath.org/

[Goo] Google, Google Documents, http://docs.google.com/

[GS] Grayson, D.R., Stillman, M.E.: Macaulay2, a software system for research in algebraic geometry, http://www.math.uiuc.edu/Macaulay2/

[H+] Hart, B., et al.: MPIR: Multiprecision Integers and Rationals, http://www.mpir.org/

[Har07] Harvey, D.: Efficient computation of p-adic heights, http://arxiv.org/abs/0708.3404

[HH] Hart, B., Harvey, D.: Flint: Fast library for number theory, http://www.flintlib.org/

[HT05] Hivert, F., Thiéry, N.M.: MuPAD-Combinat, an open-source package for research in algebraic combinatorics. Sém. Lothar. Combin., Art. B51z 51, 70 (2004) (electronic), http://www.emis.de/journals/SLC/wpapers/s51thiery.html

[Jen] Jensen, A.: Gfan: software for computing Gröbner fans and tropical varieties, http://www.math.tu-berlin.de/~jensen/software/gfan/gfan.html

[L+] Linton, S., et al.: Gap: Groups, algorithms and programming, http://www.gap-system.org/

[LHK+] Linton, S., Hammond, K., Konovalov, A., et al.: Easy Composition of Symbolic Computation Software: A New Lingua Franca for Symbolic Computation, www.win.tue.nl/~droozemo/site/pubs/1004ISSAC2010.pdf

[MST06] Mazur, B., Stein, W., Tate, J.: Computation of p-adic heights and log convergence. Doc. Math. Extra, 577–614 (2006) (electronic), MR2290599 (2007i:11089)

[PAR] PARI, A computer algebra system designed for fast computations in number theory, http://pari.math.u-bordeaux.fr/

[Ros] van Rossum, G.: Python, http://www.python.org

[Saga] Sage, Components, http://sagemath.org/links-components.html

[Sagb] Sage, Sage days workshops, http://wiki.sagemath.org/Workshops

[Sho] Shoup, V.: NTL: Number theory library, http://www.shoup.net/ntl/

[SJ05] Stein, W., Joyner, D.: Open source mathematical software. ACM SIGSAM Bulletin 39 (2005)

[SJ07] Stein, W., Joyner, D.: Open source mathematical software. Notices Amer. Math. Soc. (2007), http://www.ams.org/notices/200710/tx071001279p.pdf

[SW10] Stein, W., Wuthrich, C.: Computations About Tate-Shafarevich Groups Using Iwasawa Theory (2010) (in preparation), http://wstein.org/papers/shark/

[T+] Thiery, N., et al.: Sage Combinat Roadmap, http://trac.sagemath.org/sage_trac/wiki/SageCombinatRoadMap

[Wol] Wolfram, Why you do not usually need to know about internals, http://reference.wolfram.com/mathematica/tutorial/WhyYouDoNotUsuallyNeedToKnowAboutInternals.html

[Z+] Zimmerman, P., et al.: The MPFR Library, http://www.mpfr.org/

Sollya: An Environment for the Development of Numerical Codes

Sylvain Chevillard[1], Mioara Joldeş[2], and Christoph Lauter[2,⋆]

[1] INRIA, LORIA, Caramel Project-Team,
BP 239, 54506 Vandœuvre-lès-Nancy Cedex, France
[2] LIP (CNRS/ÉNS de Lyon/INRIA/Université de Lyon), Arénaire Project-Team,
46, allée d'Italie, 69364 Lyon Cedex 07, France

Abstract. Sollya has become a mature tool for the development of numerical software. With about 175 built-in algorithms and a broad extensibility, it offers a complete tool-chain for fixed- and floating-point software and hardware design. Its features include on-the-fly faithful rounding, specialized approximation algorithms and extensive support for floating-point code generation.

Keywords: Numerical software, faithful rounding, computer algebra, development tool, function approximation.

1 Introduction

The software tool Sollya is an interactive environment assisting developers of numerical codes in the design of mathematical routines. It provides a safe and fast experiment bench as well as tools to generate—at least partially—such codes. The users can either use it through an interactive shell (where they enter commands and get results from the tool) or as a scripting language. The language is intended to make the manipulation of numerical expressions particularly easy.

Initially, Sollya was intended more specifically for people implementing numerical functions in mathematical libraries (these functions are, e.g., exp, arccos, tanh, etc.). Since then, the tool has evolved and has now become interesting not only to developers of mathematical libraries, but also to everyone who needs to perform numerical experiments in an environment that is safe with respect to round-off errors. Recently, it has even been used for more ambitious projects, such as MetaLibm[1].

Sollya is a free software distributed under the terms of the CeCILL-C license and available at `http://sollya.gforge.inria.fr/`. The current version is Sollya 2.0, released in April 2010. A complete documentation with tutorials is available [3], and Sollya integrates an interactive help system.

⋆ C. Lauter is now with Intel Corporation, 2111 NE 25th Avenue, M/S JF1-13, Hillsboro, OR, 97124, USA, but he was in the Arénaire Project-Team when he developed Sollya.

[1] See `http://lipforge.ens-lyon.fr/www/metalibm/`

K. Fukuda et al. (Eds.): ICMS 2010, LNCS 6327, pp. 28–31, 2010.

2 Context and Competing Tools

The development of an arithmetical operator, such as a mathematical function f (e.g., $f = \log$) or a conversion operator, usually steps through four phases. In each of these phases, specific questions are addressed. Examples are given below. They can be answered using Sollya.

- Analysis phase: How to classify the numerical behavior of function f? Does it allow for any kind of range reduction (see, e.g., [10, Chapter 11])?
- Approximation phase: How to compute a polynomial p with minimal degree approximating f such that the approximation error stays below some bound?
- Code generation phase: How can an approximation polynomial p be implemented with bounded round-off error?
- Validation phase: What is a safe yet tight bound for the approximation error of p with respect to f? What are appropriate test vectors that exercise the IEEE 754 properties of the operator, such as flag settings [1]?

Most algorithms offered by Sollya are also supported by other tools such as Maple, Mathematica or Matlab. However, when used for the development of numerical codes, these tools show several deficiencies [6]. As their focus is more on general computer algebra, some of their numerical algorithms are not optimized for performance and support for floating- or fixed-point arithmetic is limited. Most importantly, while they generally offer support for arbitrary precision, the actual *accuracy* of computations in these tools is often unknown.

3 Key Features Offered by Sollya

Sollya focuses on providing arbitrary accuracy for function evaluation through on-the-fly faithful rounding. More precisely, the user may define univariate functions as expressions made up of basic functions, such as exp, sin, etc. Sollya can evaluate such function expressions at points providing results with the accuracy for which the user asked. If the working precision needs to be adapted to achieve that accuracy, Sollya takes the burden of doing so off the user.

In this process, Sollya makes sure it never lies: if ever the tool is not able to exhibit a faithful rounding or if rounding might flip the result of a comparison, it will warn the user of that problem with the result. This rigor is achieved through an extensive use of Interval Arithmetic [9], extended to cover functions with false singularities [4]. Interval Arithmetic is of course available at the Sollya interface, too.

Sollya currently supports about 175 commands in a scripting language that enables structured programming. The tool can be extended through dynamically loaded plug-ins. It is out of the scope of this paper to cover this wide range of functionalities. We shall hence concentrate on four outstanding Sollya features.

- Polynomial approximation in sparse monomial bases: Given a function f and a bounded domain I, Sollya can compute an approximation polynomial of

least error in some monomial basis which, in contrast to other tools, may be sparse. So the polynomial may be, e.g., of the form $p(x) = a_2\,x^2 + a_3\,x^3 + a_7\,x^7$. This helps with reducing the number of operations needed to evaluate such approximation polynomials [7].
- Taylor Models of univariate functions: Given a function and a bounded domain I, Sollya can compute a Taylor expansion p of f together with a rigorous interval bound Δ enclosing the error $p(x) - f(x)$ for all $x \in I$. Such Taylor Models allow the behavior of f to be rigorously analyzed. They also help with problems like validated integration [8].
- Supremum norms of approximation error functions: Sollya can compute safe bounds on the supremum norm of the error $\varepsilon = p/f - 1$ made when replacing a function f by a particular polynomial p in a bounded domain I. Such computations are a requirement for code certification for mathematical functions [4].
- Support for code generation in IEEE 754 arithmetic: Sollya offers extensive support for simulating IEEE 754 arithmetic. Additional commands enable the generation of IEEE 754-based C code with bounded round-off error [6].

4 Conclusion and Project Future

Sollya aims at providing a safe environment for numerical computations. Its main feature in comparison to competing tools such as Maple is the fact that the tool always tries to guarantee the numerical quality of the results by giving explicit error bounds.

Sollya is still under development. However it has already been used for several software projects involved with floating-point arithmetic: it has been used for the development of large parts of the CRLibm library[2] [6] and, more recently, for the FLIP library[3] [5]. Several research teams are using the tool for ongoing research (e.g., [2]). Finally, Sollya has been reported (in a private communication) to be used for research and development in industry, where it covers both the hardware as the software side of development in computer arithmetic.

In the future, the following features could be added. First, commands like numerical integration or polynomial interpolation are currently missing. Second, due to its history, Sollya only handles univariate real functions. Originally this choice was reasonable as the focus was on the development of mathematical libraries. But now that a larger community is targeted, support for multivariate functions would be interesting. Complex arithmetic could also be added, with, in particular, commands for computing rational and polynomial best approximations in the complex plane. Finally, support for linear algebra could be added in the long term: algorithms for product and inverse of matrices, linear system solvers, eigensolvers, etc. To follow the Sollya philosophy, these implementations would also adapt their working precision automatically to guarantee the correctness and accuracy of the results.

[2] See http://lipforge.ens-lyon.fr/www/crlibm/
[3] See http://flip.gforge.inria.fr/

References

1. American National Standards Institute (ANSI) and Institute of Electrical and Electronic Engineers (IEEE). IEEE Standard for Binary Floating-Point Arithmetic (IEEE Std754-2008) (2008), Revised version of the IEEE Std754-1985 Standard
2. Arnold, M.G., Collange, S., Defour, D.: Implementing LNS using filtering units of GPUs. In: IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP (2010), http://hal.archives-ouvertes.fr/hal-00423434
3. Chevillard, S., Joldeş, M., Lauter, C.: User's Manual of the Sollya Tool, Release 2.0, http://sollya.gforge.inria.fr/
4. Chevillard, S., Lauter, C.: A certified infinite norm for the implementation of elementary functions. In: Proceedings of the Seventh International Conference on Quality Software, pp. 153–160 (2007)
5. Jeannerod, C.-P., Knochel, H., Monat, C., Revy, G., Villard, G.: A new binary floating-point division algorithm and its software implementation on the ST231 processor. In: Proceedings of the 19th IEEE Symposium on Computer Arithmetic, Portland, OR, USA, pp. 95–103 (June 2009)
6. Lauter, C.: Arrondi correct de fonctions mathématiques. Fonctions univariées et bivariées, certification et automatisation. PhD thesis, École Normale Superieure de Lyon, Université de Lyon (2008)
7. Lauter, C., de Dinechin, F.: Optimizing polynomials for floating-point implementation. In: Proceedings of the 8th Conference on Real Numbers and Computers, Santiago de Compostela, Spain, pp. 7–16 (July 2008)
8. Makino, K., Berz, M.: Taylor models and other validated functional inclusion methods. International Journal of Pure and Applied Mathematics 4(4), 379–456 (2003), http://bt.pa.msu.edu/pub/papers/TMIJPAM03/TMIJPAM03.pdf
9. Moore, R.E.: Methods and Applications of Interval Analysis. Society for Industrial Mathematics, Philadelphia (1979)
10. Muller, J.-M., Brisebarre, N., de Dinechin, F., Jeannerod, C.-P., Lefèvre, V., Melquiond, G., Revol, N., Stehlé, D., Torres, S.: Handbook of Floating-Point Arithmetic. Birkhäuser, Boston (2009), http://www.springer.com/birkhauser/mathematics/book/978-0-8176-4704-9

Validated Special Functions Software

Annie Cuyt, Franky Backeljauw, Stefan Becuwe, and Joris Van Deun

Department of Mathematics and Computer Science
University of Antwerp
Middelheimlaan 1, B-2020 Antwerp, Belgium
annie.cuyt@ua.ac.be

Abstract. Because of the importance of special functions, several books and a large collection of papers have been devoted to the numerical computation of these functions, the most well-known being the NBS handbook by Abramowitz and Stegun. But up to this date, symbolic and numeric environments offer no routines for the validated evaluation of special functions. We point out how a provable correct function evaluation can be returned efficiently.

1 Introduction

Functions that are termed special have explicitly known and simple representations as infinite/asymptotic series and/or continued fractions. Together the convergence domains of these representations often cover the full area of interest for users of these functions. Hence they lend themselves easily for a variable precision implementation. While the use of series to approximate a function in numeric software is well established, that of continued fractions is far from traditional. In [1] we describe how a combination of both techniques leads to validated software. The accumulation of round-off errors is tracked and bounded above while the accumulation of truncation errors is subject to divide-and-conquer.

We assume to have at our disposal a scalable precision, IEEE 754-854 compliant, floating-point implementation of the basic operations, square root and remainder, comparisons, base and type conversions, at least in the rounding mode round-to-nearest. Such an implementation is characterized by four parameters: the internal base β, the precision p and the exponent range $[L, U]$. Here we aim at least at implementations for $\beta = 2$ with precisions $p \geq 53$, and at implementations for use with $\beta = 2^i$ or $\beta = 10^i$ where $i > 1$. The IEEE 754-854 standard was revised in 2008. For our toolkit to be widely applicable, we do not expect the available floating-point implementation to support more advanced features such as exactly rounded mixed precision fused operations (including the assignment operation). We do however assume that the base conversions (between decimal and base β) are exactly rounded.

The goal is to compute a special mathematical quantity such as $\exp(-x^2)$ or $\sqrt{\pi}$ or $\Gamma(1/x)$. We refer to this quantity as $Y = f(y_x)$, where y_x is the argument built from an exact argument x (in base β and precision p) passed by a user, and f is the mathematical expression which is to be evaluated in y_x to yield Y.

K. Fukuda et al. (Eds.): ICMS 2010, LNCS 6327, pp. 32–34, 2010.

Of course y_x and Y suffer several finite precision and truncation errors, which we now analyze.

2 Round-Off Error Accumulation and Control

We denote by $\circledast$ the exactly rounded (to the nearest, with appropriate tiebreaker) floating-point implementation of the basic operation $*$ in the chosen base β and precision p. For floating-point numbers x and y, following the IEEE standards and in the absence of overflow and underflow, the basic operations are carried out with a relative error of at most $u(p) := {}^1\!/_2\,\beta^{-p+1}$ which is also called half a unit-in-the-last-place in precision p:

$$x \circledast y = (x * y)(1+\delta), \quad |\delta| \le u(p), \quad * \in \{+,-,\times,\div\}.$$

The same holds for the square root, the remainder, the conversions between internal formats and the base conversions.

In order to compute a relative error bound for a sequence of operations, it is necessary to keep track of all these error terms. A basic result, given in [2, p. 63], says that if all $|\delta_i| \le u(p)$, $\rho_i = \pm 1$ and $nu(p) < 1$, then

$$\prod_{i=1}^{n}(1+\delta_i)^{\rho_i} = 1+\theta_n, \quad |\theta_n| \le \gamma_n(p) = \frac{nu(p)}{1-nu(p)}. \tag{1}$$

This result is very convenient, as it allows us to rewrite any number of products and quotients of factors $1+\delta_i$ in an error analysis. Note that the reverse does not hold, meaning that not any expression $1+\theta_n$ with θ_n bounded above in absolute value by $\gamma_n(p)$, can be rewritten as a product of n factors $(1+\delta_i)^{\rho_i}$.

Perturbations as in (1) appear in the error analysis of all compound expressions involving the basic operations, square root, remainder and conversions. The values θ_n and bounds $\gamma_n(p)$ keep track of the accumulation of the round-off errors involved.

3 Truncation Error Accumulation and Control

Let $\tilde{Y}_i$ be an approximation of the mathematical quantity Y_i with a relative error ϵ_i,

$$\tilde{Y}_i = Y_i(1+\epsilon_i), \quad i = 1,\ldots,m.$$

Moreover, let the exact quantity Y be given in terms of the Y_i and approximated by the value $\tilde{Y}$, such that with $\sigma_i = \pm 1$,

$$\tilde{Y} = Y(1+\eta_m), \quad 1+\eta_m = \prod_{i=1}^{m}(1+\epsilon_i)^{\sigma_i}, \quad |\eta_m| \le \kappa_m(p).$$

In [1] we show how to distribute the threshold $\kappa_m(p)$ over the individual truncation errors ϵ_i to guarantee that $|\eta_m| \leq \kappa_m(p)$. Usually, the imposed threshold $\kappa_m(p)$ for $|\eta_m|$ is a small multiple of the half unit-in-the-last-place $u(p)$. If

$$|\epsilon_i| \leq \mu_i \frac{\kappa_m(p)}{1+\kappa_m(p)}, \quad i = 1, \ldots, m, \quad \sum_{i=1}^{m} \mu_i = 1, \tag{2}$$

then

$$\prod_{i=1}^{m} |1+\epsilon_i|^{\sigma_i} \leq 1 + \kappa_m(p).$$

The weights $\mu_i, i = 1, \ldots, m$ are chosen depending on the difficulty with which the operands $\tilde{Y}_i$ in the expression for $\tilde{Y}$ are obtained.

4 Putting It All Together

Our aim eventually is to deal with the general situation where

$$\tilde{Y} = Y(1+\eta_m)(1+\theta_n), \quad |(1+\eta_m)(1+\theta_n)| \leq 1 + 2u(p) = 1 + \beta^{-p+1}. \tag{3}$$

Here the floating-point round-off errors δ_i have accumulated in θ_n and all approximation errors of a different nature ϵ_i in η_m.

To achieve (3) all floating-point operations must be carried out in a (slightly larger) working precision $\hat{p}$ than the destination precision p for $\tilde{Y}$. Then the error θ_n is bounded above in absolute value by $\gamma_n(\hat{p})$ which is a fraction of $2u(p)$. In order to guarantee (3), the accumulated error η_m must be bounded above in absolute value by $(2u(p) - \gamma_n(\hat{p}))/(1+\gamma_n(\hat{p}))$. This in turn leads to individual bounds

$$|\epsilon_i| \leq \mu_i \frac{2u(p) - \gamma_n(\hat{p})}{1+2u(p)}, \quad i = 1, \ldots, m.$$

In [1] this toolkit of ideas is illustrated for the computation of the error function and the complementary error function on the real line.

The collection of special functions that can be implemented reliably using this technique includes the incomplete gamma and Gamma functions, Dawson's integral, the error and complementary error function, the Fresnel integrals, the exponential integrals, the hypergeometric and confluent hypergeometric family, the Bessel functions and modified Bessel functions for integer and half integer argument.

References

1. Backeljauw, F., Becuwe, S., Cuyt, A., Van Deun, J.: Validated Evaluation of Special Mathematical Functions. Technical Report 2009-04, Universiteit Antwerpen (2009)
2. Higham, N.J.: Accuracy and Stability of Numerical Algorithms. Second edn. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA (2002)

The Dynamic Dictionary of Mathematical Functions (DDMF)*

Alexandre Benoit, Frédéric Chyzak, Alexis Darrasse, Stefan Gerhold, Marc Mezzarobba, and Bruno Salvy

Inria Paris-Rocquencourt, France
http://ddmf.msr-inria.inria.fr

Abstract. We describe the main features of the Dynamic Dictionary of Mathematical Functions (version 1.5). It is a website consisting of interactive tables of mathematical formulas on elementary and special functions. The formulas are automatically generated by computer algebra routines. The user can ask for more terms of the expansions, more digits of the numerical values, or proofs of some of the formulas.

1 Motivation

Dictionaries of mathematical functions are commonly used by scientists and engineers. Some of the most famous ones are Abramowitz & Stegun's *Handbook of Mathematical Functions* [1]; the Bateman project *Higher Transcendental Functions* [7]; Gradshtein & Ryzhik's *Table of Integrals, Series, and Products* [8]; and the multivolume *Integrals and Series* by Prudnikov, Brychkov, and Marichev [15]. These dictionaries gather formulas such as differential equations, definite and indefinite integrals, inequalities, recurrence relations, power series, asymptotic expansions, approximations, and sometimes graphs and numerical tables, for a large set of functions. They have been prepared by specialists of these functions and carefully checked and proofread. Their success is attested to by the hundreds of thousands of citations they have received [3].

The first editions of those books were published between 60 and 30 years ago.

Since then, the advent of the World Wide Web has changed the way people now look for information. Aware of this change, the NIST has published a new version of [1] in 2010, called the *NIST Handbook of Mathematical Functions* [13] together with a web site, the NIST Digital Library of Mathematical Functions. This site offers navigation in the formulas, active links, export to various formats, and a search engine.

In parallel, computer algebra systems have grown into huge libraries of mathematical algorithms. While the implementation of mathematical functions in these systems is often basically a coding of formulas from the dictionaries mentioned above, the algorithms have matured to a level where many of those formulas can actually be computed automatically.

* This work was supported by the Microsoft Research-Inria Joint Centre.

K. Fukuda et al. (Eds.): ICMS 2010, LNCS 6327, pp. 35–41, 2010.

The aim of the DDMF is to combine recent algorithms in computer algebra together with web interaction into a dictionary of mathematical functions that is automatically generated, easily navigable with export to various formats, and interactive[1]. Interactivity means that formulas or graphics can be adapted to the user's needs; that arbitrary precision can be given on demand; and that proofs can be displayed if desired.

At this stage, the reader is encouraged to have a look at the DDMF at the following url

`http://ddmf.msr-inria.inria.fr`

A typical page is presented in Figure 1.

The rest of this article presents the ideas underlying our current version (1.5), first from the point of view of the document system and then from the computer algebra viewpoint.

2 Dynamic Mathematics on the Web

The language we use to produce the DDMF is called DynaMoW for *Dynamic Mathematics on the Web*. The main principle on which it is based is captured by the following statement:

> *The document being generated by the symbolic computation engine is an object of the language.*

Thus, instead of using a fixed template whose fields are filled in during a computation, the structure of the document itself depends on the results of intermediate computations. For instance, the number of subsections on asymptotic expansions is a result of computing the singularities of the function; the section on symmetries only occurs if the function has been proved even or odd.

DynaMoW is a layer between a symbolic computation engine[2] and a web server. It lets one mix symbolic code together with pieces of documents in a single source code in a natural way. This provides an easy way to showcase computer algebra algorithms to users who do not know the syntax of a computer algebra system: all they need is a web browser; DynaMoW has been designed to be produce pages compatible with the most popular ones.

Moreover, once the document becomes part of the computation, new possibilities arise. For instance, being able to glue together pieces of documents during the computation lets us turn a trace of the computation into a detailed mathematical proof of its result. (See, for instance, the proof of the recurrence formula for the coefficients of the Taylor series of the Airy Ai function.) This answers a frequent request of users of computer algebra systems, who want to be able to understand where the results come from and how they can check or trust them. Traces are not the only type of proof that can be generated. For instance, we

[1] An ancestor of these ideas without interactivity was presented in [10].

[2] Currently we use Maple, but DynaMoW is designed so that other systems can be used as well.

url

The Special Function $\mathrm{erf}(x)$

change rendering

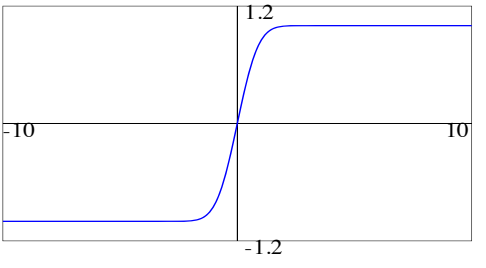
1. Differential equation

The function $\mathrm{erf}(x)$ satisfies

$$2\left(\frac{d}{dx}y(x)\right)x+\frac{d^2}{dx^2}y(x)=0$$

with initial values $y(0)=0$, $(y')(0)=2\frac{1}{\sqrt{\pi}}$.

2. Plot of $\mathrm{erf}(x)$



min = -10 max = 10 Submit

3. Numerical Evaluation

$$\mathrm{erf}(1/4+1/4\,i)\approx 0.29339518+0.26991350\,i$$

(Below, `path` may be either a point z or a broken-line path $[z_1,z_2,\ldots,z_n]$ along which to perform analytic continuation of the solution of the defining differential equation. Each z_k should be of the form `x + y*i` .)

path = 1/4+1/4 precision = 8 Submit

4. Symmetry

The function $\mathrm{erf}(x)$ is odd:

$$\mathrm{erf}(x)=-\mathrm{erf}(-x)$$

for all complex numbers x .

See the Proof That the Function $\mathrm{erf}(x)$ is Odd.

5. Taylor expansion of $\mathrm{erf}(x)$ at 0

- Expansion of erf at 0 :

$$\mathrm{erf}(x)=\sum_{n=0}^{\infty}2\,\frac{(-1)^n x^{2n+1}}{\sqrt{\pi}\,(2n+1)\,n!}$$

- See the recurrence relation for the coefficients of the Taylor expansion.

Fig. 1. The begining of the page of the DDMF on the error function

- First terms:

$$\operatorname{erf}(z) = (2\,\frac{x}{\sqrt{\pi}} - 2/3\,\frac{x^3}{\sqrt{\pi}}) + O(x^5)$$

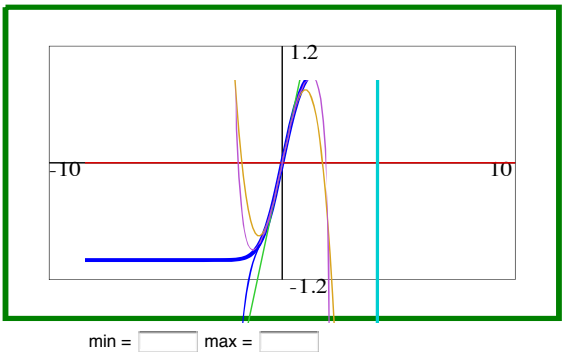
order =

- One gets a polynomial that approximates $\operatorname{erf}(x)$ uniformly on the disk $|x| \le r = 1/4$ with absolute error less than $\epsilon = \frac{1}{1000000000000000000000000000000}$ by retaining the terms up to x^{39} of this power series expansion. See this Polynomial Approximation. Proof based on the general Tail bound: General Formula.

r = epsilon =

- A majorant series for $\operatorname{erf}(x)$ in 0 is given by

$$\frac{500000065301}{1000000000000} \sum_{n=0}^{\infty} 1/6\ \frac{2^{-1/2\,n}\left(1/2\sqrt{2}\right)^{-n}(n+1)\,(n+2)\,(n+3)\,x^n}{\Gamma\left(1/2\,n+1\right)}$$



min = max =

6. Local expansions at singularities and at infinity

The differential equation above has 0 non-zero finite singular point.

- Expansion of erf at ∞ :

$$\operatorname{erf}(x) = 1 + \mathrm{e}^{-x^2} \sum_{n=0}^{\infty} -\frac{(-1)^n\,(2\,n)!\,\left(x^{-1}\right)^{2\,n}}{\sqrt{\pi}\,4^n}\,x^{-1}$$

- See the Recurrence Relation for the Power Series Coefficients of $\operatorname{erf}(x)$ at ∞ for the local coefficients.
- First terms:

$$\operatorname{erf}(z) = 1 + \mathrm{e}^{-x^{-2}}\,x\left(-\frac{1}{\sqrt{\pi}} + 1/2\,\frac{1}{\sqrt{\pi}\,x^2} + O\left(x^{-4}\right)\right)$$

order =

7. Chebyshev Recurrence

- Chebyshev expansion:

$$\operatorname{erf}(x) = \sum^{\infty} 2^{\,4^{-n}(-1)^n\,{}_1F_1(1/2+n;\,2\,n+2;\,-1)\,T_{1+2\,n}(x)}$$

Fig. 1. (*continued*)

- First terms and polynomial approximation:

$$\operatorname{erf}(x) = 0.904347\,T_1(x) - 0.0661130\,T_3(x) + 0.00472936\,T_5(x) + \ldots$$

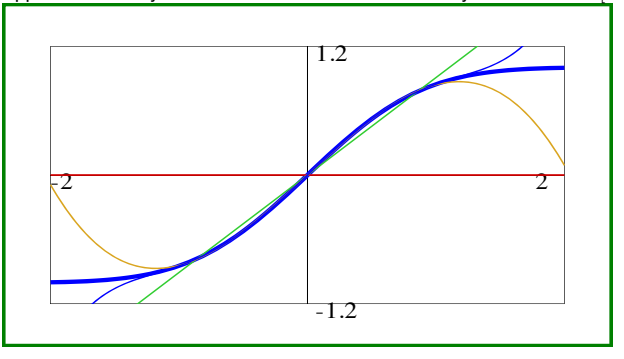
$$\operatorname{erf}(x) = 1.12633280\,x - 0.35903920\,x^3 + 0.07566976\,x^5 + \ldots$$

order = 6 Submit

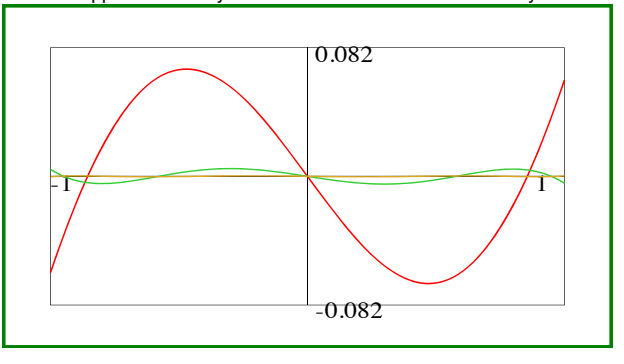
- The coefficients c_n in the Chebyshev expansion $erf(x) = \sum_{n=0}^{\infty} c_n T_n(x)$ satisfy the recurrence

$$(n^2+3n)\,c(n) + (2n^3+12n^2+24n+16)\,c(n+2) + (-n^2-5n-4)\,c(n+4) = 0$$

- Approximations by successive truncations of the Chebyshev series on $[-2, 2]$



- Errors of approximation by successive truncations of the Chebyshev series on $[-1, 1]$



8. Laplace Transform

For $\Re(s) > 0$,

$$\int_0^{\infty} e^{-ts}\operatorname{erf}(t)\,dt = -\frac{(\operatorname{erf}(1/2\,s) - 1)\,e^{1/4\,s^2}}{s}.$$

Fig. 1. (*continued*)

may present a simple proof for the solution of a recurrence once the solution has been found, instead of retracing its computation.

The implementation of the DynaMoW language itself is work in progress and a stable version will be described in due course. We believe that this language

will be of interest outside of the DDMF. For instance, we have also used it with success for an encyclopedia of combinatorial structures.

3 Computer Algebra Algorithms

From the computer algebra point of view, what is a good definition of a mathematical function such that all the desired formulas can be computed algorithmically? Our choice is to concentrate on

> *Functions given as solutions of linear differential equations or linear recurrences.*

Our basic data-structure consists of these equations and their initial conditions. In the example of Fig. 1, this is the content of Section 1.

This data-structure has become common in computer algebra, starting with works of Stanley [17], Lipschitz [9], Zeilberger [18] and more recently Chyzak, Salvy *et alii* [4,5,6]. In particular, we rely on the Maple `gfun` package [16] for many of our computations.

Given this data structure, we have used or developed algorithms to compute many relevant properties of mathematical functions. For instance, Section 3 of our example offers numerical approximations of guaranteed quality in good complexity (see [11] for the algorithm). Such approximations can be used to produce graphs as in Section 2 of the example. Section 5 is based on recurrences for the Taylor coefficients that are obtained from the differential equations. When they exist, closed-form hypergeometric solutions of these recurrences can be computed [14]. In all cases, the rest of that Section 5 (beyond the part that is visible in Fig. 1) gives the first terms of these expansions and bounds on tails of power series [12]. The same computations are performed at each singularity including infinity. Further results include Chebyshev expansions [2] and differential equations for the Laplace transform (Sections 7 and 8).

Future Work

Some of our next steps include these tasks: automatic handling of families of functions or functions with parameters, like the Bessel functions, either by letting the user choose values for the parameters, or by performing an automatic discussion according to the possible range of values; automatic generation of good numerical code at fixed precision; more integral transforms; expansions on other bases; information on the zeros of the functions; handling of branch-cuts; and support for user-defined functions.

References

1. Abramowitz, M., Stegun, I.A. (eds.): Handbook of mathematical functions with formulas, graphs, and mathematical tables. Dover Publications Inc., New York (1992); Reprint of the 1972 edition. First edition 1964

2. Benoit, A., Salvy, B.: Chebyshev expansions for solutions of linear differential equations. In: May, J. (ed.) Symbolic and Algebraic Computation, pp. 23–30. ACM Press, New York (2009); Proceedings of ISSAC 2009, Seoul (July 2009)
3. Boisvert, R., Lozier, D.W.: Handbook of Mathematical Functions. In: A Century of Excellence in Measurements Standards and Technology, pp. 135–139. CRC Press, Boca Raton (2001)
4. Chyzak, F.: Groebner bases, symbolic summation and symbolic integration. In: Buchberger, B., Winkler, F. (eds.) Groebner Bases and Applications, Proc. of the Conference 33 Years of Gröbner Bases. London Mathematical Society Lecture Notes Series, vol. 251, pp. 32–60. Cambridge University Press, Cambridge (1998)
5. Chyzak, F.: An extension of Zeilberger's fast algorithm to general holonomic functions. Discrete Mathematics 217(1-3), 115–134 (2000)
6. Chyzak, F., Kauers, M., Salvy, B.: A non-holonomic systems approach to special function identities. In: May, J. (ed.) Symbolic and Algebraic Computation, pp. 111–118. ACM Press, New York (2009); Proceedings of ISSAC 2009, Seoul (July 2009)
7. Erdélyi, A.: Higher Transcendental Functions, vol. 1-3. R. E. Krieger Publishing Company, Inc., Malabar (1981); First edition 1953
8. Gradshteyn, I.S., Ryzhik, I.M.: Table of Integrals, Series, and Products. Academic Press, London (1996); First English edition 1965
9. Lipshitz, L.: D-finite power series. Journal of Algebra 122(2), 353–373 (1989)
10. Meunier, L., Salvy, B.: ESF: An automatically generated encyclopedia of special functions. In: Sendra, J.R. (ed.) Symbolic and Algebraic Computation, pp. 199–205. ACM Press, New York (2003); Proceedings of ISSAC 2003, Philadelphia (August 2003)
11. Mezzarobba, M.: NumGfun: a package for numerical and analytic computation with D-finite functions. In: ISSAC 2010. ACM Press, New York (2010), http://arxiv.org/abs/1002.3077
12. Mezzarobba, M., Salvy, B.: Effective bounds for P-recursive sequences. Journal of Symbolic Computation (to appear)
13. Olver, F.W.J., Lozier, D.W., Boisvert, R.F., Clark, C.W. (eds.): NIST Handbook of Mathematical Functions. Cambridge University Press, Cambridge (2010)
14. Petkovšek, M.: Hypergeometric solutions of linear recurrences with polynomial coefficients. Journal of Symbolic Computation 14(2-3), 243–264 (1992)
15. Prudnikov, A.P., Brychkov, Y.A., Marichev, O.I.: Integrals and Series, 1st edn. in Moscow, Nauka, vol. 1-6 (1981)
16. Salvy, B., Zimmermann, P.: Gfun: a Maple package for the manipulation of generating and holonomic functions in one variable. ACM Transactions on Mathematical Software 20(2), 163–177 (1994)
17. Stanley, R.P.: Differentiably finite power series. European Journal of Combinatorics 1(2), 175–188 (1980)
18. Zeilberger, D.: A holonomic systems approach to special functions identities. Journal of Computational and Applied Mathematics 32(3), 321–368 (1990)

Reliable Computing with GNU MPFR

Paul Zimmermann

LORIA/INRIA Nancy-Grand Est, Équipe CARAMEL - bâtiment A,
615 rue du jardin botanique, F-54603 Villers-lès-Nancy Cedex

Abstract. This article presents a few applications where reliable computations are obtained using the GNU MPFR library.

Keywords: reliable computing, correct rounding, IEEE 754, GNU MPFR.

The overview of the 3rd International Workshop on Symbolic-Numeric Computation (SNC 2009), held in Kyoto in August 2009, says: *Algorithms that combine ideas from symbolic and numeric computation have been of increasing interest over the past decade. The growing demand for speed, accuracy and reliability in mathematical computing has accelerated the process of blurring the distinction between two areas of research that were previously quite separate. [...]* Using numeric computations certainly speeds up several symbolic algorithms, however to ensure the correctness of the final result, one should be able to deduce some *reliable facts* from those numeric computations. Unfortunately, most numerical software tools do not provide any accuracy guarantee to the user. Let us demonstrate this fact on a few examples. Can we deduce the sign of $\sin(2^{100})$ from the following computation with Maple version 13?

```
> evalf(sin(2^100));
                                    0.4491999480
```

This problem concerns other computer algebra systems, for example Sage 4.4.2, where the constant $C = e^{\pi\sqrt{163}} - 262537412640768744$ evaluated to different precisions yields different signs:

```
sage: f=exp(pi*sqrt(163))-262537412640768744
sage: numerical_approx(f, digits=15)
448.000000000000
sage: numerical_approx(f, digits=30)
-5.96855898038484156131744384766e-13
```

or Mathematica 6.0, where the same constant $C \approx -0.75 \cdot 10^{-12}$, integrated from 0 to 1, yields a result too large by several orders of magnitude:

```
In[1]:= NIntegrate[Exp[Pi*Sqrt[163]]-262537412640768744,{x,0,1}]

Out[1]= -480.
```

Another example is FFTW, which is a very efficient Fast Fourier Transform (FFT) library. On http://www.fftw.org/accuracy/comments.html one can

K. Fukuda et al. (Eds.): ICMS 2010, LNCS 6327, pp. 42–45, 2010.

read: *Our benchmark shows that certain FFT routines are more accurate than others. In other cases, a routine is accurate on one machine but not on another. [...]* This non-reproducibility from one machine to another one is quite annoying, since this means that a given program using FFTW might give different results on different computers. One reason of this non-reproducibility is the fact that FFTW uses *trigonometric recurrences* to compute the *twiddle factors* $e^{2ik\pi/2^n}$ needed in the FFT. On some architectures, those recurrences are evaluated in double-extended precision — significand of 64 bits — instead of double-precision — significand of 53 bits.

To avoid the above problems, developers of numerical software tools should provide primitives with *well-defined semantics*. This is not an easy goal. The more complex the numerical primitive is, the more difficult it is to provide rigorous bounds on the error. For example consider the implicit two-dimensional plot of $|\cos((x+iy)^4)| = 1$ for $-3 \le x, y \le 3$; the computation to 20 decimal places of $\rho(10)$ where ρ is Dickman's function, defined by the difference-differential equation $x\rho'(x) + \rho(x-1) = 0$, with initial conditions $\rho(x) = 1$ for $0 \le x \le 1$; or the computation to 20 decimal places of the singular values of the Hilbert matrix of order 50, defined by $M_{i,j} = 1/(i+j)$.

There are several ways for a numerical primitive to return valuable information. One way is to give, in addition to the numerical approximation, a bound on the absolute or relative error; or alternatively an interval enclosing the true result. The ultimate — and more difficult solution, from the implementer point of view — is to guarantee *correct rounding* as in IEEE 754 [6], i.e., that the given approximation is the best possible according to the target precision. To rigorously define what we mean by "best possible", we have to introduce *rounding directions*, which determine in which direction to round the approximation with respect to the exact result, and how to break ties.

GNU MPFR (MPFR for short) is a C library implementing correct rounding for basic arithmetic operations and mathematical functions in binary multiple-precision. We refer the reader to [4] for a technical description of MPFR. We focus here on a few applications of MPFR.

Companion Libraries MPFI and MPC. The MPFI library implements arbitrary precision interval arithmetic on top of MPFR. It was originally designed by N. Revol and F. Rouillier. For a monotonic function, implementing an interval routine is trivial: just call the corresponding MPFR function with rounding modes towards $-\infty$ and $+\infty$. However for a non-monotonic function, it requires more work; for example does $\cos([103992, 103993])$ contain 1?

The MPC library, developed by A. Enge, Ph. Théveny and the author, is another companion library to MPFR, which provides arbitrary precision complex floating-point numbers with correct rounding. MPC uses the cartesian representation $z = x + iy$, where both x and y are correctly rounded. If MPFR provides r rounding modes — $r = 5$ in MPFR 3.0.0 — then MPC can provide up to r^2 complex rounding modes. The MPC library implements all functions from the C99 standard.

Constant Folding in GCC. The GCC compilers for the C and Fortran languages use MPFR for constant folding (see details in [5]). In short, when one writes `double y = sin(17.42)` in a C program, GCC replaces this at *compile time* by `double y = -0.99004214446851813`, using MPFR to perform the corresponding computation. The advantage is twofold: on the one hand MPFR will yield the same numerical result on any configuration, whatever the operating system, the processor word size; and on the other hand MPFR guarantees correct rounding for the sine mathematical function (which de facto implies reproducibility). Up from version 4.5, GCC also uses the MPC library to provide constant folding of complex floating-point expressions.

Maple. Since version 11, the Maple computer algebra system uses MPFR for approximating real solutions of polynomial systems in the `RootFinding` package:

```
> sys:=[x^2+y^2-1, y-x^2]:
> RootFinding[Isolate](sys, [x,y], digits=1, output=interval);
       -7251005348244714239  -906369071450328423
[[x = [--------------------, --------------------],
       9223372036854775808   1152921504606846976

          1414187776304389027  5711861873363103083
    y = [--------------------, --------------------]], [...]]
          2305843009213693952  9223372036854775808
```

This computation uses the Rational Univariate Representation designed by F. Rouillier, and then the MPFI interval arithmetic library — which in turn uses MPFR — is used to isolate the roots or perform arithmetic operations on the roots.

MPFR in Sage. The open-source Sage computer algebra system (`sagemath.org`) uses MPFR for its arbitrary precision floating-point arithmetic, and MPFI for the corresponding interval arithmetic. It should be noted that the user has access to the MPFR rounding modes and to the exact representation $m \cdot 2^e$ of binary floating-point numbers, as the following example (with Sage 4.4.2) shows:

```
sage: D = RealField(42, rnd='RNDD'); U = RealField(42, rnd='RNDU')
sage: D(pi), U(pi)
(3.14159265358, 3.14159265360)
sage: D(pi).exact_rational()
3454217652357/1099511627776
sage: x = RealIntervalField(42)(pi); x.lower(), x.upper()
(3.14159265358, 3.14159265360)
```

MPFR is also used by the Magma computational number theory system, and by the Mathemagix free computer algebra system (in the `numerix` package).

Apart from the above indirect applications, where the final user is not always aware that she/he is using MPFR, we mention here a few selected "direct" applications of MPFR (more details can be found on `http://www.mpfr.org/pub.html`).

MPFR is often used as a reference correctly-rounded implementation for mathematical functions in double precision [3,8]. P. Kornerup, V. Lefèvre, N. Louvet and J.-M. Muller used MPFR to prove — among other results — that there exists no algorithm in less than 6 arithmetic operations to compute the "TwoSum" of two floating-point numbers a and b, i.e., x and y such that x is the rounding to nearest of $a+b$, and $y = a+b-x$; they used an exhaustive search approach [7]. F. Chiba and T. Ushijima used MPFR to study waves produced by a disc [1,2].

Some funny applications of MPFR are the following. K. Briggs used MPFR to find a new worst approximable pair; this result required computing 10^7 terms of the continued fraction of $2\cos(2\pi/7)$. D. de Rauglaudre used MPFR to zoom in the Mandelbrot/Julia sets, since from a given depth on, double precision is not sufficient; the corresponding videos are available on Youtube[1].

Conclusion. Numeric tools with well-defined semantics help improving the reliability and portability of mathematical software. MPFR does not solve all problems: it only guarantees correct rounding for an atomic operation, thus for a sequence of operations like the constant C in the introduction, one has to use other means like interval arithmetic or a Real RAM implementation (like the iRRAM package from N. Müller). We advise the developers of numeric tools to provide such well-defined semantics, and their users to make good use of them!

Acknowledgement. The author thanks Nathalie Revol who noticed some typos in a earlier version of that article.

References

1. Chiba, F., Ushijima, T.: Computation of the scattering amplitude for a scattering wave produced by a disc – approach by a fundamental solution method. Journal of Computational and Applied Mathematics 233(4), 1155–1174 (2009)
2. Chiba, F., Ushijima, T.: Exponential decay of errors of a fundamental solution method applied to a reduced wave problem in the exterior region of a disc. Journal of Computational and Applied Mathematics 231(2), 869–885 (2009)
3. de Dinechin, F., Ershov, A.V., Gast, N.: Towards the post-ultimate libm. In: Proceedings of 17th IEEE Symposium on Computer Arithmetic, Cape Cod, USA, pp. 288–295 (2005)
4. Fousse, L., Hanrot, G., Lefèvre, V., Pélissier, P., Zimmermann, P.: MPFR: A multiple-precision binary floating-point library with correct rounding. ACM Trans. Math. Softw., article 13 33(2) (2007)
5. Ghazi, K.R., Lefèvre, V., Théveny, P., Zimmermann, P.: Why and how to use arbitrary precision. Computing in Science and Engineering 12(3), 62–65 (2010)
6. IEEE standard for oating-point arithmetic, 2008. Revision of ANSI-IEEE Standard 754-1985, approved June 12, 2008: IEEE Standards Board (2008)
7. Kornerup, P., Lefèvre, V., Louvet, N., Muller, J.-M.: On the computation of correctly-rounded sums. In: Bruguera, J.D., Cornea, M., Das-Sarma, D., Harrison, J. (eds.) Proceedings of the 19th IEEE Symposium on Computer Arithmetic (ARITH'19), pp. 155–160. IEEE Computer Society, Los Alamitos (2009)
8. Lauter, C.Q., Lefèvre, V.: An efficient rounding boundary test for pow(x, y) in double precision. IEEE Trans. Comput. 58(2), 197–207 (2009)

[1] http://www.youtube.com/view_play_list?p=56029CB07C72B4A4

Simplicial Cohomology of Smooth Orbifolds in GAP

Mohamed Barakat[1] and Simon Görtzen[2]

[1] Department of mathematics, University of Kaiserslautern,
67653 Kaiserslautern, Germany
barakat@mathematik.uni-kl.de
[2] UMIC Research Centre, RWTH Aachen, Mies-van-der-Rohe-Str. 15,
52074 Aachen, Germany
simon.goertzen@rwth-aachen.de

Abstract. This short research announcement briefly describes the simplicial method underlying the GAP package SCO for computing the so-called orbifold cohomology of topological resp. smooth orbifolds. SCO can be used to compute the lower dimensional group cohomology of some infinite groups.

Instead of giving a complete formal definition of an orbifold, we start with a simple construction, general enough to give rise to any orbifold $\mathcal{M}$. Let X be a (smooth) manifold and G a LIE group acting (smoothly and) properly on X, i.e., the action graph $\alpha : G \times X \to X \times X : (g, x) \mapsto (x, gx)$ is a proper map. In particular, G acts with *compact* stabilizers $G_x = \alpha^{-1}(\{(x, x)\})$. Further assume that G is either

- discrete (acting discontinuously), or
- compact acting almost freely (i.e. with discrete stabilizers) on X.

In both cases G acts with *finite* stabilizers. "Enriching" $M := X/G$ with these finite isotropy groups leads to the so-called **fine** orbit space $\mathcal{M} := [X/G]$, also called the **global quotient orbifold**. We call the orbifold **reduced** if the action is faithful. The topological space M is then called the **coarse space** underlying the orbifold $\mathcal{M}$. Roughly speaking, a reduced **orbifold** $\mathcal{M}$ locally looks like the orbit space $\mathbb{R}^n/V$, where V is a *finite* subgroup of $\mathrm{GL}_n(\mathbb{R})$. It is easy to show that every effective orbifold $\mathcal{M}$ arises as a global quotient orbifold by a compact LIE group[1] G.

By considering (countable) discrete groups G acting properly and discontinuously on X we still obtain a subclass of orbifolds which includes many interesting moduli spaces. The most prominent one is the (non-compactified) moduli space $\mathcal{M}_{g,n}$ of curves of genus g with n marked points and $2g + n \geq 3$. It is the global quotient $[T_{g,n}/\Gamma_{g,n}]$ of the *contractible* TEICHMÜLLER space $T_{g,n} \approx \mathbb{C}^{3g-3+n}$ by the proper discontinuous action of the **mapping class group** $\Gamma_{g,n}$ [7].

[1] Define X as the bundle of orthonormal frames on $\mathcal{M}$. It is a manifold on which the orthogonal group $G := \mathrm{O}_n(\mathbb{R})$ acts almost freely with faithful slice representations [8, Thm. 4.1].

K. Fukuda et al. (Eds.): ICMS 2010, LNCS 6327, pp. 46–49, 2010.

A convenient way to define a cohomology theory on $\mathcal{M}$ is to consider the category $\mathrm{Ab}(\mathcal{M})$ of ABELian sheaves on $\mathcal{M}$, together with the global section functor $\Gamma_{\mathcal{M}} : \mathrm{Ab}(\mathcal{M}) \to \mathrm{Ab}$ to the category of ABELian groups. The ABELian category $\mathrm{Ab}(\mathcal{M})$ has enough injectives and the orbifold cohomology of $\mathcal{M}$ with values in a sheaf[2] $\mathcal{A}$ is then simply defined as the derived functor cohomology

$$H^n(\mathcal{M}, \mathcal{A}) := \mathrm{R}^n\Gamma_{\mathcal{M}}(\mathcal{A}).$$

This conceptually simple definition is of course highly nonconstructive. To boil it down to a constructive one we proceed in several steps following [9]:

1. If the orbifold $\mathcal{M}$ is given as a global quotient $[X/G]$, then $\mathrm{Ab}(\mathcal{M})$ is equivalent to the category $\mathrm{Ab}(X)^G$ of G-equivariant sheaves on X. The global section functor $\Gamma_{\mathcal{M}}$ then corresponds to the functor Γ_X^G of G-equivariant global sections. The space X and the action graph $\alpha : G \times X \to X \times X$ are encoded by the so-called **action groupoid**[3] $\mathcal{G} = G \ltimes X := G \times X \rightrightarrows X$ with source-target map $(s,t) = \alpha$, composition $(g,y)(h,x) = (ghx, x)$ for $y = hx$, and inversion $(g,x)^{-1} = (g^{-1}, gx)$.

   For a general groupoid $\mathcal{G}$ with base X denote by $\mathrm{Ab}(\mathcal{G})$ the ABELian category of $\mathcal{G}$-sheaves, i.e., the sheaves on X compatible with the action of the groupoid $\mathcal{G}$. Hence, if $\mathcal{G}$ is the action groupoid $G \ltimes X$, then the $\mathcal{G}$-sheaves are nothing but G-equivariant sheaves on X and $\mathrm{Ab}(\mathcal{G}) = \mathrm{Ab}(X)^G$.
2. Another way to represent an orbifold $\mathcal{M}$ was suggested by HAEFLIGER [6]. Denoting by $\mathcal{U}$ an orbifold atlas of $\mathcal{M}$, he constructed an étale proper groupoid $H := H_1 \rightrightarrows H_0$ with base space $H_0 := \coprod_{\tilde{U} \in \mathcal{U}} \tilde{U}$ and orbit space[4] H_0/H_1 homeomorphic[5] to the coarse space M underlying $\mathcal{M}$. Again it follows that $\mathrm{Ab}(H) \simeq \mathrm{Ab}(\mathcal{M})$.
3. Choose a triangulation $\mathcal{T}$ of M **adapted** to the orbifold atlas $\mathcal{U}$ of $\mathcal{M}$. Such a triangulation always exists [9, Prop. 1.2.1]. Replacing H_0 by the subspace $R_0(\mathcal{T}) := \coprod_{\sigma \in \mathcal{T}_n} \tilde{\sigma} \subset H_0$, where $\mathcal{T}_n$ is the set of maximal simplices in $\mathcal{T}$ (of dimension $n = \dim \mathcal{M}$), and pulling back H_1 over the embedding $R_0(\mathcal{T}) \times R_0(\mathcal{T}) \hookrightarrow H_0 \times H_0$ yields the set of arrows $R_1(\mathcal{T})$ of the so-called **reduced groupoid** $R(\mathcal{T}) := R_1(\mathcal{T}) \rightrightarrows R_0(\mathcal{T})$. $R(\mathcal{T})$ is a full subgroupoid of H and, again, $\mathrm{Ab}(R(\mathcal{T})) \simeq \mathrm{Ab}(H)$. In particular $H^n(R(\mathcal{T}), A_{|R(\mathcal{T})}) \cong H^n(H, A)$ for any ABELian sheaf A on H.
4. One can now show that the connected components of $R(\mathcal{T})_\bullet$ are contractible, where $R(\mathcal{T})_\bullet$ denotes the nerve of the reduced groupoid $R(\mathcal{T})$. The set

$$S_\bullet := \pi_0(R(\mathcal{T})_\bullet)$$

   of connected components of the nerve of the reduced groupoid is a simplicial set, which is, unlike $R(\mathcal{T})_\bullet$, *not* a nerve of some category, in general.

[2] One often considers the constant sheaf $\mathcal{A} = \mathbb{Z}$.

[3] The group G cannot be recovered from the action groupoid, in general.

[4] This is the space of equivalence classes of the equivalence relation given by the image of the source-target map $(s,t) : H_1 \to H_0 \times H_0$.

[5] This generalizes a similar construction for an étale proper groupoid representing a manifold, cf. [2, Chap. 2, 2.α]

5. A local system $\mathcal{A}$ on $\mathcal{M}$, i.e., a locally constant sheaf, induces a locally constant sheaf on H and by restriction one on $R(\mathcal{T})$. The induced locally constant sheaf $A^{(\bullet)}$ on the nerve[6] $R(\mathcal{T})_\bullet$ is constant on the contractible connected components and factors to a local system of coefficients (cf. [4, I.7]) on the simplicial set $S_\bullet$, which we again denote by A.
6. A spectral sequence argument finally provides the isomorphism of cohomology:
$$H^n(\mathcal{M}, \mathcal{A}) \cong H^n(S_\bullet, A).$$

The right hand side of this isomorphism is indeed constructive and was implemented in the GAP package SCO [5]. The package includes examples computing the low dimensional cohomology groups of the 2-dimensional crystallographic space groups:

In case $\mathcal{M}$ is a global quotient $[X/G]$ of a *contractible* space X by a proper action of a group G, then $H^n(\mathcal{M}, \mathbb{Z}) \cong H^n(G, \mathbb{Z})$, and we recover the ordinary group cohomology of G.

In other words: The cohomology functor cannot detect the passage from an abstract group G (regarded as a groupoid over one point) to the action groupoid $G \ltimes X$ on a contractible space X (as a contractible space does not introduce any detectable cohomological data). In favorable situations this can be exploited to replace an infinite (discrete) group by an orbifold, an object which is still group- but now also *space*-like, and where the occurring groups are all *finite*. So, roughly speaking, the cohomology of an infinite[7] (discrete) group can be computed as the cohomology of a space "intertwined" with the cohomology of some finite groups.

Starting from a fundamental domain of a proper discontinuous cocompact action of a discrete group one can easily obtain an adapted triangulation of the global quotient. This was the point of departure for computing the lower dimensional cohomology of the 2-dimensional space groups, which can easily be extended to higher dimensional space groups. The SCO package can be used to quickly produce the simplicial set $S_\bullet$ associated to an adapted triangulation, but the efficiency of computing the cohomology will depend on the growth of the cardinalities $|S_n|$, where S_n is n-th set of $S_\bullet$. It is thus desirable to construct "minimal" triangulations.

The orbifold corresponding to the 2-dimensional space group `p31m` is a contractible cone with a singular C_3-point at the bottom, as well as a C_2-edge with one D_6-isotropy (see Figures 1 and 2). Just as with D_6 itself, this leads to a 4-periodic cohomology:

$$H^i(\texttt{p31m}, \mathbb{Z}) = \begin{cases} \mathbb{Z} & i = 0 \\ 0 & i = 1 \\ \mathbb{Z}/2\mathbb{Z} \oplus \mathbb{Z}/3\mathbb{Z} & i \geq 2,\ i \equiv_4 2 \\ \mathbb{Z}/2\mathbb{Z} & i \geq 3,\ i \equiv_4 1, 3 \\ \mathbb{Z}/2\mathbb{Z} \oplus (\mathbb{Z}/3\mathbb{Z})^2 & i \geq 4,\ i \equiv_4 0 \end{cases}$$

More details are provided in the forthcoming work [1].

[6] In the sense of [3].

[7] Starting with a finite group only produces computational overhead.

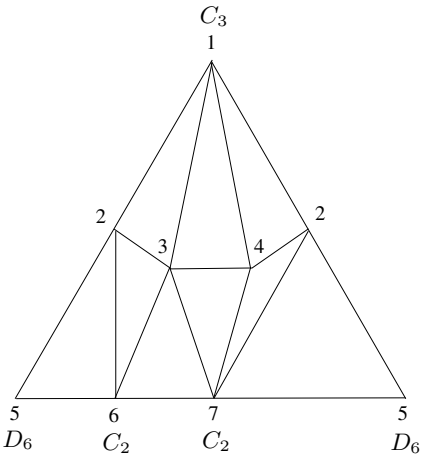


Fig. 1. Adapted triangulation of the fundamental domain of `p31m`

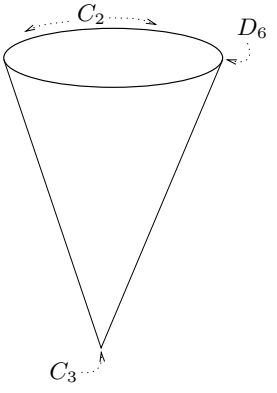


Fig. 2. The coarse space of the orbifold `p31m` enriched with the isotropy groups

References

1. Barakat, M., Görtzen, S.: Simplicial cohomology of orbifolds revisited (in preparation)
2. Connes, A.: Noncommutative geometry. Academic Press Inc., San Diego (1994), http://www.alainconnes.org/downloads.html
3. Deligne, P.: Théorie de Hodge. III. Inst. Hautes Études Sci. Publ. Math. (44), 5–77 (1974)
4. Gelfand, S.I., Manin, Y.I.: Methods of homological algebra, 2nd edn. Springer Monographs in Mathematics. Springer, Berlin (2003)
5. Görtzen, S.: GAP Package SCO (2007-2008), http://wwwb.math.rwth-aachen.de/goertzen/SCO
6. Haefliger, A.: Groupoïdes d'holonomie et classifiants. Astérisque (116), 70–97 (1984); Transversal structure of foliations (Toulouse, 1982)
7. Korkmaz, M.: Low-dimensional homology groups of mapping class groups: a survey. Turkish J. Math. 26(1), 101–114 (2002) (arXiv:math.GT/0307111)
8. Moerdijk, I., Pronk, D.A.: Orbifolds, sheaves and groupoids. K-Theory 12(1), 3–21 (1997)
9. Moerdijk, I., Pronk, D.A.: Simplicial cohomology of orbifolds. Indag. Math (N.S.) 10(2), 269–293 (1999) (arXiv:q-alg/9708021)

Computing Polycyclic Quotients of Finitely (L-)Presented Groups via Groebner Bases

Bettina Eick and Max Horn

Institut Computational Mathematics, TU Braunschweig,
Pockelsstrasse 14, 38106 Braunschweig, Germany
{beick,max.horn}@tu-bs.de
http://www.tu-braunschweig.de/icm/algebra

Abstract. We announce the development and implementation of a new GAP package **PCQL**. This facilitates the computation of consistent polycyclic presentations for polycyclic quotients of groups defined by a so-called finite L-presentation. This type of presentation incorporates all finite presentations as well as certain infinite presentations. The algorithm allows a variety of polycyclic quotients ranging from maximal nilpotent quotients of a given class to the maximal solvable quotients of a given derived length. The algorithm uses Groebner bases over integral group rings of polycyclic groups as main means of its computation.

Keywords: Polycyclic quotient, nilpotent quotient, finitely presented group, L-presented group, Groebner bases.

1 Introduction

The development and implementation of quotients methods for finitely presented groups has a long history. The simplest version of such a method is the abelian quotient algorithm. Given a finitely presented group G, this determines the abelian invariants of G/G'. We refer to the book by Sims [8] for background. There are many other types of quotients whose computation has been considered: finite p-quotients, finite solvable quotients, nilpotent quotient and, most general in this sequence, polycyclic quotients.

Nickel [7] developed a method to compute nilpotent quotients. Given a finitely presented group G and an integer n, this can compute a polycyclic presentation of the class-n quotient $G/\gamma_n(G)$ of G. This method is quite effective and has a wide range of applications. It has been generalized to finitely L-presented groups by Bartholdi, Eick and Hartung [1].

Lo [6] has developed a method to compute polycyclic quotients. Given a finitely presented group G and an integer n, this can decide whether the derived length-n quotient $G/G^{(n)}$ is polycyclic and, if so, then it can compute a polycyclic presentation for it. This method uses Groebner bases over integral group rings of polycyclic groups. It is less effective than the nilpotent quotient method by Nickel and has a rather limited range of applications. Its implementation in C is outdated and difficult to base improvements on.

K. Fukuda et al. (Eds.): ICMS 2010, LNCS 6327, pp. 50–53, 2010.

Our aim was to develop and implement a quotient algorithm which combines the advantages of Lo's method with the high effectivity of Nickel's algorithm and, at the same time, generalizes from finitely presented groups to finitely L-presented groups. We announce the development of such a method and its implementation in GAP here and describe it briefly in the following sections. We first recall the definition of a finite L-presentation in the following section.

2 L-Presentations

Let X be a finite set of abstract generators and let F be the free group on X. Let R and Q be finite subsets of F and ϕ a finite set of endomorphisms of F. Then

$$\langle X \mid Q \mid \phi \mid R \rangle$$

is called a *(finite) L-presentation*. Let ϕ^* denote the monoid generated by ϕ. Then the finite L-presentation defines the group F/K with

$$K = \langle Q \cup \bigcup_{\sigma \in \phi^*} \sigma(R) \rangle^F.$$

Every finitely presented group $\langle X \mid S \rangle$ is finitely L-presented, e.g. as $\langle X \mid S \mid \emptyset \mid \emptyset \rangle$ or as $\langle X \mid \emptyset \mid \emptyset \mid S \rangle$. There are many interesting groups for which a finite presentation is not known or does not exist, but which can be defined by a finite L-presentation. Prominent examples are the Grigorchuk group [4] and the Gupta-Sidki group [5]. Both groups arose as counterexamples to the famous Burnside conjectures and both groups have proved to have very interesting properties.

3 The Algorithm

Let G be a group given by a finite L-presentation. Our aim is to consider a certain quotient of G, check whether it is polycyclic and if so, then construct a consistent polycyclic presentation for this quotient. We proceed by induction for this purpose.

In the initial step of the induction, we determine a polycyclically presented group H together with an epimorphism $\varphi : G \to H$ with kernel G'. In the subsequent induction steps, we assume that we are given

(1) The finitely L-presented group G;
(2) A polycyclically presented group H;
(3) An epimorphism $\varphi : G \to H$ with kernel N, say;
(4) A subgroup $U \trianglelefteq H$.

Let $\varphi^{-1}(U)$ be a full preimage of U under φ and define $M = [\varphi^{-1}(U), N]$. Then M is a normal subgroup of G with $N' \leq M \leq [G, N]$. The aim in the induction step is to check whether G/M is polycyclic (or, equivalently, whether N/M is finitely generated) and if so, then determine a polycyclically presented group K and an epimorphism $\rho : G \to K$ with kernel M.

If we use $U = H$ in each step of the algorithm, then the computed quotients are the quotients of G by the groups in the lower central series of G. If we use $U = \{1\}$ in each step of the algorithm, then the computed quotients are the quotients of G by the groups in the derived series of G. The algorithm also allows for many intermediate quotients as well.

Note that N/M is a G-module, since G acts by conjugation on the quotient N/M. Since N/M is abelian, we can also consider it as an H-module. The key step in our algorithm translates the H-module N/M to a quotient V/W, where V is a free $\mathbb{Z}H$-module and W is a submodule of V. It then determines a Groebner basis for W using an improved version of the method by Lo [6]. This allows to read off whether G/M is polycyclic and it yields a polycyclic presentation for G/M in this case.

4 Two Small Examples

Due to the restricted space, we only consider two very small examples here. Let G be the group defined by the finite presentation

$$\langle a, b \mid a^4, a^{-2}ba^{-2}b, b^{-1}a^{-1}b^{-1}a^{-1}b^{-1}aba \rangle$$

and let H be the Basilica group [3] with its finite L-presentation in [2].

The following table exhibits the abelian invariants of the computed quotients in the two extreme cases: In the first case (N) we compute the quotients G_i/G_{i+1} of successive members of the lower central series and in the second case (Q) we do the same for the derived series. All quotients are described by their abelian invariants.

	G		H	
Step	(N)	(Q)	(N)	(Q)
1	(2,4)	(2,4)	(0,0)	(0,0)
2	(2)	(0,0)	(0)	(0,0,0)
3	(2)	()	(4)	(2,2,0,0,0,0,0,0,0,0)
4	(2)	()	(4)	?
5	(2)	()	(4,4)	?

The (Q) case for G exhibits the maximal solvable quotient of G after 3 steps: this has derived length 2 and is polycyclic of Hirsch length 2. The (N) case for G will never reach the maximal solvable quotient, since all nilpotent quotients of G are finite as G/G' is finite.

The (Q) case for H shows that the maximal derived length-3 quotient of H is polycyclic of Hirsch length 13. The maximal class-48 quotient of H has been determined in [1]: this has Hirsch length 3.

References

1. Bartholdi, L., Eick, B., Hartung, R.: A nilpotent quotient algorithm for certain infinitely presented groups and its applications. Internat. J. Algebra Comput. 18(8), 1321–1344 (2008)

2. Bartholdi, L., Virág, B.: Amenability via random walks. Duke Math. J. 130(1), 39–56 (2005)
3. Grigorchuk, R., Zuk, A.: On a torsion-free weakly branch group defined by a three state automaton. Internat. J. Algebra Comput. 12(1-2), 223–246 (2002)
4. Grigorchuk, R.I.: Just infinite branch groups. In: New horizons in pro-p groups. Progr. Math., vol. 184, pp. 121–179. Birkhäuser, Boston (2000)
5. Gupta, N.D., Sidki, S.N.: Some infinite p-groups. Algebra i Logika 22(5), 584–589 (1983)
6. Lo, E.H.: A polycyclic quotient algorithm. J. Symb. Comput. 25, 61–97 (1998)
7. Nickel, W.: Computing nilpotent quotients of finitely presented groups. In: Geometric and computational perspectives on infinite groups. Amer. Math. Soc. DIMACS Series, pp. 175–191 (1996)
8. Sims, C.C.: Computation with finitely presented groups. Cambridge University Press, Cambridge (1994)

Constructive Membership Testing in Black-Box Classical Groups

Sophie Ambrose[1], Scott H. Murray[2], Cheryl E. Praeger[1], and Csaba Schneider[3]

[1] School of Mathematics and Statistics, The University of Western Australia
35 Stirling Highway CRAWLEY WA 6009 Australia
Cheryl.Praeger@uwa.edu.au, alias.sqbr@gmail.com
[2] Faculty of Information Sciences and Engineering, The University of Canberra, ACT, 2601
Scott.Murray@canberra.edu.au
[3] Centro de Álgebra da Universidade de Lisboa
Av. Prof. Gama Pinto, 2, 1649-003 Lisboa, Portugal
csaba.schneider@gmail.com

The research described in this note aims at solving the constructive membership problem for the class of quasisimple classical groups. Our algorithms are developed in the black-box group model (see [HCGT, Section 3.1.4]); that is, they do not require specific characteristics of the representations in which the input groups are given. The elements of a black-box group are represented, not necessarily uniquely, as bit strings of uniform length. We assume the existence of oracles to compute the product of two elements, the inverse of an element, and to test if two strings represent the same element. Solving the *constructive membership problem* for a black-box group G requires to write every element of G as a word in a given generating set. In practice we write the elements of G as straight-line programs (SLPs) which can be viewed as a compact way of writing words; see [HCGT, Section 3.1.3].

The constructive membership problem is one of the main tasks identified in the matrix group recognition project; see [OB] for details.

The goal of our research is to develop and implement algorithms to solve the constructive recognition problem in the classes of black-box classical groups. The same problem was already treated by [KS]. The main difference between our approach and that of [KS] is that we use the standard generating set of classical groups given in [LGOB] instead of the larger generating set in [KS] and that our goal is to develop algorithms that, in addition to having good theoretical complexity, perform well in practice. Another related algorithm is that of Costi's [Cos] that solves the constructive membership problem for matrix representations of classical groups in the defining characteristic. In our algorithms we reduce the more general problem to a case treated by Costi; see Step 4 below.

In order to briefly explain the main steps of our procedures, we use $\mathsf{Sp}(2n, q)$ as an example. The natural copy of $\mathsf{Sp}(2n, q)$ is the group of $2n \times 2n$ matrices over $\mathbb{F}_q$ that preserve a given (non-degenerate) symplectic form of a vector space $V = V(\mathbb{F}_q, 2n)$. The elements of $\mathsf{Sp}(2n, q)$ are considered with respect to a given basis $e_1, \ldots, e_n, f_n, \ldots, f_1$ consisting of hyperbolic pairs (e_i, f_i). The standard

K. Fukuda et al. (Eds.): ICMS 2010, LNCS 6327, pp. 54–57, 2010.

generating set $\{\overline{s}, \overline{t}, \overline{\delta}, \overline{u}, \overline{v}, \overline{x}\}$ of $\mathsf{Sp}(2n, q)$ with odd q is described in [LGOB]. Let ω be a fixed primitive element of $\mathbb{F}_q$. Then the standard generators are as follows: $\overline{s} : e_1 \mapsto f_1,\ f_1 \mapsto -e_1$; $\overline{t} : e_1 \mapsto e_1 + f_1$; $\overline{\delta} : e_1 \mapsto \omega e_1,\ f_1 \mapsto \omega^{-1} f_1$; $\overline{u} : e_1 \leftrightarrow e_2,\ f_1 \leftrightarrow f_2$; $\overline{v} : e_1 \mapsto e_2 \mapsto \cdots \mapsto e_n \mapsto e_1,\ f_1 \mapsto f_2 \mapsto \cdots \mapsto f_n \mapsto f_1$; $\overline{x} : f_1 \mapsto e_1 + f_1,\ f_2 \mapsto f_2 + e_2$. The standard generators fix the basis vectors whose images are not listed.

Suppose that G is a black-box group that is known to be isomorphic to $\mathsf{Sp}(2n, q)$ with given n and q and let us further assume that a generating set $\mathcal{X} = \{s, t, \delta, u, v, x\}$ is identified in G such that the map $\overline{s} \mapsto s$, $\overline{t} \mapsto t$, $\overline{\delta} \mapsto \delta$, $\overline{u} \mapsto u$, $\overline{v} \mapsto v$, $\overline{x} \mapsto x$ extends to an isomorphism $\mathsf{Sp}(2n, q) \to G$. This can be achieved using the algorithms described in [LGOB]. Our aim is to write a given element g of G as an SLP in $\mathcal{X}$. For $g \in G$ let $\widetilde{g}$ denote the preimage of g in $\mathsf{Sp}(2n, q)$ and let $\widetilde{g}_{i,j}$ denote the (i, j)-entry of the matrix $\widetilde{g}$. In order to avoid conjugate towers the element $a^b = b^{-1}ab$ will be denoted by $a\hat{}b$. Suppose that q is odd, set $\mathbb{F} = \mathbb{F}_q$. Our procedure is split into several steps.

Step 1. Set $S = \{g \in G \mid \widetilde{g}_{1,2n} = 0\}$. In this step we find an element $z \in G$ as an SLP in $\mathcal{X}$ such that $gz \in S$. Set $q = t\hat{}s$. For $h \in G$, we have that $h \in S$ if and only if $q^{q^h} = q$, and thus we obtain a black-box membership test in S using $O(1)$ black-box operations. Since the elements s, u, and v induce a transitive group on the subspaces $\langle e_i \rangle$, $\langle f_i \rangle$ this test can be used to test if $\widetilde{g}_{1,i} = 0$ for all $i \in \{1, \ldots, 2n\}$. If $g \in S$ then we can choose $z = 1$; hence assume that $g \notin S$. For $\alpha \in \mathbb{F}$ let z_α be the element of G that corresponds to the transformation that maps $e_1 \mapsto e_1 - \alpha e_2$, $f_2 \mapsto \alpha f_1 + f_2$, and fixes the other basis elements. If $\widetilde{g}_{1,n-1} \neq 0$, then $gz_\alpha \in S$ with $\alpha = -\widetilde{g}_{1,n}/\widetilde{g}_{1,n-1}$. Using that $z_1 = x\hat{}s$ and $z_{\omega^k} = z_1\hat{}(\delta^{-k})$, the elements z_α ($\alpha \in \mathbb{F}$) can be enumerated using $O(q)$ black-box operations and, for each such z_α, we can test if $gz_\alpha \in S$ using $O(1)$ black-box operations. If $gz_\alpha \notin S$ for all $\alpha \in \mathbb{F}$, then we conclude that $\widetilde{g}_{1,n-1} = 0$, and so $gu \in S$. Therefore the cost of finding the suitable z is $O(q)$ black-box operations. As $z_{\omega^k} = (x\hat{}s)\hat{}(\delta^{-k})$, using fast exponentiation, the required element z can be written as a SLP of length $O(\log q)$.

Step 2. In this step we assume that $g \in S$ where S is the subset defined in Step 1. Let T denote the stabilizer of the subspace $\langle e_1 \rangle$. We may assume that $\widetilde{g}_{1,1}\widetilde{g}_{1,2n-1} \neq 0$ as this can be achieved using the membership test in Step 1 with $O(n)$ black-box operations. We want to find an element z as an SLP in $\mathcal{X}$ such that $gz \in T$. If $(\alpha_2, \ldots, \alpha_n, \beta_n, \ldots, \beta_1) \in \mathbb{F}^{2n-1}$ then let $t(\alpha_2, \ldots, \alpha_n, \beta_n, \ldots, \beta_1)$ denote the element of G corresponding to the transformation that maps $e_1 \mapsto e_1 + \alpha_2 e_2 + \ldots + \alpha_n e_n + \beta_n f_n + \ldots + \beta_1 f_1$, $e_i \mapsto e_i - \beta_i f_1$, $f_i \mapsto f_i + \alpha_i f_1$ if $i \in \{2, \ldots, n\}$, and $f_1 \mapsto f_1$. Note that $gt(-\widetilde{g}_{1,2}/\widetilde{g}_{1,1}, \ldots, -\widetilde{g}_{1,2n-1}/\widetilde{g}_{1,1}, 0) \in T$. Set $b = (x\hat{}((t\hat{}s)\hat{}g)x^{-1})\hat{}s$. Then $b = t(\gamma_2, \ldots, \gamma_{2n})$ with $\gamma_i = -\widetilde{g}_{1,i}\widetilde{g}_{1,2n-1}$ for $i = 2, \ldots, 2n-1$, and $\gamma_{2n} = -\widetilde{g}_{1,2n-1}(2\widetilde{g}_{1,1} - \widetilde{g}_{1,1}^2\widetilde{g}_{1,2n} - \widetilde{g}_{1,2n-1})$. Further, $b\hat{}(\delta^{-k}) = t(\gamma_1\omega^k, \ldots, \gamma_{2n-1}\omega^k, \gamma_{2n}\omega^{2k})$. Hence there is some k_0 such that $\gamma_i\omega^{k_0} = -\widetilde{g}_{1,i}/\widetilde{g}_{1,1}$ for all $i \in \{2, \ldots, 2n-1\}$. Set $z_0 = b\hat{}(\delta^{-k_0})$. Using the membership test for S explained in the previous paragraph and using the fact that $(\widetilde{gz_0})_{1,2n-1} = 0$, the element z_0 can be found using $O(q)$ group multiplications.

Now given the element z_0, we can recover the entries $\gamma_i\omega^{k_0}$ and we can write the element $z_0 = t(\gamma_2\omega^{k_0}, \ldots, \gamma_{2n-1}\omega^{k_0})$ as SLP as follows. For $i = 2, \ldots, n-1$ and $\alpha \in \mathbb{F}$ let $x_i(\alpha) = t(0, \ldots, 0, \alpha, 0, \ldots, 0)$ where the non-zero entry appears in the $(i-1)$-th position. Let I denote the set of indices $i \in \{2, \ldots, 2n-1\}$ for which $\gamma_i \neq 0$. For $i \in I$, let k_i be such that $\gamma_i\omega^{k_0} = \omega^{-k_i}$. Then $z_0 = \prod_{i\in I} x_i(\omega^{k_i})$. As $x_2(1) = (x^{\frown}s)^{-1}$, $x_{i+1}(1)$ can be obtained from $x_i(1)$ using $O(1)$ group multiplications, and $x_i(1)^{\frown}(\delta^{-k}) = x_i(\omega^k)$, the entries $\gamma_i\omega^{k_0}$ can be recovered and z_0 can be written as an SLP using $O(nq)$ group multiplications. The length of the SLP is $O(n \log q)$.

Step 3. Now we assume that $g \in T$. We repeat Steps 1 and 2 with g^s and obtain an element z_l and z_r as SLPs in $\mathcal{X}$ such that $z_l g z_r$ is in the intersection G_1 of the stabilizers of $\langle e_1 \rangle$ and $\langle f_1 \rangle$. The cost of this step is $O(nq)$ group multiplication and the length of the SLPs to z_l and z_r is $O(n \log q)$.

Step 4. In this step we assume that $g \in G$ lies in G_1. We have, for $i \in \{2, \ldots, 2n-1\}$, that $x_i(1)^{\frown}g = t(\widetilde{g}_{i,2}/\widetilde{g}_{1,1}, \ldots, \widetilde{g}_{i,2n-1}/g_{1,1})$. Using the procedures described in Step 2, the entries $\widetilde{g}_{i,j}$ with $i,\ j \in \{2, \ldots, 2n-1\}$ can be recovered using $O(n^2q)$ multiplications. Let M denote the $(2n-2) \times (2n-2)$ matrix formed by these entries. The procedure of Costi [Cos] is used to write M as a SLP in the standard generators of $\mathsf{Sp}(2n-2, q)$. Considering $\mathsf{Sp}(2n-2, q)$ as the subgroup G_1, we evaluate this SLP in G to obtain an element z. Now gz^{-1} is a diagonal matrix with $\widetilde{(gz^{-1})}_{2,2} = \cdots = \widetilde{(gz^{-1})}_{2n-1,2n-1}$. Hence there is some k such that $gz^{-1}\delta^k \in Z(G)$.

After the end of Step 4, the element g is written as an SLP in $\mathcal{X}$ modulo the center of G using $O(n^2q)$ group operations. The length of the SLP is $O(n^2 \log q)$. We emphasize that the procedures, while using vector and matrix notation for ease of the exposition, are actually black-box procedures requiring only the basic group operations of multiplication, inversion and equality testing. Similar algorithms are developed and implemented for the classical groups $\mathsf{SL}(n, q)$ and $\mathsf{SU}(n, q)$ (with odd q). The implementations are available in the computational algebra system Magma.

Acknowledgment. Murray would like to thank the Magma project at the University of Sydney, where some of the work was carried out. Praeger would like to acknowledge the support of the Australian Research Council Discovery Grant DP0879134 and Federation Fellowship FF0776186. Schneider was supported by the FCT project PTDC/MAT/101993/2008 (Portugal) and by the OTKA grant 72845 (Hungary).

References

[Cos] Costi, E.: Constructive membership testing in classical groups. PhD thesis, Queen Mary, University of London (2009)

[HCGT] Holt, D.F., Eick, B., O'Brien, E.A.: The Handbook of Computational Group Theory. Chapman and Hall/CRC (2005)

[KS] Kantor, W.M., Seress, Á.: Black box classical groups. Memoirs Amer. Math. Soc., vol. 149 (2001)

[LGOB] Leedham-Green, C.R., O'Brien, E.A.: Constructive recognition of classical groups in odd characteristic. Journal of Algebra 322, 833–881 (2009)

[OB] O'Brien, E.A.: Algorithms for matrix groups. Groups St Andrews (Bath) (August 2009) (accepted, to appear 2010)

Towards High-Performance Computational Algebra with GAP

Reimer Behrends[1], Alexander Konovalov[1], Steve Linton[1], Frank Lübeck[2], and Max Neunhöffer[3]

[1] School of Computer Science, University of St Andrews
{rb,alexk,sal}@cs.st-and.ac.uk
[2] LDFM, RWTH Aachen
frank.luebeck@math.rwth-aachen.de
[3] School of Mathematics and Statistics, University of St Andrews
neunhoef@mcs.st-and.ac.uk

Abstract. We present the project of parallelising the computational algebra system GAP. Our design aims to make concurrency facilities available for GAP users, while preserving as much of the existing codebase (about one million lines of code) with as few changes as possible without requiring users (a large percentage of which are domain experts in their fields without necessarily having a background in parallel programming) to have to learn complicated parallel programming techniques. To this end, we preserve the appearance of sequentiality on a per-thread basis by containing each thread within its own data space. Parallelism is made possible through the notion of migrating objects out of one thread's data space into that of another one, allowing threads to interact.

Keywords: GAP, shared memory programming, threads, data spaces.

The GAP system [4], as it is introduced on the GAP Web site, is an open-source system for computational discrete algebra, with particular emphasis on Computational Group Theory. It provides a programming language, an extensive library of functions implementing algebraic algorithms written in the GAP language as well as large data libraries of algebraic objects. The kernel of the system is implemented in C, and the library is implemented in the GAP language. Both the kernel and the library are sequential and do not support parallelism.

In the 4-year long EPSRC project "HPC-GAP: High Performance Computational Algebra and Discrete Mathematics" (http://www-circa.mcs.st-and.ac.uk/hpcgap.php), started in September 2009, we aim at reengineering the GAP system to allow parallel programming in it both in shared and distributed memory programming models.

GAP has a fairly large user base (estimated in thousands of users and large number of package authors) who have a considerable investment in the current sequential architecture. While many of them will likely be interested in leveraging the increased performance of multicore processors, it is unlikely that they will accept changes that invalidate their existing code. In addition, the GAP standard library and the packages distributed with GAP are about one million lines of code

K. Fukuda et al. (Eds.): ICMS 2010, LNCS 6327, pp. 58–61, 2010.

in size, a rewrite of which would be prohibitive. These considerations constitute the main driver for our parallel programming model which should accommodate the following types of users: (1) domain experts who do not have expertise in parallel programming or lack the resources to invest into it; (2) users who wish to leverage the benefits of parallelism, but are not expert parallel programmers themselves; and (3) parallelism experts. To ensure this, it should combine at least the appearance of a sequential environment with the same level of performance with offering a set of high-level parallel tools and providing necessary low-level functionality to control how threads interact in order to optimise performance.

Below we report on the current progress with extending the GAP system with new functionality for shared memory programming (slightly more details are given also in [2]). Our parallel programming environment builds on the notion of segregating threads through disjoint *data spaces*. Data spaces form a partition of GAP objects, i.e. each GAP object is the member of exactly one data space. Only one thread at a time can have exclusive access to a data space (checked at runtime), which ensures mutual exclusion.

We distinguish between three types of data spaces. First, there is a *thread-local data space* associated with each thread, to which that thread always has exclusive access; second, there is an indefinite number of *shared data spaces* to which threads can gain exclusive or shared access through an associated read-write lock; finally, there is a single *public data space*, to which all threads have shared access at all times. If a thread has exclusive access to a data space, that means that it can perform any operation on objects within this data space; if a thread has shared access to a data space, it can perform any operation on these objects that does not interfere with other threads performing similar operations (such as read-only operations). In order for threads to interact, objects can be migrated between data spaces. An object can only be migrated out of a data space by a thread if that thread has exclusive access to that data space. This constraint makes migration very cheap: each object has a descriptor (implemented as a C pointer) for the data space containing it, so migrating just involves updating the descriptor. Since the thread has exclusive access to the data space and the object, this can be implemented as a simple memory write (certain cases still need memory barriers for consistency).

Our programming primitives frequently use *implicit migration* so that the programmer does not have to do migration manually. Functions that communicate with other threads (such as sending to or receiving from a channel) migrate some of their arguments to a target data space if they're in the current thread's thread-local data space (objects in public and shared data spaces are left where they are). This migration occurs without the user having to explicitly specify it.

Before performing an operation on an object, a thread checks the data space descriptor of the object to ensure that the thread has the proper access to perform an operation. We optimised the most common cases (in particular, access to the thread's thread-local data space and the public data space) and eliminated checks that can be statically shown to be superfluous. Thread-local data spaces accommodate our first type of users. A piece of code that executes solely within

a single thread-local data space is indistinguishable from a purely sequential program and is guaranteed to not have race conditions or deadlocks, while incurring minimal overhead. Shared data spaces aim primarily at the third type of user, the parallelism expert. Access to them is controlled by explicit read-write locks, which may be cumbersome for the non-expert to use.

Our model still protects the programmer against the two most common types of errors, race conditions, and deadlocks. Race conditions are automatically avoided in that a thread needs to lock a shared data space before it can access the objects within. Failure to lock a data space causes the data space descriptor check to fail with a runtime error. To avoid deadlocks, we require that there be a partial order on locks and that locks are acquired following that order. That means that when a thread acquires a lock B while holding a lock A, then A must be less than B based on that partial order. This order does not have to be specified by the user, but it has to exist. To ensure that it exists, we record for each lock the set of *successor locks* that were acquired while it was being held. Then, when a lock A is acquired, we check that none of the successor locks of A are currently being held. This implementation incurs little or no overhead if there is no nesting of locks, which is a very common case [1]. It also requires no explicit annotation (like [3]), which would be cumbersome for the non-expert user, as such annotations tend to be often required even in otherwise purely sequential code that uses parallelised libraries.

The public data space only contains *atomic* objects, i.e. objects that support solely atomic operations and thus can be performed by any number of threads concurrently (this includes all fully immutable objects). This is both a convenience feature so that the programmer does not have to write explicit locking code for simple objects and allows more efficient access to objects that can be implemented without locking (such as atomic counters or lock-free queues).

A very common use case for atomic objects in the public data space is that several types of GAP objects have attributes that accumulate and cache information that is often expensive to compute. Adding to accumulated data is an idempotent operation (the result is always the same, even if calculated repeatedly) and two threads can perform it concurrently without locking. Reading and adding to the accumulated data are thus atomic, and no explicit locking is necessary. A number of synchronization primitives are similarly implemented as atomic objects, such as channels, single assignment variables, and barriers. These low-level primitives allow a parallelism expert a rich toolbox to implement parallel algorithms.

Figure 1 illustrates our model; it is a simple task farm skeleton that processes a matrix by sending each row to a dedicated channel to be processed by a separate thread associated with that channel (this example is simplified; a practical implementation would require at the very least a load balancing mechanism).

We have used the following GAP primitives to implement this example. CreateThread($function, arg_1, \ldots, arg_n$) starts a new thread that will execute $function$ with arguments $arg_1, \ldots, arg_n$. WaitThread($thread$) waits for $thread$ to finish. Channels are used to move objects from one

```
TaskFarmByRow := function(matrix, process)
  local threads, channels, i;
  threads := []; channels := [];
  for i in [1..Length(matrix)] do
    channels[i] := CreateChannel();
    threads[i] := CreateThread( function(ch)
         local row;
         row := ReceiveChannel(ch);
         process(row);
         SendChannel(ch, row);
       end,
       channels[i]);
    SendChannel(channels[i], matrix[i]);
  od;
  for i in [1..Length(matrix)] do
    WaitThread(threads[i]);
    matrix[i] := ReceiveChannel(channels[i]);
  od;
end;
```

Fig. 1. Splitting work between multiple threads

thread to another; they are implemented internally as shared data spaces. SendChannel(*channel*, *object*) will implicitly migrate *object* to *channel*'s data space. Likewise, ReceiveChannel(*channel*) will implicitly migrate the object received from *channel* to the receiving thread's local data space.

This example has some (intentional) similarities to traditional message passing systems. However, unlike a pure message passing system, objects are not copied when sent through a channel, but rather passed by reference. Access checks ensure that race conditions do not occur as a result of this. For example, in Figure 1, the main thread cannot access the rows of the matrix while they are being processed by its child threads.

References

1. Bacon, D.F., Konuru, R., Murthy, C., Serrano, M.: Thin locks: featherweight synchronization for java. In: PLDI 1998: Proceedings of the ACM SIGPLAN 1998 Conference on Programming Language Design and Implementation, pp. 258–268. ACM, New York (1998)
2. Behrends, R., Konovalov, A., Linton, S., Lübeck, F., Neunhöffer, M.: Parallelising the computational algebra system GAP. Extended abstract. In: PASCO 2010 (accepted 2010)
3. Boyapati, C., Lee, R., Rinard, M.: Ownership types for safe programming: preventing data races and deadlocks. In: Proceedings of the 17th ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications. ACM SIGPLAN Notices, vol. 37(11), pp. 211–230. ACM, New York (2002)
4. The GAP Group. GAP – Groups, Algorithms, and Programming, Version 4.4.12 (2008), `http://www.gap-system.org`

An Improvement of a Function Computing Normalizers for Permutation Groups

Izumi Miyamoto

Department of Computer Science and Media Engineering, University of Yamanashi
Takeda 4-3-11 Kofu. 400-8511, Japan
imiyamoto@yamanashi.ac.jp
http://shingen.ccn.yamanashi.ac.jp/~imiyamoto

Abstract. Computation of normalizers for permutation groups is sometimes very hard even if the degrees are small. The author previously obtained some methods to compute normalizers of permutation groups and wrote programs implementing the methods. The programs were mainly applied to groups of small degree such as transitive groups of degree up to 30 in the GAP library. The author will tune up the implementation to speed up the computation of normalizers of permutation groups and apply it to permutation groups of degree up to 100. In our experiments, the normalizers of the primitive groups up to degree 100 and their stabilizers of one point in the symmetric groups are computed.

Keywords: permutation group, normalizer, association scheme.

1 Introduction

The computation of normalizers of permutation groups is sometimes very hard even if they are of small degree. We use GAP system[1] for our computation and our programs are written in GAP programming language. Most of permutation groups of small degree are easy to compute their normalizers. The GAP function Normalizer computes them very quickly. But readers may easily find transitive groups G of degree n in the GAP library such that the normalizers of G in the symmetric groups of degree n can not be computed within a reasonable time by using the GAP function. We obtained some algorithms to compute the normalizers of permutation groups in [6,7,8] and the programs implementing our algorithms can compute rather quickly the normalizers of permutation groups of small degree which are hard for the GAP function Normalizer. In [7] block systems are used for the computation of normalizers of imprimitive permutation groups. The method in [7] can use the algorithm obtained in [6] for the groups constructed by the actions on block systems and by the stabilizers of one of the blocks. In the present paper we will improve the implementation of the algorithm shown in [6,8].

Computation of normalizers depends on a partition backtrack methods introduced in [3,4] and the method is now a fundamental one in the computation for permutation groups. But as noted in [9] that we have to find a balance between

K. Fukuda et al. (Eds.): ICMS 2010, LNCS 6327, pp. 62–68, 2010.

the cost of the base change and the gain in pruning the search tree, there exists problems in practical implementations. A polynomial-time algorithm for computing normalizers of permutation groups under some condition is obtained in [5] but the algorithm is very complicated and it has not been implemented.

In our algorithm we use an easy group theory shown in Lemma 1 and compute the automorphism groups of some combinatorial objects in the lemma by a partition backtrack method. We also use GAP Normalizer function in some small groups. Under such conditions we will improve our implementation. In section 3 we will show the improvements and in section 4 we mention about the backtrack method computing the automorphisms.

Table 2 shows that there exist not a few groups G such that the normalizer of G is very quickly computed, while that of stabilizer subgroup of one point in G can not be computed in a reasonable time by the GAP Normalizer function. So it is not true that the computation of a smaller group is easier than that of a larger group for normalizers. However in our algorithm shown in Lemma 1 we will find a sufficiently small group to compute the normalizer in it under some conditions.

In [6,7,8] some groups of degree even larger than 100 are computed, but mainly the transitive groups of degree up to 30 are considered. In the present paper we treat primitive groups of degree from 31 to 100 in the GAP library. There exist no list of imprimitive groups available. So we also consider the stabilizers of a point in the primitive groups stated above. We will go forward computing normalizers of the primitive groups of degree larger than 100 in the GAP library. We use a computer with CPU Intel Xeon E5335 2.0GHz and 16GB memory under Linux.

2 Preliminaries

We use combinatorial objects called association schemes or coherent configurations and apply an easy group theory to compute normalizers of permutation groups in the automorphism groups of the combinatorial objects formed by the groups.

Let $\Omega = \{1, 2, \cdots, n\}$ and let R_k, $1 \leq k \leq d$, be subsets of $\Omega \times \Omega$.

Definition. $(\Omega, \{R_k\}_{k=1,2,\cdots,d})$ is a *coherent configuration* if it satisfies the following:

- CC1. $\{R_1, R_2, \cdots, R_d\}$ is a partition of $\Omega \times \Omega$;
- CC2. For some $r < d$, $\{R_1, R_2, \cdots, R_r\}$ is a partition of the diagonal $\{(x, x)|x \in \Omega\}$ of Ω;
- CC3. For every k there exists k^* such that $R_{k^*} = \{(y, x)|(x, y) \in R_k\}$;
- CC4. There exist constant numbers $p_{i,j,k}$ such that for any $(x, z) \in R_k$ the number of points $y \in \Omega$ such that $(x, y) \in R_i$ and $(y, z) \in R_j$ is equal to $p_{i,j,k}$.

Set $\Omega_k = \{x \in \Omega|(x, x) \in R_k\}$ for $1 \leq k \leq r$ in CC2. If we collect relations R_i such that $R_i \subseteq \Omega_k \times \Omega_k$ for each Ω_k, $1 \leq k \leq r$, then each Ω_k with such relations

forms a coherent configuration and we call it a *fiber* of the configuration. A coherent configuration which has only one fiber is called an "*association scheme*".

A permutation g on Ω acts on $\Omega \times \Omega$ naturally by $(x,y)^g = (x^g, y^g)$ for $x, y \in \Omega$. Let G be a permutation group on Ω. Then it is well known that the orbits of G acting on $\Omega \times \Omega$ forms a coherent configuration. We consider an automorphism g of a configuration satisfying $R_k^g = \{(x^g, y^g)|(x,y) \in R_k\} = R_{k'}$ for all k. Then the normalizer of G in the symmetric group $Sym(n)$ of degree n is contained in the automorphism group of the configuration formed G. We use the following lemma in [6] to obtain the normalizer of G in small groups.

Lemma 1. *Let K be a permutation group on Ω. Let F be a tuple $[p_1, p_2, \cdots, p_r]$ of points in Ω and let G^i be the stabilizer of the subset $[p_1, p_2, \cdots, p_i]$ of F as a tuple in G for $i = 1, 2, \cdots, r$. Let N^i be the automorphism group of the configuration formed by G^i on $\Omega \backslash [p_1, p_2, \cdots, p_i]$. Set $N^0 = N$, $G^0 = G$ and set $N^{\{0..i\}} = N^0 \cap N^1 \cap \cdots \cap N^i$. Suppose that $G^i \cap K$ is transitive on the orbit of $N^{\{0..i\}} \cap K$ containing the point p_{i+1} for $i = 0, 1, \cdots, r-1$. Then the normalizer of G in K is generated by $G \cap K$ and the normalizer of G in $N^{\{0..r\}} \cap K$.*

3 Improvements and Experiments

We will explain our algorithm using examples and will show the speed up of the computations of normalizers of permutation groups G in the symmetric groups $Sym(n)$. The normalizers of G in K will be denoted by $N(K, G)$. Norm denotes the GAP Normalizer function. So N =Norm means that the normalizer is computed by the GAP function. The normalizer function introduced in [6] will be denoted by ASN and our improved function will be denoted by CCN. Both functions are written in GAP programming language. $G_n = \{g \in G | n^g = g\}$ denotes the stabilizer of the point $n \in \Omega$ in G.

For primitive groups G of degree n, $31 \leq n \leq 100$, in the GAP library the normalizers $N(Sym(n), G)$ and $N(Sym(n-1), G_n)$ were computed. The results of our computation are shown in Table 1 and 2. Computing time is in seconds. We did not list the results of $G = Sym(n)$ or $Alt(n)$, the alternating group of degree n, in the tables. The first column of Table 1 shows the time ranges and the remaining columns show the numbers of groups of which normalizer in the symmetric groups in each time range, where "fail" means that it can not be computed in a reasonable time. There exist three groups G for which it was hard to compute $N(Sym(n-1), G_n)$ and it took 6, 47 and 189 hours for the computations respectively. Some typical examples are shown in Table 2. From Table 1 most primitive groups are easy to compute Norm$(Sym(n), G)$. and the GAP function is the quickest one in many cases. Some of such groups can be seen in Table 2. For the computations $N(Sym(n-1), G_n)$, Norm does not seem to work so quickly comparing the computations $N(Sym(n), G)$. In both computations $N(Sym(n), G)$ and $N(Sym(n-1), G_n)$, there are some groups of which normalizers can not be computed in a reasonable time by the function Norm. From Table 1 we can see that all the computations by CCN and ASN were done within a reasonable time. Also we can see how faster is CCN than ASN in

general. Each improvement is explained in the examples below. All computing times for groups of degree from 31 to 100 in the following examples are listed in Table2. We start to tune up our program using groups of small degree.

Example 1. $G =$TransitiveGroup$(27, 333)$ in the GAP library. We have two points p_1 and p_2 in Lemma 1. The sizes of the orbits of $G = G^0$ and G^1 containing the points respectively are 27 and 2. Then we have three groups G^i, $0 \leq i \leq 2$, of order 1458, 54 and 27 respectively. We compute the automorphism groups to obtain three groups $N^{\{0..i\}}$, $0 \leq i \leq 2$ of order 3265173504, 120932352 and 3779136 respectively. Then the normalizer of G in $Sym(27)$ is generated by G and the group $N(N^{\{0..2\}}, G) = N(N(N(N^{\{0..2\}}, G^2), G^1), G^0)$ by Lemma1. We compute this normalizer by Norm. We note that each side of the above equation shows how to compute this group. It took 4.8 seconds for the computation of the left hand side, which is used in [6], while 0.15 seconds for that of the right hand side. So we will choose the computation following the right hand side of the equation. It took 114 seconds for the direct computation Norm$(Sym(n), G)$. Moreover we note that because the index $|G^1 : G^2| = 54/27 = 2$ is small, we had thought that we might gain little advantage and end in loss of time by this processing. But this was not true, since the index $|N^{\{0..1\}} : N^{\{0..2\}}| = 32$. An experiment revealed that this was in fact not so small.

Example 2. $G =$TransitiveGroup$(30, 1268)$. In [8] the automorphism group of the coherent configuration formed by each G^i is used, while in [6] we compute the automorphism groups of the fibers of the configuration and the isomorphisms among the fibers and consider the group generated by these permutations. For instance, if there exist r fibers and all the fibers are isomorphic mutually, then we obtain a wreath product of the automorphism of a fiber by $Sym(r)$. Usually the latter computation is by far easier than the former computation. We used a coherent configuration for the computation of N^1, if G is regular on the set of blocks and G_n has t orbits of size s and one orbit of size $s - 1$ with $n = |\Omega| = s(t + 1)$. In this case the size of N^1 is reduced 120 times by the method in [6] using the fibers. It took 1.2 seconds for the computation by CCN, 253 seconds by ASN and 5.5 seconds by ASN with the algorithm shown in Example 1. Norm$(Sym(30), G)$ can not be computed within a reasonable time.

Example 3. $G =$PrimitiveGroup$(81, 22)$ and we computed $N(Sym(80), G_{81})$. The computing times are shown in Table2. If we use a coherent configuration formed by the one point stabilizer G_{81}, we can compute the normalizer in the automorphism group of the configuration in 65 seconds. It took about 20 minutes for this computation without a coherent configuration.

On the contrary, if $G =$PrimitiveGroup$(81, 21)$, then it took 78 seconds for the computation using the coherent configuration formed by the stabilizer G_{81}, while it took about 1.3 seconds by the methods shown in Table2. We considered another version of an improvement such that we use coherent configurations for the stabilizer G_{81} in $G =$PrimitiveGroup$(81, 22)$. But there appeared 6 groups G for which it took more than 30 seconds to compute $N(Sym(n - 1), G_n)$ by the version, while the present version has only one troublesome group shown here

and the normalizers of one point stabilizers of the other groups are computed within 10 seconds.

Example 4. G =PrimitiveGroup$(100, 22)$. In this case G^4 =Group$(())$ (*identity group*). So $N^4 = Sym(96)$. It takes tens of seconds for the computation with $Sym(n)$ of this size. So we stop the application of Lemma 1 before G^r =Group$(())$.

Example 5. G =PrimitiveGroup$(85, 3)$ and we compute $N(Sym(84), G_{85})$. By ASN, $N^0 \cong$WreathProduct$(Sym(4), Sym(21))$ is computed with 1030 generators. In this case the automorphism group of the association scheme formed by G_{85} is computed within 2 seconds. The automorphisms of combinatorial objects such as association schemes of this size seem to be computed quickly like this. Our program computing the automorphism group used in [6] did not consider the group action of automorphisms obtained in each step of the backtrack search. If we use this group actions in order to prune the search tree in the backtrack search, we should compute some orbits and stabilizers of these groups. But it seems to take more time than the computation without group actions, as is noted in the introduction. We rewrote a part of this program only computing some orbits of the groups. It seems to work faster than our old program if $n = |\Omega| > 100$. We will briefly explain about this computation in the next section. However our new program computes the automorphism group with 131 generators in this case. So the remaining computations in Lemma 1 can be done quickly and the normalizer is computed in 3 seconds or so by CCN, while it took 30 seconds by ASN. Norm$(Sym(84), G_{85})$ can not be computed in 5 days.

4 A Computation of Automorphisms of Configurations

In our program computing automorphisms, we rewrote the part in which compute automorphisms g satisfying $R_k^g = R_k$ for all relations R_k in a coherent configuration.

A doubly transitive group of degree n has same orbits on $\Omega \times \Omega$ as the symmetric group $Sym(n)$. So $Sym(n)$ comes out as the automorphism group of the association scheme formed by a doubly transitive group. In fact this automorphism group is obtained without computation. We will explain our method computing automorphisms using this trivial association scheme. Suppose that we have done the backtrack search at depth more than i in the search tree. Let $G^{(i+1)}$ be the group generated by the automorphisms obtained by these searches. If we do not use the action of this already constructed group $G^{(i+1)}$, we may compute permutations $(i, i+1)$, $(i, i+2)$, $\cdots$, (i, n) as automorphisms in the search starting at depth i and point i. If we consider the action of $G^{(i+1)}$ on $\{i, i+1, \cdots, n\}$, we have a permutation $(i, i+1)$ and the search finishes, since $G^{(i+1)}$ is transitive on $\{i+1, i+2, \cdots, n\}$. In this search we may assume that, by the group action, an automorphism which we are computing moves point i to point $i+1$. So the search starts at depth i and point $i+1$. Then in the successive search we compute where $i+1$ is moved to. If we use the action of the already obtained group $G^{(i+1)}$ again, we should compute the stabilizer of point $i+1$

in $G^{(i+1)}$ and its action on the set $\{i, i+2, i+3, \cdots, n\}$ in order to prune the search tree. This process may be repeated at further depth in the search tree. In our methods we compute the action of $G^{(i+1)}$ at depth i and do not compute it in the successive search, because we found that the computation of a stabilizer seemed to be time consuming.

Therefore if we do not use the group action, we have an automorphism group with $n(n-1)/2$ generators for the trivial scheme, while we have a group with $n-1$ generators, if we use the group action. The next difficult case, which means that many automorphisms come out, is a wreath product of symmetric groups shown in Example 5. We are computing automorphisms of this type of association schemes formed by primitive groups of degree up to 1000 and of coherent configurations formed by stabilizers of one point in the groups.

Finally, we mention one more point. Let $(\Pi_1, \Pi_2, \cdots, \Pi_r)$ be a partition used in a partition backtrack. If two points $p, q \in \Pi_i$ have a different number of neighbors with respect to some relation R_j in some Π_k, then there exists no automorphism moving p to q. This argument refines the partition and the refinement process can be iterated. But this known argument seems to be time consuming, too. We do not use this argument in our old program. In our rewritten program, we use this argument under a very restricted condition such that, for instance, Π_k consists of at most two points.

Table 1. Computing times of the normalizers of primitive groups G in $Sym(n)$ and its one point stabilizer G_n in $Sym(n-1)$ with $Alt(n) \not\subseteq G$ and $31 \leq n \leq 100$ in seconds

	$N(Sym(n), G)$			$N(Sym(n-1), G_n)$		
time range	CCN	ASN	Norm	CCN	ASN	Norm
$* < 0.02$	0	0	56	0	0	4
$0.02 \leq * < 0.05$	3	0	148	11	10	28
$0.05 \leq * < 0.1$	25	6	155	44	28	43
$0.1 \leq * < 0.2$	64	16	128	78	41	80
$0.2 \leq * < 0.5$	139	69	24	160	114	117
$0.5 \leq * < 1$	167	70	9	121	65	56
$1 \leq * < 2$	111	125	3	103	119	73
$2 \leq * < 5$	62	129	9	48	120	40
$5 \leq * < 10$	5	108	0	10	41	32
$10 \leq * < 20$	0	45	2	0	30	16
$20 \leq * < 60$	0	8	4	0	5	20
$60 \leq * < 180$	0	0	0	0	2	3
$180 \leq * < 600$	0	0	0	0	0	19
$600 \leq * < 1800$	0	0	4	1	1	3
$1800 \leq * < 3600$	0	0	11	0	0	3
$3600 \leq * < 18000$	0	0	9	0	0	0
6, 47, 189 hours	0	0	0	0	0	3
fail	0	0	14	0	0	16
tot. time	558	2234		1712	2894	
max. time	7.2	42.8		1208	1227	

Table 2. Computing times of the normalizers of some primitive groups G in $Sym(n)$ and its one point stabilizer G_n in $Sym(n-1)$ with $Alt(n) \not\subseteq G$ and $31 \leq n \leq 100$ in seconds

G =PrimitiveGroup(n,k)			$N(Sym(n), G)$			$N(Sym(n-1), G_n)$		
n	k	group name	CCN	ASN	Norm	CCN	ASN	Norm
64	45	AGL(3, 4)	2.4	10.5	0.07	6.3	14.9	fail
81	21	$3^4 : D_{16} : 2$	7.2	6.0	0.06	1.3	1.3	1.2
81	22	$3^4 : (2 \times Q_8) : 2$	0.6	1.0	0.2	1207.8	1226.9	1092.7
81	35	$3^4 : 2^3 : D_8$	1.1	5.1	0.1	8.6	8.6	257.7
81	81	$3^4 : Q_8.Sym(3) : 2^2$	1.0	1.7	0.1	0.8	1.0	2494.9
81	110	ASL(4, 3)	2.9	42.8	0.06	3.5	78.6	fail
81	115	3^4 :SL(2, 9): 2^2	0.8	8.3	0.1	0.8	4.3	682279.4
85	3	PSL(4, 4)	3.2	27.2	fail	3.8	32.1	fail
91	7	PSL(3, 9)	4.5	32.2	fail	5.8	32.0	fail
97	1	C(97)	6.7	14.9	0.2	5.7	5.6	5.1
97	12	AGL(1, 97)	6.8	23.6	0.1	7.0	14.3	2.7
98	2	PGL(2,97)	6.7	30.1	3.7	6.9	25.6	0.1
100	22	$Alt(6)^2.D_8$	1.8	11.9	13913.4	1.5	10.4	3.1
100	27	$Alt(6)^2.D_8$	3.6	13.9	9102.7	1.9	11.1	1711.7

References

1. The GAP Groups. Gap - groups, algorithms and programming, version 4. Lehrstuhl D für Mathematik, Rheinisch Westfälische Technische Hochschule, Aachen, Germany and School of Mathematical and Computational Sciences. Univ. St. Andrews, Scotland (2000)
2. Higman, D.G.: Coherent configurations I. Rendiconti del Seminario Matematico della Universit a di Padova 44, 1–25 (1970)
3. Leon, J.S.: Permutation group algorithms based on partitions, I: Theory and algorithms. J. Symbolic Comput. 1(12), 533–583 (1991)
4. Leon, J.S.: Partitions, refinements, and permutation group computation. In: Finkelstein, L., Kantor, W.M. (eds.) Groups and Computation II, DIMACS 1995. Amer. Math. Soc. DIMACS Series, vol. 28, pp. 123–158 (1997)
5. Luks, L.M., Miyazaki, T.: Polynomial-time normalizers for permutation groups with restricted composition factors. In: Proceedings of the 2002 International Symposium on Symbolic and Algebraic Computation, pp. 176–183 (2002)
6. Miyamoto, I.: Computing normalizers of permutation groups efficiently using isomorphisms of association schemes. In: Proceedings of the 2000 International Symposium on Symbolic and Algebraic Computation, pp. 200–204 (2000)
7. Miyamoto, I.: An improvement of GAP Normalizer function for permutation groups. In: Dumas, J.-G. (ed.) Proceedings of the 2006 International Symposium on Symbolic and Algebraic Computation, pp. 234–238 (2006)
8. Miyamoto, I.: Computation of Isomorphisms of Coherent Configurations. ARS Math. Contemporanea (to appear)
9. Seress, A.: Permutation Group Algorithms. Cambridge Tracts in Math., vol. 152. Cambridge Univ. Press, Cambridge (2003)

A GAP Package for Computation with Coherent Configurations

Dmitrii V. Pasechnik and Keshav Kini

Division of Mathematical Sciences
School of Physical and Mathematical Sciences
Nanyang Technological University, Singapore
dima@ntu.edu.sg,
kini@member.ams.org

Abstract. We present a GAP package for computing with Schurian coherent configurations and their representations.

Keywords: GAP, coherent configuration, association scheme, permutation group, GRAPE, Sage, semidefinite programming, centralizer ring.

1 Introduction

Coherent configurations (CCs, for short) appear in the study of permutation groups, and can be considered a topic in algebraic combinatorics. Specifically, they generalize the centralizer algebra of a permutation representation of a finite group. A special class of CCs known as association schemes (cf. e.g. [1]) is also extensively studied. The need to compute with CCs and their representations arises outside their immediate domain[1].

For instance, they play an important role in the study of combinatorial properties of symmetric graphs arising in coding theory. The area was founded upon the seminal work of Delsarte, Lovasz, and Schrijver in the 1970s, and later received a new lease on life thanks to developments in semidefinite programming (SDP) solvers that made it possible to numerically optimize linear functions on cones of positive semidefinite elements of matrix algebras, subject to linear constraints. The relevant literature is extensive—we can cite only a small part of it here [5,6,7,8].

One more use of our package we can mention concerns nonassociative algebras related to the Monster sporadic simple group; see e.g. [9].

In a nutshell, CCs allow for "dimension reduction". For example, in the aforementioned coding theory/SDP setting, one reduces the computational problem

[1] Due to space constraints we will not review here the many works where association schemes and CCs are used in the theory of permutation groups; see e.g. [2], or very recent [3]. We can mention that jointly with Csaba Schneider we are using the package described here to work on *synchronizing* permutation groups; see http://www.maths.qmul.ac.uk/~pjc/LTCC-2010-intensive3/

K. Fukuda et al. (Eds.): ICMS 2010, LNCS 6327, pp. 69–72, 2010.

of SDP in the space of $2^d \times 2^d$ real symmetric matrices to the space of $d^k \times d^k$ matrices, for a small fixed k.

Another important application domain for CCs is the graph isomorphism problem. For instance, a recent polynomial-time algorithm for isomorphism of graphs with a common forbidden minor [10] constructs a certain CC, which is a generalization of a *stabilization* procedure due to Leman and Weisfeiler [11], who pioneered this approach.

We are currently developing a package for the computer algebra system GAP [12]. This package will enable a user to create, manipulate, and otherwise represent CCs within the GAP system (in particular, by linear representations), and to use upon them other tools GAP provides. A prototype is available [4]. As SDP solvers, needed for applications mentioned above, are difficult to interface with GAP, we are also developing a Sage [13] package that pulls CC data from GAP and calls one of the SDP solvers available there, such as CVXOPT.

2 Coherent Configurations

A *coherent configuration* (or CC) of *degree* n and *dimension* d is a finite collection $\mathcal{A} = \{A_1, \ldots, A_d\}$ of d nonzero 0-1 matrices of dimensions $n \times n$, satisfying the following four axioms [2].

1. $\sum_{i=1}^{d} A_i = J_n$, the all-1 matrix.
2. It is possible to find some subset of $\mathcal{A}$ whose sum is I_n.
3. If $A \in \mathcal{A}$, then $A^\top \in \mathcal{A}$.
4. There exist natural numbers $p_{i,j}^k$, $0 < i, j, k \leq d$, s.t. $A_i A_j = \sum_{k=1}^{d} p_{i,j}^k A_k$ holds for all i, j.

Two CCs are called *isomorphic* if there exists a bijection between them of the form $\varphi : A \mapsto PAP^{-1}$ for some permutation matrix P. The numbers $p_{i,j}^k$ stipulated in the fourth axiom are called the *intersection numbers* of the CC, and are invariant under isomorphism.

Note that a CC can be equivalently regarded as a partition of Ω^2 for a set Ω of size n. By the second axiom, every CC induces a partition of Ω; the parts are called the *fibers* of the CC, and a CC with only one fiber is called *homogeneous*. For a CC $\mathcal{A}$, each $A \in \mathcal{A}$ is associated with a *left* and a *right fiber*, with the nonzero entries of A occurring only in rows indexed by the left fiber and columns indexed by the right fiber.

In the algebra $M_n(\mathbb{C})$ of all $n \times n$ matrices, $\mathcal{A}$ forms a linearly independent set. This, with the fourth axiom, shows that $\mathcal{A}$ spans a subalgebra $\mathbb{C}[\mathcal{A}]$, which is called its *basis algebra*. This algebra has special properties: it is self-adjoint, it contains I, it contains J (the all-1 matrix), and it is closed under the entrywise (or Hadamard) matrix product.

A concrete example of a CC is the *Schurian CC* arising from a permutation group (G, Ω) [2]. Let the *orbitals* of (G, Ω) be the orbits of G acting on Ω^2 in the natural way; then the characteristic arrays of the orbitals together form a CC. Further, the orbits of G on Ω are the fibers of the CC. The basis algebra

of the CC is exactly the *centralizer algebra* of the permutation representation of G in GL_n, considered as a set of matrices in the matrix algebra – that is, the algebra of matrices which commute with all elements of G in the representation.

3 GAP Package Functionality

We have embarked upon a project to build a package for the computer algebra system GAP that allows users to handle CCs as encapsulated objects. The eventual goal of the project is to incorporate it into the main GAP distribution as a package that implements a wide variety of operations and computations that can be done with CCs.

The current version is based around Schurian CCs, and can do all the obvious tasks, such as computing the Schurian CC arising from a given permutation group, computing the intersection numbers, etc. While packages such as the well-known GRAPE [14] are able to do this in the homogeneous case, we have no such restriction. Since we store CCs in a nontrivial way (actually by indexing CC elements by their left and right fibers and a third parameter), the package also provides full emulation of the surface form of the CC, being able to output the CC's elements directly or in a sparse matrix form. This functionality is also available in the standalone system CoCo by I. Faradjev [15].

The array of intersection numbers $\{p_{i,j}^k\}$ of a CC $\mathcal{A}$ can be separated along the j dimension to produce a set of d many $d \times d$ matrices called the *intersection matrices* of $\mathcal{A}$. It turns out that these matrices span an algebra isomorphic to the basis algebra, called the *intersection algebra* or sometimes the *regular representation* of the CC, and denoted $\operatorname{reg}\mathcal{A}$ [2]. In fact, it is easy to turn it into a *-isomorphism, i.e. a mapping that also preserves the conjugate transpose operation, cf. [6]. This turns out to be crucial for the SDP-related applications. Our package provides functions to compute $\operatorname{reg}\mathcal{A}$ and to output its basis in a dense of a sparse form.

We are also able to use the Schurian CC $\mathcal{A}$ arising from a permutation group (G, Ω) to efficiently compute the class sum $\sum_{c \in C} \rho(c)$, where ρ is the permutation representation of (G, Ω) and C is some conjugacy class of G. As the class sums over all C span the center of $\operatorname{span}\rho(G)$, we can compute that algebra as well. Crucially, the latter, together with the knowledge of the irreducible characters $\operatorname{Irr}(G)$ of G, allows us to compute the irreducible representations of $\operatorname{reg}\mathcal{A}$. More precisely, we now have functionality to compute surjections $\phi_\chi : \operatorname{reg}\mathcal{A} \to \operatorname{reg} M_{m_\chi}(\mathbb{C})$, where m_χ is the multiplicity of the irreducible character χ in ρ.

4 Ongoing and Future Work

Presently we are developing a method to compute the aforementioned irreducible representations of $\operatorname{reg}\mathcal{A}$, starting from $\phi_\chi(\mathcal{A})$, which is an m_χ^2-dimensional subalgebra of $M_{m_\chi^2}(\mathbb{C})$ isomorphic to $M_{m_\chi}(\mathbb{C}) \otimes I_{m_\chi}$. This will allow us to construct the explicit isomorphism $\operatorname{reg}\mathcal{A} \to \oplus_{\chi \in \operatorname{Irr}(G)} M_{m_\chi}(\mathbb{C})$.

Every CC has a refinement that is Schurian – if nothing else, the discrete partition of Ω^2 is the Schurian CC arising from the trivial permutation group on Ω, and refines all CCs. Thus if we adapt our code to consider coarsenings of Schurian CCs, we can handle CCs in full generality.

References

1. Bannai, E., Ito, T.: Algebraic Combinatorics I: Association Schemes. Benjamin/Cummings (1984)
2. Cameron, P.J.: Permutation groups. London Mathematical Society Student Texts, vol. 45. Cambridge University Press, Cambridge (1999)
3. Miyamoto, I.: Computation of isomorphisms of coherent configurations. Ars Mathematica Contemporanea 3(1) (2010)
4. Pasechnik, D., Kini, K.: Cohcfg, a GAP package for coherent configurations (preliminary version) (2010), http://www1.spms.ntu.edu.sg/~dima/software/cohcfg-a1.tgz
5. Schrijver, A.: New code upper bounds from the Terwilliger algebra and semidefinite programming. IEEE Trans. Inform. Theory 51(8), 2859–2866 (2005)
6. de Klerk, E., Pasechnik, D.V., Schrijver, A.: Reduction of symmetric semidefinite programs using the regular *-representation. Math. Prog. B 109, 613–624 (2007); e-print 2005-03-1083, Optimization Online
7. de Klerk, E., Pasechnik, D.V., Maharry, J., Richter, B., Salazar, G.: Improved bounds for the crossing numbers of $K_{m,n}$ and K_n. SIAM J. Discr. Math. 20, 189–202 (2006)
8. Vallentin, F.: Symmetry in semidefinite programs. Linear Algebra Appl. 430(1), 360–369 (2009)
9. Ivanov, A.A., Pasechnik, D.V., Seress, A., Shpectorov, S.: Majorana representations of the symmetric group of degree 4. J. of Algebra (submitted)
10. Grohe, M.: Fixed-point definability and polynomial time on graph with excluded minors. In: 25th IEEE Symposium on Logic in Computer Science, LICS 2010 (to appear 2010)
11. Weisfeiler, B. (ed.): On Construction and Identification of Graphs. LNM, vol. 558. Springer, Berlin (1976)
12. The GAP Group: GAP – Groups, Algorithms, and Programming, Version 4.4.12. (2008), http://www.gap-system.org
13. Stein, W., et al.: Sage Mathematics Software (Version 4.4). The Sage Development Team (2010), http://www.sagemath.org
14. Soicher, L.H.: GRAPE: a system for computing with graphs and groups. In: Finkelstein, L., Kantor, W. (eds.) Groups and Computation. DIMACS Series in Discrete Mathematics and Theoretical CS, vol. 11, pp. 287–291. AMS, Providence (1993)
15. Faradzev, I.A., Klin, M.H.: Computer package for computations with coherent configurations. In: ISSAC 1991, Proc. Symposium on Symbolic and Algebraic Computation, pp. 219–221. Association for Computing Machinery, New York (1991)

CoCoALib: A C++ Library for Computations in Commutative Algebra ... and Beyond

John Abbott and Anna M. Bigatti

Università degli Studi di Genova, Italy
{abbott,bigatti}@dima.unige.it

1 The Main Features of CoCoA

First released in 1988, CoCoA is a *special-purpose* system for doing **Co**mputations in **Co**mmutative **A**lgebra: *i.e.* it is a system specialized in the algorithmic treatment of polynomials. It is *freely available* and offers a textual interface, an Emacs mode, and a graphical user interface common to most platforms ([6]).

One of the main purposes of the CoCoA system is to provide a "laboratory" for studying and using computational commutative algebra: it belongs to an elite group of highly specialized systems having as their main forte the capability to calculate Gröbner bases. This means that CoCoA is optimized for working with multivariate polynomials, their ideals and modules, and operations on these objects. Other special strengths of CoCoA include polynomial factorization, exact linear algebra, Hilbert functions, zero-dimensional schemes, and toric ideals.

The usefulness of these technical skills is enhanced by its programming language, the CoCoALanguage, which places great emphasis on being easy and natural to use. So CoCoA is the system of choice for teaching advanced courses in several universities, and for many researchers wanting to explore and develop new algorithms without the administrative tedium necessary when using "low-level" languages.

2 The Future of CoCoA: More Than a System

Almost 10 years ago, a new initiative began: namely, to rebuild the software laboratory but without the inherent limitations of the original. The new software comprises three main components: a C++ library (CoCoALib), an algebra computation server (CoCoAServer), and an interactive system (CoCoA-5). Of these components CoCoALib is the heart; it embodies all the "mathematical knowledge" and it is the most evolved part ([1]). The roles of the other two parts are to make CoCoALib's capabilities more easily accessible.

All the new code is free and open source software. It is downloadable from our website ([1]) and released under GPL.

K. Fukuda et al. (Eds.): ICMS 2010, LNCS 6327, pp. 73–76, 2010.

3 Philosophy behind CoCoALib

The aims of CoCoALib include offering better flexibility and performance while retaining the simplicity of use for which CoCoA has become widely appreciated.

So that the enormous investment in rebuilding is worthwhile we want to ensure that the new code will enjoy active service for a long period of time. Consequently our implementations have to satisfy various design criteria:

- be easy and natural to use
- exhibit good run-time performance
- have firm mathematical basis (following books [8,9])
- be clear and well designed
- be clean and portable
- be well documented (both for users and maintainers)

We regard clear and comprehensible code as being generally more desirable than convoluted code striving for the highest possible speed. Conveniently our experience up to now shows that this emphasis on cleanliness is also providing quite good run-time performance! In particular Gröbner basis computations are much faster than in the old C version (CoCoA-4), and are now comparable with the other specialized systems Macaulay2 ([10]) and Singular ([11]).

The inheritance mechanism of C++ plays a crucial role in the design of CoCoALib (see [2]), especially in the challenge of reconciling the traditionally conflicting goals of (mathematical) abstraction and efficiency: for example it is used to express the mathematical relationships between the various sorts of rings and their specific functions (*e.g.* `deg` for polynomial rings, `den` for fraction fields)

Being well aware that the usefulness of software is critically dependent on its documentation, we offer extensive documentation aimed both at guiding users and at aiding maintainers and contributors. And being even more aware that no one likes to read documentation, we also offer a good selection of example programs — so you can just cut-and-paste rather than read tediously through the documentation!

CoCoALib is unique in its field because, right from the outset, it was designed as an open source library. This makes it an ideal choice as a basis upon which other researchers can develop efficient and robust implementations of their algorithms. Naturally we hope that many of these implementations will then be donated as new components for the library, helping to expand it.

4 Approximately...

While being primarily concerned with computations in Commutative Algebra, and therefore with *exact* computations on polynomials, CoCoALib also offers some facilities for exploring the world of approximate algebra (see the book [5]). Two complementary approaches are: using *approximate computations* to solve *exact problems*, and applying Commutative Algebra techniques to solve *approximate problems*.

Twin-Float Arithmetic

A facility which CoCoALib offers for the first approach is twin-float arithmetic which can be used as a (generally) faster substitute for exact rational arithmetic with a heuristic guarantee of correctness.

This work is a development of ideas originally proposed by Traverso and Zanoni (ISSAC 2002) as a fast way of computing a good approximation to a Gröbner basis over the rationals. Our investigation lead to the CoCoALib implementation as a `ring`, allowing natural use in a wider range of applications. For the details see [3]; here we give just a brief intuitive outline.

Before computation begins, the user specifies a minimum acceptable accuracy. Then every (exact) rational input is converted into a *twin-float*: *i.e.* a *high precision* floating point value together with a *heuristic estimate* of the accuracy.

Every arithmetic operation on twin-float values checks that the heuristically estimated accuracy of the result is sufficient; if not, the operation fails.

A special feature of twin-floats is the ability to recover a rational number from a twin-float value. This capability allows the recovery of the exact rational answer from a twin-float result under suitable circumstances; *e.g.* an exact Gröbner basis can be obtained from one computed using twin-floats.

Approximate Border Bases

Given a set of *exact* points CoCoA can compute a Gröbner basis of the ideal of the polynomials vanishing over those points. But, when the points are measurements coming from the *real world* then the values are known only approximately.

In this approximate context, the notion of Gröbner basis, which is so important in exact commutative algebra, can exhibit a fatal weakness: it can be structurally unstable in the presence of infinitesimal perturbations. It has recently been shown that in the zero-dimensional case these problems of structural instability can be (largely) eliminated by using instead a Border Basis. Together with C Fassino and M-L Torrente we have developed the notions and theory necessary to apply the Buchberger-Möller algorithm to approximate points, and a robust prototype implementation is included in CoCoALib (see [4], [7]).

This is a rapidly developing topic, promising to be of definite interest to various aspects of both theoretical and practical research.

References

1. Abbott, J., Bigatti, A.M.: CoCoALib: a C++ library for doing Computations in Commutative Algebra, http://cocoa.dima.unige.it/cocoalib/
2. Abbott, J.: The Design of CoCoALib. In: Iglesias, A., Takayama, N. (eds.) ICMS 2006. LNCS, vol. 4151, pp. 205–215. Springer, Heidelberg (2006)
3. Abbott, J.: Twin-Float Arithmetic (preprint)
4. Abbott, J., Fassino, C., Torrente, M.L.: Stable border basis for ideals of points. Journal of Symbolic Computation 43, 883–894 (2008)
5. Abbott, J., Robbiano, L. (eds.): Approximate Commutative Algebra. Springer, Heidelberg (2009)
6. CoCoATeam: CoCoA: a system for doing Computations in Commutative Algebra, http://cocoa.dima.unige.it/

7. Fassino, C.: Almost Vanishing Polynomials for Sets of Limited Precision Points. Journal of Symbolic Computation 45, 19–37 (2010)
8. Kreuzer, M., Robbiano, L.: Computational Commutative Algebra, vol. 1. Springer, Heidelberg (2000, 2008)
9. Kreuzer, M., Robbiano, L.: Computational Commutative Algebra, vol. 2. Springer, Heidelberg (2005)
10. Grayson, D.R., Stillman, M.E.: Macaulay2: a software system for research in algebraic geometry, http://www.math.uiuc.edu/Macaulay2/
11. Greuel, G.-M., Pfister, G., Schönemann, H.: Singular 3-1-0: A computer algebra system for polynomial computations, http://www.singular.uni-kl.de/

LinBox Founding Scope Allocation, Parallel Building Blocks, and Separate Compilation

Jean-Guillaume Dumas[1], Thierry Gautier[2],
Clément Pernet[2], and B. David Saunders[3]

[1] Laboratoire J. Kuntzmann, Université de Grenoble, 51, rue des Mathématiques, umr CNRS 5224, bp 53X, F38041 Grenoble, France
Jean-Guillaume.Dumas@imag.fr
[2] Laboratoire LIG, Université de Grenoble et INRIA. umr CNRS, F38330 Montbonnot, France
Clement.Pernet@imag.fr, Thierry.Gautier@inrialpes.fr
[3] University of Delaware, Computer and Information Science Department, Newark / DE / 19716, USA
saunders@udel.edu

1 Introduction

As a building block for a wide range of applications, computational exact linear algebra has to conciliate efficiency and genericity. The goal of the LinBox project is to address this problem in the design of an efficient general-purpose C++ open-source library for exact linear algebra over the integers, the rationals, and finite fields. Matrices can be either dense, sparse or black box (i.e. viewed as a linear operator, acting on vectors only). The library proposes a set of high level linear algebra solutions, such as the rank, the determinant, the solution of a linear system, the Smith normal form, the echelon form, the characteristic polynomial, etc. Each of these solutions involves a hybrid combination of several specialized algorithms depending on the domain, and the type of matrix. Over a finite field, the building blocks are an efficient implementation of Wiedemann and block Wiedemann algorithms combined with preconditioners [1] for black box matrices, a sparse Gaussian elimination for sparse matrices and the BLAS based dense linear algebra techniques of the FFLAS library [4] for dense matrices. The solutions over the integers and rationals are lifted from modular computations by a Chinese remainder algorithm or p-adic lifting. The design is based on high genericity to allow us to write efficient algorithms independent of the many representations of domains and matrices. As a middleware, the library relies on the efficiency of kernel libraries such as GMP[1], Givaro[4], NTL[4], ATLAS[4] and can be used by general purpose computer algebra systems such as Sage[4] or Maple[4].

We describe in this paper a selection of ideas and improvements that were recently introduced into the the design of LinBox for the forthcoming 2.0 release.

[1] gmplib.org, www-ljk.imag.fr/CASYS/LOGICIELS/givaro, www.shoup.net/ntl, math-atlas.sourceforge.net, sagemath.org, www.maplesoft.com.

K. Fukuda et al. (Eds.): ICMS 2010, LNCS 6327, pp. 77–83, 2010.

2 The Lightweight Founding Scope Allocation Model

The main objects that require memory allocation in LinBox are base field or ring elements, vectors, matrices, and polynomials. The memory management for all of these object types follows the same rules, organized to maximize efficiency in time and space, and consequently requiring some efforts by the programmer. In particular no external garbage collection mechanism is used.

The input and output types of most functions are usually template types, and can be either basic types, or complicated objects. Consequently, passing arguments by value (copy) must be avoided as much as possible. Every argument is passed as a reference, including the return types. More precisely the return value of a function is also the first argument, defined as a non const reference.

```
Matrix& someFunction(Matrix& result, const XXX& args);
```

This convention was already presented in [2, §2.1] for the design of field and ring arithmetic. It does require a redefinition of the interface for some `stl`-like operators, as discussed in section 3.1. A consequence of the above convention is that the objects returned by a function, have to be declared and initialized (in particular, memory allocated, e.g. via constructors) before the function call. By enforcing this practice, we require that the programmer keep *the handle* on the objects that he allocates until all uses of the object and it's subobjects are completed. Moreover, he is responsible for object deallocation in the same scope where it was allocated. This restricts some convenient programming practices, but provides precise control of memory usage. This is particularly important when large, memory filling, matrices are in play. It also allows to avoid the costs of garbage collection or reference counting.

Many LinBox objects involve a handle containing a reference to the free store. Note that even though a function does not allocate the handle itself, it is in certain cases still free to resize and thus reallocate the free store memory referenced.

Dense Matrix allocations. The objects storing dense matrices require a special care concerning their allocations. Dense matrices are represented as a one dimensional array storing elements in the row major format: `A[i,j] = *(A+i*n+j)`. It is important to be able to define a submatrix as a view on such an array, without allocating the data. For this we propose to distinguish two classes: one for allocated (via constructors) matrices and the other for sub-matrix views. The genericity of the template mechanism or inheritance will allow to use these two types in the same code, without duplication. This allows also for an automatic decision about deallocation. Other solutions includes reference counting and explicit "end of use" functions.

3 Software Abstraction Layer for Parallelism

Efficient parallel applications must take into consideration hardware characteristics (number of cores, memory hierarchy, etc.). It is time consuming or impossible for a single developer to program a high performance computer algebra

application, with state of the art algorithms, while exploiting all the available parallelism. In order to separate the domains of expertise we have designed a software abstraction layer between computer algebra algorithms and parallel implementations which may employ automatic dynamic scheduling.

3.1 Parallel Building Blocks

Computer algebra algorithms have three main characteristics: 1) they are complex and require a deep knowledge of the problem in order to obtain the most efficient sequential algorithm; 2) they may be highly irregular. This enforces a runtime use of load balancing algorithms; 3) they are generic in the sense that they are usually designed to work over several algebraic domains.

In the case of LinBox algorithms, we have decided to base our software abstraction, called *Parallel Building Blocks (PBB)*, on the STL algorithms (Standard Template Like) principles. Indeed, C++ data structures in LinBox let us have random access iterators over containers which are naturally parallel. We have already defined several STL-like algorithms and the list will be extended in the near future:

for_each, transform, accumulate[2]: The PBB versions of these algorithms are similar to the STL versions except that the involved operators (or function object classes), given as parameters, are required to have their return value reference passed as the first parameter of the function. This is in accordance with the memory model of LinBox. The STL return-by-value semantic is not appropriate.

The fundamental idea of PBB is that at the computer algebra level, the parallelization of all the loops and more generally of all the STL-like algorithms will already enable good performance and easy switching among multiple implementations. Regarding performance, this parallelization of the inner loops of the underlying linear algebra is sufficient in many cases. Regarding implementations, this abstraction provides for programming independent of the parallel model with selection of the parallel environment depending on the target architecture. The parallel blocks can be implemented using many different parallel environments, such as OpenMP[3]; TBB[7] (Thread Building Blocks) or Kaapi [6]; using both static and dynamic work-stealing schedulers [8]. The current implementations are built on OpenMP and Kaapi.

3.2 Accumulate_until and Early Termination

To bound the complexity of many linear algebra problems, one of the key ideas is to use an accumulation with *early termination*.

For instance, this is used in Chinese Remaindering algorithms. The computation is performed modulo a sequence of (co)prime numbers and the result is built from a sequence of residues, *until* a condition is satisfied [3]. The termination of the algorithm depends on the accumulated result.

[2] `www.sgi.com/tech/stl`

[3] `openmp.org`, `threadingbuildingblocks.org`

In order to parallelize such algorithms, we proposed an extension of the STL algorithms called **accumulate_until** . The algorithm takes an array v of length N, a unary operator f to be applied to each array entry and a specific binary update operator/predicate for the accumulation. This *accumulator* with a signature like `bool accum(a, b)` behaves like an in place addition (`a+=b`) and returns `true` to indicate sufficiently many values are accumulated. Let $S_k = \sum_{i=0,..,k} f(v[i])$ with $k \in \{0, N\}$. The algorithm computes and returns $n \leq N$ and S_n such that one accumulation during the computation of S_n returned `true` or $n = N$. In indended use, we know any additional accumulation would also return `true`.

This algorithm will be used for the early termination Chinese remaindering algorithms of LinBox. Though not yet using PBB and **accumulate_until** , a sequential version and parallel versions with OpenMP and Kaapi can be found in the LinBox distributions as `linbox/algorithms/cra-domain-*.h`.

3.3 Memory Contention and Thread Safe Allocation

Many computer algebra programs allocate dynamic memory for the intermediate computations. Several experiments with LinBox algorithms on multicore architectures have shown that these allocations are quite often the bottleneck. An analysis of the memory pattern and experiments with three well known memory allocators (ptmalloc, Hoard and TCMalloc from Google Perf. Tools[4]) have been conducted. The goal was to decide whether the parallel building blocks model was suitable to high-performance exact linear algebra. We used dynamic libraries to exchange allocators for the experiments, but one can use them together in the LinBox library if needed [7, §7]. Preliminary experiments on early terminated Chinese remaindering, not the easiest to parallelize, have demonstrated the advantage, in our setting, of TCMalloc over the others [3]. One of the main reasons for that fact is that our problems required many temporary allocations. This fits precisely the thread safe caching mechanism of TCMalloc.

4 Automated Generic Separate Compilation

LinBox is developed with several kinds of genericity: 1) genericity with respect to the domain of the coefficients, 2) genericity with respect to the data structure of the matrices, 3) genericity with respect to the intermediate algorithms. While this is efficient in terms of capabilities and code reusability, there is a combinatorial explosion of combinations. Consider that each of 50 arithmetic domains may be combined with each of 50 matrix representations in each of 10 intermediate algorithm forms for a single problem as simple as matrix rank. This lengthens the compilation time and generates large executable files.

For the management of code bloat LinBox has used an "archetype mechanism" which enables, at the user's option, to switch to a compilation against abstract classes [2, §2.1]. However, this can reduce the efficiency of the library. Therefore, we propose here a way to provide a generic separate compilation.

[4] `goog-perftools.sourceforge.net/doc/tcmalloc.html`

This will not deal with code bloat, but will reduce the compilation time while preserving high performance. This is useful for instance when the library is used with unspecialized calls. This is largely the case for some interface wrappers to other Computer algebra systems such as Sage or Maple. Our idea is to automate the technique of [5] which combines compile-time instantiation and link-time instantiation, while using template instantiation instead of void pointers. The mechanism we propose is independent of the desired generic method, the candidate for separate compilation, and is explained in algorithm 1.

Algorithm 1. C++ Automatic separate compilation wrapping

Input: A generic function `func`.
Input: Template parameters `TParam` for separate specialization/compilation of `func`.
Output: A generic function calling `func` with separately compiled instantiations.

1: Create a header and a body files "func_instantiate.hpp" and "func_instantiate.cpp";
2: Add a template function `func_separate`, with the same specification as `func`, to the header;
3: Its generic default implementation is a single line calling the original function `func`. {This enables to have a unified interface, even for non specialized class.}
4: **for** each separately compiled template parameter `TParam` **do**
5: Add a non template specification `funcTParam`, to the header file;
6: Add the associated body with a single line returning the instantiation of `func` on a parameter of type `TParam`, to the body file;
7: Add an inline specialization body of `func_separate` on a parameter of type `TParam` with a single line returning `funcTParam`, to the header file;
8: **end for**
9: Compile the body file "func_instantiate.cpp".

This Algorithm is illustrated on figure 1, where the function is the `rank` and the template parameter is a dense matrix over $GF(2)$, `DenseMatrix<GF2>`.

Algorithm 1 has been simplified for the sake of clarity. To enable a more user-friendly interface one can rename the original function and all its original specializations `func_original`; then rename also the new interface simply `func`. With the classical inline compiler optimizations, the overhead of calling `func_separate` is limited to single supplementary function call. Indeed all the

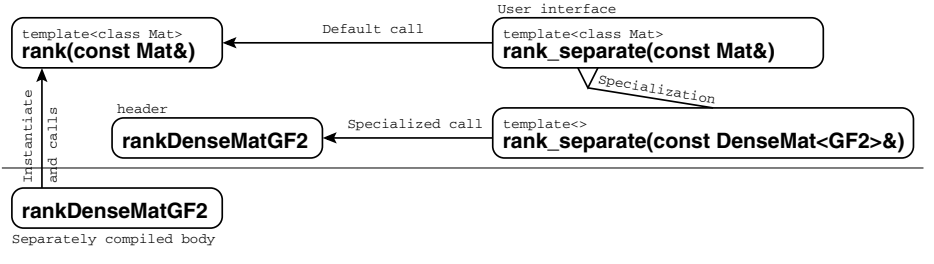


Fig. 1. Separate compilation of the rank

one line additional methods will be automatically inlined, except, of course, the one calling the separately compiled code. If this overhead is too expensive, it suffices to enclose all the non generic specializations of "func_instantiate.hpp" by a macro test. At compile time, the decision to separately compile or not can be taken according to the definition of this macro.

We show in table 1 the gains in compilation time obtained on two examples from LinBox: the `examples/{rank,solve}.C` algorithms. Indeed, without any specification the code has to invoke several specializations depending on run-time discovered properties of the input. For instance `solve.C` requires 6 specializations for sparse matrices over the Integers or over a prime field, with a sparse elimination, or an iterative method, or a dense method, if the matrix is small...

Table 1. linbox/examples/{rank,solve}.C compilation time on an AMD Athlon 3600+, 1.9GHz, with gcc 4.5 -O2. `instantiate.o` contains to the separately compiled instantiations (e.g. densegf2rank in figure 1); `{rank,solve}.o` contains to the user interface and generic implementation compilation; `link` corresponds to the linking of both `.o` and the library; `Full comp.` corresponds to the compilation without the separate mechanism.

file	real time	user time	sys. time	real time	user time	sys. time
	Rank			Solve		
`instantiate.o`	143.43s	142.47s	0.90s	171.62s	170.42s	1.12s
`{rank,solve}.o`	**18.58s**	**18.26s**	**0.30s**	**23.13s**	**22.80s**	**0.32s**
`link`	0.80s	0.64s	0.15s	0.85s	0.70s	0.14s
`Sep. comp. total`	162.81s	161.37s	1.35s	195.60s	193.92s	1.58s
`Full comp.`	162.02s	160.47s	1.21s	191.47s	189.52s	1.40s
`speed-up`	8.4	8.5	2.7	8.0	8.1	3.0s

Acknowledgment

We thank the LinBox group and especially Brice Boyer, Pascal Giorgi, Erich Kaltofen, Dan Roche, Brian Youse for many useful discussions in particular during the recent LinBox developer meetings in Delaware and Dublin.

References

1. Chen, L., Eberly, W., Kaltofen, E., Saunders, B.D., Turner, W.J., Villard, G.: Efficient matrix preconditioners for black box linear algebra. Linear Algebra and its Applications 343-344, 119–146 (2002)
2. Dumas, J.-G., Gautier, T., Giesbrecht, M., Giorgi, P., Hovinen, B., Kaltofen, E., Saunders, B.D., Turner, W.J., Villard, G.: Linbox: A generic library for exact linear algebra. In: ICMS 2002, pp. 40–50 (August 2002)
3. Dumas, J.-G., Gautier, T., Roch, J.-L.: Generic design of chinese remaindering schemes. In: PASCO 2010 (July 2010)
4. Dumas, J.-G., Giorgi, P., Pernet, C.: Dense linear algebra over word-size prime fields: the fflas and ffpack packages. ACM Trans. Math. Softw. 35(3), 1–42 (2008)

5. Erlingsson, U., Kaltofen, E., Musser, D.: Generic Gram-Schmidt orthogonalization by exact division. In: ISSAC 1996, pp. 275–282 (July 1996)
6. Gautier, T., Besseron, X., Pigeon, L.: KAAPI: a thread scheduling runtime system for data flow computations on cluster of multi-processors. In: PASCO 2007, pp. 15–23 (2007)
7. Kaltofen, E., Morozov, D., Yuhasz, G.: Generic matrix multiplication and memory management in LinBox. In: ISSAC 2005, pp. 216–223 (July 2005)
8. Traore, D., Roch, J.L., Maillard, N., Gautier, T., Bernard, J.: Deque-free work-optimal parallel STL algorithms. In: EUROPAR 2008, Las Palmas, Spain (August 2008)

FGb: A Library for Computing Gröbner Bases

Jean-Charles Faugère

SALSA Project - INRIA (Centre Paris-Rocquencourt)
UPMC, Univ Paris 06
CNRS, UMR 7606, LIP6
4, place Jussieu F-75252 PARIS CEDEX 05, France
Jean-Charles.Faugere@inria.fr

Abstract. FGb is a high-performance, portable, C library for computing Gröbner bases over the integers and over finite fields. FGb provides high quality implementations of state-of-the-art algorithms (F_4 and F_5) for computing Gröbner bases. Currently, it is one of the best implementation of these algorithms, in terms of both speed and robustness. For instance, FGb has been used to break several cryptosystems.

1 Introduction - Polynomial System Solving - Gröbner Bases

Solving efficiently polynomial system of equations is a fundamental problem in Computer Algebra with many applications. Let $\mathbb{K}$ be a field and $\mathbb{L} \supset \mathbb{K}$. The problem is:

$$\text{Find } z = (z_1, \ldots, z_n) \in \mathbb{L}^n, \text{ such that } \begin{cases} f_1(z_1, \ldots, z_n) = 0 \\ \cdots \\ f_m(z_1, \ldots, z_n) = 0 \end{cases} \text{where } f_i \in \mathbb{K}[x_1, \ldots, x_n]$$

To solve this problem several methods have been proposed : semi-numerical methods (homotopies), heuristics (for instance SAT solvers when $\mathbb{K} = \mathbb{L} = \mathbb{F}_2$), probabilistic (geometrical resolutions [13]) or exact methods. Among the exact methods one can cite Gröbner bases, Triangular sets methods or Resultant based techniques. In this paper we restrict ourselves to Gröbner basis computation [1]. It is beyond the scope of this paper to explain, in full generality, why a Gröbner basis can be used to solve a polynomial system, but in finite fields (the case $\mathbb{K} = \mathbb{L} = \mathbb{F}_p$) univariate polynomials can be computed (such polynomials can be obtained in a Gröbner basis for an appropriate elimination ordering), and, then, the solutions can be explicitly computed by factoring these polynomials in $\mathbb{F}_p[X]$.

2 Goal and Architecture of the Library

The purpose of the `FGb` library [7] is twofold. First of all, the main goal is to provide efficient implementations of state-of-the-art algorithms for computing Gröbner bases: actually, from a research point of view, it is mandatory to have such an implementation to demonstrate the *practical efficiency* of new algorithms. Secondly, in conjunction with other software, the FGb library has been used in various applications (Robotics,

K. Fukuda et al. (Eds.): ICMS 2010, LNCS 6327, pp. 84–87, 2010.

Signal Theory, Biology, Computational Geometry, ...) and more recently to a wide range of problems in Cryptology (for instance, FGb was explicitly used in [2,8,9,4,5] to break several cryptosystems). Historically, the Gb library (191 420 lines of C++ code) has first been written to implement "classical" algorithms (Buchberger's algorithm[1], FGLM [12], NormalForms, Hilbert functions, ...). The current iteration of the project – the FGb library (206 052 lines of C code) – was restarted from scratch to demonstrate the practical efficiency of a family of new algorithms (F_4[10], matrix-F_5[6], F_5[11], SAGBI-F_5[6],...). All these algorithms have in common that they rely heavily on linear algebra. Hence, whereas efficient internal representation for distributed multivariate polynomials was a key component in Gb, the critical part in FGb is the linear algebra package. The design of the library is somewhat modular: for instance, it is easy to add a new field of coefficients $\mathbb{K}$ (FGb provides already 19 different fields); it is even possible to replace the existing linear algebra package by another one (see section section 4). Even if a small portion of the code has been written in assembler code the library is *portable*; to date the library is available on several architectures: Windows (32 and 64 bits), Linux (32 and 64 bits), Mac (Universal ppc/intel 32/64 bits) and Sun Solaris.

3 Maple Interface - C Library Mode

The FGb/Gb library has no friendly interface but it can be called from any C code. In a partnership with MapleSoft, the library has been *dynamically linked* with the kernel of the Computer Algebra system Maple. Users can use the power of expressivity of a general Computer Algebra System to generate the polynomial equations while keeping the efficiency of a dedicated library. The library is shipped with all recent versions of Maple and fully integrated with high level functions in Maple (for instance the universal `solve` function in Maple can call the FGb library if needed). It is also possible to call directly the internal package (and hence have access to more options for expert users):

```
    |\^/|     Maple 13 (X86 64 LINUX)
._|\|   |/|_.Copyright (c) Maplesoft, a division of Waterloo Maple Inc.
 \  MAPLE  / All rights reserved. Maple is a trademark of
 <____ ____> Waterloo Maple Inc.
      |      Type ? for help.
> with(fgbrs):
> # the output of fgb_gbasis is a list [[lc1,lt1,p1],...]
    where lc1*lt1 is the leading monomial of p1.
> fgb_gbasis([x-y^2,x^2-y],[x,y]);
                            2   2            2   2
                     [[1, y , y  - x], [1, x , x  - y]]
> # Same computation in GF(65521) for an elimination ordering x>>y
> fgb_gbasis([x-y^2,x^2-y],[x],[y],65521);
                       4   4                                2
                [[1, y , y  + 65520 y], [1, x, x + 65520 y ]]
```

More generally, it is easy to call FGb from any C program (API and sample program available [7]): for instance, Coq can use FGb to proof some polynomial equations (the link was done by L. Pottier).

4 New High Performance Linear Algebra Package - Benchmarks

In [3], we present a recent new dedicated linear algebra package written by S. Lachartre for computing Gaussian elimination of matrices coming from Gröbner bases computations. This library is also written in C and contains new algorithms to compute Gaussian elimination as well as specific internal representation of matrices (namely sparse triangular blocks, sparse rectangular blocks and hybrid rectangular blocks). In order to demonstrate the efficiency of this combination of software we give some computational results for a well known benchmark: the Katsura problem. For instance, for a medium size problem such as Katsura 15, it takes 849.7 sec on a PC with 8 cores to compute a DRL Gröbner basis modulo $p < 2^{16}$; this is 88 faster than Magma (V2-16-1).

	Kat11	Kat12	Kat 13	Kat 14	Kat 15	Kat 16
Magma	19.5	151.2	1091.4	9460.35	74862.9	NA
FGb F_4	40.6	342.6	2550.65			
FGb F_5			191.7	1881.3	12130.8	103110.0
New linalg + F_4	**2.85**	**19.45**	**149.6**			
New linalg + F_5			**27.6**	**180.7**	**849.7**	**5687.3**

Benchmark: Katsura n modulo 65521 - PC with 8 cores.

References

1. Buchberger, B.: Ein Algorithmus zum Auffinden der Basiselemente des Restklassenringes nach einem nulldimensionalen Polynomideal. PhD thesis, Innsbruck (1965)
2. Faugère, J.-C., Joux, A.: Algebraic Cryptanalysis of Hidden Field Equation (HFE) Cryptosystems Using Gröbner bases. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 44–60. Springer, Heidelberg (2003)
3. Faugère, J.-C., Lachartre, S.: Parallel Gaussian Elimination for Gröbner bases computations in finite fields. In: Moreno-Maza, M., Roch, J.L. (eds.) ACM Proceedings of The International Workshop on Parallel and Symbolic Computation (PASCO), LIG, pp. 1–10. ACM, New York (July 2010)
4. Faugère, J.-C., Levy-dit-Vehel, F., Perret, L.: Cryptanalysis of Minrank. In: Wagner, D. (ed.) CRYPTO 2008. LNCS, vol. 5157, pp. 280–296. Springer, Heidelberg (2008)
5. Faugère, J.-C., Otmani, A., Perret, L., Tillich, J.-P.: Algebraic Cryptanalysis of McEliece Variants with Compact Keys. In: Eurocrypt 2010. LNCS, vol. 6110, pp. 279–298. Springer, Heidelberg (2010)
6. Faugère, J.-C., Rahmany, S.: Solving systems of polynomial equations with symmetries using SAGBI-Gröbner bases. In: ISSAC 2009: Proceedings of the 2009 International Symposium on Symbolic and Algebraic Computation, pp. 151–158. ACM, New York (2009)
7. Faugère, J.C.: FGb library for comptuing Gröbner bases, `http://www-salsa.lip6.fr/~jcf/Software/FGb/`
8. Faugère, J.-C., Perret, L.: Cryptanalysis of 2R– schemes. In: Dwork, C. (ed.) CRYPTO 2006. LNCS, vol. 4117, pp. 357–372. Springer, Heidelberg (2006)
9. Faugère, J.-C., Perret, L.: Polynomial Equivalence Problems: Algorithmic and Theoretical Aspects. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 30–47. Springer, Heidelberg (2006)
10. Faugère, J.C.: A new efficient algorithm for computing Gröbner bases (F4). Journal of Pure and Applied Algebra 139(1-3), 61–88 (1999)

11. Faugère, J.C.: A new efficient algorithm for computing Gröbner bases without reduction to zero F5. In: Mora, T. (ed.) Proceedings of the 2002 International Symposium on Symbolic and Algebraic Computation, pp. 75–83. ACM Press, New York (July 2002)
12. Faugère, J.C., Gianni, P., Lazard, D., Mora, T.: Efficient Computation of Zero-Dimensional Gröbner Basis by Change of Ordering 16(4), 329–344 (October 1993)
13. Lecerf, G.: Kronecker Magma package for solving polynomial systems by means of geometric resolutions, http://www.math.uvsq.fr/~lecerf/software/

Fast Library for Number Theory: An Introduction

William B. Hart

Mathematics Institute, Warwick University, Coventry, United Kingdom

Abstract. We discuss FLINT (Fast Library for Number Theory), a library to support computations in number theory, including highly optimised routines for polynomial arithmetic and linear algebra in exact rings.

1 Introduction

The Fast Library for Number Theory (FLINT) [6] is a software library, written in highly optimised C, to support computations in number theory. Its initial scope is to cover the polynomial arithmetic and linear algebra functionality of a library like Victor Shoup's Number Theory Library (NTL), [16]. However, the eventual aim of FLINT will be to provide an alternative to the Pari library [2], with a focus on higher level computations in Algebraic Number Theory.

The design motivations for FLINT are that it be:

- Written entirely in C (some assembly optimisations)
- Threadsafe design
- Implement asymptotically fast algorithms where available
- As fast or faster than other Open Source and Proprietary options
- Completely Open Source (GPL licensed)

FLINT is constructed as a set of modules, each based around a given type, e.g. the fmpz_poly module, which is based around a type for polynomials with multiple precision integer coefficients (the FLINT fmpz type).

2 Basic Integer Arithmetic

FLINT supports integer arithmetic in three ways. Firstly, the fast polynomial multiplication code (see the next section) is used to provide very fast integer multiplication for operands above about 2000 limbs. This implementation is often faster than GMP [5] (which is used for smaller multiplications), by as much as 30%.

Secondly, FLINT offers a highly optimised multiple polynomial quadratic sieve, for factoring integers. This is efficient up to about 70 decimal digits and still much faster than Pari for larger factorisations.

K. Fukuda et al. (Eds.): ICMS 2010, LNCS 6327, pp. 88–91, 2010.

Thirdly, FLINT's ulong_extras module provides fast code for operations involving C integers, i.e. long int's. This includes modular arithmetic, gcd, primality testing, efficient factorisation, via numerous optimised factoring routines. One innovation here is the One Line Factor algorithm, which is competitive with SQUFOF (also implemented). See [10] for details.

3 Polynomial Arithmetic

The bulk of code in FLINT supports polynomial arithmetic for multiprecision integer coefficients and for coefficients in $\mathbb{Z}/n\mathbb{Z}$ for up to machine word sized moduli.

FLINT implements numerous integer polynomial multiplication routines, including the classical and Karatsuba routines, Kronecker Segmentation and the Schoënhage-Strassen algorithm. The latter is based on highly optimised Fast Fourier Transform code, the final version of which was developed by David Harvey, based on the ideas presented in his paper [12].

Division of polynomials is achieved using a modified version of Mulders' algorithm (see [14], [11]), which is competitive with the usual middle product approach, but simpler to implement.

The polynomial modules also offer routines for power series operations, GCD, resultant, evaluation and composition. The latter is achieved with an algorithm implemented by Andy Novocin and the author [7].

The $\mathbb{Z}/n\mathbb{Z}$ module in the FLINT 1 series makes use of David Harvey's zn_poly library. This uses his fast Kronecker Segmentation variations [13] to achieve up to a 40% improvement over standard Kronecker Segmentation, and a highly optimised Schoënhage-Nussbaumer FFT implementation.

The $\mathbb{Z}/n\mathbb{Z}[x]$ module offers factorisation based on the Berlekamp and Cantor-Zassenhaus algorithms. The $\mathbb{Z}[x]$ module has a first implementation of the new factorisation algorithm of Novocin and van Hoeij [8] which improves on the original algorithms of van Hoeij, Belabas and others.

4 Linear Algebra

The linear algebra component of FLINT is still relatively young, providing some basic types and a FLINT rewrite of Damien Stehlé's fpLLL [15]. It also offers Hermite Normal Form and basic operations including matrix multiplication.

5 FLINT 2

The FLINT 2 series is a complete rewrite of FLINT 1 from scratch. Its focus is on very clean code and even better performance than FLINT 1. It also offers modules for multivariate polynomial arithmetic, polynomials over $\mathbb{Z}/n\mathbb{Z}$ for multiprecision n and optimised linear algebra over $\mathbb{Z}/nZ$ for word sized moduli. It should be released by the end of 2010.

6 Comparison with Other Libraries

The performance of FLINT, especially in polynomial arithmetic is usually equivalent to that of Magma [3] and in many cases faster (polynomial factoring over $\mathbb{Z}/nZ$ is still an exception to this). Magma is already up to five times faster than NTL even for polynomial multiplication.

The Pari library is not usually asymptotically fast and NTL is not threadsafe.

FLINT has been used to perform some very large computations, e.g. the computation of 10^{12} coefficients of the congruent number theta series [9].

7 Choice of Language – C

In preparing this abstract the referee requested that the author comment on the suitability of C as a language for FLINT as compared with a higher level language, say. Indeed, C is a terrible choice because it is not low level enough (it does not support carry handling or return of the high word of a product - though recent versions of GCC allow the latter via extensions), it does not comfortably support generic programming and its syntax is complex and not well suited to mathematics.

However, C is also the best choice available because it is the clear frontrunner performance-wise. C is compiled, statically typed and the GNU compiler collection and some commercial C compilers have extremely sophisticated back ends, yielding good performance.

C++ would add Object Oriented programming, however its syntax is so complex that its front end actually rivals the back end in complexity. But most importantly, the C/C++ syntax cannot be extended, its macro processing being essentially search-and-replace only.

Lisp is a high level language which is formally syntactically extensible through macros, but uses uncomfortable prefix notation. Forth is a low level language which is formally syntactically extensible, but has equally uncomfortable postfix notation and requires the programmer to think in terms of a stack.

OCaml is properly syntactically extensible through its powerful preparser, but its native syntax is uncomfortable, and once again performance has fallen behind that of good C compilers.

The ideal language would have a syntactically extensible front end sitting on an already highly optimised back end, such as that of GCC's GENERIC or LLVM's IR. To our knowledge, such a language simply doesn't exist. Even with that sophistication, no existing back ends properly support carries, meaning that for some things, assembly language is still required.

We mention one other solution of note. Sage [4] is written in Python, which has a very clean syntax, but is very high level with terrible performance (often a hundred times slower than highly optimised C). For performance critical code, Sage makes use of Cython [1], a dialect of Python, which allows static C type declarations. These can be compiled to C, which for many situations will give native C performance. In fact, FLINT is used in Sage, and Cython is used extensively in the FLINT wrapper to ensure maximum efficiency.

References

1. Behnel, S., Bradshaw, R., Seljebotn, D.: Cython: C extensions for Python, http://www.cython.org/
2. Belabas, K.: Pari/GP, http://pari.math.u-bordeaux.fr/
3. Cannon, J., Steel, A., et al.: Magma Computational Algebra System, http://magma.maths.usyd.edu.au/magma/
4. Erocal, B., Stein, W.: The Sage Project: Unifying Free Mathematical Software to Create a Viable Alternative to Magma, Maple, Mathematica and Matlab, http://wstein.org/papers/icms/icms_2010.pdf, http://www.sagemath.org/
5. Granlund, T.: GNU MP Bignum Library, http://gmplib.org/
6. Hart, W., Harvey, D., et al.: Fast Library for Number Theory, http://www.flintlib.org/
7. Hart, W., Novocin, A.: A practical univariate polynomial composition algorithm (2010) (preprint)
8. Hart, W., Novocin, A., van Hoeij, M.: Improved polynomial factorisation (2010) (preprint)
9. Hart, W., Tornaria, G., Watkins, M.: Congruent number theta coefficients to 10^{12}. In: Gaudry, et al. (eds.) Proceedings of the Algorithmic Number Theory Symposium (ANTS IX). Springer, Heidelberg (to appear 2010)
10. Hart, W.: A One Line Factoring Algorithm (2010) (preprint)
11. Hart, W.: A refinement of Mulders' polynomial short division algorithm (2007) (unpublished report)
12. Harvey, D.: A cache–friendly truncated FFT. Theor. Comput. Sci. 410, 2649–2658 (2009)
13. Harvey, D.: Faster polynomial multiplication via multipoint Kronecker substitution. J. Symb. Comp. 44, 1502–1510 (2009), http://www.cims.nyu.edu/~harvey/code/zn_poly/
14. Mulders, T.: On short multiplication and division. AAECC 11(1), 69–88 (2000)
15. Nguyen, P., Stehlé, D.: An LLL algorithm with quadratic complexity. SIAM Journal of Computation 39(3), 874–903 (2009), http://perso.ens-lyon.fr/damien.stehle/#software
16. Shoup, V.: NTL: A Library for doing Number Theory, http://www.shoup.net/ntl/

Controlled Perturbation for Certified Geometric Computing with Fixed-Precision Arithmetic*

Dan Halperin

School of Computer Science
Tel Aviv University, Israel
http://acg.cs.tau.ac.il/danhalperin

Abstract. Transforming geometric algorithms into effective computer programs is a difficult task. This transformation is particularly made hard by the basic assumptions of most theoretical geometric algorithms concerning the handling of robustness issues, namely issues related to arithmetic precision and degenerate input. Controlled perturbation, an approach to robust implementation of geometric algorithms we introduced in the late 1990's, aims at removing degeneracies and certifying correct predicate-evaluation, while using fixed-precision arithmetic. After exposing the key ideas underlying the scheme, we review the development of the approach over the past decade including variations and extensions, software implementation and applications. We conclude by pointing out directions for further development and major challenges.

1 Introduction

Most computational geometry (CG, for short) algorithms are designed under two simplifying assumptions: (i) that the computation is carried out in the Real-RAM model, allowing infinite-precision real arithmetic, and (ii) that the input is degeneracy free. These assumptions pose major difficulty to those who wish to implement the algorithms, as they are barely met in practice. For an extensive discussion of the problems in robust geometric computation and prevailing solutions, see the surveys [1,2].

The last two decades have seen tremendous headway in implementing CG algorithms, organized mostly in the carefully crafted software libraries LEDA and (nowadays primarily in) the Computational Geometry Algorithms Library, CGAL. Most robust implementations follow the Exact Geometric Computing paradigm (EGC, for short) [3]. In almost all cases the algorithms are augmented to handle degenerate input. This area has progressed so well that for certain problems the EGC solutions hardly bear any performance penalty over using machine floating-point arithmetic (when computing the desired result is at all possible with floating-point arithmetic).

* This work has been supported in part by the Israel Science Foundation (grant no. 236/06), by the German-Israeli Foundation (grant no. 969/07), and by the Hermann Minkowski–Minerva Center for Geometry at Tel Aviv University.

K. Fukuda et al. (Eds.): ICMS 2010, LNCS 6327, pp. 92–95, 2010.

However, when computing with curved objects, even of low algebraic degree and already in three-dimensional space, one is still (at the time of writing) held back by the large performance overhead of the EGC machinery. Moreover, handling all possible degeneracies in complex geometric structures in three or higher dimensions, is an excruciatingly tedious and error-prone process.

It is rare in scientific or engineering applications that we need to produce exact results. Good approximations will in most cases do the job, since the input to the geometric computation often comes from modeling or measurements that are approximate to begin with.

In view of all of the above, we proposed in the late 1990s an alternative approach to certified geometric computing, which we call *controlled perturbation*, (CP, for short). The method proceeds by perturbing the input slightly (hopefully within the accuracy requirement of the application) but in a controlled manner (hence the name) such that: (i) we are guaranteed that all the predicates used by the algorithm are evaluated correctly with floating-point arithmetic of a given precision, and (ii) degeneracies are removed. Thus, when successful, CP alleviates the problems incurred by EGC, and on the perturbed input it practically conforms with the basic assumptions underlying CG algorithms, which we mentioned above.

2 Key Ideas

Geometric predicates evaluate the sign of certain expressions (typically the sign of a polynomial). In degenerate situations the expression evaluates to zero. We say that we are far away from degeneracy when the absolute value of the expression is sufficiently large, and in particular can be safely bounded away from zero, in spite of the rounding errors incurred by floating-point arithmetic. In the remainder of this section, our description borrows from the recent full and improved version of [4] by Mehlhorn et al.[1]

To each predicate in the algorithm that we aim to "CP-fy," we attach a *guard*. The guard is a simple predicate, which is computed using floating-point arithmetic. The guard *fires* when the floating-point evaluation of the guarded predicate cannot be bounded away from zero. If throughout the entire execution of the algorithm, no guard fires, then the algorithm computes the correct result and the input is non-degenerate.

If a guard fires then there are two different strategies on how to proceed. One approach perturbs the input locally and recomputes the guard, and so on until the guard does not fire. An alternative approach aborts the current run of the algorithm, perturbs the entire input and reruns the guarded algorithm.

The method is governed by several parameters including the precision of the floating-point arithmetic (the length of the mantissa), the bit size of the input numbers, the number of objects in the input, and a prescribed limit on the allowable perturbation magnitude. Obviously the precision of the underlying

[1] The full manuscript by Mehlhorn, Osbild and Sagraloff, titled "A General Approach to the Analysis of Controlled Perturbation Algorithms" and dated 2010, is currently the most comprehensive starting point to understanding Controlled Perturbation.

arithmetic and the magnitude of the perturbation depend inversely on each other. Some of the papers that we review below include involved analysis of the connection between the various parameters of the scheme. However, CP can be successfully applied with almost no analysis whatsoever, with the exception of devising reasonable guards.

3 Brief History: Theory, Implementation, Applications

We review major milestones in the development of Controlled Perturbation. CP was first proposed [5] as a means to compute *molecular surfaces*. A molecular surface here is simply the boundary of the union of (atom) balls in $\mathbb{R}^3$, and computing it amounts to a careful walk through portions of the three-dimensional *arrangement*[2] of the spheres bounding the balls. Back in the 1990's there was no infrastructure, nor sufficient experience and knowledge, to compute arrangements of spheres exactly. Our attempts to compute the surfaces while using floating-point arithmetic failed, even when we used slight (heuristic) perturbation of the spheres. The introduction of CP led to robust and efficient, floating-point based construction of the surface.

Our next endeavor was to compute polyhedral approximation of swept volumes in $\mathbb{R}^3$ [6]. In terms of the number of different degenerate configurations that need to be considered, this turned out to be a highly demanding project. Still the CP software successfully coped with difficult and intricate configurations. Both projects (molecular surfaces and swept volumes) were primarily driven by the respective application, and the ensuing reports did not provide full analysis of the method. At about the same time Packer worked out the details of the scheme for arrangements of polygons [7]. Later on Packer investigated the effect of the order of handling objects in CP on the required perturbation magnitude [8]. The work on molecular surfaces was extended to dynamically maintain the (degeneracy-free) surface as the molecule changes its conformation [9].

The first fully certified implementation of CP was for computing arrangements of circles in the plane [10]. It relies on the extensive analysis of all the parameters involved. The efficient code comes with an instructive visualization program.[3]

All the CP algorithms described so far have the same flavor: They incrementally construct an arrangement by adding the objects inducing the arrangement one by one. Once an object is placed (possibly after perturbation) it is not moved again. In 2005, Funke et al. [11] applied the scheme to the construction of Delaunay triangulations and convex hulls in any dimension. The authors suggest and analyze a restart-upon-failure variant of CP that stops and restarts the algorithm whenever a predicate failure is detected.

Mehlhorn et al. [4] went on to propose a general approach to analyzing CP algorithms. Caroli [12] applied the approach to predicates that arise in the computation of arrangements of circles and Voronoi diagrams of line segments. It

[2] For details on arrangements and other geometric structures mentioned in this brief summary, consult the respective papers.

[3] The program is available upon request. The original code by Eran Leiserowitz has recently been transferred to work under Windows by Boris Kozorovitzky.

turned out that the analysis is rather involved, and not all predicates are covered. More recently, in a full version of [4], Mehlhorn et al. simplify the approach, and they provide an analysis for all predicates that are defined as signs of polynomials. Another interesting contribution of this recent manuscript is that it resolves the gap between the analysis that assumes that the perturbation is in the space of real numbers and the practice where implementations work with floating-point perturbations.

Further work. A lot of special tailoring is still involved in developing CP algorithms for each new problem. The ultimate and ambitious goal would be to automate the CP-fication process, or at least provide the infrastructure to ease the transformation of existing algorithms into their CP variants.

References

1. Schirra, S.: Robustness and precision issues in geometric computation. In: Sack, J.R., Urrutia, J. (eds.) Handbook of Computational Geometry, pp. 597–632 (1999)
2. Yap, C.K.: Robust geometric computation. In: Goodman, J.E., O'Rourke, J. (eds.) Handbook of Discrete and Computational Geometry, 2nd edn., pp. 927–952. CRC Press LLC, Boca Raton (2004)
3. Yap, C.K., Dubé, T.: The exact computation paradigm. In: Du, D.Z., Hwang, F.K. (eds.) Computing in Euclidean Geometry, 2nd edn., pp. 452–492 (1995)
4. Mehlhorn, K., Osbild, R., Sagraloff, M.: Reliable and efficient computational geometry via controlled perturbation. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds.) ICALP 2006. LNCS, vol. 4051, pp. 299–310. Springer, Heidelberg (2006)
5. Halperin, D., Shelton, C.R.: A perturbation scheme for spherical arrangements with application to molecular modeling. Comput. Geom. Theory Appl. 10, 273–287 (1998); A preliminary version appeared in Proc. 15th ACM Symposium on Computational Geometry (1997)
6. Raab, S., Halperin, D.: Controlled perturbation for arrangements of polyhedral surfaces. Manuscript, full version, School of Computer Science, Tel Aviv University, http://acg.cs.tau.ac.il/danhalperin/publications; A preliminary version appeared in Proc. 15th ACM Symposium on Computational Geometry (1999/2002)
7. Packer, E.: Finite-precision approximation techniques for planar arrangements of line segments. M.Sc. thesis, School of Computer Science, Tel Aviv University, Tel Aviv (2002)
8. Packer, E.: Controlled perturbation of arrangements of line segments in 2D with smart processing order. Comput. Geom. Theory Appl (to appear 2010)
9. Eyal, E., Halperin, D.: Dynamic maintenance of molecular surfaces under conformational changes. In: Symposium on Computational Geometry, pp. 45–54 (2005)
10. Halperin, D., Leiserowitz, E.: Controlled perturbation for arrangements of circles. Int. J. Comput. Geometry Appl. 14, 277–310 (2004); A preliminary version appeared in Proc. 21st ACM Symposium on Computational Geometry (2003)
11. Funke, S., Klein, C., Mehlhorn, K., Schmitt, S.: Controlled perturbation for Delaunay triangulations. In: SODA, pp. 1047–1056 (2005)
12. Caroli, M.: Evaluation of a generic method for analyzing controlled-perturbation algorithms. M.Sc. thesis, Saarland University, Saarbrücken (2007)

Exact Geometric and Algebraic Computations in CGAL*

Menelaos I. Karavelas[1,2]

[1] Dept. Applied Mathematics, University of Crete, GR-714 09 Heraklion, Greece
mkaravel@tem.uoc.gr
http://www.tem.uoc.gr/~mkaravel
[2] Institute of Applied and Computational Mathematics,
Foundation for Research and Technology - Hellas,
P.O. Box 1385, GR-711 10 Heraklion, Greece

Abstract. We summarize recent progress and on-going developments for exact geometric and algebraic computations within the Computational Geometry Algorithms Library (CGAL). We detail the existing machinery used in efficient, exact and robust implementations of various geometric entities.

1 Introduction

Implementing geometric algorithms is far from being a trivial and straightforward process. Not only do we have to accommodate potentially complicated data structures, but also account for computations involving numerical data. These computations, called *predicates*, are functions of a constant number of geometric objects (e.g., points) returning a small number of values (e.g., a sign), and affect the choices made by the algorithm. The correct or consistent evaluation of predicates is vital in order for an implementation to be robust and for the outcome of the algorithm to be usable. However, even the simplest possible calculations may lead to inconsistent or wrong results [16], especially when the geometric input data are in *degenerate* or *almost degenerate* configurations.

This phenomenon that has given rise to a variety of methodologies for ensuring robustness of the implementation, while providing some guaranties for the output. These methodologies include adaptive precision floating-point arithmetic, the "topology-oriented approach", symbolic perturbations, controlled perturbations, and exact geometric computation — see [19] for a nice overview. The exact geometric computation paradigm, along with the continued evolution of software libraries offering exact number types [9,12,18], provides a platform for implementing correct and robust geometric algorithms without the need for arithmetic considerations. When combined with arithmetic and geometric filtering techniques, as well as lazy geometric evaluation considerations the overhead of exact geometric computation can turn out to be negligible, not only when

* Partially supported by the FP7-REGPOT-2009-1 project "Archimedes Center for Modeling, Analysis and Computation".

K. Fukuda et al. (Eds.): ICMS 2010, LNCS 6327, pp. 96–99, 2010.

taking into account the easiness of development and the correctness guarantees that it provides, but also in absolute terms.

2 The Computational Geometry Algorithms Library

The Computational Geometry Algorithms Library (CGAL) [8] is a software library for computational geometry algorithms and data structures. It started in 1995 as a European project, and currently it is a mature Open Source project. It has gone through more than 15 public releases and its latest version, version 3.6 [23], is comprised of more than 600,000 lines of C++ code. The goal of the CGAL Open Source project is "to provide easy access to efficient and reliable geometric algorithms in the form of a C++ library", as it is stated in the project web page. Due to the dual nature of geometric algorithms, namely combinatorics/data structures and computations on geometric objects, the design paradigm for the algorithms in CGAL follows a clear-cut separation between the combinatorial aspects of these algorithms and the numerical computations. This separation is evident in the template parameters CGAL classes. For example, algorithms for computing triangulated Delaunay graphs have two template parameters called respectively *(triangulation) traits* and *(triangulation) data structure*. All combinatorial operations, as well as the data structures representing the Delaunay graphs are provided by the data structure template parameter, whereas all numerical issues are abstracted and encapsulated in the traits template parameter.

Currently, the CGAL library offers algorithms for tackling a variety of geometric problems — see the Package Overview section of the latest CGAL manual [23] for a complete listing of the packages offered. Algorithms in CGAL rely on *concepts*; the library typically provides at least one *model* for each concept. This design allows for modularity and exchangeability of components, making CGAL an ideal platform for benchmarking different ways of implementing the same concept. Flexibility is gained by means of template parameters and has proven to go quite far with respect to the kind and complexity of the functionality provided to the end-user; see, e.g., the case of 2D and 3D triangulations [3], as well as that of 2D arrangements [25]. Robustness is ensured via exact predicates provided by either a kernel or a geometric traits class. The use of various arithmetic/geometric filtering techniques [5,20,24,15], and symbolic perturbations (e.g., [10]) has counter-balanced the additional cost of performing exact arithmetic in degenerate or almost degenerate configurations of the input data.

3 Algebraic Computations in CGAL

During the last decade a considerable amount of work in Computational Geometry has shifted towards computing geometric entities involving non-linear geometric objects [4]. If we are to categorize these efforts, they should be split into two categories: (1) problems/implementations involving computations on algebraic numbers of small/bounded algebraic degree (2) problems/implementations involving computations on algebraic numbers of large/arbitrary algebraic degree

What is common in both cases, however, is that, either at the analysis or at the implementation level, the need for support for algebraic tools became apparent.

CGAL started as a project offering geometric algorithms for simple objects, such as points, linear segments and circles; as a result the provided algorithms required support for the exact computation of quantities involving only rational operations (or limited support for square roots). CGAL's evolution over the years reflects the shift in interest towards curvilinear objects. As of release 3.0, CGAL offers packages for computing the 2D additively weighted Voronoi diagram (it requires support for algebraic numbers of degree 2) [11], and for computing arrangements of conic curves [24]. A package for computing the 2D Euclidean segment Voronoi diagram was introduced in release 3.1 (it requires support for algebraic number of algebraic degree 4) [15], while in release 3.2 we have the introduction of the 2D Circular Kernel, the first kernel designed for non-linear geometric objects [7], accompanied by a specialized, and with limited functionality, algebraic kernel for degree 2 algebraic numbers. In the same release a framework for Kinetic Data Structures was introduced [22], while the CGAL arrangements' package extended its applicability arbitrary degree Bézier curves [13]: in both cases, support for operations on large/arbitrary degree polynomials and large/arbitrary degree algebraic numbers, namely root isolation and root comparison, has been the computational core, and relied primarily on either internal (to the package) code for algebraic computations, or on algebraic numbers provided by external libraries, such as CORE [9].

A parallel effort had been underway by the EXACUS project [1]; as of CGAL's release 3.3 the two projects started to merge, and CGAL changed its underlying structure: the library moved from being number-type centric to relying on a concrete and well-designed platform of algebraic foundations [14]. In release 3.4 we have the formal introduction on polynomials and modular arithmetic, whereas on the geometry side the 2D Circular Kernel got a three-dimensional counterpart: the 3D Spherical Kernel [6]. With the latest public release 3.6, CGAL provides the first model of an algebraic kernel for arbitrary degree univariate polynomials [17] based on the RS library [21], while work for a univariate algebraic kernel based on the bitstream Descartes' algorithm and a bivariate algebraic kernel based on the curve analysis approach, is currently underway and is expected to be part of the next public release of CGAL [2].

References

1. Berberich, E., Eigenwillig, A., Hemmer, M., Hert, S., Kettner, L., Mehlhorn, K., Reichel, J., Schmitt, S., Schömer, E., Wolpert, N.: EXACUS: Efficient and Exact Algorithms for Curves and Surfaces. In: Proc. 13th European Symposium on Algorithms (ESA). pp. 155–166 (2005)
2. Berberich, E., Hemmer, M., Kerber, M.: A generic algebraic kernel for non-linear geometric applications. Research Report 7274, INRIA (2010)
3. Boissonnat, J.D., Devillers, O., Pion, S., Teillaud, M., Yvinec, M.: Triangulations in CGAL. Comput. Geom.-Theor. Appl. 22, 5–19 (2002)
4. Boissonnat, J.D., Teillaud, M. (eds.): Effective Computational Geometry for Curves and Surfaces. Mathematics and Visualization. Springer, Heidelberg (2006)

5. Brönnimann, H., Burnikel, C., Pion, S.: Interval arithmetic yields efficient dynamic filters for computational geometry. Discrete Appl. Math. 109, 25–47 (2001)
6. de Castro, P.M.M., Cazals, F., Loriot, S., Teillaud, M.: Design of the CGAL 3D spherical kernel and application to arrangements of circles on a sphere. Comput. Geom.-Theor. Appl. 42(6-7), 536–550 (2009)
7. de Castro, P.M.M., Pion, S., Teillaud, M.: Exact and efficient computations on circles in CGAL. In: Proc. 23rd European Workshop on Computational Geometry, pp. 219–222. Technische Universität Graz, Austria (2007)
8. CGAL, Computational Geometry Algorithms Library, http://www.cgal.org
9. CORE Number Library, http://cs.nyu.edu/exact/core_pages
10. Devillers, O., Teillaud, M.: Perturbations and vertex removal in a 3D Delaunay triangulation. In: Proc. 14th ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 313–319 (2003)
11. Emiris, I.Z., Karavelas, M.I.: The predicates of the Apollonius diagram: algorithmic analysis and implementation. Comput. Geom.-Theor. Appl. 33(1-2), 18–57 (2006)
12. GMP, GNU Multiple Precision Arithmetic Library, http://www.swox.com/gmp/
13. Hanniel, I., Wein, R.: An exact, complete and efficient computation of arrangements of Bézier curves. In: Proc. ACM Symposium on Solid and Physical Modeling, pp. 253–263 (2007)
14. Hemmer, M.: Algebraic foundations. In: CGAL User and Reference Manual. CGAL Editorial Board, 3.6 edn. (2010), http://www.cgal.org/Manual/3.6/doc_html/cgal_manual/packages.html#Pkg:AlgebraicFoundations
15. Karavelas, M.I.: A robust and efficient implementation for the segment Voronoi diagram. In: Proc. International Symposium on Voronoi Diagrams in Science and Engineering (VD 2004), Hongo, Tokyo, Japan, pp. 51–62 (2004)
16. Kettner, L., Mehlhorn, K., Pion, S., Schirra, S., Yap, C.K.: Classroom examples of robustness problems in geometric computations. Comput. Geom.-Theor. Appl. 40(1), 61–78 (2008)
17. Lazard, S., Peñaranda, L., Tsigaridas, E.P.: Univariate algebraic kernel and application to arrangement. In: Vahrenhold, J. (ed.) SEA 2009. LNCS, vol. 5526, pp. 209–220. Springer, Heidelberg (2009)
18. LEDA: Library of efficient data-structures and algorithms, http://www.mpi-sb.mpg.de/LEDA/leda.html
19. Li, C., Pion, S., Yap, C.K.: Recent progress in exact geometric computation. J. Log. Algebr. Program. 64(1), 85–111 (2005)
20. Melquiond, G., Pion, S.: Formally certified floating-point filters for homogeneous geometric predicates. Informatique Théorique et Applications 41(1), 57–69 (2007)
21. The RS library, http://www-salsa.lip6.fr/~rouillie/Software.html
22. Russel, D., Karavelas, M.I., Guibas, L.J.: A package for exact kinetic data structures and sweepline algorithms. Comp. Geom.-Theor. Appl. 38(1-2), 111–127 (2007)
23. The CGAL Project: CGAL User and Reference Manual. CGAL Editorial Board, 3.6 edn. (2010), http://www.cgal.org/Manual/3.6/doc_html/cgal_manual/packages.html
24. Wein, R.: High-level filtering for arrangements of conic arcs. In: Möhring, R.H., Raman, R. (eds.) ESA 2002. LNCS, vol. 2461, pp. 884–895. Springer, Heidelberg (2002)
25. Wein, R., Fogel, E., Zukerman, B., Halperin, D.: Advanced programming techniques applied to CGAL's arrangement package. In: Proc. Library-Centric Software Design Workshop (LCSD 2005), pp. 24–33 (2005)

On Solving Systems of Bivariate Polynomials

Fabrice Rouillier

INRIA Paris-Rocquencourt Research center
Fabrice.Rouillier@inria.fr

Solving systems of bivariate polynomials is a critical operation : curves plotting, curves topology, parametric systems, general solvers which are recursive on the number of variables, etc. Several strategies currently exist from numerical algorithms (bisections for example, interval arithmetic) to general rewriting methods (Gröbner bases, triangular sets, resultants, etc.) with advantages and drawbacks.

The algorithm we propose mixes dynamically different points of view depending on the geometrical properties detected during the computations. In short, we decompose as early as possible the system to decrease the degrees of the zero-dimensional sets to be computed, by computing resultants/subresultants/triangular decompositions, and then apply either direct semi-numerical methods as in [9], [5], [7] or [6] or follow the symbolic resolution by computing a rational parametrization of the roots as in [11].

It has been shown in [11] that the first strategy performs well on systems in general position (one coordinate separates the roots) while the second one was efficient in the general case. Our new solver mixes strategies from the state of the art together with new algorithms for computing certified approximations of the real roots of univariate polynomials with a large number of significant digits (mandatory of one wants to solve, for example, a triangular system numerically) and/or for computing rational parametrization of the roots.

As a byproduct, the use of this new algorithm for computing the topology of plane curves in the algorithm from [11] speeds-up so much the overall process that new classes of curves can now be considered.

The shape of the systems. Consider $\{f_1(X,Y)=0, f_2(X,Y)=0\}$ where f_1 and f_2 are polynomials with rational coefficients of degree at most d_1 in X (resp. d_2 in Y). The number of complex roots of such system is then bounded by $d \leq d_1 d_2$.

The way we decide which algorithm to choose for solving a system depends strongly on its geometrical properties. It is well known that any system can be decomposed (using Gröbner bases, resultants, triangular sets, etc.) as the union of regular triangular systems :

$\left\{f_y(Y)=0=Y^{d_y}+R_y(Y), f_{x,y}(X,Y)=0=X^{d_x}+R_{x,y}(X,Y)\right\}$ where $d_y d_x \leq d_1 d_2 \leq d$, $\deg(R_y,Y)<d_y, \deg(R_{x,y},Y)<d_y, \deg(R_{x,y},X)<d_x$.

The way the numerical/semi-numerical approximation of the roots is obtained once the system has been simplified strongly depends on the properties of $f_{x,y}(X,Y)$ (linear in X or not, squarefree over the roots of f_y or not), but,

K. Fukuda et al. (Eds.): ICMS 2010, LNCS 6327, pp. 100–104, 2010.

in any case, one needs to be able to refine these roots with a large precision (number of significant digits) since the coefficients of $R_{x,y}$ may be huge integers.

Accurate and certified approximation of the real roots of univariate polynomials. Given $p \in \mathbb{Q}[T]$, we assume that we already have an algorithm for isolating the real roots of a univariate polynomial with rational coefficients (for example [15]) say computing intervals $(u, v) \in \mathbb{Q}^2$ such that $u \neq v \Leftrightarrow \exists! x \in]u, v[\cap \mathbb{R} | p(x) = 0$ and $u = v \Leftrightarrow p(u) = p(v) = 0$.

Isolating intervals' length may be quite large and, when dealing with univariate polynomials with integers as coefficients. Their refinement (or equivalently the accurate and certified approximation of the roots of the polynomial) may become the most costly operation if it is done straightforwardly. This step becomes costly in regards of the others steps of the global algorithm when the coefficients of the polynomial are huge integers, which is often the case when dealing with results of variables' eliminations (Resultants, Gröbner bases) and critical when the roots have to by plugged in another polynomial for a recursive resolution (triangular sets).

Our refinement algorithm is based on an computable version of Kantorovitch's criteria where we make use of multiprecision interval arithmetic to provide predicates that ensures the quadratic convergence of the Newton iterator.

Theorem 1. *Let $p \in \mathbb{Q}[X]$ be a squarefree polynomial and $\lfloor x \rceil$ be an isolating interval of one of its roots α such that $0 \notin p'(\lfloor x \rceil)$* ***(Hypothesis 1)****.*

- *Set* $\lfloor x_0 \rceil = \frac{1}{2}\left(\text{leftBound}(\lfloor x \rceil) + \text{rightBound}(\lfloor x \rceil)\right)$ *(encoded as an interval of width 0).*
- *Compute* $\lfloor K \rceil = \left\lfloor \frac{\text{diam}\left(\lfloor p' \rceil (\lfloor x \rceil)\right)}{\text{diam}(\lfloor x \rceil)} \right\rceil$, $\lfloor a \rceil = \left((\lfloor p' \rceil (\lfloor x_0 \rceil))^{-1} \right)^2$, $\lfloor b \rceil = |\lfloor p \rceil (\lfloor x_0 \rceil)|$, $\lfloor h \rceil = \lfloor 2 \rceil \lfloor K \rceil \lfloor a \rceil \lfloor b \rceil$.
- ***If*** $\lfloor h \rceil \subset [0, 1[$ ***(Criterion 1)****, then set* $t^\star \geq \frac{\lfloor 2 \rceil \lfloor a \rceil}{\lfloor h \rceil}\left(1 - \sqrt{1 - \lfloor h \rceil}\right)$ *and* $t^{\star\star} \geq \frac{\lfloor 2 \rceil \lfloor a \rceil}{\lfloor h \rceil}\left(1 + \sqrt{1 - \lfloor h \rceil}\right)$
- *If* $\lfloor y \rceil =]\underline{x_0 - t^\star}, \overline{x_0 + t^\star}[\subset \lfloor x \rceil$ ***(Criterion 2)****, then :*
  - *the iterates* $x_{n+1} = x_n - \frac{p(x_n)}{p'(x_n)}$ *exist and remain in* $\lfloor y \rceil$
  - *the sequence* $(x_n)_{n \geq 1}$ *converges quadratically to* α*, which is the only root of* p *in* $]\underline{x_0 - t^{\star\star}}, \overline{x_0 + t^{\star\star}}[$
  - $\|\alpha - x_n\| \leq \left(\frac{t^\star}{t^{\star\star}}\right)^{2^n} \|x_n - x_{n-1}\|$.

The precision to be used for the multiprecision interval arithmetic is easy to control by checking that the signs of the derivative of the polynomial are well defined at some points, keeping in mind that p is squarefree and checking that Newton mathematically converges quadratically to a unique solution under Kantorovitch's conditions.

This algorithm differs from the method proposed in [1] (QIR) by the use of multi-precision floating point arithmetic, the way the precision is dynamically controlled and the way the quadratic convergence of the Newton iterator is

ensured (or not) at some stage. Among the few examples proposed in [1], we illustrate this article with the most significant one (f_4). We follow the comparison proposed in [1] : timings for isolating and refining with a precision ϵ are given for the QIR algorithm ([1]) and for our implementation (RS). We obtain the following normalized timings (computation were not performed on the same machine):

	Isolate	$\epsilon = 10^{-100}$	$\epsilon = 10^{-1000}$	$\epsilon = 10^{-10000}$
QIR	7.4	1	12.5	660
RS	4.23	1	11.9	225

The Rational Univariate Representation of a bivariate system. When $d_x > 1$, one could combine the use of real root isolators (such as methods based on Descarte's rule of signs), tight separation bounds together with interval arithmetic to get isolating boxes around the solutions, as proposed for example in [9], [5], [7] or [6]. Another approach is to compute a so called Rational Univariate Representation of the roots (as in [11]), that is to say, an equivalent system with the following shape : $\left\{ r_t(T) = 0, X = \frac{r_{t,x}(T)}{r_{t,1}(T)}, Y = \frac{r_{t,y}(T)}{r_{t,1}(T)} \right\}$ (T is a new variable).

Computing such an intermediate object increases the symbolic processing but make easier the last stage of the algorithm (numerical approximation of the roots).

Given any ideal $I \in \mathbb{Q}[X,Y]$ we define $\boldsymbol{V}(I)$ as the zero set of I. A Rational Univariate Representation (RUR) of any zero-dimensional system is defined by a set of univariate polynomials with rational coefficients $\{r_t(T), r_{t,1}(T), r_{t,X}(T), r_{t,Y}(T)\}$ and a polynomial t (in general a linear form) that separates $\boldsymbol{V}(I)$ ($t \in \mathbb{Q}[X,,Y]$ separates $\boldsymbol{V}(I)$ if $x \longrightarrow t(x)$ is injective on $\boldsymbol{V}(I)$). According to [13], up to an algorithm in $O(d^2)$ computing a RUR is equivalent to computing the traces (for $h \in \mathbb{Q}[X,Y], Tr(h)$ is the trace of the multiplication by h modulo I) $Tr(Xt^i)$, $Tr(Yt^i)$ and $Tr(t^i)$ for $i = 1..d$, d being the number of complex zeroes of I.

Computing these traces by using a general method (say not taking care of the shape of the system) is thus the main operation (say $O(d^w)$ operations with $w \geq 2,5$ according to [13], [3] or [14]. In fact much more operations are required since these algorithms suppose some prerequisite to be already computed (such as the multiplication table of the quotient algebra). Taking into account the shape of the input (a regular triangular system), one can decrease significantly the complexity upper bound for computing a Rational Univariate Representation :

Theorem 2. *When* $I = \langle f_y(Y), f_{x,y}(X,Y) \rangle$ *with* $f_y(Y) = Y^{d_y} + R_y(Y)$, $f_{x,y}(X,Y) = X^{d_x} + R_{x,y}(X,Y)$, $\deg(R_y,Y) < d_y, \deg(R_{x,y},Y) < d_y$, $\deg(R_{x,y},X) < d_x$, *one can compute the RUR of* $\boldsymbol{V}(I)$ *related to any separating element* t *in* $\tilde{O}(d^2)$ *arithmetic operations.*

In particular, one can easily propose an algorithm for computing a Rational Univariate Representation belongs to the same class of complexity than the known algorithms for computing triangular decompositions (or at least subresultant sequences). Thus, theoretically, the computation of this intermediate object may

not increase significantly the computation time required by the symbolic processing but it simplifies a lot the next stage (numerical approximation).

Experiments and conclusion. As the initial motivation of this study was to speed up a new algorithm for the computation of the topology of plane curves [11], the new solver components have been tested on challenging curves from the state of the art which were not computable by any know algorithm.

In the following experiments we took some challenging curves (mostly from Oliver Lab's data base [10] and from [6]) and we replaced the general solver (Gröbner basis + RUR) from [11] by the following dedicated bivariate solver :

- Decompose the system as a union of regular triangular sets (add-hoc implementation in C language that is closed to the RegularChain function from Maple 14) : it is based on a lazy decomposition of the subresultant suite associated with the system.
- For each triangular component, compute a rational univariate representation of the solutions :
  - If the bivariate polynomial has degree 1, then the triangular set has the following shape : $\{f_y(Y) = 0, X - f_x(Y) = 0\}$. We then set $t = Y, r_y(T) = f_y(T), r_{t,1} = \left(\overline{f_y(T)}\right)', r_{t,y} = T r_{t,1}(T) \bmod r_{t,1}(T), r_{t,x}(T) = f_x(T) r_{t,1}(T) \bmod r_{t,x}(T)$, which defines a Rational Univariate Representation of the system.
  - If the bivariate polynomial has degree ≥ 1 we then compute a Rational Univariate Representation.
- For each Rational Univariate representation $(r_t, r_{t,1}, r_{t,y}, r_{t,y}\}$, we isolate the roots of r_t by means of non overlapping intervals with rational bounds and refine them using bisection until the hypothesis 1 of Theorem 1 is fulfilled.
- For each isolating interval $\lfloor \alpha \rceil$, we evaluate $\frac{r_{t,y}(\lfloor \alpha \rceil)}{r_{t,1}(\lfloor \alpha \rceil)}$ using interval arithmetic and refining $\lfloor \alpha \rceil$ if necessary (using theorem 1 if necessary) until the boxes $\lfloor \alpha \rceil \times \frac{r_{t,y}(\lfloor \alpha \rceil)}{r_{t,1}(\lfloor \alpha \rceil)}$ do not overlap.

The next table shows examples to illustrate the added value of the new algorithm for computing a Rational Univariate representation from a regular triangular system when it replaces the classical solver (Gröbner basis + RUR) in the strategy proposed in [11]. We also added the timings from Alcix ([6]) and Cad2d ([4]) well known efficient implementations of CAD based algorithms on these examples :

	OL-10-21-t1	**OL-13-70**	**OL-17-55**	**OL-16-11**	**OL-2-79**
New	12.1	34.4	12.3	42.3	19.2
Classical	116.3	542.2	MEM	STOP	MEM
Alcix	507.1	581.3	1038.2	STOP	507.1
Cad2d	231.2	174.7	2812.4	ERROR	ERROR

STOP = computation stopped after 1h - MEM = not enough memory. Computations were performed on a macbook pro 2.4 Ghz with 4 Mb of memory.

References

[1] Abbott, J.: Quadratic interval refinement (qir). In: ISSAC 2006 (2006) (poster session)
[2] Basu, S., Pollack, R., Roy, M.-F.: Algorithms in real algebraic geometry. Algorithms and Computations in Mathematics, vol. 10. Springer, Heidelberg (2003)
[3] Bostan, A., Salvy, B., Schost, E.: Fast algorithms for zero-dimensional polynomial systems using duality. Applicable Algebra in Engineering, Communication and Computing 14(4), 239–272 (2003)
[4] Brown, C.W.: Qepcad b: A program for computing with semi-algebraic sets using cads. Sigsam Bulletin 37, 97–108 (2003)
[5] Collins, G.-E., Johnson, J., Krandick, W.: Interval arithmetic in cylindrical algebraic decomposition. Journal of Symbolic Computation 34(2), 145–157 (2002)
[6] Eigenwillig, A., Kerber, M.: Exact and efficient 2d-arrangements of arbitrary algebraic curves. In: SODA, pp. 122–131 (2008)
[7] Emiris, I.Z., Tsigaridas, E.P.: Real solving of bivariate polynomial systems. In: Ganzha, V.G., Mayr, E.W., Vorozhtsov, E.V. (eds.) CASC 2005. LNCS, vol. 3718, pp. 150–161. Springer, Heidelberg (2005)
[8] Giusti, M., Lecerf, G., Salvy, B.: A gröbner free alternative for solving polynomial systems. Journal of Complexity 17(1), 154–211 (2001)
[9] Gonzalez-Vega, L., Necula, I.: Efficient topology determination of implicitly defined algebraic plane curves. Comput. Aided Geom. Des. 19(9), 719–743 (2002)
[10] Labs, O.: http://www.oliverlabs.net
[11] Pouget, M., Lazard, S., Tsigaridas, E., Rouillier, F., Peñaranda, L., Cheng, J.: On the topology of planar algebraic curves. In: Mathematics in Computer Science (to appear 2010)
[12] Revol, N., Rouillier, F.: Motivations for an arbitrary precision interval arithmetic and the mpfi library. In: Baker Kearfott, R. (ed.) Workshop on Validated Computing, Toronto, Canada, pp. 155–161. SIAM, Philadelphia (2002)
[13] Rouillier, F.: Solving zero-dimensional systems through the rational univariate representation. Journal of Applicable Algebra in Engineering, Communication and Computing 9(5), 433–461 (1999)
[14] Rouillier, F.: On the rational univariate representation. In: International Conference on Polynomial System Solving, pp. 75–79 (2004)
[15] Rouillier, F., Zimmermann, P.: Efficient isolation of polynomial real roots. Journal of Computational and Applied Mathematics 162(1), 33–50 (2003)

Accurate and Reliable Computing in Floating-Point Arithmetic

Siegfried M. Rump

Institute for Reliable Computing, Hamburg University of Technology,
Schwarzenbergstraße 95, Hamburg 21071, Germany,
and Visiting Professor at
Waseda University, Faculty of Science and Engineering,
3–4–1 Okubo, Shinjuku-ku, Tokyo 169–8555, Japan
rump@tu-harburg.de
http://www.ti3.tu-harburg.de

Abstract. Methods will be discussed on how to compute accurate and reliable results in pure floating-point arithmetic. In particular, verification methods with INTLAB and error-free transformations will be presented in some detail.

Floating-point computations are tremendously fast on todays computers. Even small laptops easily beat the largest available mainframes from two or three decades ago in performance. However, floating-point arithmetic carries a reputation of being unreliable and not trustworthy. Until the mid 1980's there wasn't even a commonly accepted definition of floating-point operations.

This situation changed completely with the IEEE 754 arithmetic standard, inaugurated in 1985 and revised in 2008 [2]. Today the vast majority of all computers adhere to this standard, which means that the result of all floating-point operations is precisely known. This allows to use floating-point arithmetic as a tool to assist mathematical proofs.

We will discuss in two talks two ways of assisting mathematical proofs by using floating-point arithmetic.

A first possibility is to use directed roundings. Denote the set of floating-point numbers by $\mathbb{F}$, let $a, b \in \mathbb{F}$ be floating-point numbers, and let $\circ \in \{+, -, \cdot, /\}$ be an operation. The result of the floating-point approximation of $a \circ b \in \mathbb{R}$ in rounding downwards $\mathrm{fl}_\nabla(a \circ b)$ and in rounding upwards $\mathrm{fl}_\Delta(a \circ b)$ is the uniquely defined floating-point number being less than or equal, or greater than or equal to the true (real) result $a \circ b$, respectively. It follows immediately that $a \circ b$ is a floating-point number if and only if the rounded downwards and upwards result coincide:

$$\forall a, b \in \mathbb{F}: \qquad a \circ b \in \mathbb{F} \qquad \Leftrightarrow \qquad \mathrm{fl}_\nabla(a \circ b) = \mathrm{fl}_\Delta(a \circ b)$$

Furthermore, the interval $[\mathrm{fl}_\nabla(a \circ b), \mathrm{fl}_\Delta(a \circ b)]$ is always an inclusion of the true real result. Finally, it is easy to define operations on intervals such that the true results are always included in the result interval.

K. Fukuda et al. (Eds.): ICMS 2010, LNCS 6327, pp. 105–108, 2010.

Although interval operations are easy to handle and fast implementations are available, see [3], they are generally prone to overestimations. In fact, taking some numerical algorithm such as Gaussian elimination or a Runge Kutta scheme, one may replace every operation by its corresponding interval operation. However, although the computed result intervals are always correct (if no division by an interval containing zero causes premature failure), these result intervals can also be expected to be uselessly wide - except for some special and/or toy problems.

This is due to improper use of interval arithmetic; for a detailed discussion see my overview article [5]. The purpose of so-called verification methods, which use interval arithmetic in a proper way, is ambitious: For a given problem it is proved, with the aid of floating-point arithmetic on a computer, that there exists a (unique) solution within the computed bounds. The methods are constructive, and the results are rigorous in every respect. Verification methods apply to data with tolerances as well.

Rigorous results are also the main goal in computer algebra. However, verification methods use solely floating-point arithmetic, so that the total computational effort is not too far from that of a purely (approximate) numerical method. Nontrivial problems have been solved using verification methods. For example:

Tucker received the 2004 EMS prize awarded by the European Mathematical Society for [8] "giving a rigorous proof that the Lorenz attractor exists for the parameter values provided by Lorenz. This was a long standing challenge to the dynamical system community, and was included by Smale in his list of problems for the new millennium. The proof uses computer estimates with rigorous bounds based on higher dimensional interval arithmetics."

Sahinidis and Tawaralani received the 2006 Beale-Orchard-Hays Prize for their package BARON which [6] "incorporates techniques from automatic differentiation, interval arithmetic, and other areas to yield an automatic, modular, and relatively efficient solver for the very difficult area of global optimization".

A number of verification methods are implemented in INTLAB [4], the Matlab toolbox for reliable computing. Using the operator concept implies that executable code in INTLAB is very readable, almost like a specification. Besides the basic interval operations including real and complex intervals and all elementary standard functions, packages for automatic differentiation for gradients and Hessians, Taylor series, slopes, polynomials, a long arithmetic and more are implemented. INTLAB is free for academic use, is developed and written by the author. It was, for example, used by [1] in the solution of half of the problems of the 10×10-digit challenge by [7].

We will demonstrate how to use INTLAB. In particular it will be shown that often the time penalty for verified results is less than one order of magnitude compared to the fastest floating-point algorithm, in particular for large problems: In contrast to computer algebra methods, verification methods often become relatively faster with increasing dimension. Linear and nonlinear problems with several thousand unknowns are solved.

In this talk also examples of the *wrong* use of interval operations are given. In the past such examples contributed to the dubious reputation of interval arithmetic, whereas they are, in fact, just a misuse.

In the second talk we demonstrate another use of floating-point arithmetic in so-called *error-free transformations*. For given floating-point numbers a, b it is known that the error of the result of the floating-point operation $\mathrm{fl}(a \circ b)$ in rounding to nearest to the true (real) result $a \circ b$ is itself a floating-point number. More precisely,

$$x = \mathrm{fl}(a \circ b) \quad \Rightarrow \quad y := a \circ b - \mathrm{fl}(a \circ b) \in \mathbb{F} \quad \text{for} \quad \circ \in \{+, -, \cdot\} \tag{1}$$

and

$$x = \mathrm{fl}(a/b) \quad \Rightarrow \quad y := a - b \cdot x \in \mathbb{F} \ , \tag{2}$$

provided no underflow occurred for multiplication and division. It is also well-known since a long time how to compute these errors y in pure floating-point arithmetic. Thus the pair (a, b) can be transformed into the pair (x, y) such that x is the floating-point approximation of $a \circ b$, and y is the exact error according to (1) and (2). As the name suggests, the transformation of (a, b) into (x, y) is error-free. The challenge is to create error-free transformations for composed operations and entire algorithms, or to create algorithms accompanied by (mathematically correct) error estimates.

As an example we introduce an error-free vector transformation: For a given vector $u \in \mathbb{F}^n$, a new vector $v \in \mathbb{F}^n$ is computed such that

$$v_n = \mathrm{fl}(\sum u_i) \qquad \text{and} \qquad \sum u_i = \sum v_i \ .$$

Moreover, it can be shown that the condition number of the sum $\sum v_i$ is about 10^{-16} times the condition number of the sum $\sum u_i$ when computing in IEEE 754 double precision arithmetic. Applying the error-free transformation repeatedly computes the exact sum accurately to the last bit.

Another example are geometrical predicates. For instance, it is to be decided for some point whether it is on a given hyperplane, or on which side it is. Error-free transformations as above can be used to compute results with (mathematically correct) error estimates. The transformations are applied iteratively until the problem can be solved. Since in all cases only floating-point operations are involved, the resulting algorithms are fast.

A number of reference implementations of error-free transformations are available in INTLAB. Due to the interpretation overhead they should only be used as reference; really fast algorithms evolve in an implementation in some programming language like C or Fortran.

References

1. Bornemann, F., Laurie, D., Wagon, S., Waldvogel, J.: The SIAM 100-Digit Challenge—A Study in High-Accuracy Numerical Computing. SIAM, Philadelphia (2004)
2. ANSI/IEEE 754-2008: IEEE Standard for Floating-Point Arithmetic, New York (2008)

3. Rump, S.M.: Fast and parallel interval arithmetic. BIT Numerical Mathematics 39(3), 539–560 (1999)
4. Rump, S.M.: INTLAB - INTerval LABoratory. In: Csendes, T. (ed.) Developments in Reliable Computing, pp. 77–104. Kluwer Academic Publishers, Dordrecht (1999)
5. Rump, S.M.: Verification methods: Rigorous results using floating-point arithmetic. Acta Numerica, 287–449 (2010)
6. Sahinidis, N.V., Tawaralani, M.: A polyhedral branch-and-cut approach to global optimization. Math. Programming B103, 225–249 (2005)
7. Trefethen, L.N.: The SIAM 100-Dollar, 100-Digit Challenge. SIAM-NEWS 35(6), 2 (2002), http://www.siam.org/siamnews/06-02/challengedigits.pdf
8. Tucker, W.: The Lorenz attractor exists. C. R. Acad. Sci., Paris, Sér. I, Math. 328(12), 1197–1202 (1999)

Deferring Dag Construction by Storing Sums of Floats Speeds-Up Exact Decision Computations Based on Expression Dags

Marc Mörig*

Department of Simulation and Graphics, Faculty of Computer Science,
University of Magdeburg, Universitätsplatz 2,
D-39106 Magdeburg, Germany
moerig@isg.cs.uni-magdeburg.de

Abstract. Expression-dag-based number-types are a very general and user-friendly way to achieve exact geometric computation, a widely accepted approach to the reliable implementation of geometric algorithms. Such number-types record the computation history of a numerical value in an expression dag in order to allow for recomputing the value or an approximation of it at a later stage. We describe how to defer dag construction by using error-free transformations into sums of floating-point numbers. We store a limited number of summands in statically allocated memory in order to postpone or avoid dag creation which involves expensive dynamic memory allocations. Furthermore we report on experiments where we compare different implementation strategies of our new approach. The experiments show that for small polynomial expressions typically arising in geometric applications our approach is superior to existing expression-dag-based number-types in the presence of degenerate and nearly degenerate configurations and competitive otherwise.

Keywords: expression dag, exact geometric computation, error-free transformations, algorithm engineering.

1 Introduction

Decisions in geometric predicates are based on the sign of certain arithmetic expressions. By correct sign computations we can assure correct control flow. This is the crucial idea of the so-called exact geometric computation paradigm [16]. It ensures that the implementation behaves like its theoretical counterpart, thereby ensuring correct combinatorics, whereas numerical values might still be inaccurate. However, the potential inaccuracy never leads to wrong or even contradictory decisions.

We implement a new number-type based on expression dags. It allows one to exactly compute the sign of arithmetic expressions involving the operations $\pm, \cdot, /, \sqrt[d]{}$. Recording the computation history of a numerical value in an expression tree, more precisely an acyclic directed graph, allows us to recompute the

* Partially supported by DFG grant SCHI 858/1-1

K. Fukuda et al. (Eds.): ICMS 2010, LNCS 6327, pp. 109–120, 2010.

value at a later stage of a program in a different way. For example, the created expression dags allow for lazy evaluation: First, we compute a crude numerical approximation only. If the current approximation does not suffice, we can use the expression dag to iteratively compute better and better approximations. A typical application of this scheme is verified sign computation. If the actual numerical value is far away from zero, rough numerical approximations suffice to compute the correct sign. Only if numerical values are close to zero, high precision computation is needed. Thus the running time for a sign computation becomes adaptive to its difficulty. This pays off for geometric algorithms where most sign computations are in fact easy.

For the sake of ease-of-use recording computation history and adaptive lazy evaluation is wrapped in a number-type. In programming languages providing operator overloading, such a number-type can then be used like built-in number-types. A user need not care about any implementation details in order to get verified signs. There are other techniques for exact decision computations, which usually lead to more efficient code. However to incorporate these techniques into existing implementations one needs special knowledge whereas the application of expression-dag-based number-types is straightforward. Our overall goal is to improve the performance of expression-dag-based number-types. For many minor and major implementation details it is not clear what the best choices are; therefore in our implementation we focus on a modular design. This allows us to easily generate several variants with different internals, simplifying experiments to evaluate the impact of design decisions. Additionally, modularity allows you to tailor the number-type to your specific needs. For example, the arbitrary precision software floating-point type used to evaluate the expression dag is exchangeable, allowing to either choose the most efficient number-type available or one that integrates best with the application.

Computing with expression dags comes not for free. The main cost factors are dynamic memory management (for the dag nodes as well as the software floating-point types) and computing with software arithmetic. Therefore computing even the first approximation of an expression is orders of magnitudes slower than evaluating the expression with hardware floats. In a variant of our new number-type we represent the value of an arithmetic expression exactly as a sum of hardware floating-point numbers. We limit the maximum number of summands to avoid dynamic memory management. As long as such a representation is available no expression dag is created. The effects of this are twofold. First, if the sign of an expression can be computed using the sum representation alone, we avoid the expensive dag operations completely. Second, if at some point a dag must be created, it will have different shape than the dag corresponding to the original expression. Thus our method can also be seen as a limited form of expression rewriting. We must be careful though. By computing exactly we loose adaptivity, the reason to use expression-dag-based number-types in the first place. We can only improve performance if our exact computations are faster than the creation and the first stage of adaptive evaluation of the original

dag representation. This also has to hold in the case that at some point the sum must be transformed to a dag representation.

So called error-free transformations transform an arithmetic expression involving floating-point numbers into a mathematically equivalent expression that is more suited for a particular purpose [11]. We use them to implement operations on sums of floats. We need the three ring operations addition, subtraction and multiplication as well as computing the sign of a sum. Additionally we provide an operation compressing a sum, i.e., reducing the number of summands if possible. While compression is not strictly necessary, it helps greatly to improve the performance. We call this set of five operations the basic strategy. The result of a division or root operation has a finite floating-point representation very rarely. We therefore do not attempt to implement these operations for sums of floats. The error-free transformations we use require only a few standard floating-point operations and are therefore very fast. They are however subject to overflow and underflow. Since we have to compute correct signs in any case, we need a way to handle them. There are several parameters in our implementation, e.g., the basic strategy and how to handle overflow and underflow. After some preliminaries in Section 3 we discuss alternatives for these parameters in Section 4.

For each parameter we have several alternatives, all of them can be used independently. This gives an overwhelming number of variants. To find the best set of alternatives we perform a series of experiments. We compute the Delaunay triangulation of point sets that require increasingly difficult sign computations and compare how our variants perform, typically fixing all but one parameter for which the alternatives are examined. To validate our approach we also compare with variants of our number-type that do not attempt to defer dag creation and other expression-dag-based number-types. In Section 5 we report on the results.

2 Related Work

Ours is not the first number-type based on expression dags. `CORE::Expr` [6] has been developed by Chee Yap and his co-workers at New York University, and the number-type `leda::real` [1,8] has been developed as part of the LEDA library. In its newest version `CORE::Expr` introduces dag nodes for the sum and product of arbitrarily many operands, allowing to save the creation of intermediate, binary sum or product nodes. In order to benefit from the new node types, the user has to use them explicitly [4,17]. In `leda::real` dag creation is avoided as long as operations can be performed exactly with a single double precision float. The exactness is checked using interval arithmetic. A double precision float has a mantissa of 53 bits so this approach is suited for input with very low precision only. With 32 bit integers there can be quite a few exact operations before a dag must be created. If the input itself consists of double precision floats they are likely to use all bits of their mantissa. Therefore there might be quite a few exact additions or subtractions (e.g., by Sterbenz Lemma [15] the difference of two floating-point numbers is exact, if their quotient is between $1/2$ and 2) but the product of two floats is likely to require more than 53 bits and can therefore

not be represented exactly. Unlike our approach, `leda::real` also attempts to perform divisions and roots exactly.

Shewchuk [13] provides a set of operations based on error-free transformations to compute with sums of floats and uses them to implement geometric predicates. Mörig and Schirra [9] discuss using error-free transformations and exact sign-of-sum algorithms in the implementation of geometric predicates. Both approaches achieve very efficient predicate implementations but require special knowledge and more implementation effort from the user to implement their own predicates. Here we make insights from both approaches available in a much more user-friendly form.

3 Preliminaries

At the core of all operations on sums of floats are so called error-free transformations [11]. Let $\oplus$ and $\odot$ denote floating-point addition and multiplication respectively. For example, $a + b$ can be transformed into $c^{\text{hi}} + c^{\text{lo}}$, such that $a \oplus b = c^{\text{hi}}$ and $a + b = c^{\text{hi}} + c^{\text{lo}}$. Note that c^{lo} is the actual rounding error involved in computing $a \oplus b$. Efficient algorithms for performing this transformation have been devised for IEEE 754 [12] compliant arithmetic with exact rounding to nearest. $\textsc{twosum}(a, b)$, due to Knuth [7], uses six floating-point additions and subtractions to perform this transformation, $\textsc{fasttwosum}(a, b)$, due to Dekker [3], requires $|a| \geq |b|$, but uses only three operations. The transformations are error-free unless overflow occurs. Analogously, $\textsc{twoproduct}(a, b)$, due to Veltkamp and Dekker [3] computes floating-point values c^{hi} and c^{lo} with $a \odot b = c^{\text{hi}}$ and $a \cdot b = c^{\text{hi}} + c^{\text{lo}}$. TWOPRODUCT uses 17 floating-point operations and is error-free, unless overflow or underflow occurs. All operations on sums of floats that we use are based on this three error-free transformations.

Table 1. Upper bound on $\#\text{s}(c)$, the number of summands required to represent c and $\text{msb}(c)$ the most significant bit in c. Lower bound on $\text{lsb}(c)$, the least significant bit.

c	$\#\text{s}(c)$	$\text{msb}(c)$	$\text{lsb}(c)$
$a \pm b$	$\#\text{s}(a) + \#\text{s}(b)$	$\max\{\text{msb}(a), \text{msb}(b)\} + 1$	$\min\{\text{lsb}(a), \text{lsb}(b)\}$
$a \cdot b$	$2\#\text{s}(a)\#\text{s}(b)$	$\text{msb}(a) + \text{msb}(b)$	$\text{lsb}(a) + \text{lsb}(b)$

We can estimate the number of summands required to represent an expression. The most straightforward way to implement the ring operations is to simply copy (and negate if necessary) summands in case of addition and subtraction and to apply TWOPRODUCT to each pair of summands in case of multiplication. Table 1 gives an upper bound on the number of summands $\#\text{s}(c)$ that follows from this strategy. Note that this bound is sharp, a different implementation of the ring operations can not lead to better bounds. Bounds on the most and least significant bit of a number are also given. A number c can be represented using $\lceil (\text{msb}(c) - \text{lsb}(c))/53 \rceil$ double precision summands, but $\text{msb}(c) - \text{lsb}(c)$ grows much slower than $\#\text{s}(c)$. This shows that we can reduce the number of summands

representing an expression, if the least significant bit and most significant bit in the input numbers do not differ too much.

Consider for example the in-circle test determinant D that allows to decide whether a point p is inside, outside or on the circle defined by the points q, r, s. Computing the sign of D, where all input coordinates are of double precision, is the most demanding sign computation performed by the Delaunay triangulation algorithm used in our experiments. When expanding D by its last column, as is usually done, D may require up to 1152 summands, according to Table 1.

$$D = \begin{vmatrix} p_x - s_x & p_y - s_y & (p_x - s_x)^2 + (p_y - s_y)^2 \\ q_x - s_x & q_y - s_y & (q_x - s_x)^2 + (q_y - s_y)^2 \\ r_x - s_x & r_y - s_y & (r_x - s_x)^2 + (r_y - s_y)^2 \end{vmatrix}$$

Now let l be the least significant bit and m the most significant bit in any of the input coordinates $p_x, p_y, q_x, \ldots$, then $\mathrm{msb}(D) \leq 4m+8$ and $\mathrm{lsb}(D) \geq 4l$. Thus D can be represented using $\lceil (4(m-l)+8)/53 \rceil$ double precision summands. For double precision input $m - l$ will usually be at least 53, in this case 5 summands suffice to represent the sum. Table 1 together with this example shows that there clearly is need to compress sums, i.e., try to reduce the number of summands along the computation.

4 Variations and Alternatives

We have five parameters in our implementation, they are the maximum number of summands allowed, the basic strategy, i.e., how arithmetical and other operations on sums are implemented, the compression policy, i.e., when to compress a sum, how to handle overflow and underflow and how to convert a sum into an expression dag representation.

Basic Strategy. The basic strategy provides five operations on sums of floats, they are addition, subtraction, multiplication, sign of sum and compression. We maintain a number as a sequence of floats, more precisely we store n double precision numbers $a_1, \ldots, a_n$ that represent the number $\sum_{i=1}^{n} a_i$. We have two alternative basic strategies.

The first basic strategy, `plainS`, maintains a plain sum without any special properties. Addition and subtraction are implemented straightforwardly, they simply copy summands. The multiplication performs TWOPRODUCT on all pairs of input summands. To compress a sum, for $i = 2, \ldots, n$ we compute $\mathrm{TWOSUM}(a_{i-1}, a_i)$ and replace a_{i-1} by c^{lo} and a_i by c^{hi}. If the addition $a_{i-1} \oplus a_i$ is exact, then c^{lo} is zero. In this case we eliminate zero summands. This leaves the value of the sum unchanged but improves its representation. There are possibly fewer summands and the new leading summand $a_{n'}$ is the floating-point approximation of the old sum. After several compressions, summands with higher index tend to be more significant that summands with lower index. To compute the sign of the sum we use the SIGNK algorithm presented in [9]. It first performs a compression step and then, using ordinary floating-point addition, adds the new

summands up to an approximation s. This is known as compensated summation, a well known approach to increase the accuracy of floating-point summation [5]. We use an error bound by Rump et al. [11] to verify the sign of s. When the sign can not be verified we re-iterate the algorithm. Mörig and Schirra show that SIGNK always terminates [9].

The second basic strategy, `expaS`, is based on work by Shewchuk [13]. We maintain a special type of sum called a strongly non-overlapping expansion. We do not allow zero summands unless the only summand is zero. In an expansion all summands are ordered by absolute value and summands are pairwise non-overlapping, meaning that the most significant non-zero bit of the smaller summand is smaller than the least significant non-zero bit of the larger summand. Therefore the sign of the expansion is always the sign of the leading summand a_n. Furthermore the least and most significant non-zero bit of all summands occur in a_1 and a_n respectively. Shewchuk provides algorithms to add and subtract expansions, to compress an expansion and to multiply an expansion with a single float. They are entirely based on TWOPRODUCT, TWOSUM and FASTTWOSUM. We use his implementation that is available on the web [14] and implement additional functionality following suggestions from his paper [13, section 2.8].

Plain sums and expansions employ opposing strategies. With plain sums the ring operations are lazy, at the cost of many summands since the upper bounds from Table 1 are always attained. SIGNK must do all the work. With expansions a normal form is maintained by the ring operations, this makes them more expensive but also reduces the number of summands. The sign of an expansion can be determined at no cost!

Compression Policies. One of our main concerns with sums of floats is to keep the number of summands small. All operations will be cheaper with fewer summands, since cost depends on the number of summands. More importantly we must switch to the expensive dag representation, if a ring operation can not be performed because the result, as predicted by Table 1, may exceed the maximum number of summands. We provide four different policies when to apply the compression algorithm that itself is provided by the basic strategy. The first, `noC`, is to not compress at all. The second, `alwC`, is to compress always after every ring operation. The third policy, `lazyC`, is more lazy, it compresses the operands of a ring operation if the number of summands predicted for the result exceeds the maximum number of summands. The fourth policy, `laagC`, is also lazy, but more aggressive. The operands are compressed once and then iteratively compressed again if the previous compression step reduces the number of summands. Considering the differences between plain sums and expansions, we can expect that plain sums will benefit more from additional compression.

Handling overflow and underflow. Since all our operations are based on error-free transformations and those are truly error-free only if neither overflow nor underflow occur we have to deal with such floating-point exceptions. The IEEE 754 standard provides a set of flags that are set when an exception occurs and can be checked and reset by the user. To protect from overflow and underflow, before

each operation we make a backup copy of the sum that is to be overwritten by the operation. After the operation we check the exception flags. If an exception occurs, we reset the exception flags, restore the sum from the backup and proceed by converting to an expression dag representation. This way overflow and underflow do not affect the correctness of the computation and are invisible to the user, as required. This strategy is called `restE`. Alternatively we reset the exception flags before each operation and check them afterwards. No backup is made but if an exception occurs an error handler is called. This approach, `detE`, is free from false positives but requires user interaction. For testing purposes we also have an alternative, `noE`, that does not check exception flags. It allows us to determine the cost of the former alternatives.

Checking for floating-point exceptions is only done for the operations where they may actually occur. For example the addition and subtraction of plain sums only copy summands and are therefore safe. Less obviously no exception can occur in the compression of expansions. Due to the non-overlapping property, no bit larger or smaller than already present in the expansion can be generated.

Unfortunately, using the exception flags is very expensive on the test platform. A different approach is to only perform operations if the input is both small and large enough to ensure that neither overflow nor underflow can occur. Since the summands of an expansion are ordered this can be checked in constant time. We provide another basic strategy, `safeS` that also maintains expansions but performs operations only if they are safe from overflow and underflow. We do not provide a safe basic strategy based on plain sums since checking the input will take linear time and they are already a bit slower than safe expansions.

Converting to a dag representation. We compute with sums of floats as long as a representation by sums is feasible. If not we switch to a dag representation. This can happen for three reasons. First we perform a division or root operation, second, the number of summands predicted for the result of a ring operation exceeds the maximum number of summands and third overflow or underflow occurred. We provide two alternatives to convert a sum to a dag. The first alternative, `nodeD`, creates a single node storing the value of the sum, computed using the software floating-point type used to evaluate the dag. The second alternative, `treeD`, creates a balanced binary summation tree over the summands. Another option, which we did not pursuit however, is to introduce a summation node with arbitrary arity to the dag. Such a node type is available in the newest version of `CORE::Expr`.

5 Experimental Comparison

We compare the variants described in the previous section using CGAL's [2] Delaunay triangulation algorithm that uses the 2D orientation and incircle predicate. Of these predicates, the incircle predicate is arithmetically more demanding. We use CGAL's `Simple_cartesian` kernel. As input we use randomly generated point sets from the test data generator described by Mörig and Schirra [9]. It generates sets with a certain percentage f of points on the boundary of

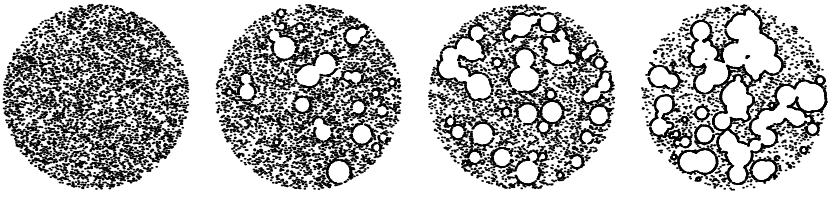


Fig. 1. Sample input data sets

a union of disks and no points in the interior. With increasing f this forces the Delaunay triangulation algorithm to perform more difficult incircle tests. For $f \in \{0\%, 25\%, 50\%, 75\%\}$ we generate 25 point sets of 5000 points each and measure the average running time in seconds. Sample input sets are shown in Figure 1. The experiments are run on a notebook with an Intel Core 2 Duo T5500 processor with 1.66 Ghz, using `g++` 4.4.1, CGAL 3.3.1, LEDA 6.0, GMP 4.3.1 and MPFR 2.4.1. When ranking different variants of sums of floats by measured running time, their relative order is invariant with respect to f. In fact the measured running time itself is nearly invariant with respect to f, since the approach is non-adaptive. For one variant this can be seen in Figure 6, most figures show only results for $f = 25\%$. We repeated our experiments on a different platform with similar results. The parameter space to search for an optimal variant is rather large. We make orthogonal cuts through the parameter space, varying one parameter and fixing the others to an alternative that performs well in the other experiments.

First we are interested in the impact of the maximum length of a sum on the performance. We vary the maximum length from 2 to 64. Looking at Figure 2 it can be seen that the computation times reaches a minimum at 8 or 16 summands and then remains constant. This is surprisingly low, since with 8 summands the product of two sums with n and m summands is transformed into a sum only if $mn \leq 4$. For 8 or more summands almost all signs are computed using the representation as sum only.

Next we evaluate the effect of different compression policies for both basic strategies. Figure 3 shows the results. Plain sums use the exception handling

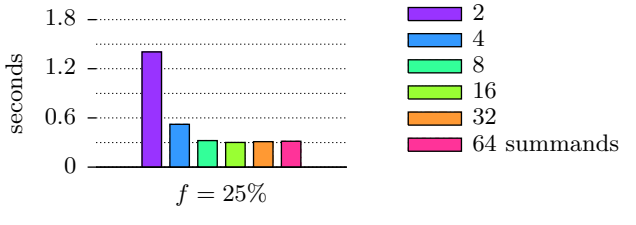


Fig. 2. Safe expansions, lazy compression and no floating-point exception handling. The dag is created as a tree and uses `leda::bigfloat`.

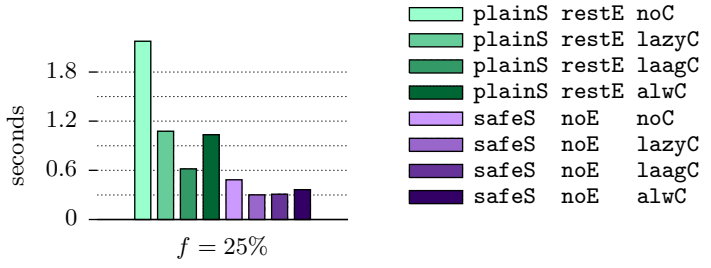


Fig. 3. Comparison of different compression policies: no compression (`noC`), lazy compression (`lazyC`), lazy aggressive compression (`laagC`) and compressing always after an operation (`alwC`). All variants use 16 summands, the dag is created as a tree and uses `leda::bigfloat`.

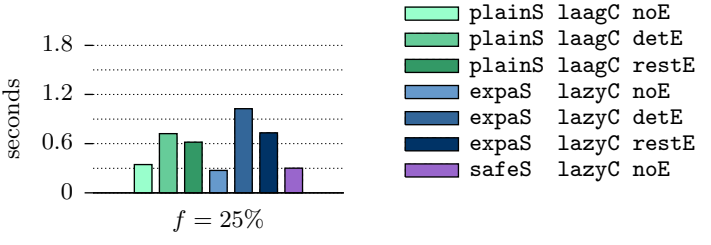


Fig. 4. Comparison of different floating-point exception handling strategies: no exception handling (`noE`), only detecting exceptions (`detE`) and restoring the sum in case of an exception (`restE`). All variants use 16 summands, the dag is created as a tree and uses `leda::bigfloat`

that is restoring sums, safe expansions need no exception handling, therefore their running time is in general much faster here. No compression is performed by the ring operations for plain sums, so any additional compression increases the performance of plain sums. The best result is provided by lazy aggressive compression. The arithmetic operations on expansions compress their results internally. This is quite effective; the running time decreases only slightly with additional compression. There is not much of a difference between lazy and lazy aggressive compression which give the best results.

The results from our experiments concerning the handling of overflow and underflow are shown in Figure 4. First we can see that plain sums and expansions perform similarly with just a slight advantage for expansions. Using any kind of floating-point exception flag checking is very slow, increasing the running time by a factor of two or even more. The cost for checking the input to avoid overflow and underflow before an operation, as is done by safe expansions is however negligible. Safe expansions are even slightly better than plain sums without exception handling.

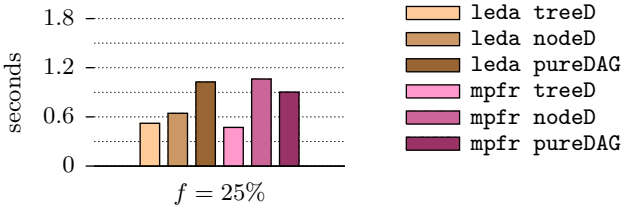


Fig. 5. Two different strategies to convert sums to a dag, creation of a single dag node (`nodeD`) and creation of a tree (`treeD`) compared to variants that do not attempt to defer dag creation (`pureDAG`). Dag evaluation is performed with either `leda::bigfloat` (`leda`) or MPFR (`mpfr`). The sums use safe expansions with lazy compression and 4 summands only.

Next we compare the alternatives for turning a sum into a dag representation. The results can be seen in Figure 5. We limit the maximum number of summands to 4 in order to ensure that sums are actually converted into expression dag representations. Following the analysis presented in Section 3 we can see that when all input coordinates have the same exponent, the term $(p_x - s_x)^2 + (p_y - s_y)^2$ and the corresponding cofactor both need at most 3 summands in an optimal representation. With our randomly generated input they will very rarely be representable by one summand only. The product of two sums with 2 summands is not transformed into a sum either, because the predicted length of the product is 8 summands. Therefore a dag representation must be created for many sums in this setting. Our experiments show that creating a tree is clearly superior to creating a single node, since it allows for adaptive evaluation of the tree, while creating a single node is completely non-adaptive. Interestingly, creating a tree is faster with MPFR while creating a single node is faster with `leda::bigfloat`. Note that although creating a dag representation is necessary for verified sign computation, using sums of floats is more efficient than directly creating a dag.

Finally we compare our best variant to other exact number-types based on expression dags. The results are shown in Figure 6. Obviously, the number-types that use dag representations adapt to the difficulty of the input while using sums of floats shows non-adaptive behavior. Deferring dag creation using sums of floats is clearly an improvement over creating the dag right away. The strategy of `leda::real`, to use a single float and interval arithmetic to avoid dag creation, implemented in `floatDAG`, does not pay off. This may be different for integer input. Both versions of `CORE::Expr` perform well for all considered input sets, although surprisingly the new version 2.0.8 is slightly slower. The focus in the new version is on expensive computations while we focus on simple cases [17,4]. Using sums of floats to defer dag creation results in a significant improvement over existing expression-dag-based number-types in the presence of degenerate and nearly degenerate configurations and is competitive for uniformly distributed input sets.

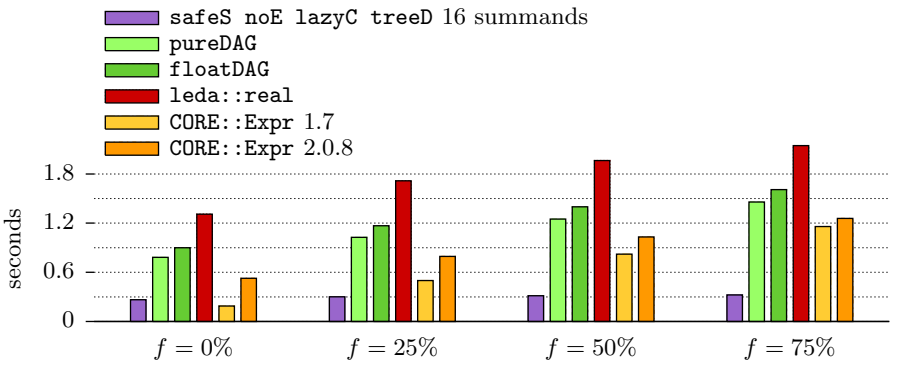


Fig. 6. Other expression-dag-based number-types and three variants of our new type: the best variant based on sums of floats, no attempt to defer dag creation (`pureDAG`), dag creation deferral using one float (`floatDAG`). All three variants use `leda::bigfloat`.

6 Conclusion and Future Work

Using sums of floats to defer dag creation can significantly improve the performance of expression-dag-based number-types when used to evaluate polynomial expressions on hardware precision input. In our experiments the approach is useful even if at some point the sum must be converted to a dag representation. This shows that to make an improvement it may not be necessary to tune the number of maximum summands for each application. Using floating-point exception flags to handle overflow and underflow is way to slow in our approach. It is crucial to keep the actual number of summands small, especially since the space requirements for direct results of ring operations are excessively large in general.

To find an optimal way of using sums of floats we use a single test scenario only. Expression-dag-based number-types are a general tool with many application. Therefore we have to verify our results for other applications, especially ones that include division and roots. An interesting test case is for example the coordinate comparison of line segment intersection points that is used by the Bentley-Ottmann sweep-line algorithm for computing the arrangement of line segments. The expression whose sign must be computed involves division of small polynomial expressions which will force dag creation.

References

1. Burnikel, C., Fleischer, R., Mehlhorn, K., Schirra, S.: Efficient exact geometric computation made easy. In: 15th ACM Symposium on Computational Geometry (SCG 1999), pp. 341–350. ACM, New York (1999)
2. CGAL: Computational Geometry Algorithms Library, `http://www.cgal.org/`
3. Dekker, T.J.: A floating-point technique for extending the available precision. Num. Math. 18(2), 224–242 (1971)

4. Du., Z.: Guaranteed Precision for Transcendental and Algebraic Computation made Easy. PhD thesis, Courant Institute of Mathematical Sciences, New York University (May 2006)
5. Kahan, W.: Further remarks on reducing truncation errors. Comm. of the ACM 8(1), 40 (1965)
6. Karamcheti, V., Li, C., Pechtchanski, I., Yap, C.: A core library for robust numeric and geometric computation. In: 15th ACM Symposium on Computational Geometry (SCG 1999), pp. 351–359. ACM, New York (1999)
7. Knuth, D.E.: Seminumerical Algorithms, 3rd edn. The Art of Computer Programming, vol. 2. Addison-Wesley, Reading (1997)
8. LEDA: Library of Efficient Data Structures and Algorithms, http://www.algorithmic-solutions.com/
9. Mörig, M., Schirra, S.: On the design and performance of reliable geometric predicates using error-free transformations and exact sign of sum algorithms. In: 19th Canadian Conference on Computational Geometry (CCCG 2007), pp. 45–48 (August 2007)
10. MPFR: A multiple precision floating-point library, http://www.mpfr.org/
11. Ogita, T., Rump, S.M., Oishi, S.: Accurate sum and dot product. SIAM Journal on Scientific Computing 26(6), 1955–1988 (2005)
12. Overton, M.L.: Numerical Computing with IEEE Floating-Point Arithmetic. SIAM, Philadelphia (2001)
13. Shewchuk, J.R.: Adaptive precision floating-point arithmetic and fast robust geometric predicates. Discrete and Computational Geometry 18(3), 305–363 (1997)
14. Shewchuk, J.R.: Companion web page to [13] (1997), http://www.cs.cmu.edu/~quake/robust.html
15. Sterbenz, P.H.: Floating-Point Computation. Prentice-Hall, Englewood Cliffs (1974)
16. Yap, C.: Towards exact geometric computation. Comput. Geom. Theory Appl. 7(1-2), 3–23 (1997)
17. Yu, J., Yap, C., Du, Z., Pion, S., Brönnimann, H.: The design of Core 2: A library for exact numeric computation in geometry and algebra. In: Fukuda, K., et al. (eds.) ICMS 2010. LNCS, vol. 6327, pp. 121–141. Springer, Heidelberg (2010)

The Design of Core 2: A Library for Exact Numeric Computation in Geometry and Algebra*

Jihun Yu[1], Chee Yap[1], Zilin Du[2], Sylvain Pion[3], and Hervé Brönnimann[4]

[1] Courant Institute, New York University,
New York, NY 10012, USA
{jihun,yap}@cs.nyu.edu
[2] Google Inc., 2400 Bayshore,
Mountain View, CA, USA
zilin@courant.nyu.edu
[3] INRIA Sophia Antipolis,
2004 route des Lucioles, BP 93,
06902 Sophia Antipolis, France
Sylvain.Pion@sophia.inria.fr
[4] CIS Department, NYU Poly,
Six MetroTech Center,
Brooklyn, NY 11201, USA
hbr@poly.edu

Abstract. There is a growing interest in numeric-algebraic techniques in the computer algebra community as such techniques can speed up many applications. This paper is concerned with one such approach called **Exact Numeric Computation** (ENC). The ENC approach to algebraic number computation is based on iterative verified approximations, combined with constructive zero bounds. This paper describes `Core 2`, the latest version of the `Core Library`, a package designed for applications such as non-linear computational geometry. The adaptive complexity of ENC combined with filters makes such libraries practical.

`Core 2` smoothly integrates our algebraic ENC subsystem with transcendental functions with ε-accurate comparisons. This paper describes how the design of `Core 2` addresses key software issues such as modularity, extensibility, efficiency in a setting that combines algebraic and transcendental elements. Our redesign preserves the original goals of the `Core Library`, namely, to provide a simple and natural interface for ENC computation to support rapid prototyping and exploration. We present examples, experimental results, and timings for our new system, released as `Core Library` 2.0.

* Yap, Du and Yu are supported by NSF Grants CCF-043836, CCF-0728977 and CCF-0917093, and with partial support from Korea Institute of Advance Studies (KIAS). Brönnimann is supported by NSF Career Grant 0133599. Brönnimann, Pion, and Yap are supported by an Collaborative Action GENEPI grant at INRIA and an NSF International Collaboration Grant NSF-04-036, providing travel support for Du and Yu to INRIA.

K. Fukuda et al. (Eds.): ICMS 2010, LNCS 6327, pp. 121–141, 2010.

1 Introduction

Most algorithms involving numbers are designed in the Real RAM model of computation. In this model (e.g., [1,37]) real numbers can be directly manipulated, comparisons are error-free, and basic arithmetic operations are exact. But in actual implementations, real numbers are typically approximated by machine doubles and this leads to the ubiquitous numerical nonrobustness issues that plague applications in scientific and engineering applications. In Computational Geometry, these numerical errors are exacerbated by the presence of discrete geometric relations defined by numbers. The survey articles [24,40] give an overview of nonrobustness issues in a geometric setting.

Now suppose P is a `C++` program using only standard libraries. When compiled, it suffers the expected nonrobustness associated with numerical errors. Imagine a software library with the property that when it is included by P, the compiled program (magically) runs like a real RAM program because all numerical quantities[1] behave like true real numbers. Such a library would be a boon towards eliminating numerical nonrobustness. The `Core Library` [22] was designed to approximate this dream: the program P only needs to insert the following two directives:

```
#define CORE_LEVEL 3
#include "CORE.h"                    (1)
```

Our library (`CORE.h`) will re-interpret the standard number type `double` as an `Expr` object (a directed acyclic graph representing a numerical expression). Indeed, by changing the `CORE_LEVEL` to 1 or 2 in (1), the program P can be compiled into other "accuracy levels", corresponding to machine precision (Level 1) or arbitrary multiprecision (Level 2). Although Levels 1 and 2 fall short of a Real RAM, the ability for a single program P to compile into different accuracy levels has interesting applications in the debug-exploration-release cycle of program development [44]. The purpose of this paper is to present the rationale and design of `Core Library` 2.0 (or `Core 2`). Towards this end, it will be compared to our original design, which refers to `Core Library` 1.7 (or `Core 1`). Thus "old system/design" refers to `Core 1` while "new system/design" refers to `Core 2`.

§1. On implementing a Real RAM. How do we implement a Real RAM? This dream in its full generality is impossible for two fundamental reasons. First, real numbers are uncountably many while any implementation is no more powerful than Turing machines which can only access countably many reals. The second difficulty is the general impossibility of deciding zeros (equivalently, making exact comparisons) [45]. The largest class beyond algebraic zeros for which zero is decidable are the elementary constants of Richardson [38,39,45]; this result depends on the truth of Schanuel's conjecture. What is possible, however, is to provide a Real RAM for interesting subsets of the reals. If program P uses only the rational operations $(\pm, \times, \div)$ then such a library could be a BigRational

[1] We are exploiting the ability of `C++` to overload operators. Otherwise, we can use some preprocessor.

number package; such a solution may have efficiency issues (e.g., [47]). If P also uses the square-root operation, then no off-the-shelf library will do; our precursor to `Core Library` [24] was designed to fill this gap. Since many basic problems in the algorithms literature involve at most irrationalities of the square-root kind, such a library is already quite useful. The natural goal of supporting all real algebraic numbers was first attained in `Core Library` 1.6 [44]. The other library that supports exact comparisons with algebraic numbers represented by floating-point approximations is `LEDA` [27,28]. Although our algebraic number subsystem is central to our library, it is not discussed in this paper since it is not the focus of our redesign effort. Interested readers are referred to [40] which describes how we achieve Real RAM capabilities efficiently for this subsystem, using constructive root bounds and filter techniques.

Another major library that is premised on exact comparison is `CGAL` [16]. Although `CGAL` does not have its own engine for general exact algebraic computation, its generic programming design supports number kernels such as the `Core Library`. Thus `Core Library` is bundled with `CGAL`, and commercially distributed by `Geometry Factory`. In the last decade, such libraries have demonstrated that the exact comparison approach is a practical means for eliminating nonrobustness in many applications.

The computation of our algebraic program P could, in principle, be carried out by computer algebra systems (CAS). Why is there a need for something like `Core Library`? First of all, if we may use a retail business analogy, many CAS systems adopt the "department store" approach to providing services while `Core Library` takes the "boutique" approach: our main service is a number type `Expr` that allows the simulation of a Real RAM. Our system is aimed at geometric applications that have salient differences from typical CAS applications. CAS are often used for one-of-a-kind computation which might be very difficult. These computations seeks to elucidate the algebraic properties of numbers while geometric applications [11,20] are interested in their analytic properties [45]. Inputs for geometric algorithms have some combinatorial size parameter n that can be moderately large. The algebraic aspects of its computation are normally encapsulated in a handful of algebraic predicates $Q(\mathbf{x})$ (e.g., orientation predicate) or algebraic expressions $E(\mathbf{x})$ (e.g., distance between two points) where $\mathbf{x} = (x_1, \ldots, x_k)$ represents the input. Evaluating $Q(\mathbf{x})$ or $E(\mathbf{x})$ is easy from the CAS viewpoint, but we must repeat this evaluation many times (as a function that grows with n). See [45,46] for more discussion.

§2. Exact Numerical Computation. There are four ingredients in our real RAM implementation:

(a) certified approximation of basic real functions (e.g., [4]),
(b) the theory of constructive zero bounds [6,29],
(c) a precision-driven evaluation mechanism [24], and
(d) filter mechanism [5,17].

The first two ingredients are essential for any Real RAM implementation; the last two ingredients are key to making the system efficient and practical. The certified approximations in (a) are ultimately dependent on interval techniques [33]. The

constructive zero bound in (b) is a systematic way to compute a bound $B(E)$ for a numerical expression E such that if E is defined and non-zero, then $|E| > B(E)$. Using this, we are able to do exact comparisons. We can view (c) as a pro-active kind of lazy evaluation – this is expanded below in Section 3.3. Finally, a simplified view of "filters" in (d) is to regard them as certified machine arithmetic. Using them we can cheaply perform exact comparisons in the majority of input instances, despite the fact that exact comparisons are very difficult in the worst case. This form of computation is[2] characterized as **Exact Numeric Computation** (ENC) in [45,46]. Computer algebra textbooks (e.g., [9]) list several alternatives for computing with algebraic numbers; to this list, we may now add the ENC approach.

There are many libraries (e.g., [21,31,32,43]) for arbitrary precision real computation. But lacking the critical ingredient (b), they cannot support exact comparison. As substitute for exact comparison, they use "ϵ-comparison" that compares numbers up to any desired $\epsilon > 0$ accuracy. Brattka and Hertling [3] provides a theoretical study of Real RAM with ϵ-comparisons. Numerical analysts also use this ϵ-accuracy approach. In this paper, we will need to integrate an exact subsystem for algebraic numbers with a new ϵ-accurate part for transcendental numbers.

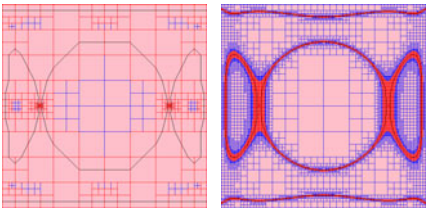


Fig. 1. Isotopic Approximation of curve $\sin_3(x^2) - \cos_3(y^2) = 0$ with `Core 2`

As an illustration of ENC applications, Figure 1 shows the curve $f(x, y) = \sin_3(x^2) - \cos_3(y^2) = 0$ approximated by `Core 2`, using a recent algorithm [25]. Here $\sin_n, \cos_n$ means we use the first n terms of their Taylor expansions. Our computation in the left figure stops once the isotopy-type is determined; in the right figure, we continue to a user-specified Hausdorff distance. Until recently, most exact computation on algebraic curves and surfaces are based on strong algebraic techniques such as resultant computation (e.g., [2]). In ENC, our main techniques are evaluation and domain subdivision (such subdivision boxes are seen in Figure 1). Superficially, this resembles the traditional numerical approaches, but ENC can provide the topological guarantees [25,36] that are normally only associated with algebraic algorithms. ENC algorithms have many advantages: adaptive complexity, relatively easy to implement, and locality (i.e., we can restrict computational effort to a local region, as in Figure 1).

§3. Goals of this Paper. There are three main motivations for the present redesign effort. The first is the desire to incorporate transcendental functions

[2] Also known as Exact Geometric Computation (EGC) in the context of geometric applications.

in our expressions. Many computations need transcendental constants ($\pi, e, \ln 2$, etc) or transcendental functions ($\sin x, \exp x, \ln x$, etc.). For instance, the problem of shortest path amidst planar disc obstacles [8], or motion planning in robotics involving holonomic or dynamic constraints or helical motion are all transcendental problems.In molecular simulations where we compute Coulombic forces, we need the error function erf, which is an instance of hypergeometric functions [14,15]. But we no longer guarantee the sign of such expressions.

The second motivation is to make the `Expr` class more flexible, extensible and modular. Although these are standard concerns of software engineering, we will discuss their special manifestations in an ENC software. There are many opportunities to introduce specialized operators into `Expr`, and we would like to introduce mechanisms to support this. Invisible to users, the evaluation of expressions relies on two critical functions: **filters** [5,7,17] and **zero bounds** [6,35]. In `Core 1`, both functionalities are integrated into the `Expr` class, making them hard to maintain and extend.

The third motivation is the perpetual quest for improved efficiency. There are two major sources of inefficiency that we address. The centerpiece of any ENC library is a poly-algorithm[3] to evaluate a numerical expression [24]. First, we re-examine this evaluation poly-algorithm. The optimal design of this poly-algorithm is far from understood, but we will see much room for improvement. The other efficiency issue arises in the numerical engine that delivers high precision approximations. Intuitively, this engine is a BigFloat number system combined with interval arithmetic. We will see that it plays two distinct roles but these roles are conflated in `Core 1`.

Lastly, our redesign must preserve the simple numerical API of `Core Library` as illustrated by (1), and is thus backward compatible with `Core 1`.

§4. Overview. Section 2 reviews the original design of `Core Library` and discusses the issues. Sections 3 and 4 present (resp.) the new design of the main `C++` classes for expressions and bigFloats. Section 5 describes new facilities to make `Expr` extensible. We conclude in Section 6. Many topics in this paper appear in greater detail in the Ph.D. thesis [13] of one of the authors. The source code for all experiments reported here are found in the subdirectory `progs/core2paper`, found in our open source (QPL license) `Core 2` distribution [10]. Experiments are done on an Intel Core Duo 2.4 GHz CPU with 2 GB of memory. The OS is Cygwin Platform 1.5 and compiler is `g++-3.4.4`.

2 Review of `Core Library`, Version 1

The `Core Library` features an object-oriented design, implemented in `C++`. A basic goal of the `Core Library` is to make ENC techniques transparent and easily accessible to (non-specialist) programmers through a simple numerical[4]

[3] By a "poly-algorithm", we mean a suite of complementary algorithms that work together to solve a specific problem.

[4] API stands for "Application Programmers Interface".

API (illustrated by (1)). User convenience is high priority because we view `Core Library` as a tool for experimentation and rapid prototyping.

There are three main subsystems in `Core 1`: the expression class (`Expr`), the real number class (`Real`) and the big float number class (`BigFloat`). These are number classes, built over standard big number classes (`BigInt`, `BigRat`) which are wrappers around corresponding types from `GNU`'s multiprecision package `GMP`. The `Expr` class provides the critical functionalities of ENC. In theory, `Expr` is the only number type users need, but experienced users can also access the underlying number classes directly (with `CORE_LEVEL` set to 4 in (1)). An instance of `Expr` is a directed acyclic graph (DAG) representing a numerical constant constructed from arbitrary real algebraic number constants. In the following, we raise some issues in the old design of the `Expr` and `BigFloat` classes.

- Some critical facilities in `Expr` should be modularized and made extensible. Specifically, the filter and root bound facilities have grown considerably over the course of development and are now hard to maintain, debug, or extend.
- The main evaluation algorithm (the "poly-algorithm" in the introduction) of `Expr` has three co-recursive subroutines. The old design does not separate their roles clearly, and this can lead to costly unnecessary computations.
- `Core 1` supports only algebraic expressions. An overhaul of the entire design is needed to add support for non-algebraic expressions.
- Currently, users cannot easily add new operators to `Expr`. E.g., it is useful to add diamond operator [6,41], product, summation (see below), etc.

Next consider the `BigFloat` class. It is used by `Expr` to approximate real numbers, and is the workhorse for the library. It is implemented on top of `BigInt` from `GMP`. The old `BigFloat` is represented by a triple $\langle m, err, e\rangle$ of integers, representing the interval $[(m - err)B^e, (m + err)B^e]$ where $B = 2^{14}$ is the base. We say the bigfloat is **normalized** when $err < B$ and **exact** when $err = 0$. The following issues arise:

- The above representation of `BigFloat` has performance penalty as we must do frequent error normalization. Some applications do not need to maintain error. E.g., in self-correcting Newton-type iterations, it is not only wasteful but may fail to converge unless we zero out the error by calling `makeExact`. Users can manually call `makeExact` but this process is error-prone.
- Our `BigFloat` assumes that for exact bigfloats, the ring operations $(+, -, \times)$ are computed exactly. This is important for ENC but we see situations below where this is undesirable and the IEEE model of round-off is preferable.
- The old `BigFloat` supports only $\{+, -, *, /, \sqrt[2]{\ }\}$. For `Expr` to support transcendental functions such as exp, sin, etc., we need their `BigFloat` analogues (recall ingredient (a) in ¶2). This implementation is a major effort, and the correct rounding for transcendental functions is quite non-trivial [12,30].

To bring out these performance penalties, we compare our old `BigFloat` implementation of `sqrt` based on Newton iteration against `MPFR` [18]: `Core 1` was 25 times slower as seen in Figure 8. The `MPFR` package satisfies all three criteria

above. A key feature of MPFR is its support of the IEEE rounding modes (the "R" in MPFR refers to rounding). Hence a critical decision of Core 2 was to capitalize on MPFR.

3 Redesign of the Expr Package

We first focus on expressions. The goal is to increase modularity and extensibility of expression nodes, and also to improve efficiency.

3.1 Incorporation of Transcendental Nodes

What is involved in extending expressions to transcendental operators? In Core 1, we classify nodes in Expr into rational or irrational ones as such information is critical for root bound computation. We now classify them into integer, dyadic, rational, algebraic, and transcendental. A node is transcendental if any of its descendants has a transcendental operator (e.g., a leaf for $\pi = 3.1415\ldots$, or a unary node such as $\sin(\cdot)$). This refined classification of nodes is exploited in root bounds. There is a natural total ordering on these types, and the type of a node is the maximum of the types in descendant nodes. As transcendental expressions do not have root bounds, we introduce a user-definable global value called **escape bound** to serve as their common root bound. Another bound called **cutoff bound** is used for a different purpose; both are explained below.

3.2 New Template-Based Design of ExprRep

The Expr class in Core 2 is templated, unlike in Core 1. It remains only a thin wrapper around a "rep class" called ExprRep, which is our focus here.

§5. ExprRep and ExprRepT. The filter and root bound facilities were embedded in the old ExprRep class. We now factor them out into two functional modules: Filter and Rootbd. The Real class (see Section 2), which was already an independent module, is now viewed as an instance of an abstract number module called Kernel. The role of Kernel is to provide approximate real values. We introduce the templated classes ExprT and ExprRepT, parametrized by these three modules:

```
template <typename Rootbd,
          typename Filter, typename Kernel>
   class ExprT;

template <typename Rootbd,
          typename Filter, typename Kernel>
   class ExprRepT;
```

Now, `Expr` and `ExprRep` are just typedefs:

```
typedef ExprT<BfmssRootBd<BigFloat2>,
          BfsFilter, BigFloat2> Expr;
typedef ExprRepT<RootBd,
          Filter, Kernel> ExprRep;
```

The actual template arguments for `Rootbd`, `Filter`, and `Kernel` for `Expr` are passed to `ExprRep`, `ExprT`, and `ExprRepT`. The benefit of this new design is that now we can replace `Filter`, `Rootbd` or `Kernel` at the highest level without any changes in `ExprRep`, `ExprT` or `ExprRepT`. We see here that the default `Expr` class in `Core 2` uses the k-ary BFMSS root bounds [6, 35] and the new `BigFloat2` kernel (below). But users are free to plug in other modules. E.g., one could substitute a better filter and root bound for division-free expressions. This design of `Expr` follows the "delegation pattern" in Object-Oriented Programming [42]: the behavior of `Expr` is delegated to other objects (filters, etc).

§6. `ExprRepT` class hierarchy. The class `ExprRepT` defines abstract structures and operations which are overridden by its subclasses. This hierarchy of subclasses is shown in 2.

There are four directly derived classes, corresponding to the arities of the operator at the root of the expressions: constants, unary, binary, and anary operators. Each of them has further derived classes – for instance, the binary operator class is further derived into three subclasses corresponding to the four arithmetic operators (`AddSubRepT`, `MulRepT`, `DivRepT`). The class `AddSubRepT` combines addition and subtraction as these two operations share basically the same code. The introduction of anary operators is new. An anary operator is one without a fixed arity, such as summation $\sum_{i=1}^{n} t_i$ and product $\prod_{i=1}^{n} t_i$. Another is the diamond operator $\diamond(a_0, \ldots, a_n, i)$ to extract the ith real root of a polynomial $p(x) = \sum_{i=0}^{n} a_i x^i$ [6] where a_i's are expressions. Below we show the usefulness of these extensions.

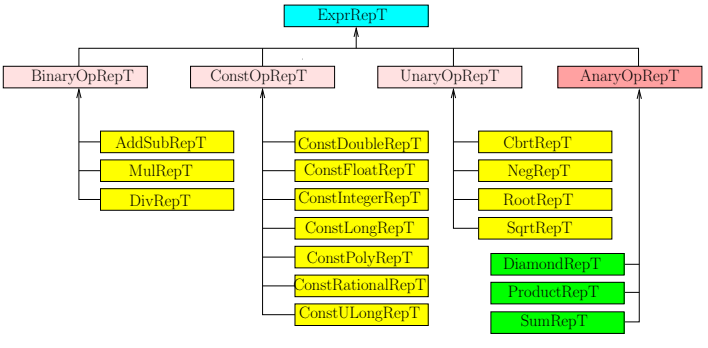
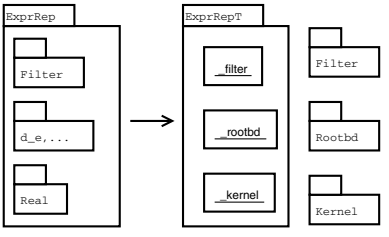


Fig. 2. `ExprRepT` Class hierarchy



(a)

Field name	Size in bytes
Dynamic type information	4
Reference counter	4
Operands : Node * [arity]	4×arity
Filter _filter	16
Kernel *_kernel	4
Rootbd *_rootbd	4
Cache *_cache	4
int _numType	4

(b)

Fig. 3. (a) Comparing `ExprRep` and `ExprRepT`. (b) Layout in 32-bit architecture.

§7. Memory layout of `ExprRepT`. Size of expression nodes can become an issue (see Section 5). Our design of `ExprRepT` optimizes the use of space (see Figure 3(a)). Each `ExprRepT` node has three fields `_filter, _rootbd` and `_kernel`. Here, `_filter` is stored directly in the node, while `_rootbd` and `_kernel` are allocated on demand, and only pointers to them are stored in the node. This is because the filter computation will always be done, but root bound and high precision approximations (from `_kernel`) may not be needed. No memory will be allocated when they are not needed. The memory layout of `ExprRepT` is shown in Figure 3(b). E.g., a binary `ExprRepT` node uses a total of 48 bytes on a 32-bit architecture. The field `_cache` is added to cache important small but potentially expensive information such as `sign, uMSB, lMSB`. The field `_numType` is used for node classification.

§8. Some Timings. We provide two performance indicators after the above redesign. In Figure 4(a), we measure the time to decide the sign of determinants, with and without (w/o) the filter facility, in `Core 1` and `Core 2`. The format '$N \times d \times b$' in the first column indicates the number N of matrices, the dimension d of each matrix and the bit length b of each matrix entry (entries are rationals). Interestingly, for small determinants, the filtered version of `Core 1` is almost twice as fast. All times are in microseconds.

MATRIX	Core 1.7 Time with (w/o) filter	Core 2.0 Time with (w/o) filter	Speedup
1000x3x10	9 (621)	19 (232)	0.5 (2.7)
1000x4x10	26 (1666)	43 (530)	0.6 (3.1)
500x5x10	449 (1728)	204 (488)	2.2 (3.5)
500x6x10	1889 (3493)	597 (894)	3.2 (3.9)
500x7x10	4443 (6597)	1426 (1580)	3.1 (4.2)
500x8x10	8100 (11367)	2658 (2820)	3.0 (4.0)

(a)

bit length L	`Core 1`	`Core 2`	Speedup
1000	0.82	0.59	1.4
2000	6.94	1.67	4.2
8000	91.9	11.63	7.9
10000	91.91	30.75	3.0

(b)

Fig. 4. (a) Timing filter facility (b) Timing root Bound facility

In Figure 4(b), we test the new root bound facility by performing the comparison $\sqrt{x}+\sqrt{y} \;:\; \sqrt{x+y+2\sqrt{xy}}$ where x, y are b-bit rational numbers. As this expression is identically zero, filters do not help and root bounds will always be reached.

3.3 Improved Evaluation Algorithm

Since the evaluation algorithm is the centerpiece of an ENC library, it is crucial to tune its performance.

§9. Algorithms for `sign()`, `uMSB()` and `lMSB()`. `Core 1` has two main evaluation subroutines, `computeApprox()` and `computeExactSign()` (see [24]). The former computes an approximation of the current node to some given (composite) precision bound. The latter computes the sign, upper and lower bounds on the magnitude of the current node. The sign of an expression node is critical in many places. E.g., the division operator must check the sign of the right child to detect divisions by zero. But to get the sign of an expression E, we may need to estimate upper (E^+) or lower (E^-) bounds on the magnitude of the expression. These three values are maintained in `Expr` as `sign()`, `uMSB()` and `lMSB()`. In `Core 1`, `computeExactSign()` computes them simultaneously using the rules in Table 5.

E	Case	$E.sgn()$	E^+	E^-
Constant x		$sign(x)$	$\lceil \log_2 x \rceil$	$\lfloor \log_2 x \rfloor$
$E_1 \pm E_2$	if $E_1.sgn() = 0$	$\pm E_2.sgn()$	E_2^+	E_2^-
	if $E_2.sgn() = 0$	$E_1.sgn()$	E_1^+	E_1^-
	if $E_1.sgn() = \pm E_2.sgn()$	$E_1.sgn()$	$\max\{E_1^+, E_2^+\}+1$	$\max\{E_1^-, E_2^-\}$
	if $E_1.sgn() \neq \pm E_2.sgn()$ and $E_1^- > E_2^+$	$E_1.sgn()$	$\max\{E_1^+, E_2^+\}$	$E_1^- - 1$
	if $E_1.sgn() \neq \pm E_2.sgn()$ and $E_1^+ < E_2^-$	$\pm E_2.sgn()$	$\max\{E_1^+, E_2^+\}$	$E_2^- - 1$
	otherwise	unknown	$\max\{E_1^+, E_2^+\}$	unknown
$E_1 \times E_2$		$E_1.sgn() * E_2.sgn()$	$E_1^+ + E_2^+$	$E_1^- + E_2^-$
$E_1 \div E_2$		$E_1.sgn() * E_2.sgn()$	$E_1^+ - E_2^-$	$E_1^- - E_2^+$
$\sqrt[k]{E_1}$		$E_1.sgn()$	$\lceil E_1^+/k \rceil$	$\lfloor E_1^-/k \rfloor$

Fig. 5. Recursive rules for computing `sign`, `uMSB`, `lMSB`

There are two "unknown" entries in Table 5. In these cases, `computeApprox()` will loop until the sign is determined, or up to the root bound. To compute such information, we recursively compute sign and other information over the children of this node, whether needed or not. This can be unnecessarily expensive. In `Core 2`, we split the 2 routines into five co-recursive routines in `ExprRepT`: `get_sign()`, `get_uMSB()`, `get_lMSB()`, `refine()` and `get_rootBd()`. In `Expr` the corresponding methods `sign()`, `uMSB()` and `lMSB()` simply calls `get_sign()`, etc. Depending on the operator at a node, these co-routines can better decide which information from a child is really necessary. The structure of these algorithms are quite similar, so we use `get_sign()` as an example:

SIGN EVALUATION ALGORITHM, `get_sign()`:
1. Ask the filter if it knows the sign;
2. Else if the cache exists, ask if sign is cached;
   Note: the cache may contain non-sign information
3. Else if the approximation (`_kernel`) exists, ask if it can give the sign;
4. Else if the virtual function `compute_sign()` returns `true`, return `sgn()` (sign is now in the cache);
5. Else call `refine()` (presented next) to get the sign.

Thus it is seen that, for efficiency, we use five levels of computation in `get_sign()`: filter, cache, kernel, recursive rules (called `compute_sign()`), and `refine()`. Note that we do not put the cache at the first level. We do not even cache the `sign`, `uMSB` and `lMSB` information when the filter succeeds because a `Cache` structure is large and we try to avoid costly memory allocation.

The object oriented paradigm used by the above design is called the "template method pattern" [19, p. 325]: define the skeleton of an algorithm in terms of abstract operations which is to be overridden by subclasses to provide concrete behavior. In the derived classes of `ExprRepT`, it is sufficient to just override the virtual function `compute_sign()` when appropriate. For example, `MulRepT` may override the default `compute_sign()` function as follows:

```
1 virtual bool compute_sign() {
2   sign() = first->get_sign() * second->get_sign();
3   return true; }
```

§10. Algorithm for `refine()`. As seen in the `get_sign()` algorithm, if the first four levels of computation fail, the ultimate fall-back for obtaining sign (and also for lower bounding magnitude) is the `refine()` algorithm. We outline this key algorithm to obtain sign via refinement:

1. If the node is transcendental, get the global escape bound. Otherwise, compute the constructive root bound.
2. Take the minimum of the bound from step 1 and the global cutoff bound.
3. Compute an initial precision. If an approximation exists, use its precision as the initial precision. Otherwise use 52 bits instead which is the relative precision that a floating-point filter can provide.
4. Initialize the current precision to the initial precision. Then enter a for-loop that doubles the current precision each time, until the current precision exceeds twice the bound computed in step 2.
5. In each iteration, call `a_approx()` (see below) to approximate the current node to an absolute error less than the current precision. If this approximation suffices to give a sign, return the sign immediately (skip the next step).
6. Upon loop termination, set the current node to be zero.
7. Check if the termination was caused by reaching the escape bound or cutoff bound. If so, append **zero assertion** to a diagnostic file in the current directory. This assertion says that "the current node is zero".

§11. Conditional Correctness. The cutoff bound in the above `refine()` algorithm is a global variable that is set to `CORE_INFTY` by default. While escape bound affects only transcendental nodes, the cutoff bound sets an upper bound on the precision in `refine()` for all nodes. Thus it may override computed zero bounds and escape bounds. During program development, users may find it useful to set a small cutoff bound using `set_cut_off_bound()`. Thus, *our computation is correct, conditioned on the truth of all the zero assertions in the diagnostic file.*

§12. Computing Degree Bounds. In the `refine()` algorithm above, the first step is to compute a constructive root bound. Most constructive root bounds need an upper bound on the degree of an algebraic expression [40]. For radical expressions, a simple upper bound is obtained as the product of all the degrees of the radical nodes (a radical node $\sqrt[k]{E}$ has degree k). A simple recursive rule can obtain the degree bound of E from the degree bounds of its children (e.g., [23, Table 2.1]). But this bound may not be tight when the children share nodes. The only sure method is to traverse the entire DAG to compute this bound. To support this traversal, in `Core 1` we store an extra flag `visited` with each `ExprRep`. Two recursive traversals of the DAG are needed to set and to clear these flags, while computing the degree bound. To improve efficiency, we now use the `std::map` data structure in `STL` to compute the degree bound: we first create a map M and initialize the degree bound D to 1. We now traverse the DAG, and for each radical node u, if its address does not appear in M, we multiply its degree to the cumulative degree bound D and save its address in M. At the end we just discard the map M. This approach requires only one traversal of the DAG.

3.4 Improved Propagation of Precision

An essential feature of precision-driven evaluation is the need to propagate precision bounds [40]. Precision propagation can be illustrated as follows: if we want to evaluate an expression $z = x+y$ to p-bits of absolute precision, then we might first evaluate x and y to $(p+1)$-bits of absolute precision. Thus, we "propagate" the precision p at z to precision $p+1$ at the children of z. This propagation is correct provided x and y have the same sign (otherwise, $p+1$ bits might not suffice because of cancellation). In general, we must propagate precision from the root to the leaves of an expression. In `Core 1`, we use a pair $[a, r]$ of real numbers that we call "composite precision" bounds. If $x, \widetilde{x} \in \mathbb{R}$, then we say $\widetilde{x}$ is an $[a, r]$**-approximation** of x (written, "$\widetilde{x} \approx x[a, r]$") if $|\widetilde{x} - x| \leq 2^{-a}$ or $|\widetilde{x} - x| \leq |x|2^{-r}$. If we set $a = \infty$ (resp., $r = \infty$), then $\widetilde{x}$ becomes a standard **relative** r**-bit** (resp., an **absolute** a**-bit**) **approximation** of x. It is known that a relative 1-bit approximation would determine the sign of x; so relative approximation is generally infeasible without zero bounds. The propagation of composite bounds is tricky, and various small constants crop in the code, making the logic hard to understand and maintain (see [34]). Our redesign offers

a simpler and more intuitive solution in which we propagate either absolute or relative precision, not their combination.

§13. Algorithms for `r_approx()` and `a_approx()`. `Core 1` has one subroutine `computeApprox()` to compute approximations; we split it into two subroutines `a_approx()` and `r_approx()`, for absolute and relative approximations (respectively). Above, we saw that `refine` calls `a_approx()`. There are two improvements over `Core 1`: first, propagating either absolute or relative precision is simpler and can avoid unnecessary precision conversions. Second, the new algorithms do not always compute the sign (which can be very expensive) before approximation.

§14. Overcoming inefficiencies of Computational Rings. Another issue relates to the role **computational rings** in ENC (see §16 in [45]). This is a countable set $\mathbb{F} \subseteq \mathbb{R}$ that can effectively substitute for the uncountable set of real numbers. To achieve exact computation, $\mathbb{F}$ needs a minimal amount of algebraic structures [45]. We axiomatize $\mathbb{F}$ to be a subring of $\mathbb{R}$ that is dense in $\mathbb{R}$, with $\mathbb{Z}$ as a subring. Furthermore, the ring operations together with division by 2, and comparisons are effective over $\mathbb{F}$. BigFloats with exact ring operations is a model of $\mathbb{F}$, but IEEE bigFloats is not. For computations that do not need exactness, the use of such rings may incur performance penalty. To demonstrate this, suppose we want to compute $\sqrt{2} \cdot \sqrt{3}$ to relative p-bits of precision. We describe two methods for this. In Method 1, we approximate $\sqrt{2}$ and $\sqrt{3}$ to relative $(p+2)$-bits, then perform the exact multiplication of these values. In Method 2, we proceed as in Method 1 except that the final multiplication is performed to relative $(p+1)$-bits. The timings (in microseconds) are shown in Figure 6. We use loops to repeat the experiment since the time for single runs is short. It is seen that Method 2 can be much more efficient; this lesson is incorporated into our refinement algorithm.

Precision	Loops	Method 1	Method 2	Speedup
10	1000000	345	191	45%
100	100000	60	46	23%
1000	10000	72	71	1%
10000	1000	267	219	18%
100000	100	859	760	12%

Fig. 6. Timing for computing $\sqrt{2} \cdot \sqrt{3}$ w/ and w/o exact multiplication

4 Redesign of the `BigFloat` System

The `BigFloat` system is the "engine" for `Expr`, and `Core 1` implements our own `BigFloat`. In Section 2, we discussed several good reasons to leverage our system on `MPFR`, an efficient library under active development for bigFloat numbers with directed rounding. Our original `BigFloat` plays two roles: to implement a computational ring [45] (see section section 3.4), and to provide arbitrary

precision interval arithmetic [33]. Computational ring properties are needed in exact geometry: e.g., to compute implicit curve intersections reliably, we can evaluate polynomials with exact `BigFloat` coefficients, at exact `BigFloat` values, using exact ring operations. Interval arithmetic is necessary to provide certified approximations. For efficiency, `Core 2` splits the original BigFloat class into two new classes: (1) A computational ring class, still called `BigFloat`. (2) An interval arithmetic class called `BigFloat2`, with each interval represented by two `MPFR` bigFloats. This explains[5] the "2" in its name.

4.1 The `BigFloat` Class as Base Real Ring

The new class `BigFloat` is based on the type `mpfr_t` provided by `MPFR`. `MPFR` follows the IEEE standard for (arbitrary precision) arithmetic. The results of arithmetic operations are rounded according to user-specified output precision and rounding mode. If the result can be exactly represented, then `MPFR` always outputs this result. E.g., a call of `mpfr_mul(c, a, b, GMP_RNDN)` will compute the product of a and b, rounding to nearest `BigFloat`, and put the result into c. The user must explicitly set the precision (number of bits in the mantissa) of c before calling `mpfr_mul()`. To implement the computational ring `BigFloat`, we just need to automatically estimate this precision. E.g., we can use the following:

Lemma 1. *Let $f_i = (-1)^{s_i} \cdot m_i \cdot 2^{e_i}$ (for $i = 1, 2$) be two bigFloats in `MPFR`, where $1/2 \leq m_i < 1$ and the precision of m_i is p_i. To guarantee that all bits in the mantissa of the sum $f = f_1 \pm f_2$ is correct, it suffices to set the precision of f to*

$$\begin{cases} 1 + \max\{p_1 + \delta, p_2\} & \text{if } \delta \geq 0 \\ 1 + \max\{p_1, p_2 - \delta\} & \text{if } \delta < 0 \end{cases}$$

where $\delta = (e_1 - p_1) - (e_2 - p_2)$. Similarly, for multiplication, it suffices to set the precision of f to be $p_1 + p_2$ in computing $f = f_1 \cdot f_2$.

See [13] for a proof. While this lemma is convenient to use, it may over-estimate the needed precision. In binary notation, think of the true precision of c as the minimum number of bits to store the mantissa of c. Trailing zeros in the mantissa contributes to over-estimation. To avoid this, we provide a function named `mpfr_remove_trailing_zeros()` whose role is to remove the trailing zeros. In an efficiency tradeoff, it only removes zeros by chunks (chunks are determined by `MPFR`'s representation). To understand the effect of overestimation, we conduct an experiment in which we compute the factorial $F = \prod_{i=1}^{n} i$ using two methods: In Method 1, we initialize $F = 1$ and build up the product in a for-loop with $i = 2, 3, \ldots, n$. In the i-th loop, we increase the precision of F using Lemma 1, then call `MPFR` to multiply F and i, storing the result back into F. In Method 2, we do the same for-loop except that we call `mpfr_remove_trailing_zeros()` on F after each multiplication in the loop. Instead of F, we can repeat the experiment with the arithmetic sum $S = \sum_{i=1}^{n} i$. The speedup for the second method over the first method is shown in Figure 7 (time in microseconds, precision in bits).

[5] Happily, it also coincides with the "2" in the new version number of `Core Library`.

n	$F = \prod_{i=1}^{n} i$ trailing zeros (prec/msec)	$F = \prod_{i=1}^{n} i$ no zeros (prec/msec)	$S = \sum_{i=1}^{n} i$ trailing zeros (prec/msec)	$S = \sum_{i=1}^{n} i$ no zeros (prec/msec)
10^2	575/0	436/0	102/0	31/0
10^3	8979/0	7539/0	1002/0	31/0
10^4	123619/62	108471/47	10002/15	31/16
10^5	1568931/9219	1416270/8891	100002/437	31/110
10^6	timeout	timeout	1000002/57313	63/1078

Fig. 7. Timing for computing F and S w/ and w/o removing trailing zeros

§15. Benchmarks of the redesigned `BigFloat`. By adopting `MPFR`, our `BigFloat` class gains many new functions such as `cbrt()` (cube root) and the elementary functions (`sin()`, `log()`, etc). The performance of the `BigFloat` is also greatly improved. We compared the performance of `Core 1` and `Core 2` on `sqrt()` using the following experiment: compute $\sqrt{i}$ for $i = 2, \ldots, 100$ with precision p. The timing in Figure 8 show that `Core 2` is about 25 times faster, thanks purely to `MPFR`.

Precision	`Core 1`	`Core 2`	Speedup
1000	25	1	25
10000	716	32	22
100000	33270	1299	25

Fig. 8. Timing comparisons for `sqrt()`

4.2 The Class `BigFloat2`

`BigFloat2` is the second class split off from the original `BigFloat`. An instance of `BigFloat2` is just an interval represented by a pair of bigFloat numbers. Call this the **endpoint representation** of intervals. Besides serving as numerical engine for `Expr`, the `BigFloat2` class is also useful for various ENC applications (e.g., in meshing algorithms of the kind producing Figure 1). In `Core 1`, we use the **centered representation** where an interval $[a, b]$ is represented by its center $c = (a + b)/2$ and an error bound $err = (b - a)/2$, with c a bigFloat number and err a machine long. In view of the limited precision in err, it is necessary to "normalize" the representation when it gets too large. This is one of the disadvantages of the centered representations. In the worst case, the endpoint representation can be less efficient than the centered representation by a factor of 2, both in speed and in storage. But this loss in efficiency is compensated by ease of implementation, and in sharper error bounds. It can also be beneficial in low precision computation. We note that van der Hoeven's `Mmxlib` [31] combines the advantages of both representations by switching from the endpoint representation to the centered representation when the precision exceeds some threshold value. Since our kernel class is a template parameter, we may experiment with such an interval class in the future.

5 Extending the Expr Class

We provide facilities for adding new operators to `Expr`. `Core 2` uses such facilities to implement the standard elementary functions. Future plans include extending elementary functions to all hypergeometric functions, following the analysis in [13,14]. We give two examples of how users can use these facilities for their own needs. We refer to Zilin Du's thesis [13] for more details about these facilities.

5.1 Summation Operation for Expr

When an `Expr` is very large, we not only lose efficiency (just to traverse the DAG) but we often run out of memory. Consider the following code to compute the harmonic series $H = \sum_{i=1}^{n} \frac{1}{i}$:

```
Expr harmonic(int n) {
  Expr H(0);
  for (int i=1; i<=n; ++i)
    H = H + Expr(1)/Expr(i);
  return H; }
```

This function builds a deep unbalanced DAG for large n. This can easily cause segmentation faults through stack overflow (column 2 in Figure 9). In ¶6, we said that `Core 2` supports a new class of **anary** (i.e., "without arity") nodes. In particular, we implemented the `summation` and `product` operators. Using `summation`, we can rewrite the harmonic function:

```
Expr term(int i) {
  return Expr(1)/Expr(i); }
Expr harmonic(int n) {
  return summation(term, 1, n); }
```

The improvements from this new implementation is shown in Figure 9. We can now compute the harmonic series for a much larger n, and achieve a speedup as well.

Similarly, the use of `product` operator leads to speed-ups.

5.2 Transcendental Constants π, e and All That

We present our *first* transcendental node π, which is a leaf node derived from `ConstRepT`:

n	Time w/o summation	Time w/ summation	Speedup
1000	24	7	3.4
10000	3931	67	58.6
100000	(segmentation fault)	752	N/A
1000000	(segmentation fault)	12260	N/A

Fig. 9. Timings for computing harmonic series $\sum_{i=1}^{n} \frac{1}{i}$ (in microseconds)

```
template <typename T>
class PiRepT: public ConstRepT<T> {
public:
  PiRepT() : ConstRepT<T>() {
    compute_filter();
    compute_numtype();
  }
  // functions to compute filter and number type
  void compute_filter() const {
    filter().set(
      3.14159265358979323846264338327950288F);
      /* value is not exact*/
  }
  void compute_numtype() const
  { _numType = NODE_NT_TRANSCENDENTAL; }

  // virtual functions for sign, uMSB, lMSB
  virtual bool compute_sign() const
  { sign() = 1; return true; }
  virtual bool compute_uMSB() const
  { uMSB() = 2; return true; }
  virtual bool compute_lMSB() const
  { lMSB() = 1; return true; }

  // virtual functions for r_approx, a_approx
  virtual void compute_r_approx(prec_t prec) const
  { kernel().pi(prec); }
  virtual bool compute_a_approx(prec_t prec) const
  { kernel().pi(abs2rel(prec)); }
};
```

Now the new π expression is given by:

```
template <typename T>
ExprT<T> pi()
{ return new PiRepT<T>(); }
```

Note how easy it is to do this extension — it could equally be used to introduce e or $\ln 2$ or any constant, provided the kernel class knows how to approximate it. Such constants can now freely appear in an expression, and our precision-propagation mechanism can automatically approximate the expression to any desired absolute error bound. Figure 10 gives timings for π and other elementary functions after incorporation into `Expr`.

Precision (bits)	Expr.	`Core 2`	`Core 1`	Speedup
10,000	π	20	—	—
	$\sqrt{\pi}$	90	—	—
	e^2	80	—	—
	sin(0.7)	50	1240	25
	cos(0.7)	50	1230	25
	tan(0.7)	110	2490	23
100,000	π	710	—	—
	$\sqrt{\pi}$	2200	—	—
	e^2	830	—	—
	sin(0.7)	4780	—	—
	cos(0.7)	4650	—	—
	tan(0.7)	9450	—	—

Fig. 10. Transcendental Constants and Functions

6 Conclusion

The goal of `Core Library` is to approximate the ideal real RAM. To support rapid prototyping of algorithmic ideas in geometry and algebra, ease of use and functionality is prized above sheer efficiency. With `Core 2`, we combine exact algebraic computation with transcendental functions. Our redesigned package is more modular, extensible, and flexible. We gained efficiency from the design and better evaluation algorithms. We adopt the highly efficient `MPFR` library to improve maintainability and to gain transcendental functions. Despite this overhaul, the original simple Core Numerical API is preserved.

In the transcendental aspects, we are just leveraging the speed of `MPFR` into our environment. What we give back is a new convenient way to access `MPFR`. In 2005, `MPFR` won the **Many Digits Competition** [26] in a field of 9 teams that included `Maple` and `Mathematica`. Their solutions are "hand-coded" in the sense that each algorithm is preceded by an error analysis to determine the needed precision, plus a hand-coded implementation of these error bounds. By incorporating `MPFR` into `Core 2`, we can now do these competition problems *automatically*: `Core 2` provides the automatic error analysis. Another "added value" that `Core Library` provides `MPFR` is access to a computational ring to support exact geometric computation. A future research is to understand and reduce any performance penalties of this automation.

A major open problem is to better understand the expression evaluation algorithms. There is no satisfactory theoretical basis for the optimal evaluation of such expressions (see some attempts in [43]). The second major problem is to provide constructive bounds for non-algebraic constants. Instead of escape bounds, we could use Richardson's algorithm to decide zero [38,39]. This is because all the constants in `Core 2` are elementary constants in the sense of Richardson. As Richardson suggested, this is a win-win situation because if our computation is ever wrong, we would have found a counter-example to Schanuel's conjecture. Unfortunately, current versions of Richardson's algorithm do not appear practical enough for general application.

`Core Library` represents a new breed of real number libraries to support exact numerical computation (ENC). It is made feasible through sophisticated built-in functionalities such as filters and constructive zero bounds. There remain ample opportunities for exploring the design space of constructing such libraries. Our `Core Library` offers one data point. Such libraries have many potential applications. Besides robust geometric algorithms, we can use them in geometric theorem proving, certifying numerical programs, and in mathematical explorations.

References

1. Aho, V., Hopcroft, J.E., Ullman, J.D.: The Design and Analysis of Computer Algorithms. Addison-Wesley, Reading (1974)
2. Berberich, E., Kerber, M., Sagraloff, M.: Exact geometric-topological analysis of algebraic surfaces. In: 24th ACM Symp. on Comp. Geometry, pp. 164–173 (2008)

3. Brattka, V., Hertling, P.: Feasible real random access machines. J. of Complexity 14(4), 490–526 (1998)
4. Brent, R.P.: Fast multiple-precision evaluation of elementary functions. J. ACM 23, 242–251 (1976)
5. Brönnimann, H., Burnikel, C., Pion, S.: Interval arithmetic yields efficient dynamic filters for computational geometry. Discrete Applied Mathematics 109(1-2), 25–47 (2001)
6. Burnikel, C., Funke, S., Mehlhorn, K., Schirra, S., Schmitt, S.: A separation bound for real algebraic expressions. Algorithmica 55(1), 14–28 (2009)
7. Burnikel, C., Funke, S., Seel, M.: Exact geometric computation using cascading. Int'l. J. Comput. Geometry and Appl. 11(3), 245–266 (2001)
8. Chang, E.-C., Choi, S.W., Kwon, D., Park, H., Yap, C.: Shortest paths for disc obstacles is computable. Int'l. J. Comput. Geometry and Appl. 16(5-6), 567–590 (2006); Gao, X.S., Michelucci, D.: Special Issue of IJCGA on Geometric Constraints
9. Cohen, H.: A Course in Computational Algebraic Number Theory. Springer, Heidelberg (1993)
10. Core Library homepage, since, Software download, source, documentation and links (1999), http://cs.nyu.edu/exact/core/
11. de Berg, M., van Kreveld, M., Overmars, M., Schwarzkopf, O.: Computational Geometry: Algorithms and Applications. Springer, Berlin (1997)
12. Defour, D., Hanrot, G., Lefévre, V., Muller, J.-M., Revol, N., Zimmermann, P.: Proposal for a standardization of mathematical function implementation in floating-point arithmetic. Numerical Algorithms 37(1-4), 367–375 (2004)
13. Du, Z.: Guaranteed Precision for Transcendental and Algebraic Computation made Easy. Ph.D. thesis, New York University, Department of Computer Science, Courant Institute (May 2006) http://cs.nyu.edu/exact/doc/
14. Du, Z., Eleftheriou, M., Moreira, J., Yap, C.: Hypergeometric functions in exact geometric computation. In: Brattka, V., Schoeder, M., Weihrauch, K. (eds.) Proc. 5th Workshop on Computability and Complexity in Analysis, Malaga, Spain, July 12-13. ENTCS, vol. 66(1), pp. 55–66 (2002), http://www.elsevier.nl/locate/entcs/volume66.html
15. Du, Z., Yap, C.: Absolute approximation of the general hypergeometric functions. In: il Pae, S., Park, H. (eds.) Proc. 7th Asian Symposium on Computer Mathematics (ASCM 2005), December 8-10, pp. 246–249. Korea Institute for Advanced Study, Seoul (2005)
16. Fabri, A., Giezeman, G.-J., Kettner, L., Schirra, S., Schoenherr, S.: The CGAL kernel: a basis for geometric computation. In: Lin, M.C., Manocha, D. (eds.) FCRC-WS 1996 and WACG 1996. LNCS, vol. 1148, pp. 191–202. Springer, Heidelberg (1996)
17. Fortune, S.J., van Wyk, C.J.: Static analysis yields efficient exact integer arithmetic for computational geometry. ACM Transactions on Graphics 15(3), 223–248 (1996)
18. Fousse, L., Hanrot, G., Lefèvre, V., Pélissier, P., Zimmermann, P.: Mpfr: A multiple-precision binary floating-point library with correct rounding. ACM Trans. Math. Softw. 33(2), 13 (2007)
19. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley Professional Computing Series. Addison-Wesley Publishing Company, New York (1995)
20. Goodman, J.E., O'Rourke, J. (eds.): Handbook of Discrete and Computational Geometry. CRC Press LLC, Boca Raton (1997)

21. Gowland, P., Lester, D.: A survey of exact arithmetic implementations. In: Blank, J., Brattka, V., Hertling, P. (eds.) CCA 2000. LNCS, vol. 2064, pp. 30–47. Springer, Heidelberg (2001)
22. Karamcheti, V., Li, C., Pechtchanski, I., Yap, C.: A Core library for robust numerical and geometric computation. In: 15th ACM Symp. Computational Geometry, pp. 351–359 (1999)
23. Li, C.: Exact Geometric Computation: Theory and Applications. Ph.D. thesis, New York University, Department of Computer Science, Courant Institute (January 2001), http://cs.nyu.edu/exact/doc/
24. Li, C., Pion, S., Yap, C.: Recent progress in Exact Geometric Computation. J. of Logic and Algebraic Programming 64(1), 85–111 (2004); Special issue on Practical Development of Exact Real Number Computation
25. Lin, L., Yap, C.: Adaptive isotopic approximation of nonsingular curves: the parametrizability and non-local isotopy approach. In: Proc. 25th ACM Symp. on Comp. Geometry, Aarhus, Denmark, June 8-10, pp. 351–360 (2009); Accepted for Special Issue of SoCG 2009 in DCG (2009)
26. Many digits friendly competition, The details of the competition, including final results (October 3-4, 2005), http://www.cs.ru.nl/~milad/manydigits/
27. Mehlhorn, K., Näher, S.: LEDA: a platform for combinatorial and geometric computing. Comm. of the ACM 38, 96–102 (1995)
28. Mehlhorn, K., Schirra, S.: Exact computation with leda real – theory and geometric applications. In: Alefeld, G., Rohn, J., Rump, S., Yamamoto, T. (eds.) Symbolic Algebraic Methods and Verification Methods, Vienna, pp. 163–172. Springer, Heidelberg (2001)
29. Mignotte, M.: Identification of algebraic numbers. J. Algorithms 3, 197–204 (1982)
30. Muller, J.-M.: Elementary Functions: Algorithms and Implementation. Birkhäuser, Boston (1997)
31. Müller, N.T.: The iRRAM: Exact arithmetic in C++. In: Blank, J., Brattka, V., Hertling, P. (eds.) CCA 2000. LNCS, vol. 2064, pp. 222–252. Springer, Heidelberg (2001)
32. Müller, N.T., Escardo, M., Zimmermann, P.: Guest editor's introduction: Practical development of exact real number computation. J. of Logic and Algebraic Programming 64(1), Special Issue (2004)
33. Neumaier, A.: Interval Methods for Systems of Equations. Cambridge University Press, Cambridge (1990)
34. Ouchi, K.: Real/Expr: Implementation of an Exact Computation Package. Master's thesis, New York University, Department of Computer Science, Courant Institute (January 1997), http://cs.nyu.edu/exact/doc/
35. Pion, S., Yap, C.: Constructive root bound method for k-ary rational input numbers. Theor. Computer Science 369(1-3), 361–376 (2006a)
36. Plantinga, S., Vegter, G.: Isotopic approximation of implicit curves and surfaces. In: Proc. Eurographics Symposium on Geometry Processing, pp. 245–254. ACM Press, New York (2004)
37. Preparata, F.P., Shamos, M.I.: Computational Geometry. Springer, Heidelberg (1985)
38. Richardson, D.: How to recognize zero. J. Symbolic Computation 24, 627–645 (1997)
39. Richardson, D.: Zero tests for constants in simple scientific computation. Mathematics in Computer Science 1(1), 21–38 (2007); Inaugural issue on Complexity of Continuous Computation

40. Schirra, S.: Robustness and precision issues in geometric computation. In: Sack, J., Urrutia, J. (eds.) Handbook of Computational Geometry. Elsevier Science Publishers, B.V., North-Holland, Amsterdam (1999)
41. Schmitt, S.: The diamond operator – implementation of exact real algebraic numbers. In: Ganzha, V.G., Mayr, E.W., Vorozhtsov, E.V. (eds.) CASC 2005. LNCS, vol. 3718, pp. 355–366. Springer, Heidelberg (2005)
42. Stroustrup, B.: The Design and Evolution of C++. Addison-Wesley, Reading (April 1994)
43. van der Hoeven, J.: Effective real numbers in Mmxlib. In: Proc. ISSAC 2006, Genova, Italy, pp. 138–145 (2006)
44. Yap, C., Li, C., Pion, S., Du, Z., Sharma, V.: Core Library Tutorial: a library for robust geometric computation (1999–2004); Version 1.1 was released in January 1999. Version 1.6 in June 2003, Source and documents from, http://cs.nyu.edu/exact/
45. Yap, C.K.: In praise of numerical computation. In: Albers, S., Alt, H., Näher, S. (eds.) Efficient Algorithms. LNCS, vol. 5760, pp. 380–407. Springer, Heidelberg (2009)
46. Yap, C.K.: Tutorial: Exact numerical computation in algebra and geometry. In: Proc. 34th Int'l Symp. Symbolic and Algebraic Comp. (ISSAC 2009), KIAS, Seoul, Korea, July 28-31, pp. 387–388 (2009)
47. Yu, J.: Exact arithmetic solid modeling. Ph.D. dissertation, Department of Computer Science, Purdue University, West Lafayette, IN 47907, Technical Report No. CSD-TR-92-037 (June 1992)

Introducing HOL Zero

(Extended Abstract)

Mark Adams

Proof Technologies

Theorem provers are now playing an important role in two diverse fields: computer system verification and mathematics. In computer system verification, they are a key component in toolsets that rigorously establish the absence of errors in critical computer hardware and software, such as processor design and safety-critical software, where traditional testing techniques provide inadequate assurance. In mathematics, they are used to check the veracity of recent high-profile proofs, such as the Four Colour Theorem and the Kepler Conjecture, whose scale and complexity have pushed traditional peer review to its limits.

The reason for using theorem provers is, of course, for the high assurance they provide, through their implementation of formalised deduction. Furthermore, theorem provers at the top end of the "assurance league" have a robust software architecture called *LCF-style*, that reduces the part trusted to perform sound deduction down to a relatively small inference kernel. This is achieved by only allowing theorems to be constructed inside the kernel, forcing all deduction to ultimately go via the kernel. LCF-style systems, such as the HOL family, Isabelle and Coq, represent a large proportion of current worldwide theorem prover usage.

However, there still remain various concerns with the trustworthiness even of LCF-style systems. Firstly, the implementation of the LCF-style architecture is sometimes not watertight, and tricks like writing bogus theory export files and updating mutable strings can lead to the construction of arbitrary theorems. Secondly, pretty printers produce ambiguous output in certain circumstances, such as for theorems involving overloaded variables or irregular names, potentially misleading the user into thinking their conjecture has been proved when actually something else has. Thirdly, the clarity of the trusted core of source code can suffer from a variety of problems, including overly long file/line lengths, irregular formatting, ultra-sparse commenting and excessive complexity, thus increasing the possibility that soundness errors are still lurking undiscovered.

HOL Zero is a new, open-source HOL system that aims to tackle these problems. It is a bare-bones system, and does not have the advanced support for interactive or automated proof that other HOL systems have, which enable them to be employed on large projects. However, it has a simple, watertight inference kernel, a pretty printer that always produces unambiguous output, and extremely carefully written and explained source code, and so pushes theorem prover trustworthiness to a new level.

By adhering to a new API for HOL, called the Common HOL Platform, it can import proofs that have been recorded and exported on existing HOL systems (that have been suitably adapted for Common HOL), thus acting as a generic HOL proof checker. It has successfully imported various large bodies of proof from HOL Light, including the entire HOL Light base system, the Jordan Curve Theorem and the Consistency of HOL

K. Fukuda et al. (Eds.): ICMS 2010, LNCS 6327, pp. 142–143, 2010.

Light. The intention is to remove any lingering doubts about the correctness of proofs performed on HOL systems.

The clarity and simplicity of HOL Zero's implementation also make it a good basis for those wishing to understand how theorem provers work, potentially extending understanding well beyond a small handful of computer scientists to broader communities of mathematicians, logicians and programmers eager to learn. Perhaps HOL Zero could become the basis for university courses on theorem prover implementation. This would increase scrutiny of its source code, and thus further improve its trustworthiness.

In this presentation, we will give examples of trustworthiness problems in existing HOL systems (HOL is the presenter's area of expertise), and give an overview of HOL Zero's features and its implementation, including how it avoids these problems. We will also provide a live demonstration of proof exporting/importing in action.

Euler's Polyhedron Formula in mizar

Jesse Alama*

Center for Artificial Intelligence
Department of Computer Science, Faculty of Science and Technology
New University of Lisbon, Portugal
j.alama@fct.unl.pt

Abstract. Euler's polyhedron formula asserts for a polyhedron p that $V - E + F = 2$, where V, E, and F are, respectively, the numbers of vertices, edges, and faces of p. Motivated by I. Lakatos's philosophy of mathematics as presented in his *Proofs and Refutations*, in which the history of Euler's formula is used as a case study to illustrate Lakatos's views, we formalized a proof of Euler's formula formula in the mizar system. We describe some of the notable features of the proof and sketch an improved formalization in progress that takes a deeper mathematical perspective, using the basic results of algebraic topology, than the initial formalization did.

1 Introduction

I. Lakatos's philosophy of mathematics, as expounded in *Proofs and Refutations* [5], challenges formalism on the grounds that formalism is unable to satisfactorily account for the growth and development of mathematical knowledge. In *Proofs and Refutations* Lakatos delves into the history of Euler's polyhedron formula, arguing that formalist philosophy of mathematics does not satisfactorily account for the changes that the theorem underwent.

Inspired by Lakatos's philosophy and the challenge that it evidently raises to contemporary work in proof checking, we set out to formalize a proof of Euler's polyhedron formula. The purpose of this note is to discuss the resulting formalization [1] that was carried out in the mizar system.

2 Euler's Polyhedron Formula

Euler's polyhedron formula is a basic result of topology asserting, for a polyhedron p, the invariant relation

$$V - E + F = 2,$$

* Partially supported by the ESF research project *Dialogical Foundations of Semantics* within the ESF Eurocores program *LogICCC* (funded by the Portuguese Science Foundation, FCT LogICCC/0001/2007).

K. Fukuda et al. (Eds.): ICMS 2010, LNCS 6327, pp. 144–147, 2010.

where V, E, and F are, respectively, the numbers of vertices, edges, and faces of p. The relation—an extension to three dimensions of the familiar relation $V - E = 0$ for two-dimensional polygons—was first conjectured by Euler in a 1750 letter to C. Goldbach [4]. Eventually, Euler was able to prove his result [2], though his first attempt [3] was incomplete, and his second attempt is not without its flaws [10].

A generalization of Euler's relation to polyhedra of arbitrary finite dimension by L'Huillier states that

$$\sum_{k=0}^{d-1}(-1)^k N_k = 1 + (-1)^{d+1}, \tag{1}$$

where d is the dimension of the polyhedron and N_k is the number of k-dimensional polytopes of the polyhedron (so that V, E, and F in the old notation become N_0, N_1, and N_2).

Poincaré gave a proof of Euler's formula in this general form in 1893 [7] (though, like Euler's first proof, Poincaré's first proof was flawed but was later corrected [8]). Lakatos treats Poincaré's proof in *Proofs and Refutations*; indeed, the book essentially ends with a discussion of Poincaré's proof. Poincaré's proof, as presented by Lakatos, is thus a natural target for formalization.

3 A Formalization in mizar

The formalization of Poincaré's proof of Euler's formula was carried out in the mizar system [6]. Although mizar is clearly only one among a variety of systems with which to formalize Poincaré's proof, mizar is a natural candidate because of its declarative proof language based on natural deduction, classical first-order logic, and set theory and its foundation in set theory. Such "classical" foundations are criticized by Lakatos. Less straightforwardly "classical" proof systems based on typed λ-calculus or various higher-order logics appeared, when searching for a suitable interactive proof assistant with which to carry out Poincaré's proof, to be rather less natural candidates, especially given the aim of staying true to Lakatos's version of Poincaré's proof of Euler's polyhedron formula. In retrospect, there would likely be some advantages to formalizing Poincaré's proof in some other alternatives to mizar, owing to mizar's lack of "automation", compared to similar systems.

As with most non-trivial formalizations, the proof in mizar of Euler's polyhedron formula required some auxiliary mathematical knowledge to be formalized. At the time the formalization was carried out, the two main pieces of mathematical knowledge that were unavailable in the mizar Mathematical Library but were needed for the formalization of Poincaré's proof were:

- the so-called rank-nullity theorem of linear algebra, which says that for a linear transformation T from a finite-dimensional vector space V to a finite-dimensional vector space W, we have that $\dim V = \operatorname{rank} T + \operatorname{nullity} T$, where $\operatorname{rank} T$ and $\operatorname{nullity} T$ are, respectively, $\dim(\operatorname{im}(T))$ and $\dim(\ker(T))$.
- the fact that the powerset of a set can be considered as a vector space over the two-element field Z_2 where vector addition is understood as symmetric difference.

With these ingredients in place the formalization of Poincaré's proof of the general Euler polyhedron formula was straightforward. Polyhedra are defined as finite sequences of incidence matrices, which in turn are defined simply as functions from a cartesian product $P_m \times P_{m+1}$ of sets of m- and $m+1$-dimensional polytopes into the two-element field Z_2 (which is simply the set $\{0, 1\}$ made into a field in the obvious way). Poincaré gives a elegant sufficient condition for Euler's relation which, in Lakatos's work, is called *simple connectedness*:

```
let p be polyhedron;
attr p is simply-connected
 means
for k being Integer
  holds k -circuits p = k -bounding-chains p;
```

This property roughly says that a polyhedron has no "holes", unlike, say, a polyhedron on the surface of a torus.

The theorem to be proved, in the mizar syntax, is:

```
for p being polyhedron st p is simply-connected holds p is eulerian
```

where the property `eulerian` is defined as

```
definition
let p be polyhedron;
attr p is eulerian
 means
Sum (alternating-proper-f-vector p)
  = 1 + ((- 1) |^ ((dim p) + 1));
```

which reminds us of (1). Altogether the proof in mizar required about 2000 lines of text.

4 A Deeper Formalization

The current formalization is not entirely satisfactory, because it proves Euler's polyhedron formula only in its purely combinatorial, graph-theoretical sense. No real numbers are mentioned, and standard algebraic topological concepts hardly appear at all.

The narrowness from which the current formalization suffers is due largely to the choice of proof of Euler's formula. As discussed earlier, the motivation for Euler's formula (among possible mathematical results to formalize) and Poincaré's proof (among possible proofs of Euler's formula) was I. Lakatos's *Proofs and Refutations*. The book ends with a presentation and discussion of Poincaré's proof. Fidelity to Lakatos was, at the time the proof was conceived, essential, so alternatives to Poincaré's proof (as given by Lakatos) were not seriously considered.

A more satisfactory formalization would adopt the perspective of algebraic topology, where polyhedra are understood as certain configuration of points in a real cartesian space. Such a formalization, based on Pontryagin's [9] and Spanier's [11] is ongoing. The requirements for this wider project are considerably greater than the current, narrower formalization.

References

1. Alama, J.: Euler's polyhedron formula. Formalized Mathematics 16(1), 7–17 (2008), http://mizar.org/fm/2008-16/fm16-1.html
2. Euler, L.: Demonstratio nonnullarum insignium proprietatum quibus solida hedris planis inclusa sunt praedita. Novi Commentarii Academiae Scientarum Petropolitanae 4, 94–108 (1758)
3. Euler, L.: Elementa doctrinae solidorum. Novi Commentarii Academiae Scientarum Petropolitanae 4, 109–140 (1758)
4. Juskevich, A.P., Winter, E. (eds.): Leonhard Euler und Christian Goldbach: Briefwechsel 1729-1764. Akademie-Verlag, Berlin (1965)
5. Lakatos, I.: Proofs and Refutations: The Logic of Mathematical Discovery. Cambridge University Press, Cambridge (1976)
6. mizar, http://www.mizar.org
7. Poincaré, H.: Sur la généralisation d'un théorème d'Euler relatif aux polyèdres. Comptes Rendus de Séances de l'Academie des Sciences 117, 144 (1893)
8. Poincaré, H.: Complément à l'analysis situs. Rendiconti del Circolo Matematico di Palermo 13, 285–343 (1899)
9. Pontryagin, L.S.: Foundations of Combinatorial Topology. Dover Publications, New York (1999)
10. Samelson, H.: In defense of Euler. L'Enseignement Mathématique 42, 377–382 (1996)
11. Spanier, E.H.: Algebraic Topology. Springer, Heidelberg (1968)

Building a Library of Mechanized Mathematical Proofs: Why Do It? and What Is It Like to Do?

R.D. Arthan

Lemma 1 Ltd. & Queen Mary, University of London
rda@lemma-one.com

Abstract

A few years ago I used the ProofPower mechanized proof tool to develop the calculus from the ϵ-δ definitions of limits through to the derivation of the properties of the trigonometric functions from their differential equations. At the time, I wrote:

> "Undertaking this kind of work is a mathematical activity of an unusual, often entertaining, albeit sometimes frustrating nature. It is rather like preparing the material for a course or a textbook with the assistance of an amanuensis who is an *idiot savant* of an unusual kind. Firstly, he insists on and gets absolute editorial control over what purports to be a proof: every proof step is checked with complete accuracy and no lacunae slip his attention. On the more constructive side, he [can be] capable of amazing feats of calculation."

In this talk, I will discuss what it is like to develop mathematical theories with such an unusual assistant and on why the results will eventually be seen as highly worthwhile for mathematicians, computer scientists and engineers.

K. Fukuda et al. (Eds.): ICMS 2010, LNCS 6327, p. 148, 2010.

Linear Programs for the Kepler Conjecture
(Extended Abstract)

Thomas C. Hales*

University of Pittsburgh

Abstract. The Kepler conjecture asserts that the densest arrangement of congruent balls in Euclidean three-space is the face-centered cubic packing, which is the familiar pyramid arrangement used to stack oranges at the market. The problem was finally solved in 1998 by a long computer proof. The Flyspeck project seeks to give a full formal proof of the Kepler conjecture. This is an extended abstract for a talk in the formal proof session of ICMS-2010, which will describe the linear programming aspects of the Flyspeck project.

The original proof of the Kepler conjecture in 1998 was about three hundred pages long and relied long computer calculations that were done by custom computer code [6]. The amount of custom computer code was originally estimated to be about 40,000 lines, but later estimates give a number closer to 180,000 lines [3]. The larger figure includes a large difficult-to-estimate number of duplicated lines.

The computer code consists of three separate programs. The first is a program that generates all hypermaps up to isomorphism with prescribed properties. (A hypermap, defined below, is a combinatorial structure that conveniently encodes the structure of a planar graph.) The output of this program is a set X of about 18 thousand hypermaps. The second is a program that proves nonlinear inequalities. The third program generates and solves linear programs.

It is this third program that is the subject of this extended abstract. This third program was considered from a formal proof perspective in Obua's thesis [7]. The thesis gives a formal verification of over 90% of the cases. Why did he not complete the remaining 10% of the cases? He did not encounter any difficulties with the technology. Indeed, his work demonstrates that the formal proof of linear programs is entirely feasible.

Rather, the difficulties are with the slopppy documentation in the original proof of the Kepler conjecture. The linear programming part of the proof is most poorly documented part of the proof. The original linear programs were generated in interactive Mathematica sessions [4]. There are voluminous notes about these interactive sessions, but these notes are not in the form of executable code. In the end, the remaining cases were not formalized because it was not clear what precisely was to be formalized.

* Research supported by NSF grant 0804189 and a grant from the Benter Foundation. The author places this abstract in the public domain.

K. Fukuda et al. (Eds.): ICMS 2010, LNCS 6327, pp. 149–151, 2010.

1 Linear Programs

The linear programming part of the proof has been reworked from the start with the formal proof in mind [5]. The methods described in this extended abstract treat 99.93% of the cases (that is, all but 12 hypermaps).

The linear programs are specified in the *MathProg* language, which is a domain specific language designed for linear programming in the GNU Linear Programming Kit (GLPK) [1]. MathProg is a subset of the *AMPL* language [2]. According to the design of MathProg, the linear program is split into two parts: a model and data. The model contains the declarations of variables, parameters, and indexing sets. There is a single model file shared by all linear programs. There is a separate data file for each linear program. The model is further divided into a header and a body.

The computer code for this project has been written in Objective Caml. The data files as well as the body of the model are automatically generated. The computer code has not yet been formalized, but it is now in a formalization-ready state. There are a few different parts to the code.

- (200 lines) interface with GLPK,
- (200 lines) MathProg format model header,
- (80 lines) model body code generator,
- (300 lines) data generator and control flow.

The are two additional data files:

- (4MB) a set X of about 18 thousand hypermaps,
- (1000 lines) archive of nonlinear inequalities (in HOL Light format).

This is a significant improvement over the original program from the 1998 proof of the Kepler conjecture, which involved several thousand lines of computer code, 3GB of data, and long interactive sessions.

2 The Main Theorem

As we mentioned above, in the original proof of the Kepler conjecture, it is difficult even to state the theorem that has been proved by the linear programming part of the proof. Here we give a simple statement that captures the linear programming part of the proof of the Kepler conjecture.

Definition 1. *A hypermap is a finite set D with two permutations $n, f : D \to D$. A third permutation e is defined by the relation $enf = I$. We represent the hypermap as a tuple $x = (D, e, n, f)$.*

Let X be the set of over 18 thousand hypermaps that is mentioned above. Each of these hypermaps can be augmented by various *markings* to produce what we call marked hypermaps. Let Y be the finite set of all marked hypermaps and let $Y_x \subset Y$ be those that come from $x \in X$. Certain subsets of Y_x are called *covers* of x. Associated with each marked hypermap $y \in Y$ is a polyhedron $P(y)$. The main theorem about linear programs takes the following form.

Theorem 1. *For every hypermap $x \in X$, there exists a cover $U \subset Y_x$ such that for every $y \in U$, the polyhedron $P(y)$ is empty.*

The proof is by a direct construction carried out by computer. Each polyhedron is explicitly presented as a finite system of linear inequalities. Linear programming methods show that each system of linear inequalities has no solutions. The existence of a cover is established by a direct computer search. It takes about 2.5 hours to run the program on a laptop computer.

Finally, we describe the relationship between this theorem and the Kepler conjecture. The Kepler conjecture is initially expressed as a statement about packings of congruent balls in an unbounded region of space. Various reduction arguments reduce the proof of the conjecture to a conjecture about packings of finitely many balls.

The proof of the Kepler conjecture is by contradiction. If the Kepler conjecture is false, then there exists a finite packing V of at most 15 balls that has various remarkable properties. The balls in the packing form the set of nodes of a graph (V, E), and the combinatorial properties of the graph can be encoded as a hypermap x in the set X. With a counterexample V in hand, for every cover U of x, it is possible to find a marked hypermap $y \in U$ for which the polyhedron $P(y)$ is nonempty. The existence of a counterexample V contradicts the theorem, which asserts on the contrary that $P(y)$ is empty.

References

1. GLPK (GNU Linear Programming Kit), http://www.gnu.org/software/glpk/
2. Fourer, R., Gay, D.M., Kernighan, B.W.: The AMPL book. Brooks/Cole, Monterey (2002)
3. Hales, T., Harrison, J., McLaughlin, S., Nipkow, T., Obua, S., Zumkeller, R.: A revision of the proof of the Kepler Conjecture. In: DCG (2009)
4. Hales, T.C.: Computer resources for the Kepler conjecture (2003), http://www.math.princeton.edu/~annals/KeplerConjecture/ (2005 snapshot)
5. Hales, T.C.: The Flyspeck Project (2010), http://code.google.com/p/flyspeck
6. Hales, T.C., Ferguson, S.P.: The Kepler conjecture. Discrete and Computational Geometry 36(1), 1–269 (2006)
7. Obua, S.: Flyspeck II: The basic linear programs, Ph.D. thesis, Technische Universität München (2008), http://deposit.d-nb.de/cgi-bin/dokserv?idn=992033632&dok_var=d1&dok_ext=pdf&filename=992033632.pdf, http://mediatum2.ub.tum.de/doc/645669/645669.pdf

A Formal Proof of Pick's Theorem

(Extended Abstract)

John Harrison

Intel Corporation, JF1-13
2111 NE 25th Avenue, Hillsboro OR 97124, USA
johnh@ichips.intel.com

Given a polygon with vertices at integer lattice points (i.e. where both x and y coordinates are integers), Pick's theorem [4] relates its area A to the number of integer lattice points I in its interior and the number B on its boundary:

$$A = I + B/2 - 1$$

We describe a formal proof of this theorem using the HOL Light theorem prover [2]. As sometimes happens for highly geometrical proofs, the formalization turned out to be quite challenging. In this case, the principal difficulties were connected with the triangulation of an arbitrary polygon, where a simple informal proof took a great deal of work to formalize.

Elementary Triangle

We start by establishing the result for an *elementary* triangle: one whose vertices are lattice points and which contains no other lattice points either in its interior or boundary. Pick's theorem for such a triangle simply asserts that it has area $1/2$, or 0 in the degenerate cases where it is flat.

Given two vectors A and B, we can consider them as defining the linear transformation of the plane $f : (x, y) \mapsto Ax + By$. It is not hard to show that if the image under f of the set of integer lattice points is exactly this same set of integer lattice points, then the determinant of the matrix of f is ± 1:

```
|- ∀f:real^N->real^N.
       linear f ∧ IMAGE f integral_vector = integral_vector
       ⇒ abs(det(matrix f)) = &1
```

Given an elementary triangle OAB, where we take O as the origin, one can show that the integer multiples of the two other vertices generate precisely the integer lattice points. The determinant in the previous theorem is precisely twice the area of the triangle formed by the three vertices, which therefore has area $1/2$:

```
|- ∀a b c:real^2.
       {x | x IN convex hull {a,b,c} ∧ integral_vector x} = {a,b,c}
       ⇒ measure(convex hull {a,b,c}) =
              if collinear {a,b,c} then &0 else &1 / &2
```

Arbitrary Triangle

Next, we proceed inductively to establish the result for an arbitrary triangle with all its vertices at integer lattice points. If a triangle ABC is not elementary, then it must have a lattice point D either:

K. Fukuda et al. (Eds.): ICMS 2010, LNCS 6327, pp. 152–154, 2010.

– On one if its sides, say AB, in which case we can subdivide it into triangle ADC and BCD.
– In its interior, in which case we can divide it into three triangles ADB, ADC and BDC.

Although this is straightforward enough, we can make it even simpler, by reformulating things slightly so that the first case becomes a degenerate case of the second, and we avoid handling degenerate cases of the theorem itself separately. By a general result on additivity of a function defined on a set of lattice points, we deduce the following variant of Pick's theorem for a triangle. It is straightforward to show it equivalent to the usual formulation with a proviso of nondegeneracy.

```
|- ∀a b c:real^2.
    integral_vector a ∧ integral_vector b ∧ integral_vector c
    ⇒ measure(convex hull {a,b,c}) =
        &(CARD {x | x IN convex hull {a,b,c} ∧ integral_vector x}) -
        (&(CARD {x | x IN convex hull {b,c} ∧ integral_vector x}) +
         &(CARD {x | x IN convex hull {a,c} ∧ integral_vector x}) +
         &(CARD {x | x IN convex hull {a,b} ∧ integral_vector x})) / &2 +
        &1 / &2
```

Arbitrary Polygon

Again, we proceed inductively, showing that any polygon can be subdivided into two by a line joining two vertices and otherwise lying entirely in the interior. (This can also be used to drive an inductive proof that any polygon can be triangulated, and that is where we looked for a proof, though we don't explicitly deduce this general result.) The informal proof, essentially excerpted from [1], seems relatively straightforward:

> Pick the coordinate axis so that no two vertices have the same y coordinate. Let B be the lowest vertex on the polygon, and let A and C be adjacent to B. If AC is an interior diagonal, we draw the diagonal AC, forming a triangle ABC and a polygon without the vertex C. Otherwise, let D be a vertex of the polygon at maximal distance from the line AC in the direction of B. Cut the polygon into two along the edge BD.

The first challenge is to formalize the 'pick the coordinate axis' step. An earlier paper [3] described an extensive framework for such 'without loss of generality' reasoning, but to support the present proof, this had to be generalized from 'first order' concepts like points and lines to 'higher order' concepts like polygonal paths and sets of points.

The next difficulty is to formalize the general notion of being 'inside' and 'outside' a polygon. In fact, we define this concept for a general closed curve, and use the Jordan curve theorem to deduce some of its important properties. For example, if one chops the inside of a closed curve in two with an arc across it, the inside divides into two in a fairly obvious way, and we use this to drive the main induction. But the proof of this in the general setting of simple closed curves turned out to be far from obvious; we formalized a 14-line proof from [5] (1.4, page 31), giving rise to a laborious 788-line formal counterpart.

Finally, we can follow through the informal proof. However, even this turned out to be quite difficult. One needs to work quite hard to establish that certain points lie

'inside' or 'outside' the polygon; we exploit a 'parity lemma' precisely characterizing how the inside/outside status switches as a line segment crosses segments of the polygon. It is also necessary to translate several completely obvious geometric arguments to do with orientation into tedious pieces of vector algebra. But finally, we obtain the overall result:

```
|- (∀x. MEM x p ⇒ integral_vector x) ∧
   simple_path(polygonal_path p) ∧
   pathfinish(polygonal_path p) = pathstart(polygonal_path p)
   ⇒ measure(inside(path_image(polygonal_path p))) =
       &(CARD
         {x | x IN inside(path_image(polygonal_path p)) ∧ integral_vector x}) +
       &(CARD
         {x | x IN path_image(polygonal_path p) ∧ integral_vector x}) / &2 -
       &1
```

References

1. Hales, T.C.: Easy piece in geometry (2007), http://www.math.pitt.edu/~thales/papers/
2. Harrison, J.: HOL Light: A tutorial introduction. In: Srivas, M., Camilleri, A. (eds.) FMCAD 1996. LNCS, vol. 1166, pp. 265–269. Springer, Heidelberg (1996)
3. Harrison, J.: Without loss of generality. In: Berghofer, S., Nipkow, T., Urban, C., Wenzel, M. (eds.) Theorem Proving in Higher Order Logics. LNCS, vol. 5674, pp. 43–59. Springer, Heidelberg (2009)
4. Pick, G.: Geometrisches zur Zahlenlehre. Sitzungsberichte des deutschen naturwissenschaftlich-medicinischen Vereines für Böhmen "Lotos" in Prag, Series 2 19, 311–319 (1899)
5. Whyburn, G.T.: Topological Analysis. Princeton Mathematical Series, vol. 23. Princeton University Press, Princeton (1964) (revised edn.)

Evaluation of Automated Theorem Proving on the Mizar Mathematical Library

Josef Urban[1,⋆], Krystof Hoder[2], and Andrei Voronkov[2,⋆⋆]

[1] Radboud University, Nijmegen
[2] University of Manchester

Abstract. This paper investigates the strength of first-order automatic theorem provers (ATPs) in proving theorems and lemmas from the Mizar proof assistant's formal mathematical library. Several Mizar use-cases are described and evaluated, as well as various ATP systems and strategies. The new version of the leading Vampire ATP system is included in the evaluation, experiments with Mizar-specific strategy-selection are performed with E the prover, and the SInE axiom selection is evaluated on large Mizar problems with both E and Vampire. A rough mathematical division of the Mizar library is introduced, and the ATP performance is evaluated on it.

1 Introduction and Motivation

In the last five years there was a considerable increase of the use of fully automatic first-order theorem provers as assistants to interactive theorem provers (ITPs). A number of formal knowledge bases and core logics have been translated to first-order ATP formats such as TPTP.[1] For example, let us mention the related work on the Isabelle/Sledgehammer ATP link [MP09], and the export of the SUMO [PS07] and CYC [MJWD06] real-world formal knowledge bases.[2]

One of the main goals of the MPTP[3] project is to make the large Mizar Mathematical Library[4] (MML) accessible to ATP and AI experiments and techniques. The particular value of MML/MPTP in comparison to the above mentioned related projects is that this is a comparatively large library focused primarily on standard mathematics as done by mainstream mathematicians (using first-order logic and set theory as the foundations). The first-order setting allows a practically complete and reasonably efficient translation for first-order ATPs, which is harder to do for higher-order systems. The size of the library and its consistency on the symbol-naming and theorem-naming level also allows experimenting with

⋆ Supported by the NWO project "MathWiki a Web-based Collaborative Authoring Environment for Formal Proofs".
⋆⋆ Supported by an EPSRC grant.

[1] Thousands of Problems for Theorem Provers, see `www.tptp.org`
[2] `www.ontologyportal.org` and `www.cyc.com`
[3] Mizar Problems for Theorem Proving.
[4] `www.mizar.org`

K. Fukuda et al. (Eds.): ICMS 2010, LNCS 6327, pp. 155–166, 2010.

all kinds of "knowledge-based" ATP/AI techniques, which might be relevant for emulating the thinking of learned mathematicians, and bringing new insights to the fields of ATP and AI.

The first inclusion of the MML/MPTP problems in ATP benchmarks (the TPTP library) happened in 2006, when also the large-theory MPTP Challenge[5] was announced. Since then the CASC[6] Large Theory Batch (LTB) competition was introduced, and run already twice in 2008 and 2009. This influences the performance and tuning of existing ATP systems, and gives rise to new techniques and interesting metasystems.

The purpose of the current paper is to evaluate the progress made over the past five years in the area of reasoning in large formal mathematical theories, and particularly evaluate the strongest ATP systems and metasystems on sufficiently recent MML/MPTP and the mathematical subfields contained in it.

1.1 Recent Evolution of Mizar and MPTP

Despite its age, Mizar is a living and evolving system with a number of users around the world. Since the last published MPTP experiments done on MML version 938 (938 articles), a number of articles have been added to the library resulting in thousands of new "Mizar theorems"[7]. These range from a number of standard calculus results developing, e.g., the Riemann integral, to abstract algebra results like Sylow theorems [Ric07], to formalization of special fields like BCI/BCK algebras [Din07] to results from mathematical theory of social choice like Arrow's Impossibility Theorem [Wie07].

At least the following developments have been tried/done with the Mizar system between these two versions:

- The Mizar type system mechanisms (Horn-like mechanisms automatically inferring monadic adjectives about the objects of the set-theoretical universe) have been constantly strengthened, becoming one of the main automation tools in Mizar.
- Experiments have been done with strengthening the matching/unification mechanisms in the Mizar kernel module.
- Identifications (i.e., registered automated equalities applied implicitly by the system) have been introduced by Mizar and used in MML.
- Further elements of computer algebra have been introduced in the kernel module, to allow automated normalization and solving of systems of linear equations.

The development of MPTP has to reflect the Mizar/MML changes. Also, as a relatively young system, MPTP has a number of its own developments to do. Here is a short summary of the recent ones:

[5] `http://www.tptp.org/MPTPChallenge/`

[6] The CADE ATP System Competition, see `http://www.tptp.org/CASC/`

[7] We put Mizar theorems in quotes at least once to deliver the message that only very few of these "theorems" would be called a "theorem" by mathematicians. Large majority of these propositions are lemmas useful and re-usable for proving further results, and this property makes them "theorems" in the Mizar parlance.

- Probably the largest change is that initial methods for ATP-export of Mizar internal arithmetics have been implemented. This is a constant cat-and-mouse pursuit with the experiments done with computer algebra in the Mizar kernel,[8] however it is now possible to do ATP experiments over Mizar problems containing arithmetics. The export is correct, but not always complete.[9] However, as can be seen in Section 3, counter-satisfiability is detected only reasonably rarely in practice by ATPs.
- MPTP changes in ATP problem creation, accommodating the new developments in the Mizar type automations, and introduction of identifications.
- Changes making MPTP faster and more real-time, including:
- More advanced (graph-like) datastructures to speed-up the process of selecting necessary parts of the library for generating the ATP problems.
- Larger use of available Prolog indexing and the asserted database for various critical parts of the code.
- Instead of working always with the whole loaded MML, MPTP was refactored to allow working only with the (usually much smaller) part of the MML needed for the newly processed article. This is specifically required for the new ATP-for-Mizar (MizAR) service running now in real time at the RU Foundations' group server[10] [US10].

The summary of data from previous experiments with SPASS (version 2.1) and E (version 0.9) from 2005 on MML version 938 using MPTP 0.2 is given in Table 1 (see [Urb06] for details).

Table 1. Reproving of the theorems from non-numerical articles by MPTP 0.2 in 2005

description	proved	countersatisfiable	timeout or memory out	total
E 0.9	4309	0	8220	12529
SPASS 2.1	3850	0	8679	12529
together	4854	0	7675	12529

Note that these experiments have been done in 2005 only on "non-numerical" articles (containing 12529 theorems/problems), i.e., on Mizar articles guaranteed not to contain any arithmetical evaluations. The current experiments described in Section 3 are however performed on the whole MML, because a basic ATP-export of Mizar computer algebra is now available.

[8] This is the main reason why we choose a version of MML that is not completely recent at the time of writing this evaluation. The MML version 1011 that we choose for the evaluation here has now been sufficiently tested, and the ATP-export of Mizar internal arithmetics sufficiently debugged. It would be possible to experiment with a more recent MML version, however for a large-scale evaluation it is preferable to use a reasonable recent version for which MPTP is known to work well.

[9] This also really depends on the particular version of the Mizar kernel.

[10] `http://mws.cs.ru.nl/~mptp/MizAR.html`

2 Mizar Data, Experimental Setup

The experiments described in Section 3 are performed on three classes of data,[11] all coming from the proofs of all Mizar theorems from MML version 1011. There are 51424 theorems in this MML version. The classes differ by the average number of axioms (previous theorems and definitions from MML) included in the problems, coming from different Mizar use-cases. The classes and use-cases are as follows:

- *SMALL*: Problems with smallest number of included axioms. This use-case models a user who knows relatively well how a proof should proceed (what MML knowledge should roughly be used). In the HOL Light (established for its ITP/ATP inventions) terminology: MESON_TACTIC[12] . The average size of an MPTP problem in this class is 218 formulas. Many of these theorems have long Mizar proofs - tens to hundreds of lines - and can contain nontrivial mathematical ideas. See the listings at these web pages.[13]
- *ENVIRON*: Problems that include all axioms contained in article's environment (that is: articles imported by the current article). This use-case models Mizar authors who selected a particular combination of mathematical areas (previous articles) to base their articles on, and thus limited the Mizar knowledge to a smaller subset of MML. Inside this MML subset they however do not provide any additional guidance to the ATPs. Such problems can already be very large: their average size is 5830 formulas.
- *ALL*: Problems that include all of the Mizar knowledge available in the MML at the time of proving a particular theorem. This models users who do not want to limit their search to the articles imported in their environment, and provide no guidance to ATPs. The price for such intellectual laziness is obviously a large number of axioms in such ATP problems, the average size of a problem in this class is 40898 formulas.

All these three use-cases are interesting and relevant. As mentioned above, the *SMALL* case is used a lot in ITPs like HOL (Light) as a general method for solving a goal once the user feels that it is sufficiently simply derivable from other established premises. The Mizar system actually also works in a similar way (using a custom weak theorem prover for the "by" inference), however, the emphasis there is not on strength, but on capturing the notion of *obvious inference* [Dav81, Rud87]. Another advantage of the *SMALL* case is that the 218 average formulas (which means much less in a significant number of cases) can be reasonably attacked by existing standard resolution and tableaux techniques, and ATPs based on them, without introducing any novel techniques for dealing

[11] Available at `http://mws.cs.ru.nl/~mptp/mptp_1011/noint/`

[12] `http://www.cl.cam.ac.uk/~jrh13/hol-light/HTML/MESON_TAC.html`. Actually, MPTP does here more than MESON: it adds a lot of "background" formulas to the problem including knowledge used implicitly by Mizar (reflexivity of $\leq$, etc.).

[13] `http://mmlquery.mizar.org/mmlquery/fillin.php?filledfilename=mml-facts.mqt&argument=number+102`, and `http://www.cs.ru.nl/~freek/100/`

with a large number of axioms. Thus, for metasystems that combine custom axiom-selection methods with standard ATPs, the *SMALL* case can be thought of as a benchmark for the ATP component of such metasystems, i.e., telling how good the performance of the whole metasystem could be, if the axiom-selection component of the metasystem was perfect.

This is no longer true for the *ENVIRON* and *ALL* classes. As will be seen in Section 3, using standard ATP techniques on these problems is currently not productive, and axiom-preselection methods are necessary on the *ENVIRON* and *ALL* classes to make use of ATPs.

There are other possible classes of data and divisions of the *SMALL*, *ENVIRON* and *ALL* classes along various axes. A common objection to these three classes is that they are too hard: for example, the data for testing Isabelle/Sledgehammer come typically from goals that are easier than the "full Isabelle theorems". The answer is that using Mizar *simple justifications* ("by" steps – steps provable using the Mizar built-in limited checker [Wie00]) has with the development of MPTP and ATP methods over Mizar become too easy, and such data are no longer suitable as a Mizar/MPTP/ATP benchmark. The success rate of various combined ATP/AI methods on large pieces of "by" data is now around 99.9%, actually allowing for using such methods together with the GDV [Sut06] ATP-based verifier (enhanced to handle TPTP proofs with Jaskowski-like assumptions) to completely ATP-cross-verify large pieces of MML (see [US08] for details). Other classes of problems that could come to mind are:

- Internal Mizar sublemmas that serve to prove another theorem/lemma, but are not themselves "too easy" (i.e., are not proved by simple justification). Such sublemmas could be considered an easier dataset than theorems, but harder than the simple justifications.
- De-lemmatized theorems. This would be a dataset created from *SMALL*, where the references (other theorems) used to prove a theorem are (recursively, to some level of recursion) replaced by their own references, in the extreme case expanding them all the way to axioms and definitions. Such de-lemmatized theorems could be considered a harder dataset than the standard theorems.

The reason why it seems unnecessary to test also on such classes of data is that already the theorem dataset provides a variety of both easy and hard data. There are a number of Mizar theorems proved using a simple justification, and on the other hand, there are theorems (like ROLLE:1 in [KRS90] - Rolle's theorem) that take more than four hundred lines and a large number of references to prove, i.e., the amount of lemmatization varies greatly across various Mizar articles and with various Mizar authors.

The ATP success rates reported in Section 3 on the *SMALL* theorem dataset indicate that also from the practical "benchmark" point of having data that are not too easy and not always very hard, this dataset seem to work well with current off-the-shelf ATPs. Again, this is not yet true with the large *ENVIRON* and *ALL* datasets which, on the other hand, can be considered to be hard

Table 2. Evaluation of E, SPASS, and Vampire on all *SMALL* problems in 30s

description	proved	countersatisfiable	timeout or memory out	total
E 1.1-004	16191	4	35229	51424
SPASS 3.7	17550	12	33862	51424
Vampire 0.6	20109	0	31315	51424
together	22607	12	28817	51424

benchmarks for ATP/AI metasystems that complement standard ATP with systems for axiom selection.

Divisions along various further axes of these datasets are certainly possible. In section 4 we attempt to define a "reasonable" crude mathematical categorization of 80% of MML articles, and provide an initial evaluation across this division.

3 Experiments

3.1 Overall Evaluation on *SMALL* Problems

The large-scale experimental evaluation of the standard ATPs focuses on the *SMALL* class of problems (for the reasons mentioned above in Section 2). The three main evaluated ATPs are the latest versions of the SPASS [WDF+09] (version 3.7) system, the E prover [Sch02] (version 1.1-004 Balasun), and the Vampire [RV02] prover (version 0.6 - preliminary version for CASC-J5). SPASS and E are evaluated on the server of the Foundations group at Radboud University Nijmegen (RU), which is eight-core Intel Xeon E5520 2.27GHz with 8GB RAM and 8MB CPU cache. The time limit for the evaluations is 30s,[14] and the memory limit is 900MB for each problem. Vampire is evaluated on computers at the laboratory of the University of Manchester (UM), each of them being Intel Core2 Duo E7300 2.66GHz PC with 1G RAM and 3MB cache. The time limits used are again 30s. The relative performances of the two hardware platforms have been compared by evaluation on common ATP problems. The UM platform turns out to be approximately 10% faster.[15] No parallelization is used, each problem is always run serially. The Table 2 shows the results. Note that there are some counter-satisfiable problems (very likely arithmetical) however their number is insignificant. It turns out that E, SPASS, and Vampire can in 30s together (that is: if run in parallel) solve 44% of the MML *SMALL* problems.

In Table 3, the results of SPASS used in (the incomplete) SOS mode are shown, and compared to the results of standard SPASS. This is done on randomly selected 1000 *SMALL* problems. The number of countersatisfiable results is not relevant for SPASS-SOS however: it is incorrect when SPASS is used in

[14] The experience from previous experiments with E and SPASS is that only a small fraction of problems is solved after 30s. This is obviously different with strategy-scheduling ATP systems like Vampire.

[15] This difference obviously does not translate to 10% more solved problems, however particularly with strategy-scheduling ATP like Vampire, it is quite significant.

Table 3. Comparison of SPASS-SOS and SPASS on 1000 *SMALL* problems in 30s

description	proved	countersatisfiable	timeout or memory out	total
SPASS 3.7-SOS	292	55	653	1000
SPASS 3.7	345	0	655	1000
together	377	0	623	1000

SOS mode together with ordering-based ATP techniques. SPASS-SOS turns out to be significantly worse than SPASS on the same data (proving 345 of these 1000 problems), however the SOS strategy is reasonably complementary to the standard one: together, the both methods of running SPASS solve 377 problems from this dataset (Vampire solves 39% of these problems).

3.2 Overall Evaluation on *ENVIRON* and *ALL* Problems, SInE

To a smaller extent (the above mentioned dataset of 1000 problems) we also evaluate E and Vampire on the large *ENVIRON* and *ALL* problems, and focus on evaluation of a new heuristic axiom pre-selector SInE.[16]

The SInE selection algorithm (yet to be published) uses a syntactic approach based on symbol presence in formulas of the problem. SInE builds a relation D (as in "Defines") between symbols and axioms. The D-relation represents the fact that for a symbol there are some axioms that "give it its meaning." In order to construct the D-relation, for each symbol the number of axioms in which it appears is computed, this number is called the *generality index* of the symbol. Then each axiom is put into the D-relation with the least general symbol it contains.[17] After the relation is built, the actual axiom selection starts. All problem-specific formulas are selected, and in each iteration the selection is extended by all included formulas that are D-related to any of the symbols used in already selected formulas.[18] The iterating is done until the set of selected axioms becomes stable. The stable set of formulas is then passed to a theorem prover.

This standard fixpoint algorithm however tends to give too many axioms on MML problems.[19] To deal with this, we have introduced a depth limit parameter — a limit on the number of selection-extending iterations. With the depth limit equal to one ("-d1" parameter), for example, only the included axioms D-related to symbols in the problem-specific axioms are included.

[16] SUMO Inference Engine - originally developed by the second author for reasoning in the large SUMO knowledge base.

[17] If there are more symbols with lowest generality index, axiom is put in relation with all of them.

[18] Note that the current implementation relies on reasonable presentation of large-theory problems using TPTP-includes. This can be easily changed if needed.

[19] The structure of MML significantly differs from KBs like SUMO and CYC. There are many more nontrivial theorems in MML, while SUMO and CYC contain a lot of definitions.

Table 4. Evaluation of Vampire and E with SInE(-d1) on random 1000 *ENVIRON* and *ALL* MML.1011 problems in 30s

problems	Vampire+SInE	Vampire+SInE(-d1)	E	E+SInE	E+SInE(-d1)
ENVIRON	181	205	65	135	161
ALL	84	141	21	64	153

The Table 4 presents the results of evaluation of E and Vampire on 1000 *ENVIRON* and *ALL* problems run with 30s time limit on the UM PCs.[20] Note that the combination of Vampire and SInE run with -d1 solves 205 of the *ENVIRON* problems, and the combination of E and SInE with -d1 solves 153 of the *ALL* problems. This is a very good performance on problems with average size of 5830 formulas resp. 40898 formulas. E in this mode solves about half of the *SMALL* versions of the problems. This is very likely also due to improved E heuristics for dealing with large problems. The SInE preprocessing time needs to be added to the 30s given to E prover. For the *ENVIRON* problems this time is on average 1s, and for the *ALL* problems, this is on average 4s.

3.3 Evaluation of Strategy Selection and Combination

Design, selection and combination of sufficiently orthogonal useful ATP strategies has been for some time a well known technique significantly raising performance of ATPs. Both Vampire and E use strategy selection and machine learning on the TPTP library to select a collection of useful strategies. However, they use the found strategies in different ways. Vampire selects sequences of strategies, while E selects one "best" strategy when run in auto-mode. The effect of strategy combination in Vampire can be estimated by comparing Vampire's performance in shorter and longer times (here in 5s and 30s on a random set of 1000 *SMALL* problems). For E and SPASS (running a single strategy depending on the problem) this difference is relatively small. Only 41 problems out of 345 solved in 30s by SPASS are solved in time longer than 5s, which is approximately 13% increase. For Vampire, this improvement is much more significant: out of 384 problems solved in 30s, only 310 are solved in 5s, which gives a 24% increase. This significant difference in comparison to SPASS is due to Vampire running not only for a longer time, but also switching to a different strategy during proof-search.

This clue leads to a strategy-evaluation experiment done with E again on this random set of 1000 *SMALL* problems: Each of the 196 E strategies predefined by the E developer Stephan Schulz is tested on this set of problems with a 5s time limit. It turns out that the strategy selected by E as potentially the best solves 310 problems with the 30 seconds time limit, while the strategy that turns out to be the best in reality solves 317 problems with a 5s time limit.

This clearly demonstrates the potential of domain-based ATP strategy-tuning. All the 196 E strategies together solve 386 of the 1000 problems. This confirms

[20] The new version of Vampire uses SInE automatically, thus we do not provide data for Vampire without SInE.

a previous conjecture by the first author and the E developer that E with a suitably-tuned strategy combination mode will be considerably stronger. It also demonstrates that strategy combination is more robust on new problems. The strategies of E are defined using smaller building blocks and a special ("programming") language using a number of parameters. Given the performance potential gained by this strategy-tuning, it would be very interesting (and feasible) AI experiment to try to invent new E strategies by AI methods for (e.g., genetic) parameter-optimization/programming. Particularly on a large knowledge base like MML, this might lead to some surprising ATP-strategy inventions, in the same way as combining ATPs with learning on MML sometimes finds completely novel and significantly shorter proofs than those written by the Mizar authors.

4 Evaluation of ATPs on Different Mathematical Domains in MML

Formal mathematics can be clustered according to many aspects, and just thinking about organizing mathematics can lead to all kinds of theoretical investigations in Foundations, Category Theory, but also into practical investigations with various classification schemes like Mathematics Subject Classification 2000[21], and also into very pragmatic classifications based on the shape of the formal theories, important for performing automated reasoning in various domains[22].

For the purpose of ATP evaluation in this paper we attempt a manual division of the MML articles into (currently) seventeen subdomains of main MML developments. This is motivated by the curiosity to verify experimentally various intuitions developed over the years in the ATP field, like "algebra is ATP-easier than calculus". As mentioned above, there can be many approaches to this, and for example a proper MathSC2000 classification would certainly serve even better than the coarse-grained division started by us, however detailed classification of more than 1000 formal articles requires a non-trivial amount of work, and making fine-grained decisions would be beyond our resources. The (evolving) division can be viewed at our web page[23]. The categories and the numbers of articles in each category are shown in Table 5. The categorization now includes 804 articles out of the total 1011.

To the extent to which MML is approximation of "real mathematics", and to the extent to which this rough categorization is valid, these seventeen large classes of problems (with the three different problem sizes coming from the different use-cases described in Section 2) express a large mathematically-oriented (and particularly Mizar/MPTP-oriented) ATP benchmark.

[21] See http://wiki.mizar.org/twiki/bin/view/Mizar/MathematicsSubjectClassification for a so far 25%-successful attempt to classify MML according to MathSC2000.

[22] For example, the strategy-selection tuning typically works by defining suitable clustering based on the term and formula structure of the problems.

[23] http://github.com/JUrban/MPTP2/raw/master/MMLdivision.1011

Table 5. Categorization of MML 1011, 804 articles covered, SPASS, E, Vampire, and overall success rates on the categories

description	articles	probs	S	E	V	All	S %	E %	V %	All %
Algebra	50	2798	1182	1086	1314	1481	42.24	38.81	46.96	52.93
Algebraic Topology	5	215	52	50	89	94	24.19	23.26	41.4	43.72
Arithmetic, Number theory	70	4095	1587	1515	1741	1943	38.75	37	42.52	47.45
Calculus (real, complex)	54	3255	585	538	651	783	17.97	16.53	20	24.06
Category theory	21	1023	298	305	406	455	29.13	29.81	39.69	44.48
Computers, Algorithms	81	3809	971	932	1120	1304	25.49	24.47	29.4	34.23
Functional analysis	30	1320	395	330	445	507	29.92	25	33.71	38.41
General Topology	65	3191	1199	1115	1441	1594	37.57	34.94	45.16	49.95
Geometry	36	1593	666	659	806	876	41.81	41.37	50.6	54.99
Graph theory,Finite structs	43	2756	1186	1094	1331	1455	43.03	39.7	48.29	52.79
Lattices	50	2434	707	570	764	917	29.05	23.42	31.39	37.67
Linear Algebra	32	1752	496	493	630	700	28.31	28.14	35.96	39.95
Logic, Model theory	52	2832	1042	1084	1196	1369	36.79	38.28	42.23	48.34
Probability and Measure	23	1123	348	274	448	489	30.99	24.4	39.89	43.54
Real plane,Euclidean spaces	84	4555	1018	897	1290	1439	22.35	19.69	28.32	31.59
Set Theory	74	4060	2412	2278	2570	2735	59.41	56.11	63.3	67.36
Universal Algebra	34	1093	391	372	434	502	35.77	34.03	39.71	45.93
together	804	41904	14535	13592	16676	18643	34.69	32.44	39.8	44.49

The table indeed seems to confirm quite convincingly the "algebra is ATP-easier than calculus" theory. The set-theoretical domain outperforms even the algebraic and is the most ATP-friendly. This is quite likely because of two related factors:

- Set theoretical articles belong to the more basic ones, not much previous implicit knowledge is included in the articles and their size is thus likely smaller, making them easier for ATPs.
- As the needed implicit knowledge (encoding e.g. the type system) gets more involved in more advanced areas like calculus, the ATP-emulation of these Mizar type mechanisms becomes more costly, and the ATP performance suffers.

There are a number of ways that these data can be further analyzed, providing useful feedback to the MPTP algorithms and also to ATP (meta) systems.

5 Conclusions, Future Work

The most important message of this evaluation is that a combination of three recent ATP systems can solve 44% of the MML theorems coming from many different parts of mathematics, regardless of how much computer algebra is done in them. The leading Vampire ATP system alone can solve 39%. Another important message is that off-the-shelf ATPs are still weak in large theories, however fast axiom-selection heuristics like SInE can help them and improve this state very significantly. Other one-problem-at-atime large-theory heuristic methods similar

to SInE are used for axiom pruning in Isabelle/Sledgehammer [MP09], and also for axiom ordering in the SRASS system [SP07]. If these other methods could be run easily on arbitrary and large TPTP problems, it would be interesting to evaluate and compare them on our benchmarks, or at least in the CASC LTB competition which includes a small selection of the older MPTP problems in its MZR category. The problem of axiom selection in large theories and the possible gains from good solutions seem to be sufficiently important to warrant such benchmarking and further research in this field, possibly leading also to smarter clause-selection algorithms implemented directly inside ATPs.

In this evaluation, axiom-selection methods that use transfer of knowledge between problems (machine learning) like MaLARea [USPV08] are not considered. Methods using learning are very interesting and quite novel in the ATP context, and we are currently investigating various suitable methods for machine learning , characterizations of problems and proofs, and also suitable combinations with strategy-selection and with other axiom-selection methods working in the one-problem-at-a-time setting like SInE. We also plan to do a thorough strategy evaluation of a much larger set of strategies available with Vampire, and their Mizar/MPTP-oriented tuning similar to the current CASC-tuning of Vampire, possibly again with adding machine learning technology.

An important future work is translation of ATP proofs to a presentable ITP format. The (technically certainly admirable) solution used by Isabelle/ Sledgehammer (and obviously also of HOL Light) is inclusion of a reasonably strong ATP system directly into their cores. This is however against the philosophy of readable proofs and "obvious inferences" of Mizar, and with strong external ATPs also potentially causing all kinds of other problems: not all proofs can be internalized, and the hard internalized proof s make the library refactoring slow and fragile.[24] With the growing strength of ATPs, a proper human-readable presentation of ATP proofs is a more and more pressing (and also very interesting) AI task. Some previous work in this direction has been done in the context of the Omega and ILF systems. A recent initial effort in this direction is described in [VSU10].

References

[Dav81] Davis, M.: Obvious logical inferences. In: Hayes, P.J. (ed.) IJCAI, pp. 530–531. William Kaufmann, San Francisco (1981)

[Din07] Ding, Y.: Several classes of BCI-algebras and their properties. Formalized Mathematics 15(1), 1–9 (2007)

[KRS90] Kotowicz, J., Raczkowski, K., Sadowski, P.: Average value theorems for real functions of one variable. Formalized Mathematics 1(4), 803–805 (1990)

[24] Note that this philosophy is no longer just Mizar's: the Math Components project targeted at the large formalization of Feit-Thompson theorem in Coq is avoiding Coq mechanisms that keep "too much automation" inside proofs for very similar reasons as Mizar does.

[MJWD06] Matuszek, C., Cabral, J., Witbrock, M., DeOliveira, J.: An Introduction to the Syntax and Content of Cyc. In: Baral, C. (ed.) Proceedings of the 2006 AAAI Spring Symposium on Formalizing and Compiling Background Knowledge and Its Applications to Knowledge Representation and Question Answering, pp. 44–49 (2006)

[MP09] Meng, J., Paulson, L.C.: Lightweight relevance filtering for machine-generated resolution problems. J. Applied Logic 7(1), 41–57 (2009)

[PS07] Pease, A., Sutcliffe, G.: First Order Reasoning on a Large Ontology. In: Urban, J., Sutcliffe, G., Schulz, S. (eds.) Proceedings of the CADE-21 Workshop on Empirically Successful Automated Reasoning in Large Theories (2007)

[Ric07] Riccardi, M.: The Sylow theorems. Formalized Mathematics 15(3), 159–165 (2007)

[Rud87] Rudnicki, P.: Obvious inferences. J. Autom. Reasoning 3(4), 383–393 (1987)

[RV02] Riazanov, A., Voronkov, A.: The design and implementation of VAMPIRE. Journal of AI Communications 15(2-3), 91–110 (2002)

[Sch02] Schulz, S.: E – a brainiac theorem prover. Journal of AI Communications 15(2-3), 111–126 (2002)

[SP07] Sutcliffe, G., Puzis, Y.: SRASS - a semantic relevance axiom selection system. In: Pfenning, F. (ed.) CADE 2007. LNCS (LNAI), vol. 4603, pp. 295–310. Springer, Heidelberg (2007)

[Sut06] Sutcliffe, G.: Semantic Derivation Verification. International Journal on Artificial Intelligence Tools 15(6), 1053–1070 (2006)

[Urb06] Urban, J.: MPTP 0.2: Design, implementation, and initial experiments. J. Autom. Reasoning 37(1-2), 21–43 (2006)

[US08] Urban, J., Sutcliffe, G.: ATP-based cross-verification of Mizar proofs: Method, systems, and first experiments. Mathematics in Computer Science 2(2), 231–251 (2008)

[US10] Urban, J., Sutcliffe, G.: Automated reasoning and presentation support for formalizing mathematics in Mizar. In: Autexier, S., Calmet, J., Delahaye, D., Ion, P.D.F., Rideau, L., Rioboo, R., Sexton, A.P. (eds.) AISC 2010. LNCS (LNAI), vol. 6167, pp. 132–146. Springer, Heidelberg (2010)

[USPV08] Urban, J., Sutcliffe, G., Pudlak, P., Vyskocil, J.: MaLARea SG1: Machine Learner for Automated Reasoning with Semantic Guidance. In: Armando, A., Baumgartner, P., Dowek, G. (eds.) IJCAR 2008. LNCS (LNAI), vol. 5195, pp. 441–456. Springer, Heidelberg (2008)

[VSU10] Vyskocil, J., Stanovsky, D., Urban, J.: Automated proof shortening by invention of new definitions. In: LPAR 2010. LNCS (LNAI). Springer, Heidelberg (to appear 2010)

[WDF+09] Weidenbach, C., Dimova, D., Fietzke, A., Kumar, R., Suda, M., Wischnewski, P.: SPASS version 3.5. In: Schmidt, R.A. (ed.) Automated Deduction – CADE-22. LNCS, vol. 5663, pp. 140–145. Springer, Heidelberg (2009)

[Wie00] Wiedijk, F.: CHECKER - notes on the basic inference step in Mizar (2000), `http://www.cs.kun.nl/~freek/mizar/by.dvi`

[Wie07] Wiedijk, F.: Arrow's impossibility theorem. Formalized Mathematics 15(4), 171–174 (2007)

On Local Deformations of Planar Quad-Meshes

Tim Hoffmann

Zentrum Mathematik
Technische Universität München
tim.hoffmann@ma.tum.de

Abstract. Planar quad-meshes (meshes with planar quadrilateral faces – PQ-meshes for short) are an important class of meshes (see e.g. Bobenko and Suris [2008]). Although they are often desirable in computer graphics – since planar quads can be rendered with out triangulating them – and architectual geometry (see Pottmann et al. [2007]) – because building with planar tiles is more cost effective – modelling freeform surfaces with planar quadrilaterals is problematic (in fact in practical applications one deforms or subdivides PQ-meshes without obeying the planarity constraint and ensures it afterwards in a global optimization step). In this paper we present a method that allows local deformations of PQ-meshes (with square grid combinatorics) that makes it possible to modify a PQ-mesh while keeping all quadrilaterals planar through the whole process (without a minimization step). In principle the method allows for PQ-mesh subdivision as well.

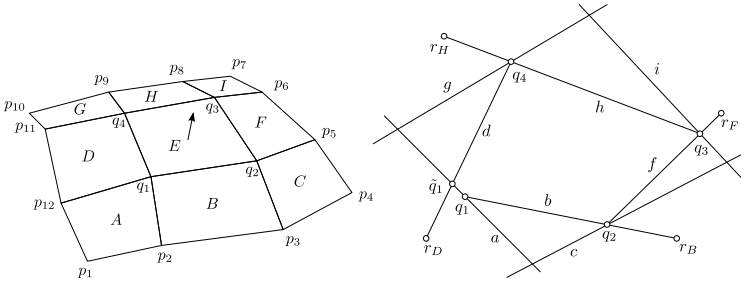


Fig. 1. A 3×3-quad patch and the configuration in the plane E

The deformation scheme outlined in the following is a new approach and it is currently implemented in a prototype stage by Christian Hick. The general idea is that while moving a single vertex will generically destroy the planarity of the adiacent faces, moving the plane of a face and allowing its four points to adjust as necessary, has exactly enough freedom to generically allow for planarity of the central face as well as its neighbours.

The Construction

Given a 3×3-quad patch from a PQ-mesh as shown in Fig. 1-a. Assume that the exterior should be fixed. In particular this means, that – using the labeling in

K. Fukuda et al. (Eds.): ICMS 2010, LNCS 6327, pp. 167–169, 2010.

Fig. 1-a – the points $p_1, \ldots, p_{12}$ are fixed. Now, generically prescribing the plane E in which the central quadrilateral lies the points $q_1, \ldots, q_4$ can be computed.

Since the points $q_1, \ldots, q_4$ all lie in the plane E, we can reformulate the problem of finding their locations into a problem in that plane only. Note that the planes (*not* the quadrilaterals) A, C, I, and G are fixed as well, since three points are already known in each of them. Let a, c, i, and g denote the lines in E that are the intersections of the planes A, C, I, and G with E. The remaining planes B, F, H, and D are yet to be determined, but we know at least the intersections of the lines through the point pairs (p_2, p_3), (p_5, p_6), (p_8, p_9), and p_{11}, p_{12}) with E. Denote them by r_B, r_F, r_H, and r_D respectively.

Since q_0 lies in E and A it must be in a. Choosing any point in a as q_1 fixes the plane B and in particular the intersection b of B with E (it is just the line through q_1 and r_b in E). This in turn determines q_2 as the intersection of the lines b and c. Knowing q_2 determines F and the intersecting line f (again as line through q_2 and r_F). Continuing this way we end up with a point $\tilde{q}_1$ on a as intersection of d and a (see Fig. 1-b). A priori there is no reason q_1 and $\tilde{q}_1$ should coincide, but the construction furnishes a projective map $q_1 \mapsto \tilde{q}_1$ from a onto itself. We are interested in fixed points of this map of which there are two if the map is hyperbolic, one if it is parabolic and none if it turns out to be elliptic – the case that the map degenerates to the identity can happen as well. However if one starts with a planar configuration and deforms it by smoothly varying the plane E there should be a neighbourhood in which a deformation is possible.

Since the whole problem and configuration is a projective one, we assume that E is a projective plane. Thus, points and lines are dual to each other. Writing them in homogenous coordinates, we have the convenient fact that the line through two points p and q can be computed by the vector cross-product $p \times q$. The dual construction is the intersection point of two lines r and s which is again the cross-product $r \times s$. For our construction above this means that $b = q_1 \times r_B$, $q_2 = b \times c$, $f = q_2 \times r_F$, and so forth (note the $\mathbb{R}^3$- representations are homogenous coordinates, so the order in the cross-product (which makes for a sign) is irrelevant here).

For a given point $v = (v_1, v_2, v_3)$, the map $w \mapsto v \times w$ can be written as $w \mapsto M_v w$ with M_V being a skew symmetric 3×3 matrix

$$M_v = \begin{pmatrix} 0 & -v_3 & v_2 \\ v_3 & 0 & -v_1 \\ -v_2 & v_1 & 0 \end{pmatrix}.$$

Using this notation the map $q_1 \mapsto \tilde{q}_1$ becomes $\tilde{q}_1 = M q_1$ with

$$M := M_a M_{r_D} M_g M_{r_H} M_i M_{r_F} M_c M_{r_b}.$$

Eigenvectors to non-vanishing eigenvalues of M will be exactly the fixed points we are interested in, since each of them will furnish a valid choice for q_1 (and subequently $q_2 = M_c M_{r_b} q_1$, $q_3 = M_i M_{r_F} q_2$ etc.). In general M has three eigenvalues, one of them being 0, since r_B will always map to 0. The other two – if present – will give us one or two distinct eigenvectors/fixed points. Thus generically there are two solutions. In a modeling application one starts with a given

configuration and calculates whether it corresponds to the bigger or smaller eigenvalue and then pick the corresponding one for the deformations.

The Dual Construction

In $P\mathbb{R}^3$ points and planes are dual to each other, just like points and lines are in $P\mathbb{R}^2$. So naturally, our local deformation has a dual version that we can state as follows. By just replacing each vertex with a plane and each plane with a vertex we get a configuration as in Fig: 2. We now can move the central vertex and calculate its direct neighbours in such a way that all except the four planes q_1, q_2, q_3, and q_4 are fixed. This implies in particular that the vertices B, F, H, and D move only on the lines that are the intersections of the outer planes p_2 and p_3, p_6 and p_6, p_8 and p_9, and p_{11} and p_{12}.

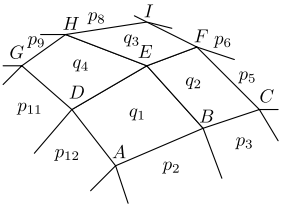


Fig. 2. The dual configuration (cf. Fig 1)

References

[2007] Pottmann, H., Asperl, A., Hofer, M., Kilian, A.: Architectural Geometry. Bentley Institute Press, Exton (2007)

[2008] Bobenko, A.I., Suris, Y.B.: Discrete Differential Geometry: Integrable Structure. Graduate Studies in Mathematics, vol. 98. AMS, Providence (2008)

Construction of Harmonic Surfaces with Prescribed Geometry

Matthias Weber

Indiana University
Department of Mathematics, Bloomington, IN 47401, USA
matweber@indiana.edu
http://www.indiana.edu/~matweber

Abstract. In this note we explain how a well-understood construction method for minimal surfaces can be used as flexible tool to explicitly parametrize harmonic surfaces with prescribed geometry of arbitrary finite topological type.

1 Introduction

A fundamental problem in surface geometry has been the construction of surfaces with given local properties like constant curvature and prescribed global geometric features. In the case of minimal surfaces, this has reached a state where amazingly complex surfaces can be constructed using relatively elementary means. Furthermore, the availability of rather explicit parametrizations has allowed to numerically and visually explore phenomena that have furthered the theory in a way unconceivable 10 years ago.

This success in the realm of minimal surfaces is mostly due to the availability of the Weierstrass representation, which is limited to minimal surfaces (and a few other related situations).

Thus, if one wants to use these techniques as a general modeling tool, the limitation to minimal surfaces limits also the topological possibilities. For instance, by a theorem of Schoen [1], there is no catenoid with a handle, i.e. no embedded, complete minimal surface with two ends and genus one. To overcome this limitation and still be able to make use of the machinery developed for minimal surfaces, we suggest to consider the class of harmonic surfaces where the parametrization is given by a harmonic map into $\mathbb{R}^3$.

In this note, we will illustrate how to implement this idea using a concrete example.

2 The Minimal Catenoid with a Handle

Definition 1. *Let Ω be a Riemann surface, $G(z)$ a meromorphic function and $dh(z)$ a holomorphic 1-form on Ω. Then*

$$f(z) = \operatorname{Re} \frac{1}{2} \int^z (1/G - G, i/G + iG, 2)\, dh \tag{1}$$

K. Fukuda et al. (Eds.): ICMS 2010, LNCS 6327, pp. 170–173, 2010.

parametrizes a minimal surface away from the singularities of G and dh. This parametrization is conformal.

The most simple instance of applying the Weierstrass representation to construct concrete minimal surfaces is the

Lemma 1. *Let Σ be a minimal surface, conformally equivalent to the disk, so that the boundary of $\bar{\Sigma}$ consists of finitely many planar symmetry curves that lie in vertical planes. Assume that all points with vertical normal occur only at intersection points of the boundary and possibly at the ends of the surface. Then Σ can be parametrized using a Gauss map $G(z)$ and a height differential dh that are given as*

$$G(z) = \rho \prod (x - a_j)^{\alpha_j} \tag{2}$$

$$dh = \mu \prod (x - a_j)^{\beta_j}\, dz \tag{3}$$

where the a_j are points on the real axis in one-to-one correspondence to the vertices of the surface boundary, the α_j, and β_j are real numbers uniquely determined by the angles between the symmetry lines at the vertices, ρ is the López-Ros parameter, and μ a scale factor.

For more sophisticated applications and a proof, see [2].

To construct an (impossible) catenoid with a handle, let's assume symmetries at all three coordinate planes (see [3]). The two vertical coordinate planes then cut the surface into four congruent simply connected pieces. The points with vertical normal occur at the catenoidal ends C_1 and C_2, and at two points H_1 and H_2 in the handle. Let's focus on the piece in which the boundary curves meet counter clockwise in the points C_1, C_2, V_2, V_1 in this order. Using the horizontal symmetry and a Mobius transformation of the upper half plane, we can assume that these points correspond to ∞, 0, $1/a$, a for some real number $a > 1$. Observe that the assumed symmetry about the horizontal coordinate plane is realized by a reflection at the unit circle in the upper half plane. Then it is easy to see that by

$$G(z) = \frac{1}{\sqrt{a}} z^{-1/2}(z - 1/a)^{-1/2}(z - a)^{1/2}$$
$$dh = dz/z$$

any catenoid with a handle and the assumed symmetry properties would be given by this formula. The problem is that no matter how one tries to adjust the parameter a, it is impossible to close all periods: The two boundary curves between C_1 and C_2 and between V_1 and V_2 are supposed to lie in plane $x_1 = 0$, but the formula only guarantees that they lie in parallel planes, see the left image in Figure 1.

3 The Harmonic Catenoid with a Handle

To understand how it is possible to *canonically* overcome this problem, it helps to switch to the global picture for a moment: A catenoid with a handle would

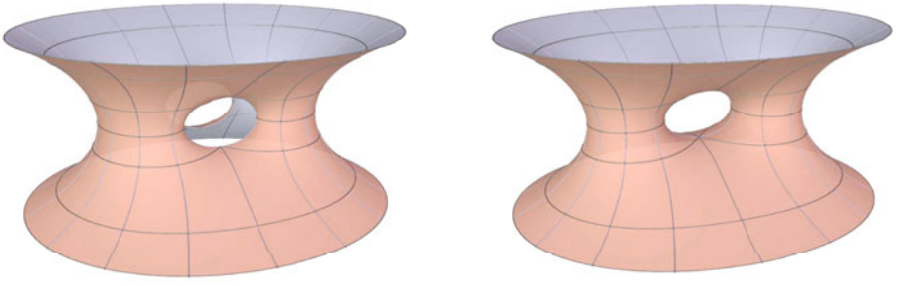


Fig. 1. Minimal Catenoid with Handle and Unclosed Period and Harmonic Catenoid with a Handle

be defined on a rectangular torus, where the Weierstrass data $G(z)$ and dh have become a meromorphic function and 1-form. The representation formula 1 parametrizes a well-defined surface if and only if *all* the periods of the forms ω_j are purely imaginary. This is not the case, as ω_1 has still a real period (while both other forms have purely imaginary periods). This can be remedied by adding a suitable multiple of the holomorphic 1-form dz of the torus to ω_1. In fact, this problem can be corrected uniquely on any compact Riemann surface in a unique way, as the real parts of its periods determine a holomorphic 1-form uniquely. This uniqueness guarantees that all assumed symmetries of the putative minimal surface will persist for the harmonic surface. Moreover, as we only modify the surface by adding a holomorphic form, the asymptotic behavior of the surface won't change. In our case, this means that the surface will still have ends asymptotic to a catenoid. This shows that for any $a > 1$, there is a unique harmonic catenoid with a handle and with all symmetry properties as claimed.

More generally, using the same approach, we can prove:

Theorem 1. *Let Σ be an immersed surface, diffeomorphic to a disk, so that the boundary of $\bar{\Sigma}$ consists of finitely many symmetry curves that lie in vertical planes. Assume that all points with vertical normal occur only at intersection points of the boundary and possibly at the ends of the surface. Also assume that reflection at the vertical symmetry planes extends the surface to a surface $\hat{\Sigma}$ of finite topological type whose ends are asymptotic to catenoidal or planar ends. Then there is a harmonic surface $\tilde{\Sigma}$ whose boundary is isotopic to that of Σ in each symmetry plane. It is given by a Gauss map $G(z)$ and a height differential dh as in equation (2) using a modified Weierstrass representation $\tilde{\omega}_j = \omega_j + \phi_j$, where the ϕ_j extend to holomorphic 1-forms $\tilde{\omega}_j$. Any choice of the parameters a_j determines the ϕ_j uniquely.*

References

1. Fujimori, S., Weber, M.: Triply periodic minimal surfaces bounded by vertical symmetry planes. In: Manuscripta Mathematica, pp. 29–53 (2009)
2. Hoffman, D., Karcher, H.: Complete embedded minimal surfaces of finite total curvature. In: Osserman, R. (ed.) Encyclopedia of Mathematics, pp. 5–93. Springer, Heidelberg (1997)
3. Schoen, R.: Uniqueness, symmetry, and embeddedness of minimal surfaces. Journal of Differential Geometry 18, 791–809 (1983)

A Library of OpenGL-Based Mathematical Image Filters

Martin von Gagern[1] and Christian Mercat[2]

[1] Zentrum Mathematik (M10), TU München, 85748 Garching, Germany
[2] I3M/LIRMM, cc 51, Université Montpellier 2, France

Abstract. There are a lot of transformations that can turn one raster image into a derived one in a mathematically interesting way. This article describes a collection of such filters, implemented in OpenGL in order to use the high degree of parallelism modern GPUs provide, thereby providing performance required to process e.g. live camera images in real-time. The filters contained in this library include wallpaper groups, conformal maps described by meromorphic functions, as well as hyperbolic symmetry groups. Using examples of increasing complexity, several key implementation techniques are explained, including texture wrap configurations, user-configurable control points, and custom fragment shader programs. This work might exhibit aesthetic aspects of mathematics to the masses and provide useful building blocks for scientists as well as artists.

Keywords: Transformation, tiling, wallpaper group, conformal map, meromorphic function, hyperbolic geometry, GPU, parallelism, webcam.

1 Introduction

1.1 Definition

In this article, an *image filter* is a piece of program code which can transform an input raster image to generate an output raster image.

Most filters have a set of parameters tuning their behavior. Besides common numeric parameters, control points are of special interest. A control point is a point that can be moved around in the input image and that will influence the resulting output image. Parameters can be interactive, so that modifying one parameter might implicitly adjust other parameters as well. Some filters might allow mathematical formulas as parameter input.

Our image filters are implemented in OpenGL so that most of the actual image transformation can be computed on the GPU. With this approach we expect to be able to provide sizable result images based for example on a live video stream from a webcam, while maintaining a decent frame rate.

K. Fukuda et al. (Eds.): ICMS 2010, LNCS 6327, pp. 174–185, 2010.

1.2 Motivation and Prior Art

The basic ideas for this collection of image filters come from a number of previous projects. There is *morenaments euc*, an application written by Martin von Gagern which allows a user to draw in any of the 17 wallpaper groups of the Euclidean plane. From that project derived *morenaments hyp*, a similar application for hyperbolic ornaments, for which some essential design principles were demonstrated at the ICMS 2006.

The combination of these two areas led to the investigation of possible conformal transformations of Euclidean ornaments into hyperbolic ones, in close cooperation with Jürgen Richter-Gebert. In the course of this project the use of OpenGL fragment shader programs for rendering of hyperbolic ornaments was developed as well.

Christian Mercat is author of *Conformal Webcam*[1], a Java application applying a freely chosen meromorphic function to a live image from a webcam.

Each author's project could benefit a lot from the ideas of the other. Evaluating the complex function for the *Conformal Webcam* in a fragment shader program would greatly increase performance and therefore mitigate the problem of slow frame rates. On the other hand, using a live image as input for a hyperbolic ornament would give another stunning visual effect, impressively exhibiting the performance of this GPU-based approach. This gave rise to the ideas outlined below.

1.3 Composition of the Library

We intend to build a library of image filters which are of pedagogical or artistic interest. The idea is to visualize an abstract mathematical notion as the deformation of a picture.

In the long run the library will likely cover topics from complex analysis, crystallography, group theory, hyperbolic geometry, differential equations, differential geometry, and surfaces.

Currently, the real-time computability and the use of OpenGL are central criteria for the implementation as part of this library. However, in the long run it might make sense to extend the library beyond what is possible under these restrictions.

In the beginning the collection of filters will consist of two major groups: *symmetric ornaments* and *conformal maps*. Symmetric ornaments include the well known 17 Euclidean wallpaper groups as well as the infinite number of hyperbolic counterparts.

The filters for conformal maps describe transformations of the plane onto itself via an arbitrary meromorphic function. This allows visualization of a large class of interesting non-linear transformations.

Other groups of filters are likely to be added in the future.

The latest source code as well as news on the current development status can be obtained from `http://martin.von-gagern.net/2010-icms/`

1.4 A Word on Symmetry

The image filters described below will conceptually map a single rectangular image to the whole (Euclidean or hyperbolic) plane, usually using some form of repetition. At least in the case of the Euclidean plane, the resulting image is a portion of the infinite result. This repetitiveness is not a strong requirement for future additions to the library, though.

When describing the form such a repetition takes, it is useful to name the underlying symmetry group. In most cases, this will be one of the 17 wallpaper groups of the Euclidean plane. This article will usually state both the crystallographic group name (starting with a letter p or c) as well as the corresponding *orbifold symbol*.[2] The latter provide a concept which easily extends to the hyperbolic plane as well.

1.5 Short Introduction to OpenGL

OpenGL is a complex and powerful graphics API. This section introduces only those aspects relevant to explanations below. While the primary application of OpenGL is the preferably photorealistic rendering of three-dimensional scenes, it can be used to render two-dimensional abstract images as well, simply by placing all objects in a plane, using an orthogonal projection and disabling all lighting effects.

An OpenGL application describes a scene as a set of geometric primitives, e.g. triangles. Attributes can be associated with each vertex of these primitives. The attribute most important here is the so-called texture coordinate. After loading an image as a texture, these texture coordinates can be used to fill the interior of the primitive by projective interpolation of the corresponding part of the texture image.

The texture interpolation process described above is the common case, called the *fixed functionality pipeline* and hard-wired into older graphics cards. More recent hardware supports the application to replace this fixed functionality with custom code, called a *fragment shader program*. This allows for very powerful calculations. The important fact is that these calculations are executed on the GPU for several pixels in parallel. Due to the large number of shading units on the GPU, this tends to be a lot faster than the same calculations would be if calculated on a single CPU core. Exploiting this parallelism for mathematical visualization is one of the aims of the project described in this article.

Each instance of a fragment shader program is responsible for the color of a single pixel (except when supersampling). As a consequence of the parallelism, there can be no communication between these programs, so the program should be structured in such a way that it can handle calculations for every pixel independently. Shader programs for OpenGL are written in the OpenGL Shading Language, which looks a lot like C, but provides built-in support for vectors, matrices and other things useful for geometric calculations. The source code of this program is compiled by the graphics card driver software.

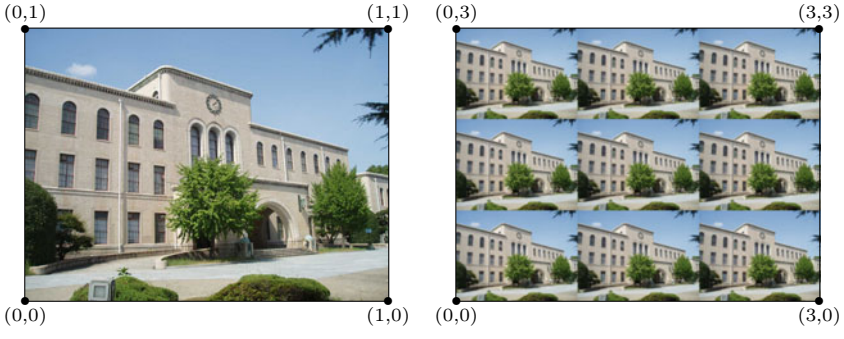
2 Explanation by Examples

Some things are best explained by example. So this section will describe a number of image filters in some detail, in order to demonstrate implementation techniques employed by other filters as well. The examples start simple, and their descriptions build on those of preceding examples.

2.1 Tiled Rectangles and Wraparound Parameters

Probably one of the simplest transformations is *simple tiling*. Copies of a rectangular input image are placed adjacently to cover the whole plane. The underlying symmetry group consists only of translations. It is the wallpaper group p1, also denoted with the orbifold symbol ∘.

Implementing this kind of transformation is very simple indeed. The input image is loaded into the OpenGL context as a texture. An orthogonal projection is chosen for the result image. The scene, as described in OpenGL, can consist of a single quad, spanning the area of the result image. Suitable texture coordinates for the corners of the quad will determine the number of repeats of the texture within the result image. The texture has to be configured with `GL_REPEAT` wrap parameters for both directions.



(a) The input texture (b) A simple tiling

Fig. 1. A simple tiling with texture coordinates

As a variation of this approach, one can replicate the input image using reflections instead of translations along one or both its axes. Translations in one direction and reflections in the other give the wallpaper group pm (orbifold symbol ∗∗), while reflections along all the edges of the input image yield the wallpaper group pmm (orbifold symbol ∗2222). Both of these are implemented using the above setup with `GL_MIRRORED_REPEAT` instead of `GL_REPEAT` for one or both directions.

The techniques presented up to here are far from new; many computer games employ the wraparound parameters to render all kinds of background textures, including wallpapers.

(a) Wallpaper group pm (b) Wallpaper group pmm

Fig. 2. Using different reflections

2.2 Wallpaper Groups and Fixed Functionality Pipeline

To make things more interesting, let's look at *kaleidoscopes* next. A kaleidoscope consists of a set of mirrors arranged in such a way that the generated reflections line up and form a periodic pattern. Replacing the physical mirrors with mirror reflections, the whole setup can be flattened into the Euclidean plane.

There are four possible kaleidoscope groups. Three of them use three mirrors, while the last one requires four mirrors.

Table 1. Kaleidoscope groups

Interior angles	$(\frac{\pi}{3}, \frac{\pi}{3}, \frac{\pi}{3})$	$(\frac{\pi}{4}, \frac{\pi}{2}, \frac{\pi}{2})$	$(\frac{\pi}{6}, \frac{\pi}{3}, \frac{\pi}{2})$	$(\frac{\pi}{2}, \frac{\pi}{2}, \frac{\pi}{2}, \frac{\pi}{2})$
Symmetry group	p3m1	p4m	p6m	pmm
Orbifold symbol	$*333$	$*442$	$*632$	$*2222$

While the group pmm might be applied to the whole rectangular input image, the kaleidoscopes using three mirrors cut out a triangular part of any image, to be used as the fundamental domain of the resulting symmetric pattern. This is where parameters come into play: we want to give users the freedom to choose the section of the input image on which they want to operate. This is best achieved by three control points, corresponding to the three corners of the triangle.

Moving one control point should move the triangle as a whole, while changes to the other two control points can be used to control size and orientation of the selected triangle. The shape is determined by the symmetry group and will remain fixed. Obviously, adjusting one of these points will adjust the others as well, which is the reason why control points are not independent input parameters, but instead modification of one will influence the others as well. The user interface has to take this kind of interaction into account.

In the OpenGL scene, this kind of pattern can no longer be expressed using a single quad and suitable wrap parameters for the input texture. Instead, every copy of the fundamental domain has to be expressed separately, using one triangle or quad each. The texture coordinates associated with the vertices align it with the portion of the input texture selected using the control points.

(a) Input with control points (b) Output

Fig. 3. The wallpaper group p3m1

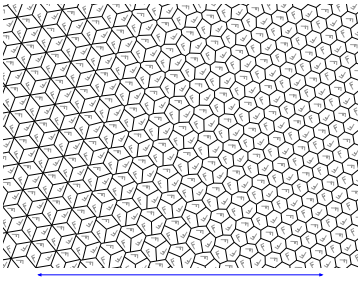


Fig. 4. The spectrum of fundamental domains for p3

Moving from kaleidoscopes to general wallpaper groups, new challenges arise. We'll demonstrate them using the wallpaper group p3 (orbifold symbol 333) as an example. That's the symmetry group obtained by taking only the orientation-preserving elements of p3m1.

The fundamental domain of p3 has twice the size of that of p3m1. But where the shape of the fundamental domain of p3m1 had to be an isosceles triangle, bounded by the lines of reflection, no such restriction applies to p3. Even if we restrict the fundamental domain to be a convex polygon, there is still a whole spectrum of possible shapes, illustrated in Figure 4. Of these, the rhombus and the regular hexagon are probably the most regular and thus the most aesthetic ones, but certainly not the only possibilities.

This calls for yet another set of three control points, to control the shape of the fundamental domain. Moving any one of these three points will move the others as well.

While most applications probably never use it, it is possible to draw arbitrary convex polygons in OpenGL. Therefore the scene description can be implemented in a straight-forward way, without having to cut the polygons into triangles first.

Similar techniques can be used to implement all 17 wallpaper groups.

They all rely only on the so called fixed functionality pipeline: textures and geometric primitives only, without any custom shader code. This is because the map

from destination positions to source positions is piecewise projective, in this case even piecewise affine.

2.3 Conformal Maps and Programmable Vertex Shaders

Everybody is used to visualizing a function from the plane to the real numbers, like precipitation maps: simply color the target space $\mathbb{R}$ with colors and plot each point (x, y) of the domain space $\mathbb{R}^2$ by the color $f(x, y)$.

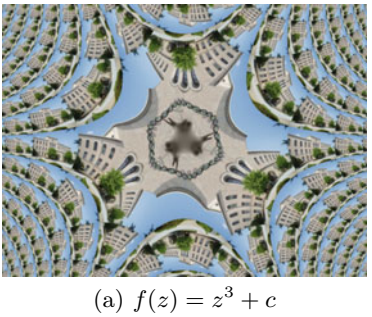
A generic *conformal filter* operates on $\mathbb{C}$ instead of $\mathbb{R}^2$. It simply paints a point z of the domain space (i.e. output image) with the color of the point $f(z)$ in the target space (i.e. input image) where f is some meromorphic function such as tan, exp, or log. Most keys on a calculator are seen as real functions but have an analytic continuation as a meromorphic function.

The complex differentiability is visualized by the fact that the picture in the domain space, away from singularities, is to the first order around z a simple similitude of parameter $1/f'(z)$, since locally the function behave as $f(z+z_0) = f(z_0) + z \cdot f'(z_0) + o(|z|)$. In particular, the zeros of the derivative are very easy to spot as the similitude ratio tends to infinity. There, the function is no longer conformal, it behaves locally as a monomial, $f(z+z_0)-f(z_0) = \frac{f^{(k)}(z_0)}{k!}\, z^k + o(z^k)$ and the angles through z_0 are divided by k, replicating the features k times. Other striking points are the logarithmic singularities. Christian Mercat used his Conformal Webcam during a master course on complex analysis.

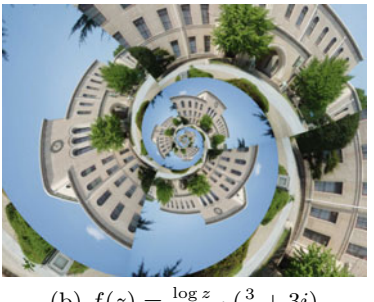
As these meromorphic functions cannot reasonably be expressed as piecewise projective maps, they have to be implemented using a different technique.

In order to get a whole plane from the single rectangular input, we can again wrap that input at its edges, with or without reflections, just as we did for the most simple tilings. Another thing we can take from these simple tilings is using a single quad spanning the output image as the whole scene description.

In order to map the texture onto that quad according to the exponential function, we have to replace the fixed functionality affine mapping with our own generic fragment shader. This fragment shader calculates suitable texture coordinates for every pixel of the result image by evaluation of the specified complex function. Then it determines the fragment color through a texture lookup.



(a) $f(z) = z^3 + c$



(b) $f(z) = \frac{\log z}{2\pi} \cdot (\frac{3}{4} + 3i)$

Fig. 5. Some conformal maps

Control points in the user interface could be passed as complex-valued constants into the formula, e.g. to configure specific features of the transformation, like the position of a singularity.

2.4 Basic Hyperbolic Tilings

A similar approach could be used for *tilings in the hyperbolic plane.*

In our opinion, the most aesthetic model of the hyperbolic plane is the Poincaré disk model. It represents the whole hyperbolic plane as the interior of the unit circle. Hyperbolic lines are modeled as circle arcs perpendicular to the unit circle, including lines through the origin as the special case of circles with infinite radius. The hyperbolic angle measure corresponds to the Euclidean angle between tangents. Due to this fact the model is conformal. The measure of length appears distorted, so that lines of equal hyperbolic length appear smaller the closer they are to the rim of the unit circle.

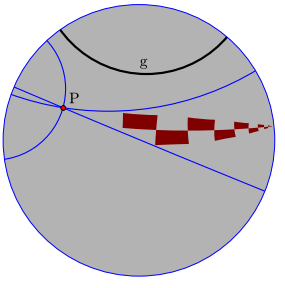


Fig. 6. The Poincaré model of the hyperbolic plane, illustrating multiple lines through P not intersecting g as well as a rule with steps of equal hyperbolic length

Orientation-preserving isometric transformations of the hyperbolic plane can be expressed as projective transformations of the complex line. Orientation-reversing transformations additionally require complex conjugation. To be more precise, we describe hyperbolic transformations using this formula:

$$\begin{pmatrix} x+yi \\ z+wi \end{pmatrix} \mapsto \begin{pmatrix} c-di & a+bi \\ a-bi & c+di \end{pmatrix} \cdot \begin{pmatrix} x+pyi \\ z+pwi \end{pmatrix} \qquad \begin{matrix} x,y,z,w \in \mathbb{R} \\ a,b,c,d \in \mathbb{R} \\ p \in \{-1,1\} \end{matrix} \tag{1}$$

where a, b, c, d, p characterize the transformation and x, y, z, w represent coordinates of a point in $\mathbb{CP}^1$. Scaling transformation matrices so that their determinants equal 1 (i.e. choosing a representative from $\mathrm{SL}(2, \mathbb{C})$) will help to avoid overflow when combining several transformations.

In order to render a hyperbolic tiling, we choose one fundamental domain at the center of the unit disk as the one providing the image data. The custom shader code will repeatedly apply generators of the group until the resulting position falls within that fundamental domain. Then a texture containing at least that fundamental domain is used to provide the actual color information.

In a crude approach, the fundamental domain can be simply cut from the input image. That means that a portion of the input image, delineated by circle arcs, is simply interpreted as a portion of the Poincaré disk. Figure 8?? illustrates this method. However, while being simple and fast, this approach is somewhat clumsy from the mathematical point of view. Hyperbolic isometries which change the placement of the fundamental domain within the unit disc also affect the shape of the portion cut out from the input image, and the interpretation of that portion with respect to the hyperbolic distance measure.

2.5 Conformal Hyperbolization and Chained Transformations

More elegant would be an explicit and mathematically sound transformation step between the Euclidean input image and the hyperbolic fundamental domain. Conformality turns out to be a suitable requirement for such a transformation. On the one hand, as the angle measure is the same for the Euclidean plane and the Poincaré model of the hyperbolic plane, conformality is a concept that easily bridges the gap between both worlds. On the other hand, angles between objects have a great impact on how we perceive an image composition. So preserving angles will also preserve a lot of how an image actually looks to the human eye.

There is a way to conformally turn the fundamental domain of an Euclidean ornament into a fundamental domain for a related hyperbolic ornament[3]. The orbifold of the resulting hyperbolic ornament has the same topology as that of the Euclidean ornament, but different combinatorics. In other words, it changes (some of) the numbers in an orbifold symbol, while leaving all other parts of the orbifold symbol unchanged. In general, as the sum of interior angles in hyperbolic geometry is always smaller than in Euclidean geometry, the angles have to decrease and the numbers indicating the order of rotational centers therefore have to increase. However, if some of the numbers increase enough, it is possible for a single number to decrease as well.

The method used for the actual transformation of the fundamental domain comes from the field of discrete differential geometry. The central concept is called discrete conformal equality of triangle meshes[4], which I'll briefly describe here. Input consists of a triangle mesh, including full combinatoric information as well as the lengths of all edges, and a target angle sum for each vertex. There exists an algorithm to calculate a scale factor for each vertex, such that scaling all edges by the factors associated with both their endpoints will result in a new mesh, which will have the desired angle sums.

In particular, if the original triangulation corresponds to a topological disc with designated corners, and if furthermore inner vertices are assigned a target angle sum of 2π and edge vertices an angle sum of π, then corner angles can be chosen at will, and the result will be a flat polygon with straight edges and the desired angles at the corners.

This association of scale factors with vertices of a mesh is in a way a discretization of the concept of continuous conformality. If the triangulation is sufficiently fine, the map between both meshes will be close to what a continuous conformal map would produce.

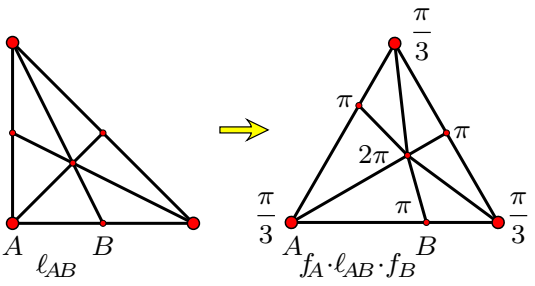


Fig. 7. Small example of a discretely conformal map

I will not go into the details of the algorithm. Suffice it to say that it is based on convex optimization. We are using an implementation of that algorithm called Confoo, written by Martin von Gagern.

After the triangle mesh has been transformed, the interior of the triangles has to be mapped as well. It turns out that superior to a simple affine interpolation is a specific projective interpolation, chosen such that it maps not only corners onto their images, but also the circumcircle of the source triangle to that of the target triangle. This kind of interpolation will be continuous along the edges of the triangles if and only if the two meshes are discretely conformally equivalent. The corresponding projective transformation can be easily computed from the scale factors associated with the vertices, and can be implemented in the projective geometry model provided by OpenGL.

Putting it all together, this gives the following processing chain: First the Euclidean input image is transformed into a hyperbolic image using discretely conformal triangle meshes. This can be achieved using the fixed functionality pipeline, with the input image as a texture, the target mesh expressed using triangle primitives, and the texture coordinates chosen to express the required projective interpolation. The resulting image is stored in an off-screen buffer and then used in a second step as the texture for the calculation of a hyperbolic tiling using a fragment shader program.

3 Applications

3.1 Edutainment

One major application of this library of filters is demonstrating mathematical concepts to the masses in a pleasing and fascinating way. It is fairly easy to combine a projection with a live camera feed, and as moving images tend to attract attention, and the interaction possible via the live image invites experimentation, it should be possible to rouse people's interest in the concepts behind these images. Seeing as many people still associate mathematics with dry formulas, it is important to show that the subject can be aesthetically beautiful as well.

(a) Fundamental domain simply cut from input image

(b) Conformal distortion of whole input image

Fig. 8. Hyperbolic patterns with orbifold symbol ∗3333

3.2 Education

The pedagogical interest of such a setup, besides its appeal to students who see their faces distorted, is to be able to change on the spot the image in order to interactively show specific features of the transformation by simply pointing with the finger. The functional relation between the actual image and its deformation is rendered much more lively and concrete by the webcam animation, where a still image doesn't provide just enough information to make the connection.

In the classroom, some students will be named by their position on the plane for example, this one is called zero, this other one $+i$, or -1 and so on, they can hold signs to show to the camera. It helps making the learning process more tangible.

3.3 Arts

While a simple setup as described above might be suitable for scientific institutions, exhibitions and schools, artists can probably make a lot more of it by including these filters or their output into their own installations or other artworks. The authors would appreciate notice of such applications.

Some of the filters discussed above are already implemented as Pure-Data GEM plugins, but without using OpenGL and the benefits this can bring. Pure-Data is a programming environment used in particular by artists and performers to produce interactive sounds and images. It might be worthwhile to make the OpenGL versions of these filters available to users of Pure-Data in order to reach a wider audience.

For live performances, it would be nice if the artist could control the transformations, in particular the position of control points, using features detected in the input image. That way, an item moved around by the artist could actually influence the transformation, adding even more dynamic to the performance.

3.4 Derived Scientific Applications

Scientists might build on this library for their own applications. There are many example applications out there doing complex 3D scenes in OpenGL, but rather few examples of how to do elaborate 2D scientific computations in this way. This set of filters and the corresponding front end application provide such an example, and may serve as the basis for their own implementations, so they won't have to start from scratch.

All of these applications are freely available, as the library will be published under the GNU General Public License. Other licenses might be available from the authors upon request.

Acknowledgments. The authors would like to thank Jürgen Richter-Gebert for his contributions to the hyperbolization of ornaments, as well as for his valuable comments on a draft of this article. Thanks also to the user Hasec of Wikimedia Commons, who took the photo used as the example input throughout this paper.

References

1. Mercat, C.: Conformal webcam, Images des Math., CNRS (March 2009), http://images.math.cnrs.fr/Applications-conformes.html
2. Conway, J.H., Burgiel, H., Goodman-Strauss, C.: The Symmetries of Things. A.K. Peters, Wellesley (2008)
3. von Gagern, M., Richter-Gebert, J.: Hyperbolization of Euclidean Ornaments. Electronic Journal of Combinatorics 16(2), R12 (2009)
4. Springborn, B., Schröder, P., Pinkall, U.: Conformal equivalence of triangle meshes. In: ACM SIGGRAPH (2008)

MD-jeep: An Implementation of a Branch and Prune Algorithm for Distance Geometry Problems

Antonio Mucherino[1], Leo Liberti[2], and Carlile Lavor[3]

[1] INRIA Lille Nord Europe, Villeneuve d'Ascq, France
antonio.mucherino@inria.fr
[2] LIX, École Polytechnique, Palaiseau, France
liberti@lix.polytechnique.fr
[3] Dept. of Applied Mathematics, State University of Campinas, Campinas-SP, Brazil
clavor@ime.unicamp.br

Abstract. We present MD-jeep, an implementation of a Branch & Prune (BP) algorithm, which we employ for the solution of distance geometry problems related to molecular conformations. We consider the problem of finding the conformation of a molecule from the distances between some pairs of its atoms, which can be estimated by experimental techniques. We reformulate this problem as a combinatorial optimization problem, and describe a branch and prune solution strategy. We discuss its software implementation, and its complexity in terms of floating-point operations and memory requirements. MD-jeep has been developed in the C programming language. The sources of the presented software are available on the Internet under the GNU General Public License (v.2).

1 Introduction

The Distance Geometry Problem (DGP) [4,5,8,13] is the problem of finding the coordinates of a set of points from some known distances between pairs of such points. There are different real-life applications where a DGP needs to be solved, and the most interesting and challenging application arises in biology. Distances between pairs of atoms of a molecule can be estimated through experiments of Nuclear Magnetic Resonance (NMR), and such distances can be used for formulating a DGP. DGPs arising in biology are usually referred to as Molecular DGPs (MDGPs — whence the name of our software).

Proteins are important molecules, because they perform many important functions in living beings. There is a web database, the Protein Data Bank (PDB) [1] (web address: http://www.rcsb.org/), which is completely devoted to the three-dimensional conformations of proteins. In fact, the conformation of a protein can give insights on its dynamics in the cells, and therefore on its function. Currently, the conformation and the function of many proteins are still not known: each gene of recently sequenced genomes is potentially able to code for a protein (or for more than one), but the corresponding conformation and function are still

K. Fukuda et al. (Eds.): ICMS 2010, LNCS 6327, pp. 186–197, 2010.

unknown. One way of solving this problem is to isolate each of such proteins and to perform experiments of NMR in order to obtain a subset of distances between pairs of their atoms. The successive step is to solve an MDGP.

The MDGP is, in its basic form, a constraint satisfaction problem, where molecular conformations, that satisfy all the constraints based on the distances, must be identified. This problem is often reformulated as a global continuous optimization problem, where a penalty function, measuring the satisfaction of the set of constraints, needs to be minimized. Different penalty functions have been proposed over the years for the MDGP, and all of them contain several local minima, where many traditional nonlinear descent methods can easily get stuck at. The most common penalty function is the Largest Distance Error (LDE):

$$LDE(X) = \frac{1}{m} \sum_{i,j} \frac{| \, ||x_i - x_j|| - d_{ij} \, |}{d_{ij}}, \tag{1}$$

where $X = \{x_1, x_2, \ldots, x_n\}$ is a three-dimensional conformation of n atoms, and m is the number of known distances d_{ij}.

Many techniques have been proposed for the MDGP, and the reader is referred to [8,13] for a survey. However, there are only a few software for the MDGP that are freely available for the scientific community. As an example, `DGSOL` (`http://www.mcs.anl.gov/~more/dgsol/`) is based on the idea of approximating the penalty function (in the continuous reformulation of the problem) with a sequence of smoother functions converging to the original objective function [14]. Other available software products are based on general meta-heuristic searches for global optimization. `Xplor-NIH` (`http://nmr.cit.nih.gov/xplor-nih/`) has been particularly designed for solving MDGPs arising from NMR experiments [19], and it includes different functionalities. In particular, for the solution of MDGPs, it makes use of heuristic methods (such as Simulated Annealing) and local search methods (such as Conjugate Gradient Minimization). Finally, `TINKER` (`http://dasher.wustl.edu/tinker/`) is a package for molecular modeling and design. It includes many force fields for attempting the prediction of protein conformations from their chemical structure only. One of its functionalities, however, is to solve MDGPs. `TINKER` implements the method for distance geometry proposed in [6]. Solutions are found by applying the meta-heuristic Simulated Annealing and they are successively equilibrated with Molecular Dynamics techniques.

Ever since 2006 [7,8,9,10,11,12,13,15,16,17], we have been working on a combinatorial reformulation of the MDGP. When some particular assumptions are satisfied [12], the domain of the penalty function can be discretized, and, in particular, it can be seen as a binary tree containing positions for the atoms of the considered molecule. Therefore, the optimization problem to be solved becomes combinatorial, and we refer to this combinatorial reformulation as the Discretizable MDGP (DMDGP). Both the MDGP and the DMDGP are NP-hard [7,18].

In order to solve instances of the reformulated problem, we employ a Branch & Prune (BP) algorithm [12], which is strongly based on the binary tree structure of the penalty function domain. The basic idea is to construct the binary tree during the execution of the algorithm. At each iteration, two new nodes of the

tree are added, which represent two new positions for a current atom x_i. Then, the feasibility of the two positions is checked, and branches of the tree containing infeasible positions are pruned. This pruning phase allows for reducing the binary tree very quickly, and for solving the DMDGP in a reasonable amount of time. The BP algorithm has been shown to provide very accurate solutions on sets of instances related to protein conformations.

In this paper, we present the software `MD-jeep`, which is an implementation of the BP algorithm in the C programming language. We present in detail the implementation strategies that are employed for an efficient execution of the BP algorithm. In particular, we describe the strategy we consider for reducing to the minimum possible the memory requirement and the floating-point operations. Computational experiments are shown, and implementation issues regarding future versions of the software are also discussed. `MD-jeep` is distributed under the GNU General Public License (v.2) and it can be downloaded from the following web address: `http://www.antoniomucherino.it/en/mdjeep.php`

The rest of the paper is organized as follows. In Section 2 we describe the BP algorithm and we discuss several implementation details regarding the development of `MD-jeep`. In Section 3 we present some computational experiments obtained by using the developed software, and show how the outputs it provides can be visualized by using visualization software. In Section 4 we discuss some implementation issues related to future versions of `MD-jeep`. Section 5 concludes the paper.

2 An Implementation of the BP Algorithm

2.1 The DMDGP and the BP Algorithm

Let $G = (V, E, d)$ be a weighted undirected graph, where vertices in $V = \{1, 2, \ldots, n\}$ correspond to the atoms of the considered molecule, and there is an edge between two vertices if and only if the corresponding distance is known. The weights d associated to the edges provide the numerical value of the known distances. Instances of the DMDGP must satisfy the following two assumptions, for a given ordering on V:

- $\{1, 2, 3\} \subset V$ must be a clique, and, for each atom $x_i \in V$ with rank $i > 3$, the set E must contain the three edges $(i-1, i)$, $(i-2, i)$ and $(i-3, i)$;
- for each triplet of consecutive atoms x_i, x_{i-1} and x_{i-2}, the triangular inequality on the corresponding distances must hold strictly:

$$d_{i-2,i} < d_{i-2,i-1} + d_{i-1,i}.$$

When these two assumptions are satisfied, a binary tree of atomic positions can be built and explored for solving the DMDGP (see Figure. 1). In fact, if the positions for the first $i-1$ atoms are already known, then there are only two possible positions for the atom x_i, because of the two assumptions. The binary tree can be simply built by repeating recursively the same procedure on all the

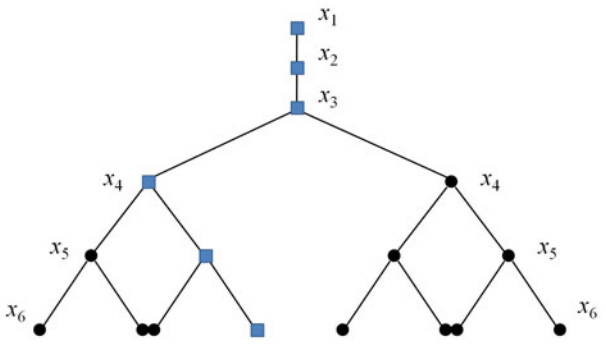


Fig. 1. An example of binary tree for a small molecule with $n = 6$ atoms. The boxes show a complete path on tree, which corresponds to a solution to the DMDGP.

atoms forming the molecule (we provide more details regarding this procedure in Section 2.4). Note that the binary tree has n layers, and all the possible positions for the same atom x_i can be found on the layer i of the tree.

The BP algorithm [12], that we use for solving instances of the DMDGP, is strongly based on the structure of this binary tree. At each iteration of the algorithm, two new positions for the current atom are computed by exploiting the distances that must be known because of the assumptions. However, other distances (which are not required by the assumptions) may also be known, and they can be used for checking the feasibility of the computed atomic positions. Therefore, during the search, branches of the binary tree are pruned as soon as one of its positions are discovered to be infeasible. This pruning phase helps in reducing the binary tree quickly, so that an exhaustive search of the remaining branches is not computationally expensive.

More details on the assumptions of the DMDGP, on the construction of the binary tree and on the BP algorithm can be found in [7,12]. Alg. 1 is a sketch of the BP algorithm. The input parameters for the algorithm are i, the current atom whose positions are searched, n, the total number of atoms forming the molecule, and d, the subset of available distances. The condition $|\,||x_i - x_j|| - d_{ij}\,| < \varepsilon, \forall j < i$, represents the pruning test that we employ for discovering infeasible atomic positions. Since a perfect match on the floating-point arithmetic of a computer machine is impossible, a tolerance ε is used (usually set to 0.001). Note that the algorithm invokes itself recursively for working on the successive atoms of the molecule. The output provided by the BP algorithm is the set of solutions to the DMDGP.

2.2 Input Arguments

`MD-jeep` is written in the C programming language. It accepts as input a list of distances on pairs of atoms of a molecule through a text file with a predefined format. In particular, since we mainly work with protein conformations, some additional information related to these molecules can also be specified in the

Algorithm 1. The BP algorithm.

```
0: BP(i, n, d)
  for (k = 1, 2) do
    compute the k^th atomic position for the i^th atom: x_i;
    check the feasibility of the atomic position x_i:
    if (| ||x_i − x_j|| − d_ij | < ε, ∀j < i) then
      the atomic position x_i is feasible;
      if (i = n) then
        a solution is found;
      else
        BP(i + 1,n,d);
      end if
    else
      the branch containing x_i is pruned;
    end if
  end for
```

input file. Such additional information are currently not used during the execution of the BP algorithm, but they are included in the output files so that other software can use them, together with the solutions provided by the BP algorithm.

The general format of each single row of the input text file must be:

```
i  j  l  u  i_atom  j_atom  i_amino  j_amino ,
```

where

- `i` the label of the first atom to which the distance refers;
- `j` the label of the second atom to which the distance refers;
- `l` the lower bound on the distance;
- `u` the upper bound on the distance;
- `i_atom` the name of the atom `i`;
- `j_atom` the name of the atom `j`;
- `i_amino` the name of the amino acid the atom `i` belongs to;
- `j_amino` the name of the amino acid the atom `j` belongs to.

Note that the names of the amino acids can be expressed in the standard 3-digit code (e.g.: *glycine* is GLY). Naturally, `i_amino` and `j_amino` regard protein conformations only. If there are no amino acids in the molecule, or if the names of the amino acids are unknown, the symbol UNK (*unknown*) can be used.

It is important to note that the BP algorithm is currently able to solve DMDGPs where exact distances d are provided, rather than lower and upper bounds. However, the decision to include in the input text file two values `l` and `u` has been taken in order to guarantee the compatibility of this format with the future versions of the software, when lower and upper bounds will be considered. Currently, only instances in which the lower bound `l` coincides with the upper bound `u` can be solved.

2.3 Instance Preprocessing

Once the input text file is read, some checks are performed before invoking the BP algorithm. First of all, we need to verify if the instance in memory is actually a DMDGP. In order to check this, the first assumption of the DMDGP is verified: for each atom x_i, the distances between x_i and the three preceding atoms must be known. Instead, we do not spend computational time for checking the second assumption, for which all the triangular inequalities on the triplets on consecutive atoms must hold strictly. We avoid this check because the probability for this assumption not being satisfied is zero. If the distances contain errors or noise, the (non-strict) triangular inequality can be checked for all the possible triplets of atoms of the molecule. This is a necessary condition for the compatibility of the distances given in input, and can be performed by `MD-jeep` by setting the appropriate option.

2.4 An Efficient Implementation

The data from the input text file are stored into a predefined data structure, `PROBL`, where each distance is represented by all the information provided on the generic row of the input file. As a consequence, an array of n elements of this data structure represents an entire instance of the DMDGP.

Let us suppose that the distance between the two atoms `i` and `j` is needed sometimes during the execution of the BP algorithm. In order to find information on the distance, it is necessary to scan the array `PROBL` until the corresponding distance is found (if it is actually included in the considered instance). To avoid scanning this array every time a distance is needed, we use a matrix of pointers which is able to provide the location in `PROBL` of the needed distance by using the two labels `i` and `j`. Of course, in this way, a bi-dimensional array of n^2 integers needs to be defined, but it is worth using this memory for speeding the algorithm up.

Before invoking the BP algorithm, the angles θ among consecutive triplets of atoms are computed, as well as the cosine of each torsion angle ω that is defined by each quadruplet of consecutive atoms (details about these computations are given in [7]). Each $\cos(\omega)$ implies the definition of two possible values for ω, which in turn implies two possible atomic positions for the corresponding atom. At each iteration of the algorithm, the two atomic positions are computed as follows. The matrix:

$$B'_i = \begin{bmatrix} -\cos\theta_{i-2,i} & -\sin\theta_{i-2,i} & 0 & -d_{i-1,i}\cos\theta_{i-2,i} \\ \sin\theta_{i-2,i}\cos\omega_{i-3,i} & -\cos\theta_{i-2,i}\cos\omega_{i-3,i} & -\sin\omega_{i-3,i} & d_{i-1,i}\sin\theta_{i-2,i}\cos\omega_{i-3,i} \\ \sin\theta_{i-2,i}\sin\omega_{i-3,i} & -\cos\theta_{i-2,i}\sin\omega_{i-3,i} & \cos\omega_{i-3,i} & d_{i-1,i}\sin\theta_{i-2,i}\sin\omega_{i-3,i} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

is considered for computing the first position for x_i, and the matrix:

$$B''_i = \begin{bmatrix} -\cos\theta_{i-2,i} & -\sin\theta_{i-2,i} & 0 & -d_{i-1,i}\cos\theta_{i-2,i} \\ \sin\theta_{i-2,i}\cos\omega_{i-3,i} & -\cos\theta_{i-2,i}\cos\omega_{i-3,i} & \sin\omega_{i-3,i} & d_{i-1,i}\sin\theta_{i-2,i}\cos\omega_{i-3,i} \\ -\sin\theta_{i-2,i}\sin\omega_{i-3,i} & \cos\theta_{i-2,i}\sin\omega_{i-3,i} & -\cos\omega_{i-3,i} & d_{i-1,i}\sin\theta_{i-2,i}\sin\omega_{i-3,i} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

is considered for the second position. Note that the only difference between B'_i and B''_i is the sign of the sine of the torsion angle $\omega_{i-3,i}$. In the following discussion, we will consider the symbol B_i for referring to any of the two matrices, and the symbols B'_i and B''_i when it will be important to discriminate between the two matrices. In order to obtain the two sets of coordinates for x_i, the two matrices are multiplied by all the preceding matrices B_j, $\forall j < i$:

$$Q'_i = B_4 \cdots B_{i-1} B'_i \qquad Q''_i = B_4 \cdots B_{i-1} B''_i \tag{2}$$

and the coordinates for the two positions for x_i are given by:

$$[Q'_i(1,4), Q'_i(2,4), Q'_i(3,4)] \qquad [Q''_i(1,4), Q''_i(2,4), Q''_i(3,4)]\,,$$

where $Q'_i(k,h)$ and $Q'_i(k,h)$ refer to the element (k,h) of the two matrices.

Let us analyze the complexity of this procedure. All the matrices that are needed for computing an atomic position can be associated to the corresponding vertex of the binary tree. For each atomic position, we need to compute the matrix B'_i or the matrix B''_i, and then we need to compute the matrix Q'_i or Q''_i, respectively. In the worst case (in which BP never prunes), we would need memory for representing the full unpruned tree: the memory requirement would be $O(2 \times 2^{n-3})$, where n is the number of considered atoms. This memory requirement is huge for large molecules. Moreover, every time a new atomic position is computed for x_i, the product among $i-3$ matrices needs to be performed. For all the atomic positions belonging to the same layer i of the binary tree, the complexity is $O(i-3)$.

In order to reduce both memory requirement and floating-point operations, we consider the following strategy. The matrices B_i are needed for computing the matrices Q_i, from which the coordinates of the atomic positions can be extracted. However, the matrix Q_i related to the atom x_i can also be computed as:

$$Q_i = Q_{i-1} B_i, \tag{3}$$

where Q_{i-1} is the matrix related to x_{i-1}. Therefore, instead of considering all the matrices B_j, with $j < i$, only the matrix Q_{i-1} can be exploited for calculating Q_i. This brings to two consequences. First, we can avoid to keep in memory all the matrices B_i, because they are never used again after the computation of Q_i. Secondly, by using the expression (3) instead of (2), the calculation of each atomic position, on any layer i of the binary tree, only needs the computation of the product between two 4×4 matrices.

If all the matrices Q_i are kept in memory, the new memory requirement in the worst case is $O(2^{n-3})$, which is still too large. Let us analyze a single iteration of the BP algorithm. Two new matrices Q'_i and Q''_i are computed for identifying the two possible positions for x_i. Both Q'_i and Q''_i depend by the preceding matrix Q_{i-1}, which, after this computation, will never be used again. Until the search is not backtracked on higher layers of the binary tree, the matrix Q_{i-1} keeps the coordinates of the atom x_{i-1}. However, when the search is backtracked, the memory for Q_{i-1} can be released and used for storing the new matrix Q_{i-1}

corresponding to the branch currently being explored. In this way, the memory requirement is decreased to $O(n-3)$.

The only evident flaw of this strategy is that found solutions are lost when the search is backtracked: once a solution is found, the algorithm can continue the exploration of the remaining branches of the binary tree, and the arrays where the matrices Q_i are stored are overwritten. For this reason, we print on text files the solutions as soon as they are found. If only the best solution is required by the user, we allocate memory for storing only another solution, where we keep the best solution ever found during the search, and we print it at the end of the execution.

2.5 Solutions in PDB Format

The solutions found by the BP algorithm are printed in text files in PDB format. Details about this format can be found on the web site of the Protein Data Bank (`http://www.rcsb.org/`). The advantage in using this format is that it is compatible with many other software for the management or for the visualization of molecules. In particular, we use RasMol (`http://www.rasmol.org/`) for visualizing the conformations obtained by the BP algorithm.

3 Experiments

`MD-jeep` was compiled by the GNU C compiler v.4.1.2 with the `-O3` flag. We performed the following experiments on an Intel Core 2 CPU 6400 @ 2.13 GHz with 4GB RAM, running Linux.

The instances we consider were generated artificially by employing a commonly used technique [3,7,20]. We chose a subset of proteins from the PDB and we extracted the backbone atoms from these molecules, i.e. the sequence of atoms $\mathrm{N}-\mathrm{C}_\alpha-\mathrm{C}$. We computed all the possible distances between pairs of such atoms, and we kept only the distances smaller than 6Å. This is done for simulating distances obtained through experiments of Nuclear Magnetic Resonance (NMR). Actually, in order to simulate real NMR data [9,10], such distances should be mainly related to hydrogen atoms, and they should be noisy. However, the aim of the presented experiments is only to show how the developed software works. For considering more realistic (and more complex) instances of the DMDGP, the BP algorithm can be adapted as described, as an example, in [9,10,11,15,17].

Table 1 shows some computational experiments on a set of generated instances, which can be downloaded at the same address as the sources of `MD-jeep`. The label given to the each instance is the label of the corresponding downloaded PDB file. In the table, n is the total number of atoms contained into the protein conformation, m is the total number of available distances, #Sol is the number of solutions found by the BP algorithm, *best* LDE is the penalty function value (1) in correspondence with the best found solution, and, finally, the CPU time is given in seconds for each experiment. All the experiments show that this implementation of the BP algorithm is able to find very accurate solutions to the

Table 1. Computational experiments on a set of artificially generated instances

instance name	n	m	#Sol	*best* LDE	CPU time
`1crn`	138	846	2	5.79e-14	0.00
`1hoe`	222	1259	2	7.26e-14	0.00
`1jk2`	270	1816	8	5.63e-14	0.01
`1a70`	291	1628	2	3.25e-13	0.00
`1fs3`	372	2209	2	1.84e-13	0.01
`1mbn`	459	3200	8	2.08e-10	0.00
`1rgs`	792	4936	8	1.55e-13	0.06
`1m40`	1224	13823	2	2.46e-13	0.03
`1bpm`	1443	9303	2	8.87e-14	0.03
`1n4w`	1610	10920	2	2.58e-13	0.04
`1mqq`	2032	13016	8	5.40e-13	0.09
`1rwh`	2265	14057	2	4.49e-14	0.12
`3b34`	2790	19563	4	4.91e-12	0.15
`2e7z`	2907	27706	2	1.22e-12	0.18

problem (the best LDE is very close to 0 in all the cases), while the computational time is only a small fraction of seconds, even when the largest instances are solved.

By setting the appropriate options, our software can provide the found solutions in PDB format. The results can then be analyzed by using visualization software for molecular conformations. In Figure 2 we show two different representations of the best found solution corresponding to the instance `1mbn`. These two pictures have been created by using the software RasMol (`http://www.rasmol.org/`), which accepts as input a protein conformation in PDB format. In the picture on the left, all the atoms of the molecule are represented by small spheres having different

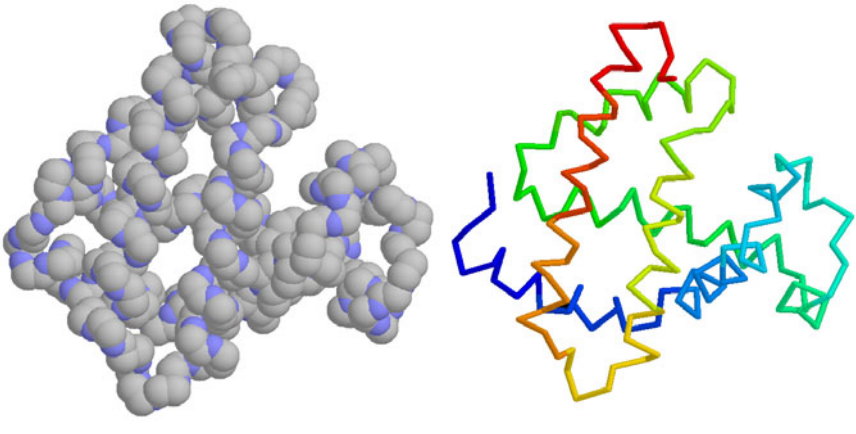


Fig. 2. Two different ways to represent with RasMol one of the solutions obtained by the BP algorithm

colors. The choice of the colors is made by RasMol by analyzing the additional information that we inserted in our output files (in this specific case, the labels for the atoms). In the picture on the right, only the trace of the protein backbone is represented.

4 New Developments

We recently proposed an extention of the DMDGP, to which we refer as the Discretizable Distance Geometry Problem (DDGP) [16]. In the DDGP in $\Re^3$, the assumptions for the discretization are relaxed:

- $\{1, 2, 3\} \subset V$ must be a clique, and, for each atom $x_i \in V$ with rank $i > 3$, there must exist three vertices j, k, h such that

$$j < i,\ k < i,\ h < i, \qquad (j, i), (k, i), (h, i) \in E, \qquad d_{jk} < d_{kh} + d_{hi}.$$

This new assumption allows for discretizing a larger subclass of distance geometry problems, which are not necessarily related to molecular conformations. In [16], a wide discussion on the main differences between the DMDGP and the DDGP is presented. We point out that the DDGP can be also defined in spaces with several dimensions.

Future versions of `MD-jeep` will also solve DDGPs. Even though the DDGP can be seen an extention of the problem that `MD-jeep` is currently able to solve, the extention of `MD-jeep` is not trivial. In particular, the strategy which is implemented for the computation of the atomic positions cannot be used anymore: such a strategy can be employed only when all torsion angles ω are defined by consecutive quadruplets of atoms. As a consequence, we need to use an alternative strategy.

Let us suppose that all the atoms with rank smaller than x_i have been already placed somewhere and that the two possible positions for x_i need to be found. By the new assumption, there are three atoms x_j, x_k and x_h that precede x_i and for which the three distances d_{ji}, d_{ki}, d_{hi} are known. Therefore, three spheres having center in x_j, x_k and x_h and radius d_{ji}, d_{ki} and d_{hi}, respectively, can be defined. Since the triangular inequality $d_{jk} < d_{kh} + d_{hi}$ holds, the intersection of these three spheres can result in two different points, which are the two possible positions for the atom x_i.

The intersection of the three spheres can be computed by solving two linear systems, as explained in [2]. As a consequence, two linear systems need to be solved for finding the coordinates of each atomic position on the binary tree. It is important to note that, differently from the strategy based on the torsion angles, round-off errors can more easily propagate when solving these linear systems. The main reason is that the new assumption for the DDGP does not require the consecutivity among the three preceding atoms used for placing the current atom x_i. Therefore, spheres having different sizes are generally intersected, and this helps the propagation of numerical errors.

In order to keep low the propagation of errors, we plan to implement two main strategies. First, spheres having very different diameters bring to the definition of

linear systems where the coefficient matrix is badly-scaled: elements on the rows or on the columns of the matrix can be much larger than the others. Therefore, we need to use a strategy for scaling the coefficient matrix before the solution of the linear system. Secondly, the lost of the consecutivity assumption allows us to choose which distances to use for building the binary tree, and which distances to use for pruning. Since more than three distances between the current atom x_i and the predecessors may be available, there are different possible combinations of distances that can be used for computing the two atomic positions. We plan to develop a strategy for finding out which is the best triplet of distances, i.e. which is the triplet of distances for which the propagation of errors is as low as possible.

5 Conclusions

We presented `MD-jeep`, an implementation of the BP algorithm for solving instances of the DMDGP. We discussed many aspects related to the development of `MD-jeep`, from the input and output formats to the strategies that are considered for reducing the memory requirements and the floating-point operations. The presented software is freely downloadable and usable, and it is distributed under the GNU General Public License (v.2). Future releases of the software will consider more general DMDGPs (for example, considering lower and upper bounds on the distances) and the recently proposed DDGP.

Acknowledgments

The authors wish to thank Sonia Cafieri for the discussions on badly-scaled matrices, and Virginia Costa and Luiz M. Carvalho for their help in the development of the presented software. We would also like to thank the Brazilian research agencies FAPESP and CNPq, the French research agency CNRS and École Polytechnique, for financial support.

References

1. Berman, H.M., Westbrook, J., Feng, Z., Gilliland, G., Bhat, T.N., Weissig, H., Shindyalov, I.N., Bourne, P.E.: The Protein Data Bank. Nucleic Acids Research 28, 235–242 (2000)
2. Coope, I.D.: Reliable Computation of the Points of Intersection of n Spheres in n-space. ANZIAM Journal 42, 461–477 (2000)
3. Biswas, P., Toh, K.-C., Ye, Y.: A Distributed SDP Approach for Large-Scale Noisy Anchor-Free Graph Realization with Applications to Molecular Conformation. SIAM Journal on Scientific Computing 30, 1251–1277 (2008)
4. Crippen, G.M., Havel, T.F.: Distance Geometry and Molecular Conformation. John Wiley & Sons, New York (1988)
5. Havel, T.F.: Distance Geometry. In: Grant, D.M., Harris, R.K. (eds.) Encyclopedia of Nuclear Magnetic Resonance, pp. 1701–1710. Wiley, New York (1995)

6. Hodsdon, M.E., Ponder, J.W., Cistola, D.P.: The NMR Solution Structure of Intestinal Fatty Acid-binding Protein Complexed with Palmitate: Application of a Novel Distance Geometry Algorithm. Journal of Molecular Biology 264, 585–602 (1996)
7. Lavor, C., Liberti, L., Maculan, N.: Discretizable Molecular Distance Geometry Problem, Tech. Rep. q-bio.BM/0608012, arXiv (2006)
8. Lavor, C., Liberti, L., Maculan, N.: Molecular Distance Geometry Problem. In: Floudas, C., Pardalos, P. (eds.) Encyclopedia of Optimization, 2nd edn., pp. 2305–2311. Springer, New York (2009)
9. Lavor, C., Mucherino, A., Liberti, L., Maculan, N.: Discrete Approaches for Solving Molecular Distance Geometry Problems using NMR Data. International Journal of Computational Biosciences (to appear 2010)
10. Lavor, C., Mucherino, A., Liberti, L., Maculan, N.: Computing Artificial Backbones of Hydrogen Atoms in order to Discover Protein Backbones. In: IEEE Conference Proceedings, International Multiconference on Computer Science and Information Technology (IMCSIT 2009), Workshop on Computational Optimization (WCO 2009), Mragowo, Poland, pp. 751–756 (2009)
11. Lavor, C., Mucherino, A., Liberti, L., Maculan, N.: An Artificial Backbone of Hydrogens for Finding the Conformation of Protein Molecules. In: Proceedings of the Computational Structural Bioinformatics Workshop (CSBW 2009), Washington DC, USA, pp. 152–155 (2009)
12. Liberti, L., Lavor, C., Maculan, N.: A Branch-and-Prune Algorithm for the Molecular Distance Geometry Problem. International Transactions in Operational Research 15(1), 1–17 (2008)
13. Liberti, L., Lavor, C., Mucherino, A., Maculan, N.: Molecular Distance Geometry Methods: from Continuous to Discrete. International Transactions in Operational Research (to appear 2010)
14. Moré, J.J., Wu, Z.: Distance Geometry Optimization for Protein Structures. Journal of Global Optimization 15, 219–223 (1999)
15. Mucherino, A., Lavor, C.: The Branch and Prune Algorithm for the Molecular Distance Geometry Problem with Inexact Distances. In: Proceedings of World Academy of Science, Engineering and Technology (WASET), International Conference on Bioinformatics and Biomedicine (ICBB 2009), Venice, Italy, pp. 349–353 (2009)
16. Mucherino, A., Lavor, C., Liberti, L.: The Discretizable Distance Geometry Problem. Optimization Letters (in revision)
17. Mucherino, A., Liberti, L., Lavor, C., Maculan, N.: Comparisons between an Exact and a MetaHeuristic Algorithm for the Molecular Distance Geometry Problem. In: ACM Conference Proceedings, Genetic and Evolutionary Computation Conference (GECCO 2009), Montréal, Canada, pp. 333–340 (2009)
18. Saxe, J.B.: Embeddability of Weighted Graphs in k-space is Strongly NP-hard. In: Proceedings of 17th Allerton Conference in Communications, Control, and Computing, Monticello, IL, pp. 480–489 (1979)
19. Schwieters, C.D., Kuszewski, J.J., Clore, G.M.: Using Xplor-NIH for NMR Molecular Structure Determination. Progress in Nuclear Magnetic Resonance Spectroscopy 48, 47–62 (2006)
20. Wu, D., Wu, Z.: An Updated Geometric Build-Up Algorithm for Solving the Molecular Distance Geometry Problem with Sparse Distance Data. Journal of Global Optimization 37, 661–673 (2007)

TADD: A Computational Framework for Data Analysis Using Discrete Morse Theory

Jan Reininghaus, David Günther,
Ingrid Hotz, Steffen Prohaska, and Hans-Christian Hege

Zuse Institute Berlin (ZIB), Takusstr. 7, 14195 Berlin, Germany
{reininghaus,david.guenther,hotz,prohaska,hege}@zib.de
http://www.zib.de

Abstract. This paper presents a computational framework that allows for a robust extraction of the extremal structure of scalar and vector fields on 2D manifolds embedded in 3D. This structure consists of critical points, separatrices, and periodic orbits. The framework is based on Forman's discrete Morse theory, which guarantees the topological consistency of the computed extremal structure. Using a graph theoretical formulation of this theory, we present an algorithmic pipeline that computes a hierarchy of extremal structures. This hierarchy is defined by an importance measure and enables the user to select an appropriate level of detail.

Keywords: Discrete Morse theory, data analysis, scalar fields, vector fields.

1 Motivation

We propose a computational framework to extract the extremal structure of scalar and vector fields on 2D manifolds embedded in $\mathbb{R}^3$. The extremal structure of a scalar field consists of critical points and separatrices – the streamlines of the gradient field that connect the critical points. The extremal structure of a vector field additionally includes periodic orbits – the streamlines that are closed.

These structures are of great interest in many applications and have a long history [2,12]. Typically, the critical points are computed by finding all zeros of the gradient or vector field. The critical points of a scalar field are classified into minima, saddles, and maxima by the eigenvalues of its Hessian, while the critical points of a vector field are classified into sinks, saddles, and sources by the eigenvalues of its Jacobian. The respective eigenvectors can be used to compute the separatrices as the solution of an autonomous ODE. For the numerical treatment of these problems and the extraction of the periodic orbits, we refer to [18,20,5].

One of the biggest challenges that such numerical algorithms face is the discrete nature of the extremal structure which necessitates a lot of binary decisions. For example, the type of a critical point depends on the sign of the eigenvalues.

K. Fukuda et al. (Eds.): ICMS 2010, LNCS 6327, pp. 198–208, 2010.

Depending on the input data, the resulting extremal structure may therefore strongly depend on the algorithmic parameters and numerical procedures.

From a topological point of view this can be quite problematic. Morse theory relates the extremal structure of a generic function to the topology of the manifold, e.g. by the Poincaré-Hopf Theorem, or by the strong Morse inequalities [13]. The topology of the manifold therefore restricts the set of the admissible extremal structures.

Forman has developed a discrete version of Morse theory [7,8] for cell complexes. A gradient or vector field is therein directly encoded in the combinatorial structure of the cell complex, and their extremal structure is defined in a combinatorial fashion. A finite cell complex of a 2D manifold can therefore only carry a finite number of combinatorial (gradient) vector fields, and their respective extremal structure is consistent with the topology of the manifold.

The basic idea of our computational framework is to compute a combinatorial (gradient) vector field that represents our input data. Its extremal structure can then be easily extracted and is always consistent with the topology of the manifold. This topological consistency greatly improves the robustness of our algorithm. In some sense it serves as an error correcting code: a single misclassification of a critical point cannot occur, as this would result in an inadmissible extremal structure.

Note that the first implementation of Forman's theory was presented by Lewiner [11]. His combinatorial (gradient) vector fields were thereby based on the construction of hypergraphs and hyperforests.

For the purpose of data analysis, the computed extremal structure is in general too complex. This is especially true if one deals with noisy data. One is therefore interested in a meaningful and consistent simplification of the extremal structure. Our framework allows for this by computing a sequence of combinatorial (gradient) vector fields that represents the input field. The user is then able to select an appropriate level of detail to efficiently analyze the data.

2 Computational Discrete Morse Theory

This section begins with a short introduction to discrete Morse theory in a graph theoretical formulation. We then formulate an optimization problem that results in a hierarchy of combinatorial (gradient) vector fields representing a given (gradient) vector field [14]. For simplicity, we restrict ourselves to 2D manifolds while the mathematical theory for combinatorial (gradient) vector fields is defined in a far more general setting [7,8].

Let C denote a finite regular cell complex [9] of a 2D manifold. In this paper, we call a cell complex regular if the boundary of each d-cell is contained in a union of $(d-1)$-cells.

Examples of such cell complexes that arise in practice are triangulations or quadrangular meshes. We first define its cell graph $G = (N, E)$, which encodes the combinatorial information contained in C in a graph theoretic setting.

The nodes N of the graph consist of the cells of the complex C and each node u^p is labeled with the dimension p of the cell it represents. The edges E of the

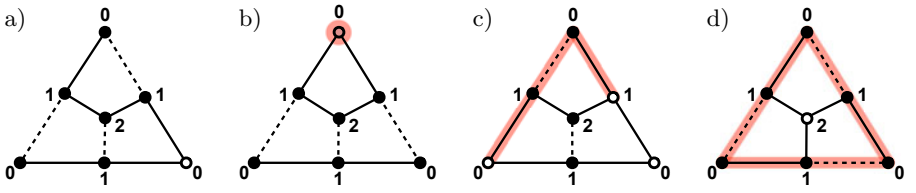


Fig. 1. Basic definitions. a) a combinatorial vector field (dashed) on the cell graph of a single triangle. The numbers correspond to the dimension of the represented cells, and matched nodes are drawn solid. b) a critical point of index 0. c) a 0-separatrix. d) an attracting periodic orbit.

graph encode the neighborhood relation of the cells in C. If the cell u^p is in the boundary of the cell w^{p+1}, then $e^p = \{u^p, w^{p+1}\} \in E$. Note that we label each edge with the dimension of its lower dimensional node.

A subset of pairwise non-adjacent edges is called a matching. Using these definitions, a *combinatorial vector field* V on a regular cell complex C can be defined as a matching of the cell graph G (see Figure 1a). The set of combinatorial vector fields on C is thereby given by the set of matchings $\mathcal{M}$ of the cell graph G.

We now define the extremal structure of a combinatorial vector field. The unmatched nodes are called critical points. If u^p is a critical point, we say that the critical point has index p. A critical point of index p is called sink ($p = 0$), saddle ($p = 1$), or source ($p = 2$). A combinatorial p-streamline is a path in the graph whose edges are of dimension p and alternate between V and its complement. A p-streamline connecting two critical points is called a p-separatrix. If a p-streamline is closed, we call it either an attracting periodic orbit ($p = 0$) or a repelling periodic orbit ($p = 1$). An illustration of the combinatorial extremal structure is shown in Figure 1.

As shown in [4], a *combinatorial gradient vector field* V^ϕ can be defined as a combinatorial vector field that contains no periodic orbits. A matching of G that gives rise to such a combinatorial vector field is called a Morse matching. The set of combinatorial gradient vector fields on C is therefore given by the set of Morse matchings $\mathcal{M}^\phi$ of the cell graph G. In the context of gradient vector fields, we refer to a critical point u^p as a minimum ($p = 0$), saddle ($p = 1$), or maximum ($p = 2$).

We now define the optimization problem that results in a meaningful combinatorial representative of our input data f. Assume that f leads to edge weights $\omega : E \to \mathbb{R}$ – we postpone their computation to Section 3.2. Assume further that f is represented well if the weight of the matching is high. We can then compute a combinatorial vector field to represent f by finding the maximum weight matching in G

$$V = \arg\max_{M \in \mathcal{M}} \omega(M). \tag{1}$$

If we want to compute a combinatorial gradient vector field V^ϕ we simply replace $\mathcal{M}$ by $\mathcal{M}^\phi$. Note that this restriction of the admissible matchings makes (1) an NP-hard problem in general [10].

Due to the matching property, the number of critical points is given by $|N| - 2|V|$. We can therefore compute a combinatorial vector field with a prescribed number of critical points by computing

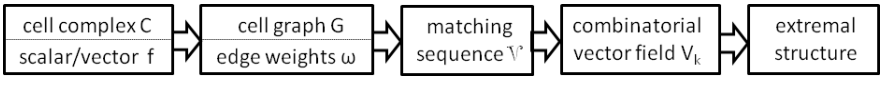
$$V_k = \arg \max_{M \in \mathcal{M}, |M|=k} \omega(M). \tag{2}$$

Let $k_0 = |V|$ denote the size of the maximum weight matching, and let $k_n = \max_{k \in \mathbb{N}} |V_k|$ denote the size of the heaviest maximum cardinality matching. From a data analysis point of view, V_{k_0} is a fine grained while V_{k_n} is the coarsest possible representation of the input data f. A hierarchy of combinatorial (gradient) vector fields $\mathcal{V}$ can now be defined as the the sequence of matchings

$$\mathcal{V} = (V_k)_{k=k_0,\ldots,k_n}. \tag{3}$$

The main task of our computational framework is to compute the sequence (3).

3 Algorithmic Pipeline



Our computational framework consists of five main parts, which the following subsections describe in detail.

3.1 Input Data

Our computational framework requires as input a finite regular cell complex of a 2D manifold embedded in $\mathbb{R}^3$ and a scalar or vector valued function f. We assume that f is defined on the 0-cells of the complex. Because we will later need data values on all cells, we extend f to the higher dimensional cells by taking the average value of the incident lower dimensional cells.

3.2 Edge Weighted Cell Graph

Using the regular cell complex, we first construct its cell graph $G = (N, E)$ as defined in Section 2. A spatial embedding $c : N \to \mathbb{R}^3$ of G can be defined using the embedding of the cell complex in $\mathbb{R}^3$. The embedding of the nodes that represent higher dimensional cells is thereby computed by taking the average of the coordinates from the incident lower dimensional cells.

We now define the edge weights $\omega : E \to \mathbb{R}$ of this graph. In Section 2 we assumed that f is represented well by a matching M, if the weight of M is large (1). Let $e^p = \{u^p, w^{p+1}\}$ denote an edge of the graph. If e^p is a matching edge it can be thought of as an arrow pointing from u^p to w^{p+1}. We therefore assign a large weight to e^p if such an arrow reflects the flow behavior f well. In this paper, we propose to measure the tangential flow of f along e^p to achieve this. Using Stokes Theorem, the edge weight ω for scalar input data f is thereby defined by

$$\omega(e^p) = f(w^{p+1}) - f(u^p), \tag{4}$$

whereas in case of vector field data f (assuming linear interpolation), its edge weight is defined by

$$\omega(e^p) = \left(f(w^{p+1}) + f(u^p)\right) \cdot \left(c(w^{p+1}) - c(u^p)\right)/2. \tag{5}$$

3.3 Matching Sequence

Given the edge weighted cell graph G, we now compute the sequence of maximum weight matchings (3). We begin with an algorithm for the vector field case $\mathcal{M}$. We then use the introduced notation to describe an algorithm that approximates (3) for the scalar field case $\mathcal{M}^\phi$.

Due to the assumed regularity of the cell complex C, the cell graph G is bipartite – a simple bipartition can be derived from the dimension of the represented cells. We can therefore employ the Hungarian method. This method is usually employed to compute V_{k_0}, but can be directly applied to compute (3).

The following presentation of the Hungarian method closely follows [16]. The basic idea is to start with V_0 and then to iteratively compute the sequence (2). In each iteration, the augmenting path of maximum weight is computed. An augmenting path of a matching V_j is a path in the graph whose start and end nodes are not covered by the matching and whose links alternate between V_j and its complement. The weight of an augmenting path is defined as the alternating sum of the weights of its links. Given an augmenting path p of maximum weight we can augment the matching V_j to get V_{j+1} by taking the symmetric difference $\triangle$ of V_j and p. For an efficient computation of the augmenting path of maximum weight we refer to [16].

To store the sequence of matchings (3) efficiently, we only store V_{k_n} and the sequence of augmenting paths that lead from V_{k_0} to V_{k_n} [14]. Because the augmenting paths of edge weighted cell graphs are usually rather short, this is a lot more efficient in our context than storing all matchings of (3) individually.

The computation of (3) for the scalar field case $\mathcal{M}^\phi$ is a lot more involved – in general it is NP-hard [10]. We therefore propose a simple approximation algorithm for this problem. The basic idea is to make use of Forman's cancellation theorem [8]. Using the graph theoretic formulation introduced in Section 2 this theorem can be stated as follows:

If two unmatched nodes are connected by a unique p-separatrix s in a Morse matching $M \in \mathcal{M}^\phi$, then $M \triangle s$ is a Morse matching.

The pseudo code for our approximation algorithm is shown in Algorithm 1. The input consists of the cell Graph G and its edge weights ω. The output consists of $V_{k_n}^\phi$ and a list of augmenting paths. Together, these can be used to reconstruct an arbitrary element of the sequence (3). The subfunction $getMaxUniqueSeparatrix(\ldots)$ returns the unique p-separatrix of maximum weight of the saddle u^1. The 2D manifold structure of the cell graph G implies that at most four p-streamlines emanate from u^1 and that these cannot split. The subfunction $getMaxUniqueSeparatrix(\ldots)$ therefore simply iterates all (up to four) p-separatrices that start in u^1. It then checks for uniqueness by comparing their end nodes and returns the unique p-separatrix with the largest

weight. If there is no unique p-separatrix at all, then an empty path is returned with a weight of $-\infty$. Note that there are always two 0-streamlines emanating from u^1 and that these are always 0-separatrices. The 1-streamlines that emanate from u^1 however may end in the boundary of the manifold.

Algorithm 1. MorseMatchingSequence(G, ω)

Output: $AugPaths$, $V_{k_n}^{\phi}$
1: $M \leftarrow \emptyset$, $AugPaths \leftarrow \emptyset$, $heap \leftarrow \emptyset$
2: **for all** $u^1 \in N$ **do**
3: $(path, weight) \leftarrow getMaxUniqueSeparatrix(G, \omega, M, u^1)$
4: $heap.push(u^1, weight)$
5: **while** $heap \neq \emptyset$ **do**
6: $(u^1, weight) \leftarrow heap.pop()$
7: $(path, weight) \leftarrow getMaxUniqueSeparatrix(G, \omega, M, u^1)$
8: $(nextNode, nextWeight) \leftarrow heap.top()$
9: **if** $weight \geq nextWeight$ **then**
10: $M \leftarrow M \triangle path$
11: **if** $weight < 0$ **then**
12: $AugPaths.push(path)$
13: **else if** $-\infty < weight$ **then**
14: $heap.push(u^1, weight)$
15: $V_{k_n}^{\phi} \leftarrow M$

Line 1 initializes M as the empty matching, the list of augmenting paths $AugPaths$, and a priority queue $heap$. All nodes representing 1-cells are then inserted into this queue, ordered by the weight of their heaviest unique p-separatrix, in Lines 2-4. We then iterate over the queue (Line 5), remove the top element of the $heap$ (Line 6) and compute its heaviest unique p-separatrix (Line 7). This is necessary, as previous iterations may have affected this node. We now check whether this p-separatrix is the heaviest of all available unique p-separatrices. Assuming that augmenting the matching only decreases the weight returned by $getMaxUniqueSeparatrix(\ldots)$, it suffices to check whether the weight of u^1 is larger than the weight of the next element of the $heap$ (Lines 8-9). If this is the case, we augment the matching M by taking the symmetric difference of M and $path$ (Line 10), and store the augmenting path if its weight is negative (Line 11-12). Otherwise, we reinsert u^1 into the $heap$ with its new weight if it is larger than $-\infty$ (Line 13-14). When the $heap$ is empty the algorithm terminates and returns an approximation of the heaviest maximum cardinality Morse matching $V_{k_n}^{\phi}$.

3.4 Combinatorial (Gradient) Vector Field

The heaviest maximum cardinality (Morse) matching V_{k_n} and the list of augmenting paths computed in the Section 3.3 allows for the reconstruction of an arbitrary element of the sequence (3). Each matching can be restored by iteratively taking the symmetric difference of V_{k_n} with the augmenting paths. This

enables the user to select a combinatorial (gradient) vector field with prescribed number of critical points.

Alternatively, we can make use of the associated weight of each matching as an importance measure. The user can set a fraction $\theta \in [0,1]$ to select a combinatorial (gradient) vector field with a weight as close as possible to $\omega(V_{k_0}) + \theta\,(\omega(V_{k_n}) - \omega(V_{k_0}))$. This approach can be useful in dealing with noisy data. Noise induces a very complex extremal structure. The augmenting paths corresponding to the spurious extremal structure, however, have a very large weight. Setting θ to a small value therefore removes all spurious extremal structures while the dominant structure remains unchanged.

Note that taking the symmetric difference of a matching M with an augmenting path corresponds to the cancellation of a pair of critical points. In the scalar case, the weight of such an augmenting path equals the difference of the scalar values of these two critical points. The above importance measure is thereby closely related to the persistence measure in [6].

3.5 Extremal Structure

Given a combinatorial (gradient) vector field, we can now extract its extremal structure. As the critical points are the unmatched nodes u^p they can be easily extracted. The classification into sources, saddles, sinks (minima, saddles, maxima) is given by p. To compute all separatrices, we iterate over all saddles u^1 and compute the incident p-separatrices.

In the vector case, we also need to extract all periodic orbits. Due to the $2D$ manifold structure, p-streamlines can not split when the first node is a 1-node. Therefore, the extraction of periodic orbits is quite simple. First, we iterate over all 1-nodes. Given a node u^1, we start the computation of the p-streamlines that emanates at u^1. Each streamline is continued as long as the following node w^1 is not yet labeled, in which case it is labeled with u^1. If the label of w^1 equals u^1 we add w^1 to a set of seed points. We then iterate over all seed points to compute all periodic orbits.

4 Examples

In this section, we present three applications of the computational framework presented in Section 3. The framework was implemented as a module in the visualization and data analysis software Amira [17]. It can be made available to researchers for evaluation purposes. The integrated visualization capability of Amira was used to assess the relevance of the computed extremal structures and the practical quality of the approximation Algorithm 1.

The visualization of the abstract representation of the input data as a matching in an edge weighted graph proved to be very useful in the development of correct and efficient algorithms.

We first illustrate the ability to extract the extremal structure of a scalar field where noise is present. We then apply our framework to a vector field on a 2D manifold to show the physical relevance of the hierarchy of extremal structures

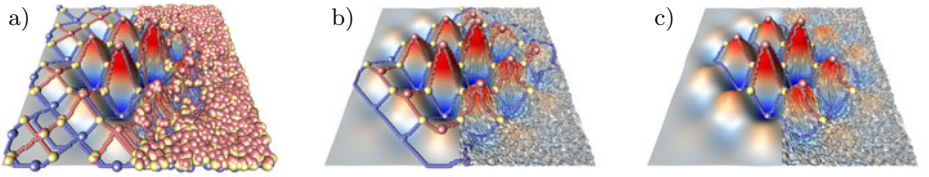


Fig. 2. Synthetic noisy scalar field. Extremal structure of a) $V^{\phi}_{k_0}$, b) $V^{\phi}_{k_n-23}$ and c) $V^{\phi}_{k_n-11}$. Minima, saddles and maxima are depicted as blue, yellow and red spheres, while 0-separatrices and 1-separatrices are shown as blue and red lines.

(3). The paper is concluded with an application to extremal lines of curvature fields of a discrete $2D$ manifold. All examples were computed on a workstation containing an Intel Core i7 860 CPU.

4.1 A Synthetic Noisy Scalar Field

To illustrate the robustness of our data analysis framework, we applied it to a synthetic data set depicted as a height field in Figure 2. The data set was produced by sampling the analytic function $f : [-1, 1]^2 \to \mathbb{R}$

$$f(x, y) = \sin(10\,x)\,\sin(10\,y)\,e^{-3\,(x^2+y^2)} \tag{6}$$

on a uniform triangulation with $16k$ vertices. We then added uniform noise in the the range of $[-0.05, 0.05]$ to the sub domain $[0, 1] \times [-1, 1]$. We applied our algorithmic pipeline presented in Section 3 to this input data. The runtime for the computation of (3) using Algorithm 1 was less than a second on a standard workstation. Figure 2 shows the extremal structure of the initial combinatorial gradient field $V^{\phi}_{k_0}$, and two elements, $V^{\phi}_{k_n-23}$ and $V^{\phi}_{k_n-11}$, of the matching sequence (3). As can be seen in Figure 2a, $V^{\phi}_{k_0}$ includes the extremal structure induced by the noise. The simplified combinatorial gradient fields, however, only contain the dominant extremal structure present in f.

4.2 A Vector Field from Biofluid Mechanics

Figure 3 depicts a surface velocity field of a simulation of blood flow through a cerebral aneurysm done by the Biofluid Mechanics Lab of the Charité - Universitätsmedizin Berlin [3]. The cell graph of the triangulation consists of $60k$ nodes. The runtime for the computation of (3) using a simple implementation of the Hungarian method was about 30 minutes.

The critical points in this vector field are stagnation points and thus of interest for the flow analysis. Our algorithm delivers a hierarchy of extremal structures which captures the dominant nature of the flow (see Figure 3 bottom-left). The blood enters the aneurysm at the bottom, and leaves it horizontally. This behavior is found by our algorithm and the global separation on the surface is extracted. This reduced flow structure may serve as a basis when comparing different cerebral aneurysms.

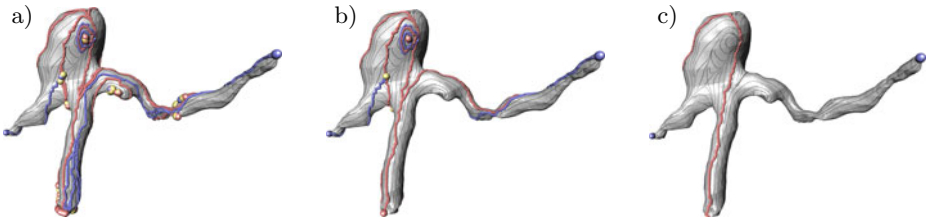


Fig. 3. Vector field from biofluid mechanics. The vector field is visualized using the streamline seeding technique described in [15]. The extremal structures of a) V_{k_0}, b) V_{k_n-4} and c) V_{k_n} are shown. Sinks, saddles and sources are depicted as blue, yellow and red spheres. 0-separatrices and attracting periodic orbits are depicted as blue lines, while 1-separatrices and repelling periodic orbits are shown as red lines.

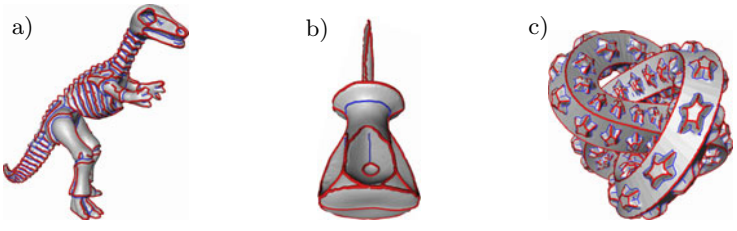


Fig. 4. Extremal lines in curvature fields. For all surface models, the first and second principal curvatures κ_1 and κ_2 are computed. a), b) and c) show the most dominant parts of 0-separatrices (blue) in κ_1 and 1-separatrices (red) in κ_2.

Table 1. Running time for Algorithm 1 with separatrix persistence [19] computation

Surface Model	triangles	nodes in G	edges in G	time (sec)
screwdriver	54300	162902	325800	1
dinosaur	112384	337154	674304	2
knot	957408	2872224	5744448	24

4.3 Extremal Lines in Curvature Fields

Figure 4 illustrates the extraction of extremal lines in curvature fields of different surfaces. As described in [19], each point of a p-separatrix can be assigned an importance value, called separatrix persistence. The main idea of separatrix persistence is to measure the strength of monotony breaks with respect to the sequence of combinatorial gradient vector fields (3). For details how to incorporate this measure into our computational pipeline, we refer to [19].

Separatrix persistence allows to discriminate spurious from dominant extremal lines. These lines are shown in Figure 4. Note that a reduction to the most dominant extremal parts destroys the connectivity of the extremal structure. The total running time for the computation of (3) using Algorithm 1 and the computation of separatrix persistence is shown in Table 1. The worst case complexity

of Algorithm 1 is $O(n^3)$, where n denotes the number of edges in the triangulation. However, the empirical running time for practical applications is almost linear. The models are provided by Aim@Shape [1].

Acknowledgements

The authors would like to thank the anonymous reviewers for their valuable comments that significantly improved the quality of this paper. The first author and third author are funded by the DFG Emmy-Noether research program, while the second author is funded by the Max-Planck Institute of Biochemistry, Martinsried. Finally, we would like to thank Jens Kasten, Michael Koppitz, Britta Weber, Daniel Baum, and Tino Weinkauf for many fruitful discussions on this topic.

References

1. Aim@Shape, http://shapes.aim-at-shape.net/
2. Cayley, A.: On contour and slope lines. The London, Edinburg and Dublin Philosophical Magazine and Journal of Science 18, 264–268 (1859)
3. Cebral, J., Castro, M., Appanaboyina, S., Putman, C., Millan, D., Frangi, A.: Efficient pipeline for image-based patient-specific analysis of cerebral aneurysm hemodynamics: technique and sensitivity. IEEE Transactions on Medical Imaging 24(4), 457–467 (2005)
4. Chari, M.K.: On discrete Morse functions and combinatorial decompositions. Discrete Math. 217(1-3), 101–113 (2000)
5. Chen, G., Mischaikow, K., Laramee, R.S., Pilarczyk, P., Zhang, E.: Vector field editing and periodic orbit extraction using Morse decomposition. IEEE Transactions on Visualization and Computer Graphics 13(4), 769–785 (2007)
6. Edelsbrunner, H., Harer, J., Zomorodian, A.: Hierarchical Morse complexes for piecewise linear 2-manifolds. Discrete Computational Geometry 30, 87–107 (2003)
7. Forman, R.: Combinatorial vector fields and dynamical systems. Mathematische Zeitschrift 228(4), 629–681 (1998)
8. Forman, R.: Morse theory for cell complexes. Advances in Mathematics 134, 90–145 (1998)
9. Hatcher, A.: Algebraic Topology. Cambridge University Press, Cambridge (2002)
10. Joswig, M., Pfetsch, M.E.: Computing optimal Morse matchings. SIAM J. Discret. Math. 20(1), 11–25 (2006)
11. Lewiner, T.: Geometric discrete Morse complexes. Ph.D. thesis, Department of Mathematics, PUC-Rio (2005), advised by H. Lopes, G. Tavares
12. Maxwell, J.C.: On hills and dales. The London, Edinburg and Dublin Philosophical Magazine and Journal of Science 40, 421–425 (1870)
13. Milnor, J.: Topology from the differentiable viewpoint. Univ. Press Virginia (1965)
14. Reininghaus, J., Hotz, I.: Combinatorial 2d vector field topology extraction and simplification. In: Pascucci, V., Tricoche, X., Hagen, H. (eds.) Topology in Visualization 2009 (to appear 2010)
15. Rosanwo, O., Petz, C., Prohaska, S., Hotz, I., Hege, H.C.: Dual streamline seeding. In: Eades, P., Ertl, T., Shen, H.W. (eds.) Proceedings of the IEEE Pacific Visualization Symposium, pp. 9–16 (2009)

16. Schrijver, A.: Combinatorial Optimization. Springer, Heidelberg (2003)
17. Stalling, D., Westerhoff, M., Hege, H.C.: Amira: A highly interactive system for visual data analysis. In: The Visualization Handbook, pp. 749–767 (2005)
18. Weinkauf, T.: Extraction of Topological Structures in 2D and 3D Vector Fields. Ph.D. thesis, University Magdeburg and Zuse Institute Berlin (2008)
19. Weinkauf, T., Günther, D.: Separatrix persistence: Extraction of salient edges on surfaces using topological methods. Computer Graphics Forum (Proc. SGP 2009) 28(5), 1519–1528 (2009)
20. Wischgoll, T., Scheuermann, G.: Detection and visualization of closed streamlines in planar flows. IEEE Transactions on Visualization and Computer Graphics 7(2), 165–172 (2001)

Introduction to Normaliz 2.5

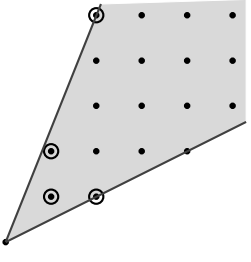
Winfried Bruns, Bogdan Ichim, and Christof Söger

Winfried Bruns, Universität Osnabrück, FB Mathematik/Informatik,
49069 Osnabrück, Germany
wbruns@uos.de
Bogdan Ichim, Institute of Mathematics "Simion Stoilow" of the Romanian Academy,
C.P. 1-764, 010702 Bucharest, Romania
bogdan.ichim@imar.ro
Christof Söger, Universität Osnabrück, FB Mathematik/Informatik,
49069 Osnabrück, Germany
csoeger@uos.de

Abstract. In this paper we introduce the version 2.5 of Normaliz , a program for the computation of Hilbert bases of rational cones and the normalizations of affine monoids. It may also be used for solving diophantine linear systems of inequalities, equations and congruences. We present some of the new features of the program, as well as some recent achievements.

1 Introduction

Let C be a finitely generated pointed rational cone in $\mathbb{R}^d$, i.e. the set of linear combinations $\sum_{i=1}^n a_i v_i$ of finitely many integral vectors v_i with nonnegative real coefficients a_i such that $x, -x \in C$ is only possible for $x = 0$. The set of lattice points $C \cap \mathbb{Z}^d$ is an affine monoid with unique finite minimal system of generators, called its *Hilbert basis*. For the theory of affine monoids and the notions of commutative algebra used in this paper we refer the reader to [3]. In the figure the Hilbert basis is marked by open circles.



The program Normaliz [5], version 2.5 (in the following simply called Normaliz), is mainly a tool for computing Hilbert bases. Note that this task is NP-hard [11].

Several related computations are also integrated. Using Normaliz, one may compute the following:

(1) the Hilbert basis and the support hyperplanes of a rational cone. The cone may be given by:
  (i) a system of generators;
  (ii) a linear system of inequalities, equations and congruences (congruences since version 2.5);
  (iii) the binomial equations of the (monoid) generators (since version 2.5).

K. Fukuda et al. (Eds.): ICMS 2010, LNCS 6327, pp. 209–212, 2010.

(2) the lattice points and the support hyperplanes of an integral polytope;
(3) the generators of the integral closure of the Rees algebra of a monomial ideal $I \subseteq K[X_1, \ldots, X_n]$ and the generators of the integral closure of I.

If the associated monoid is homogeneous in a certain sense, then one may also compute the h-vector and Hilbert polynomial of the monoid.

For the algorithms implemented see [9] (starting with version 1.0), [6] (introduced in version 2.0) and [4] (some of the recent additions in version 2.5). A description of the user interface (version 2.2) is contained in [7].

2 Interactions with other Software Systems

We provide the library `normaliz.lib` that make Normaliz accessible from Singular and also the package `Normaliz.m2` [8] that make Normaliz accessible from Macaulay2. Thus Singular or Macaulay2 can be used as a comfortable environment for the work with Normaliz, and, moreover, Normaliz can be applied directly to objects belonging to the classes of toric rings and monomial ideals.

Thanks to Andreas Paffenholz, Normaliz has been made accessible from polymake (see [12]).

3 New Features of the Program

In the following, we present the changes in the new version of Normaliz.

(1) First, there are some deep changes in the *implementation* of the program.
  (i) The new version of Normaliz has full support for parallel computing. Much better computation times can now be obtained on a multi-core processor system.
  (ii) Memory usage has been optimized.
(2) We have also done some *algorithmic improvements*, which have allowed us to solve some computationally difficult questions (presented in [4]).
  (i) We have introduced a new method for computing the Hilbert basis using a partial triangulation (see [4] for details).
  (ii) A new algorithm for computing the support hyperplanes has been implemented. It uses Fourier-Motzkin elimination recursively and allows computations in cones with many support hyperplanes and big triangulations.
  (iii) The shelling algorithm [6] has been improved.
(3) We have added a *graphical interface* called jNormaliz [2]. This interface is written in Java. It allows us to combine the good portability (on different operating systems) of the graphical elements provided by Java with the computational advantages of the C++ implementation of Normaliz.
(4) The *input modes* have been augmented.
  (i) We have introduced a new format for the input files allowing us to combine systems of inequalities, equations and congruences. (In the previous versions only a system of inequalities or a system of equations was allowed as input). The new format is fully compatible with the old format.

(ii) We have also added a new type of input called "lattice ideal". The input is a matrix of vectors in $\mathbb{Z}^n$, representing binomial equations of the monoid generators (see [3] Section 4.C for details).

(5) Now a better and finer *specialization of the computations performed* is available.
   (i) The user can now chose to compute only the height 1 elements of the Hilbert basis of a homogeneous monoid. This is much faster than computing all the elements of the Hilbert basis and allows the fast computation of the lattice points of an integral polytope.
   (ii) The Hilbert basis can be computed using only a partial triangulation. This is very fast in some particular but interesting cases (like the one presented in Section 4).
   (iii) Specific computations in cones with a large number of support hyperplanes and big triangulations can now be made.

4 One Computational Example

We call a monoid $M \subseteq \mathbb{Z}^d$ *normal* if $M = C \cap$ lattice(M) where C is the cone generated by M. After an identification lattice$(M) \cong Z^r$, checking normality of M amounts to verifying whether the Hilbert basis of C (with respect to $\mathbb{Z}^r$) is contained in M.

An $r_1 \times r_2 \times \cdots \times r_N$ *contingency table* is a function $T : \{1, \ldots, r_1\} \times \cdots \times \{1, \ldots, r_N\} \to \mathbb{Z}_+$ where $\mathbb{Z}_+$ denotes the nonnegative integers. The j-th $(N-1)$-*marginal* T_j of T is the $r_1 \times \cdots \times r_{j-1} \times r_{j+1} \times \cdots \times r_N$ contingency table defined by $T_j(i_1, \ldots, i_{j-1}, i_{j+1}, \ldots, i_N) = \sum_{k=1}^{r_j} T(i_1, \ldots, i_{j-1}, k, i_{j+1}, \ldots, i_N)$.

The $r_1 \times r_2 \times \cdots \times r_N$ contingency tables form the monoid $\mathcal{O}$ of integral points in the nonnegative orthant of $\mathbb{R}^D$ where $D = r_1 \cdots r_N$. The assignment $T \mapsto (T_1, \ldots, T_N)$ is a monoid homomorphism $\mathcal{M}$ from $\mathcal{O}$ into the monoid of nonnegative integer points in $\mathbb{R}^{d_1 + \cdots + d_N}$ where $d_j = r_1 \cdots r_{j-1} r_{j+1} \cdots r_N$. The image $\mathcal{M}(\mathcal{O})$ is called the *monoid derived from* $r_1 \times r_2 \times \cdots \times r_N$ *contingency tables* (by taking line sums). For the role of these monoids and their normality in algebraic statistics we refer the reader to Drton, Sturmfels and Sullivant [10] and Sullivant [15].

Normality of monoids derived from $r_1 \times r_2 \times \cdots \times r_N$ contingency tables by taking $N-1$-marginals was settled almost completely by Ohsugi and Hibi [14]. Using Normaliz one can now show computationally, that the monoids of the missing cases $5 \times 5 \times 3$, $5 \times 4 \times 3$ and $4 \times 4 \times 3$ are normal. For details we refer the reader to [4]. Another (independent) computational solution to this problem can be found using the software LattE for tea, too [13].

Note that this normality problem cannot be settled directly by using previously available software such as Normaliz version 2.2 or `4ti2` version 1.3.2 [1]. Both codes fail to return an answer due to time and to memory requirements of intermediate computations.

Acknowledgement. B. Ichim was partially supported by CNCSIS grant RP-1 no. 7/01.07.2009 during the preparation of this work.

References

[1] 4ti2 team. 4ti2 – A software package for algebraic, geometric and combinatorial problems on linear spaces, http://www.4ti2.de

[2] Almendra, V., Ichim, B.: jNormaliz. A graphical interface for Normaliz, http://www.math.uos.de/normaliz

[3] Bruns, W., Gubeladze, J.: Polytopes, rings, and K-theory. In: Springer Monographs in Mathematics (2009)

[4] Bruns, W., Hemmecke, R., Ichim, B., Köppe, M., Söger, C.: Challenging computations of Hilbert bases of cones associated with algebraic statistics. Exp. Math. (in Press)

[5] Bruns, W., Ichim, B., Söger, C.: Normaliz. Algorithms for rational cones and affine monoids, http://www.math.uos.de/normaliz

[6] Bruns, W., Ichim, B.: Normaliz: algorithms for affine monoids and rational cones. J. Algebra (in Press)

[7] Bruns, W., Ichim, B.: Introduction to Normaliz 2.2. In: Breaz, D., et al. (eds.) Proceedings of ICTAMI 2009, pp. 113–132. Acta Univ. Apulensis (2009)

[8] Bruns, W., Kämpf, G.: A Macaulay 2 interface for Normaliz (preprint), http://arxiv.org/abs/0908.1308

[9] Bruns, W., Koch, R.: Computing the integral closure of an affine semigroup. Univ. Iagel. Acta Math. 39, 59–70 (2001)

[10] Drton, M., Sturmfels, B., Sullivant, S.: Lectures on algebraic statistics. In: Oberwolfach Seminars, vol. 39. Birkhäuser, Basel (2009)

[11] Durand, A., Hermann, M., Juban, L.: On the complexity of recognizing the Hilbert Basis of a linear diophantine system. In: Kutyłowski, M., Wierzbicki, T., Pacholski, L. (eds.) MFCS 1999. LNCS, vol. 1672, pp. 92–102. Springer, Heidelberg (1999)

[12] Joswig, M., Müller, B., Paffenholz, A.: Polymake and lattice polytopes. In: Krattenthaler, C., et al. (eds.) DMTCS Proc. AK, Proceedings of FPSAC 2009, pp. 491–502 (2009)

[13] LattE for tea, too team. LattE for tea, too – A joint source code distribution of the two software packages Latte macchiato und 4ti2, http://www.latte-4ti2.de

[14] Ohsugi, H., Hibi, T.: Toric ideals arising from contingency tables. In: Commutative Algebra and Combinatorics, Ramanujan Mathematical Society Lecture Note Series, vol. 4, pp. 87–111 (2006)

[15] Sullivant, S.: Normal binary graph models. Ann. Inst. Stat. Math. (to appear) (preprint) (arXiv:0906.178)

Computer Algebra Methods in Tropical Geometry

Thomas Markwig

Fachbereich Mathematik, Technische Universität Kaiserslautern
Postfach 3049, 67653 Kaiserslautern, Germany
keilen@mathematik.uni-kl.de
http://www.mathematik.uni-kl.de/~keilen

Tropical geometry is a young field of mathematics which allows to study properties of objects from algebraic geometry with the aid of methods from discrete mathematics, like convex geometry and combinatorics. There are different ways to introduce tropical varieties and to derive the connection between these and their algebraic counterparts. We use a way, where the connection is concrete and where Gröbner basis techniques can be used to to establish it in both directions.

For this we consider the field $\mathbb{K}$ of *Puiseux series*. The elements of $\mathbb{K}$ are generalised power series of the form $a = c_1 \cdot t^{\alpha_1} + c_2 \cdot t^{\alpha_2} + \ldots$, where $c_i \in \mathbb{C}$ and the $\alpha_i \in \mathbb{Q}$ have a common denominator and $\alpha_1 < \alpha_2 < \alpha_3 < \ldots$. This field is algebraically closed and thus suitable for algebraic geometry. Moreover, it comes with a *non-archimedean valuation* $\mathrm{val} : \mathbb{K}^* \longrightarrow \mathbb{Q} : a \mapsto \alpha_1$. We call $\mathrm{lc}(a) := c_1$ the leading coefficient of a.

We are interested in studying the vanishing locus

$$V(I) := \{\mathbf{p} \in (\mathbb{K}^*)^n \mid f(\mathbf{p}) = 0 \; \forall \; f \in I\}$$

of an ideal $I \trianglelefteq \mathbb{K}[\mathbf{x}]$, $\mathbf{x} = (x_1, \ldots, x_n)$. For this we degenerate our variety $V(I)$ by applying componentwise the valuation map, and we call

$$\mathrm{Trop}(I) := \{-\mathrm{val}(\mathbf{p}) \mid \mathbf{p} \in V(I)\} \subset \mathbb{Q}^n$$

the *tropical variety* defined by I. The definition gives a clear connection between the algebraic variety $V(I)$ and its tropical counterpart $\mathrm{Trop}(I)$, but it does not reveal its piecewise linear structure and it gives no clue on how to compute $\mathrm{Trop}(I)$. The key to both questions lies in the lifting lemma, which can be attributed in the hypersurface case to Kapranov (see [3]), and which is otherwise connected to many people (see e.g. [9]).

Theorem 1 (Lifting Lemma). $\mathrm{Trop}(I) = \{\omega \in \mathbb{Q}^n \mid \mathrm{in}_\omega(I) \textit{ is monomial free}\}$.

In this statement $\mathrm{in}_\omega(I)$ denotes some kind of *initial ideal* that we still have to introduce. We first want to emphasise that this description suggests that Gröbner basis methods might be a suitable to study $\mathrm{Trop}(I)$.

In this note we want to address the following computational questions.

A) How can we compute $\mathrm{in}_\omega(I)$?
B) How can we lift a point ω from $\mathrm{Trop}(I)$ to a point $\mathbf{p} \in V(I)$?
C) How can we compute $\mathrm{Trop}(I)$?

K. Fukuda et al. (Eds.): ICMS 2010, LNCS 6327, pp. 213–216, 2010.

1 How to Compute $\mathrm{in}_\omega(I)$?

For a point $\omega \in \mathbb{Q}^n$ and a polynomial $f = \sum_\beta a_\beta \cdot \mathbf{x}^\beta$ with $\mathbf{x}^\beta = x_1^{\beta_1} \cdots x_n^{\beta_n}$ we define the *ω-degree* $\deg_\omega(f) := \max\{-\mathrm{val}(a_\beta) + \beta \cdot \omega \mid a_\beta \neq 0\}$ with $\beta \cdot \omega := \sum_{i=1}^n \beta_i \cdot \omega_i$, and we then define the *initial form* of f with respect to ω as $\mathrm{in}_\omega(f) := \sum_{-\mathrm{val}(a_\beta)+\beta\cdot\omega=\deg_\omega(f)} \mathrm{lc}(a_\beta) \cdot \mathbf{x}^\beta \in \mathbb{C}[\mathbf{x}]$ and the *initial ideal* of I with respect to ω as $\mathrm{in}_\omega(I) = \langle \mathrm{in}_\omega(f) \mid f \in I \rangle \trianglelefteq \mathbb{C}[\mathbf{x}]$.

We can use Gröbner basis techniques to compute the initial ideal of I with respect to ω. For this we have to consider the uniformising parameter t in $\mathbb{K}$ as an additional variable, and we need a suitable monomial ordering on monomials in t and $\mathbf{x}$. We fix some monomial ordering $>$ on the monomials in $\mathbf{x}$ and define

$$t^\alpha \cdot \mathbf{x}^\beta \ >_\omega \ t^{\alpha'} \cdot \mathbf{x}^{\beta'} \ :\Longleftrightarrow \ \deg_\omega\left(t^\alpha \cdot \mathbf{x}^\beta\right) > \deg_\omega\left(t^{\alpha'} \cdot \mathbf{x}^{\beta'}\right) \quad \text{or}$$
$$\left(\deg_\omega\left(t^\alpha \cdot \mathbf{x}^\beta\right) = \deg_\omega\left(t^{\alpha'} \cdot \mathbf{x}^{\beta'}\right) \text{ and } \mathbf{x}^\beta > \mathbf{x}^{\beta'}\right).$$

It is then straight forward to see the following result (see [7]).

Theorem 2. *Given $f_1, \ldots, f_k \in \mathbb{Q}[t, \mathbf{x}]$, $I = \langle f_1, \ldots, f_k \rangle \trianglelefteq \mathbb{K}[\mathbf{x}]$ and a Gröbner basis G of $J = \langle f_1, \ldots, f_k \rangle \trianglelefteq \mathbb{Q}[t, \mathbf{x}]$ w.r.t. $>_\omega$. Then $\mathrm{in}_\omega(I) = \langle \mathrm{in}_\omega(g) \mid g \in G \rangle$.*

Multiplying the generators by some power of t and applying a transformation of the form $t \mapsto t^N$ we can assume that I is generated by elements in $\mathbb{C}[[t]]$, and Theorem 2 holds with $\mathbb{Q}[t]$ replaced by $\mathbb{C}[[t]]$ as well. But from a practical point of view we can only deal with polynomial coefficients over $\mathbb{Q}$. Thus the above statement is good enough to treat the general situation in practice.

2 Two Interesting Cases of Tropical Varieties

2.1 $I = \langle f \rangle$ Defines a Hypersurface

Then $\mathrm{in}_\omega(I)$ is generated by $\mathrm{in}_\omega(f)$, and $\mathrm{Trop}(I)$ consists of those points ω for which the initial form is not a monomial. Tropicalising the polynomial f to the piecewise linear function $\mathrm{trop}(f) : \omega \mapsto -\mathrm{val}(a_\beta) + \beta \cdot \omega$ the initial form $\mathrm{in}_\omega(f)$ is not a monomial if and only if $\mathrm{trop}(f)$ is not differentiable at ω. Thus $\mathrm{Trop}(I)$ is a *piecewise linear hypersurface* in $\mathbb{Q}^n$ in this case.

Using the library `tropical.lib` [5] from `Singular` we can visualise tropical curves ($n = 2$). Using the software `TropicalSurface` by Lars Allermann [1] one can also visualise tropical surfaces ($n = 3$).

2.2 Constant Coefficient Case: I Is Generated by Polynomials in $\mathbb{Q}[\mathbf{x}]$

By Theorem 2 it then suffices to compute Gröbner bases over $\mathbb{Q}[\mathbf{x}]$ and to check if their initial forms are monomial free. If we call two points ω and ω' in $\mathbb{Q}^n$ equivalent if the corresponding initial ideals of I coincide, then the equivalence classes are open convex cones forming the Gröbner fan of I. An immediate corollary is thus the following.

Corollary 1. *If I is generated by polynomials in $\mathbb{Q}[\mathbf{x}]$ then $\mathrm{Trop}(I)$ is a subfan of the Gröbner fan of I.*

3 How to Lift a Point from Trop(I) to $V(I)$?

We describe an algorithm (see [6]) for lifting a point $\omega \in \mathrm{Trop}(I)$ to a point $\mathbf{p} \in V(I)$ such that $-\mathrm{val}(\mathbf{p}) = \omega$. The algorithm induces a proof of the inclusion $\{\omega \in \mathbb{Q}^n \mid \mathrm{in}_\omega(I) \text{ is monomial free}\} \subseteq \mathrm{Trop}(I)$ in the Lifting Lemma.

The idea is to intersect $V(I)$ by certain linear forms , so that $V(I)$ is zero-dimensional. Then one constructs the point inductively. For this we choose a zero $(c_1, \ldots, c_n)$ of $\mathrm{in}_\omega(I)$ in $(\mathbb{C}^*)^n$ and set $p_i = c_i \cdot t^{\omega_i} + h.o.t.$. In order to compute the higher order terms, we transform the ideal basically by $x_i \mapsto t^{-\omega_i} \cdot (x_i + c_i)$, continue by computing a point ω' in the tropical variety of the transformed ideal I' and go on with this new ideal I' and this new point ω'.

The algorithm is a generalisation of the Newton-Puiseux algorithm for computing parametrisations of plane curve singularities respectively generalisations thereof for space curves (see [8] and [2]). However, in practice one has to deal with a couple of problems:

- We can only compute the lifting up to a given order.
- To reduce to the zero-dimensional case one has to test if the linear forms are general enough. This is done by a dimension computation.
- Even if we start with polynomials over $\mathbb{Q}[t, \mathbf{x}]$ it will be necessary to pass to field extensions of $\mathbb{Q}$ in order to compute zeros of the initial ideal.
- The recursion step may stop in some components and may go on in others. To get rid of the components elimination and saturation are necessary.
- In each step we have to compute a tropical variety.

The algorithm is implemented in the SINGULAR library `tropical.lib` [5].

4 How to Compute a Tropical Variety?

Right now, there is basically only one software package which allows to compute tropical varieties for a larger class of ideals. This is the programme `gfan` by Anders Nedergaard Jensen [4]. In `gfan` several algorithms for computing a tropical variety are implemented which deal with different special cases. We will only comment briefly on the most important one which allows to compute the tropical variety of a *prime ideal* in the constant coefficient case. As mentioned, the tropical variety is then a subfan of the Gröbner fan. Moreover, if I is a prime ideal of dimension d, then $\mathrm{Trop}(I)$ is a pure d-dimensional fan and it is connected in codimension one. The idea of the algorithm is now the following:

- Find a first d-dimensional cone C in $\mathrm{Trop}(I)$ by some heuristics.
- Compute all facets of C.
- Compute the adjacent d-dimensional cones of the Gröbner fan and go on in the same way with these.

Since there are only finitely many d-dimensional cones and since one can pass from any one to any other one in this way after finitely many steps one has found all cones of $\mathrm{Trop}(I)$. Passing from one cone to another one involves lifting Gröbner bases of initial ideals to full Gröbner bases, and can thus be done by Gröbner basis techniques.

References

1. Allermann, L.: Tropical hypersurfaces (2008), http://www.mathematik.uni-kl.de/~allermann
2. Alonso, M.-E., Mora, T., Niesi, G., Raimondo, M.: An algorithm for computing analytic branches of space curves at singular points. In: Proceedings of the 1992 International Workshop on Mathematics Mechanization, pp. 135–166. International Academic Publishers (1992)
3. Einsiedler, M., Kapranov, M., Lind, D.: Non-archimedean amoebas and tropical varieties. J. Reine Angew. Math. 601, 139–157 (2006)
4. Jensen, A.N.: Gfan, a software system for Gröbner fans (2007), http://www.math.tu-berlin.de/~jensen/software/gfan/gfan.html
5. Jensen, A.N., Markwig, H., Markwig, T.: tropical.lib. A SINGULAR 3.0 library for computations in tropical geometry (2007)
6. Jensen, A.N., Markwig, H., Markwig, T.: An algorithm for lifting points in a tropical variety. Collect. Math. 59, 129–165 (2008)
7. Markwig, T.: Standard bases in $k[[t_1, \ldots, t_m]][x_1, \ldots, x_n]$. J. Symb. Comp. 43, 765–786 (2008)
8. Maurer, J.: Puiseux expansions for space curves. Manuscripta Math. 32, 91–100 (1980)
9. Speyer, D., Sturmfels, B.: Tropical mathematics. math.CO/0408099 (2004)

A New Desingularization Algorithm for Binomial Varieties in Arbitrary Characteristic

Rocío Blanco*

Universidad de Castilla-La Mancha. Departamento de Matemáticas,
E.U. de Magisterio, Edificio Fray Luis de León,
Avda, de los Alfares 42, 16071 Cuenca, Spain
mariarocio.blanco@uclm.es

Abstract. We construct a combinatorial algorithm of resolution of singularities for binomial ideals, over a field of arbitrary characteristic. This algorithm is applied to any binomial ideal. This means ideals generated by binomial equations without any restriction, including monomials and p-th powers, where p is the characteristic of the base field.

In particular, this algorithm works for toric ideals. However, toric geometry tools are not needed, the algorithm is constructed following the same point of view as Villamayor algorithm of resolution of singularities in characteristic zero.

Keywords: Resolution of singularities, Binomial ideals.

1 Introduction

In the particular case of binomial ideals, there exist some specific methods of resolution of singularities for binomial varieties with suitable restrictions, specially in the case of toric ideals. For normal toric varieties over an algebraically closed field of arbitrary characteristic see [7] and [4]. Non necessarily normal toric varieties are treated in [5] and [8].

Bierstone and Milman construct in [1] an algorithm of resolution of singularities, free of characteristic, for reduced binomial ideals with no nilpotent elements. In particular, their algorithm applies to toric ideals. During this resolution process p-th powers are never obtained at the transform ideals. In fact, this algorithm can not treat p-th powers of the type $(y^{\gamma}x_1 - bx^{\beta})^{p^s}$.

In this paper we consider binomial ideals without any kind of restriction, and we construct an algorithm of resolution of singularities for these binomial ideals in arbitrary characteristic that provides combinatorial centers of blowing-up. This type of centers preserve the binomial structure of the ideal after blowing-up, what let us ensure the existence of a hypersurface of maximal contact which to make induction on the dimension of the ambient space.

Blowing up only combinatorial centers we obtain a locally monomial ideal as output. By applying our algorithm again, we can assure to obtain a log-resolution

* Research partially supported by MTM2007-64704, MTM2009-07291.

K. Fukuda et al. (Eds.): ICMS 2010, LNCS 6327, pp. 217–220, 2010.

of the beginning ideal and an embedded desingularization of the corresponding binomial variety with good properties. Full text of this paper is available at [2].

One of the key points of this new algorithm is that it can be implemented. The SINGULAR library "resbin.lib" (RESolution of BINomial ideals) allows to compute explicitly a resolution of singularities of a binomial ideal over a field of arbitrary characteristic. This library is a joint work with Professor Gerhard Pfister, from Technical University Kaiserslautern.

1.1 Combinatorial Algorithm

Let K be an algebraically closed field of arbitrary characteristic, $\mathbf{W}$ will be the regular ambient space. At any stage of the resolution process, $\mathbf{W} = \cup_i U_i$, where $U_i \cong \mathbb{A}^n_K$. Locally, inside any affine chart U_i, we consider an open set W.

At the beginning of the resolution process $W = Spec(K[x_1, \ldots, x_n])$, where $dim(W) = n$. Fix the normal crossing divisor $E = \{V_1, \ldots, V_n\}$, where $V_i = V(x_i)$ for each $1 \leq i \leq n$.

Let $J \subset K[x] = K[x_1, \ldots, x_n]$ be a binomial ideal (generated by monomial and binomial equations). After a blowing up $W' \to W$, binomial equations of the type

$$1 - \mu x^\delta, with\ \mu \in K, \delta \in \mathbb{N}^n$$

appear naturally in the transform ideal of J. The points $\xi' \in W'$ outside the exceptional divisor where $1 - \mu x^\delta$ vanishes, satisfy $x^\delta(\xi') \neq 0$. We denote as y_i each variable x_i that do not vanish anywhere over $V(J) \cap V(1 - \mu x^\delta)$.

The binomial equations of J of the form $1 - \mu y^\delta$ are said to be *hyperbolic equations* of J. In what follows we work in localized rings of the type $K[x, y]_y$.

Remark 1. At any stage of the resolution process, inside any chart U_i we consider the open set

$$W = Spec(K[x, y]_y) = Spec(K[x_1, \ldots, x_s, y_1, \ldots, y_{n-s}]_y) \subset \mathbb{A}^n_K.$$

The normal crossing divisor E is a set of normal crossing regular hypersurfaces in $\mathbb{A}^n_K$, such that

$$E = \{V(x_1), \ldots, V(x_s), V(y_1), \ldots, V(y_{n-s})\}.$$

In the open set $Spec(K[x, y]_y)$ we have $E \cap Spec(K[x, y]_y) = \{V(x_1), \ldots, V(x_s)\}$.

Remark 2. Fixed a normal crossing divisor E as above, we define a modified order function, the E-order. Given a binomial ideal J, the E-order function (associated to J), $E-ord_J$, computes the order of the ideal J along $E \cap W$.

Definition 1. *Let $J \subset \mathcal{O}_W$ be an ideal, c a positive integer. We call* E-singular locus *of J with respect to c to the set,*

$$E-Sing(J, c) = \{\xi \in W / \ E-ord_\xi(J) \geq c\}.$$

We give the idea of the running of the algorithm.

Algorithm 1. *Let $J \subset \mathcal{O}_W$ be a binomial ideal without hyperbolic equations, with respect to a normal crossing divisor E. Fix a reduced Gröbner basis of J. Consider the factorization $J = M \cdot I$ where the ideal M is defined by a normal crossings divisor supported by the current exceptional locus.*
At the beginning $\mathcal{O}_W = K[x]$, $E = \{V(x_1), \ldots, V(x_n)\}$ and $J = I$.

1. *We proceed by blowing-up combinatorial centers inside the E-singular locus of J, whereas $E{-}Sing(J, max\ E{-}ord(J)) \neq \emptyset$.*
   At some step we obtain $J' = M' \cdot I'$ with $E{-}Sing(J', max\ E{-}ord(J)) = \emptyset$. This means that the maximal E-order of the ideal J has dropped.
2. - *If $max\ E{-}ord(I') > 0$ then consider $E{-}Sing(J', max\ E{-}ord(J')) \neq \emptyset$ and go to step 1.*
   - *Otherwise, $max\ E{-}ord(I') = 0$. In this case, there are two possibilities:*
     * *$I' = 1$, J' is already a principal ideal.*
     * *There are hyperbolic equations in I'. So we pass to the localization, go to step 1 and continue the resolution process with the non hyperbolic equations of I'.*

We continue the resolution process until either $I' = 1$ (J' is a principal ideal) or I' is given only by hyperbolic equations.

Example 1. Consider the binomial ideal $J =< (x_1 - x_2x_3)^2, x_2^2 - x_3^3 > \subset K[x]$, $char(K) = 2$. Take the origin as center of blowing up. At the affine chart where we divide by x_1, the total transform of J is of the form:

$$J^* =< x_1^2 > \cdot < (1 - x_1x_2x_3)^2, x_2^2 - x_1x_3^3 > .$$

We notice that a hyperbolic equation has appeared in J^*. In a neighborhood of a point ξ where $x_i(\xi) \neq 0$ for $i = 1, 2, 3$, the total transform of J can be written:

$$J^*_\xi =< y_1^2 > \cdot < (1 - y_1y_2y_3)^2, 1 - y_1y_2^{-2}y_3^3 > .$$

Remark 3. It is known that the reduced Gröbner basis of a binomial ideal is binomial, see [6].

Remark 4. Given a binomial ideal J, the algorithm 1 provides a *locally monomial resolution* of J. This means that the total transform of the ideal J is a monomial ideal with respect to a local system of parameters.

Our aim is to achieve a log-resolution of J, which is reached by applying algorithm 1 again.

Remark 5. The step from the locally monomial resolution to the log-resolution modifies the singular points included in the hyperbolic hypersurfaces.

Remark 6. Let X be a binomial variety. Algorithm 1 provides an embedded desingularization of X. See [2] for details.

References

1. Bierstone, E., Milman, P.: Desingularization of toric and binomial varieties. J. Algebraic Geom. 15(3), 443–486 (2006)
2. Blanco, R.: Combinatorial resolution of binomial ideals in arbitrary characteristic (preprint) arXiv:0902.2887v1 [math.AG]
3. Blanco, R., Encinas, S.: Embedded desingularization of toric varieties. Journal of Symbolic Computation: Special Issue, Symbolic and Algebraic Computation (accepted for publication) (preprint) arXiv:0901.2211v2 [math.AG]
4. Cox, D.: Toric varieties and toric resolutions. In: Resolution of Singularities (Obergurgl, 1997). Progr. Math., vol. 181, pp. 259–284. Birkhäuser, Basel (2000)
5. González Pérez, P.D., Teissier, B.: Embedded resolutions of non necessarily normal affine toric varieties. C. R. Math. Acad. Sci. Paris 334(5), 379–382 (2002)
6. Eisenbud, D., Sturmfels, B.: Binomial ideals. Duke Math. J. 84(1), 1–45 (1996)
7. Kempf, G., Knudsen, F.F., Mumford, D., Saint-Donat, B.: Toroidal embeddings. I. LNM, vol. 339. Springer, Berlin (1973)
8. Teissier, B.: Monomial ideals, binomial ideals, polynomial ideals. In: Trends in Commutative Algebra. Math. Sci. Res. Inst. Publ., vol. 51, pp. 211–246. Cambridge Univ. Press, Cambridge (2004)

An Algorithm of Computing Inhomogeneous Differential Equations for Definite Integrals

Hiromasa Nakayama and Kenta Nishiyama

Department of Mathematics,
Graduate school of Science, Kobe University,
1-1 Rokkodai, Nada-ku, 657-8501, Kobe, Japan
JST CREST,
5 Sanbancho, Chiyoda-ku, 102-0075, Tokyo, Japan
{nakayama,nisiyama}@math.kobe-u.ac.jp

Abstract. We give an algorithm to compute inhomogeneous differential equations for definite integrals with parameters. The algorithm is based on the integration algorithm for D-modules by Oaku. Main tool in the algorithm is the Gröbner basis method in the ring of differential operators.

Keywords: integration algorithm, holonomic functions, D-module, Gröbner basis.

1 Introduction

Let us denote by $D = K\langle x_1, \ldots, x_n, \partial_1, \ldots, \partial_n\rangle$ the Weyl algebra in n variables, where K is $\mathbb{Q}$ or $\mathbb{C}$ and ∂_i is the differential operator standing for x_i. We denote by $D' = K\langle x_{m+1}, \ldots, x_n, \partial_{m+1}, \ldots, \partial_n\rangle$ the Weyl algebra in $n-m$ variables, where $m \leq n$ and D' is a subring of D.

Let I be a holonomic left D-ideal ([8]). The integration ideal of I with respect to $x_1, \ldots, x_m$ is defined by the left D'-ideal

$$(I + \partial_1 D + \cdots + \partial_m D) \cap D'.$$

Oaku ([6]) gave an algorithm computing the integration ideal. This algorithm is called *the integration algorithm for D-modules*. The Gröbner basis method in D is used in this algorithm.

We give a new algorithm computing not only generators of the integration ideal J but also $P_0 \in I$ and $P_1, \ldots, P_m \in D$ such as

$$P = P_0 + \partial_1 P_1 + \cdots + \partial_m P_m$$

for any generator $P \in J$. Our algorithm is based on Oaku's one. We call these $P_1, \ldots, P_m$ *inhomogeneous parts of P*. As an important application of our algorithm, we can obtain inhomogeneous differential equations for definite integrals with parameters by using generators of the integration ideal and inhomogeneous parts.

K. Fukuda et al. (Eds.): ICMS 2010, LNCS 6327, pp. 221–232, 2010.

For example, we compute an inhomogeneous differential equation for the integral $A(x_2) = \int_a^b e^{-x_1-x_2x_1^3} dx_1$. This is the case of $m = 1, n = 2$. The annihilating ideal of the integrand $f(x_1, x_2) = e^{-x_1-x_2x_1^3}$ in D is $I = \langle \partial_1 + 1 + 3x_2x_1^2, \partial_2 + x_1^3 \rangle$. The integration ideal of I with respect to x_1 is $J = \langle 27x_2^3\partial_2^2 + 54x_2^2\partial_2 + 6x_2 + 1 \rangle = \langle P \rangle$. The operator $P_1 = -(\partial_1^2 + 3\partial_1 + 3)$ is an inhomogeneous part of P. We apply the operator P to the integral $A(x_2)$ and obtain

$$\begin{aligned} P \cdot A(x_2) &= \int_a^b \partial_1 (P_1 \cdot e^{-x_1-x_2x_1^3}) dx_1 = \left[P_1 \cdot e^{-x_1-x_2x_1^3} \right]_{x_1=a}^{x_1=b} \\ &= -\left[(9x_2^2x_1^4 - 3x_2x_1^2 - 6x_2x_1 + 1) e^{-x_1-x_2x_1^3} \right]_{x_1=a}^{x_1=b}. \end{aligned}$$

In this way, we get an inhomogeneous differential equation for the integral $A(x_2)$.

We will give an algorithm to compute inhomogeneous parts of the integration ideal and give some examples. Other algorithms to compute differential equations for definite integrals are the Almkvist-Zeilberger algorithm ([1], [10], [2]), the Chyzak algorithm ([4]) and the Oaku-Shiraki-Takayama algorithm ([7]). A comparison with these algorithms are also given.

We implement our algorithms on the computer algebra system Risa/Asir ([11]). They are in the program package `nk_restriction.rr` ([14]). Packages `Mgfun` in Maple and `HolonomicFunctions` in Mathematica offers an analogous functionality, and are based on the Chyzak algorithm ([12], [13]).

2 Review of the Integration Algorithm for D-Modules

We will review the integration algorithm for D-modules. We define the ring isomorphism $\mathcal{F} : D \to D$ satisfying

$$\mathcal{F}(x_i) = \begin{cases} -\partial_i & (1 \leq i \leq m) \\ x_i & (m < i \leq n) \end{cases}, \mathcal{F}(\partial_i) = \begin{cases} x_i & (1 \leq i \leq m) \\ \partial_i & (m < i \leq n) \end{cases}.$$

This map is called *the Fourier transformation in D*.

The integration ideal of a left holonomic D-ideal I with respect to $x_1, \ldots, x_m$ is defined by the left D'-ideal $J = (I + \partial_1 D + \cdots + \partial_m D) \cap D'$.

Algorithm 1 (Integration algorithm for D-modules, [6], [8]).

Input: Generators of a holonomic left D-ideal I and
a weight vector $w = (w_1, \ldots, w_m, w_{m+1}, \ldots, w_n)$ such that $w_1, \ldots, w_m > 0, w_{m+1} = \cdots = w_n = 0$.
Output: Generators of the integration ideal of I with respect to $x_1, \ldots, x_m$.

1. Compute the restriction module of the left D-ideal $\mathcal{F}(I)$ with respect to the weight vector w. The details of the computation are as follows.
   (a) Compute the Gröbner basis of the left D-ideal $\mathcal{F}(I)$ with respect to the monomial order $<_{(-w,w)}$. Let the Gröbner basis be $G = \{h_1, \ldots, h_l\}$.

(b) Compute the generic b-function $b(s)$ of $\mathcal{F}(I)$ with respect to the weight vector $(-w, w)$.
(c) If $b(s)$ has a non-negative integer root, then we set
$s_0 =$ (the maximal non-negative integer roots).
Otherwise, the integration ideal is 0 and finish.
(d) $m_i = \mathrm{ord}_{(-w,w)}(h_i)$,
$\mathcal{B}_d = \{\partial_1^{i_1} \cdots \partial_m^{i_m} \mid i_1 w_1 + \cdots + i_m w_m \leq d\} \quad (d \in \mathbb{N})$,
$r = \#\{(i_1, \ldots, i_m) \mid i_1 w_1 + \cdots + i_m w_m \leq s_0\} = \#\mathcal{B}_{s_0}$.
(e) $\tilde{\mathcal{B}} = \bigcup_{i=1}^{l} \{\tilde{h}_{i\beta} := \partial^\beta h_i \mid \partial^\beta \in \mathcal{B}_{s_0 - m_i}\}$,
$\mathcal{B} = \{h_{i\beta} := \tilde{h}_{i\beta}|_{x_1 = \cdots = x_m = 0} \mid \tilde{h}_{i\beta} \in \tilde{\mathcal{B}}\}$.
Here, $h_{i\beta} = \sum_{\partial^\alpha \in \mathcal{B}_{s_0}} g_\alpha \partial^\alpha \quad (g_\alpha \in D')$.

2. Let $(D')^r$ be the left free D'-module with the base $\mathcal{F}^{-1}(B_{s_0})$, i.e. $(D')^r = \sum_{\partial^\alpha \in \mathcal{B}_{s_0}} D' x^\alpha$. Regard elements in $\mathcal{F}^{-1}(\mathcal{B})$ as elements in the left D'-module $(D')^r$. In other words, $\mathcal{F}^{-1}(h_{i\beta}) = \sum_{\partial^\alpha \in \mathcal{B}_{s_0}} g_\alpha x^\alpha \quad (g_\alpha \in D')$ is regarded as an element in $(D')^r$. Let M be the left D'-submodule in $(D')^r$ generated by $\mathcal{F}^{-1}(\mathcal{B})$.
3. Compute the Gröbner basis G of M with respect to a POT term order such that the position corresponds to $x^0 = 1$ is the minimum position. Output $G' = G \cap D'$. This set G' generates the integration ideal of I.

We consider the following definite integral of a holonomic function $f(x_1, \ldots, x_n)$.

$$A(x_{m+1}, \ldots, x_n) = \int_R f(x_1, \ldots, x_n) dx_1 \cdots dx_m, \quad R = \prod_{i=1}^{m} [a_i, b_i]$$

Let $I = \mathrm{Ann}_D f := \{P \cdot f = 0 \mid P \in D\}$ be the annihilating ideal of the integrand, and J be the integration ideal of I. For every $p \in J$, there exist $p_1, \ldots, p_m \in D$ such that

$$p - \sum_{i=1}^{m} \partial_i p_i \in I$$

and we have

$$\begin{aligned} p \cdot A(x_{m+1}, \ldots, x_n) &= \int_R p \cdot f dx_1 \cdots dx_m = \int_R \sum_{i=1}^{m} (\partial_i p_i) \cdot f dx_1 \cdots dx_m \\ &= \sum_{i=1}^{m} \int_R \partial_i (p_i \cdot f) dx_1 \cdots dx_m. \end{aligned} \tag{1}$$

Therefore, if we take an integration domain such that the right hand side of (1) equals to zero, we can regard the integration ideal as a system of homogeneous differential equations for the integral $A(x_{m+1}, \ldots, x_n)$. If the right hand side is not zero, the equation (1) gives an inhomogeneous differential equations for the function A.

3 Computing Inhomogeneous Parts of the Integration Ideal

In this section, we give a new algorithm of computing inhomogeneous differential equations for definite integrals. For the purpose, we must find an explicit form p_i $(1 \leq i \leq m)$ in the equation (1) in the section 2.

Theorem 1. *Let $J \subset D'$ be the integration ideal of a holonomic left D-ideal I. For any $p \in J$, there exists an algorithm to compute differential operators $p_i \in D$ $(1 \leq i \leq m)$ such that*

$$p - \sum_{i=1}^{m} \partial_i p_i \in I. \tag{2}$$

Proof. We will present an algorithm of obtaining operators p_i. By applying Algorithm 1, we obtain a generating set $\{g_1, \ldots, g_t\}$ of the integration ideal of I. It is sufficient to compute inhomogeneous parts for each generator g_j. From the step 3 of Algorithm 1, g_j can be expressed as $g_j = \sum q_{ji\beta} \mathcal{F}^{-1}(h_{i\beta})$ where $q_{ji\beta} \in D$. Then these $q_{ji\beta} \in D$ can be computed by referring the history of the Gröbner basis computation in the step 3. Therefore, we have

$$\begin{aligned} I \ni \sum q_{ji\beta} \mathcal{F}^{-1}(\tilde{h}_{i\beta}) &= g_j - \left(g_j - \sum q_{ji\beta} \mathcal{F}^{-1}(\tilde{h}_{i\beta}) \right) \\ &= g_j - \sum q_{ji\beta} \left(\mathcal{F}^{-1}(h_{i\beta}) - \mathcal{F}^{-1}(\tilde{h}_{i\beta}) \right) \\ &= g_j - \sum q_{ji\beta} \left(\mathcal{F}^{-1}(\tilde{h}_{i\beta}|_{x_1=\cdots=x_m=0}) - \mathcal{F}^{-1}(\tilde{h}_{i\beta}) \right) \\ &= g_j - \sum q_{ji\beta} \mathcal{F}^{-1}(\tilde{h}_{i\beta}|_{x_1=\cdots=x_m=0} - \tilde{h}_{i\beta}). \end{aligned}$$

Since each term of $\tilde{h}_{i\beta}|_{x_1=\cdots=x_m=0} - \tilde{h}_{i\beta}$ can be divided from the left by either of $x_1, \ldots, x_m$, each term of $\mathcal{F}^{-1}(\tilde{h}_{i\beta}|_{x_1=\cdots=x_m=0} - \tilde{h}_{i\beta})$ can be divided from the left by either of $\partial_1, \ldots, \partial_m$. Thus we can rewrite

$$\sum q_{ji\beta} \mathcal{F}^{-1}(\tilde{h}_{i\beta}|_{x_1=\cdots=x_m=0} - \tilde{h}_{i\beta}) = \sum_{i=1}^{m} \partial_i p_{ij}.$$

Let us present our algorithm.

Algorithm 2

Input: Generators of a holonomic left ideal $I \subset D$ and
a weight vector $w = (w_1, \ldots, w_m, w_{m+1}, \ldots, w_n)$ such that $w_1, \ldots, w_m > 0, w_{m+1} = \cdots = w_n = 0$.

Output: Generators $\{g_1, \ldots, g_t\}$ of the integration ideal of I w.r.t. $x_1, \ldots, x_m$ and operators $p_{ij} \in D$ satisfying $g_j - \sum_{i=1}^{m} \partial_i p_{ij} \in I$ for each generator g_j $(1 \leq j \leq t)$.

1. Apply Algorithm 1.
2. Compute $q_{ji\beta}$ satisfying $g_j = \sum q_{ji\beta}\mathcal{F}^{-1}(h_{i\beta})$ by referring the history of the Gröbner basis computation in the step 3 of Algorithm 1.
3. Rewrite $R_j := g_j - \sum q_{ji\beta}\mathcal{F}^{-1}(\tilde{h}_{i\beta})$ to the form of $R_j = \sum_{i=1}^{m} \partial_i p_{ij}$.

   Output p_{ij}.

Example 1 (Incomplete Gauss's hypergeoemtric integral).
We set

$$F(x) = \int_p^q t^{b-1}(1-t)^{c-b-1}(1-xt)^{-a}dt.$$

We will compute a differential equation for the integral $F(x)$. A holonomic ideal annihilating the integrand $f(x,t) = t^{b-1}(1-t)^{c-b-1}(1-xt)^{-a}$ is

$$\begin{aligned} I_f =& \langle (-x^2+x)\partial_x^2 + ((-t+1)\partial_t + (-a-b-1)x + c - 1)\partial_x - ab, \\ & (-t+1)x\partial_x + (t^2-t)\partial_t + (-c+2)t + b - 1, (tx-1)\partial_x + at\rangle \end{aligned}$$

which is obtained by using Oaku's algorithm to compute the annihilating ideal of a power of polynomials. The generic b-function of $\mathcal{F}(I_f)$ with respect to the weight vector $w = (1,0)$ (i.e. t's weight is 1 and x's weight is 0) is $s(s-a+c-1)$. We assume that $a-c+1$ is not a non-negative integer. Then the maximal non-negative integer root s_0 of $b(s)$ is 0. Therefore, the integration ideal of I_f with respect to t is

$$\langle (-x^2+x)\partial_x^2 + ((-a-b-1)x + c)\partial_x - ab\rangle = \langle P\rangle.$$

The differential equation $P \cdot g = 0$ is Gauss's hypergeometric equation. The inhomogeneous part of P is $\partial_t(-t+1)\partial_x$. We apply P to the integral $F(x)$ and obtain the inhomogeneous differential equation

$$P \cdot \int_p^q f(x,t)dt = \int_p^q (\partial_t(t-1)\partial_x) \cdot f(x,t)dt = \left[(t-1)\frac{\partial f}{\partial x}(x,t)\right]_p^q.$$

We present the output for this problem by the program `nk_restriction.rr` on the computer algebra system Risa/Asir ([11]). We use the command `nk_restriction.integration_ideal` to compute the integration ideal. The option `inhomo=1` make the system compute inhomogeneous parts and the option `param = [a,b,c]` means that parameters are a, b, c. The `sec` shows the exhausting time of each steps. This example and next example are executed on a Linux machine with Intel Xeon X5570 (2.93GHz) and 48 GB memory.

```
[1743] load("nk_restriction.rr");
[1944] I_f=[-dx^2*x^2+(-dx*a-dx*b+dx^2-dx)*x-dx*dt*t-b*a+dx*c+dx*dt-dx,
(-dx*t+dx)*x+dt*t^2+(-c-dt+2)*t+b-1,dx*t*x+a*t-dx];
[(-x^2+x)*dx^2+((-t+1)*dt+(-a-b-1)*x+c-1)*dx-b*a,
(-t+1)*x*dx+(t^2-t)*dt+(-c+2)*t+b-1,(t*x-1)*dx+a*t]
[1945] nk_restriction.integration_ideal(I_f,[t,x],[dt,dx],[1,0]|param=
[a,b,c],inhomo=1);
-- nd_weyl_gr :0.004sec(0.000623sec)
-- weyl_minipoly_by_elim :0sec(0.000947sec)
-- generic_bfct_and_gr :0.004sec(0.001922sec)
generic bfct : [[1,1],[s,1],[s-a+c-1,1]]
S0 : 0
B_{S0} length : 1
-- fctr(BF) + base :0sec(0.000277sec)
-- integration_ideal_internal :0sec(0.000499sec)
[[(-x^2+x)*dx^2+((-a-b-1)*x+c)*dx-b*a],[[[[dt,(t-1)*dx]],1]]]
```

Example 2 ($F(x) = \int_0^\infty e^{-t-xt^3}dt$).

We consider the integral $F(x) = \int_0^\infty e^{-t-xt^3}dt$. A holonomic ideal annihilating the integrand $f(t,x) = e^{-t-xt^3}$ is $I_f = \langle \partial_t + 1 + 3xt^2, \partial_x + t^3 \rangle$. The integration ideal of I_f with respect to t is $J = \langle 27x^3\partial_x^2 + 54x^2\partial_x + 6x + 1 \rangle = \langle P \rangle$. The inhomogeneous part of P is $-\partial_t(\partial_t^2 + 3\partial_t + 3)$. We apply P to the integral $F(x)$ and obtain

$$\begin{aligned} P \cdot \int_0^\infty e^{-t-xt^3}dt &= -\int_0^\infty (\partial_t(\partial_t^2 + 3\partial_t + 3)) \cdot e^{-t-xt^3}dt \\ &= -\left[(\partial_t^2 + 3\partial_t + 3) \cdot e^{-t-xt^3}\right]_0^\infty \\ &= -\left[(-6xt + (1+3xt^2)^2 - 3 - 9xt^2 + 3)e^{-t-xt^3}\right]_0^\infty = 1. \end{aligned}$$

```
[1946] load("nk_restriction.rr");
[2146] I_f=[dt+1+3*x*t^2, dx+t^3];
[dt+3*t^2*x+1,dx+t^3]
[2147] nk_restriction.integration_ideal(I_f,[t,x],[dt,dx],[1,0] |
inhomo=1);
-- nd_weyl_gr :0sec(0.000526sec)
-- weyl_minipoly :0sec(0.0002439sec)
-- generic_bfct_and_gr :0sec(0.001016sec)
generic bfct : [[1,1],[s,1]]
S0 : 0
B_{S0} length : 1
-- fctr(BF) + base :0sec + gc : 0.008sec(0.00691sec)
-- integration_ideal_internal :0sec(0.0003109sec)
[[27*x^3*dx^2+54*x^2*dx+6*x+1],[[[dt,-dt^2-3*dt-3]],1]]]
```

Theorem 2. *We consider the following multiple integral,*

$$F(x_{m+1},\dots,x_n)=\int_{a_1}^{b_1}\cdots\int_{a_m}^{b_m} f(x_1,\dots,x_n)dx_1\cdots dx_m \quad (m\le n). \tag{3}$$

Let I be a holonomic left D-ideal annihilating the integrand $f(x_1,\dots,x_n)$. There exists an algorithm to compute inhomogeneous differential equations for the multiple integral $F(x_{m+1},\dots,x_n)$ from the holonomic ideal I. The algorithm is described below.

For simplicity, we will explain the algorithm in the case of $m=2$. We set

$$F(x_3,\dots,x_n)=\int_{a_1}^{b_1}\int_{a_2}^{b_2} f(x_1,\dots,x_n)dx_1dx_2 \quad (2\le n),$$

and will compute an inhomogeneous differential equation of F.

Let I be a holonomic left D-ideal annihilating the integrand $f(x_1,\dots,x_n)$. We compute the integration ideal J of I with respect to x_1,x_2, i.e.

$$J=(I+\partial_1 D+\partial_2 D)\cap D' \quad (D'=K\langle x_3,\dots,x_n,\partial_3,\dots,\partial_n\rangle).$$

We take an element $P\in J$. There exist $P_0\in I$ and $P_1,P_2\in D$ such that $P=P_0+\partial_1P_1+\partial_2P_2\in D'$. We apply the operator P to the integral F, and obtain

$$P\cdot F=\int_{a_2}^{b_2}(P_1\cdot f|_{x_1=b_1}-P_1\cdot f|_{x_1=a_1})dx_2+\int_{a_1}^{b_1}(P_2\cdot f|_{x_2=b_2}-P_2\cdot f|_{x_2=a_2})dx_1. \tag{4}$$

Let F_1,F_2 be the first term and the second term of the right hand side and let f_1,f_2 be the integrand of F_1,F_2.

To obtain a holonomic ideal annihilating the integral F_1, we must compute a holonomic ideal I_1 annihilating the integrand f_1. When the integrand f_1 is the power of polynomial, we can use Oaku's algorithm to obtain the holonomic ideal I_1 ([6]). In general case, we can compute the holonomic ideal I_1 from I by the following method.

The ideal quotient $I:P_1$ is holonomic and annihilates the function $P_1\cdot f$. To obtain a holonomic ideal J_1 annihilating $P_1\cdot f|_{x_1=b_1}$, we compute the restriction ideal of $I:P_1$ with respect to $x_1=b_1$. Applying the same procedure for $x_1=a_1$ instead of $x_1=b_1$, we obtain a holonomic ideal J_2 annihilating $P_1\cdot f|_{x_1=a_1}$. Since $J_1\cap J_2$ is holonomic and annihilates $f_1(=P_1\cdot f|_{x_1=b_1}-P_1\cdot f|_{x_1=a_1})$, we obtain $J_1\cap J_2$ as I_1.

We compute the integration ideal K_1 of I_1 with respect to x_2, i.e.

$$K_1=(I_1+\partial_2 D_1)\cap D' \quad (D_1=K\langle x_2,x_3,\dots,x_n,\partial_2,\partial_3,\dots,\partial_n\rangle).$$

We take an element $P^{(1)}\in K_1$. There exist $P_0^{(1)}\in I_1$ and $P_2^{(1)}\in D_1$ such that $P^{(1)}=P_0^{(1)}+\partial_2P_2^{(1)}$. We apply $P^{(1)}$ to the integral F_1, and obtain

$$P^{(1)}\cdot F_1=P_2^{(1)}\cdot f_1|_{x_2=b_2}-P_2^{(1)}\cdot f_1|_{x_2=a_2}. \tag{5}$$

Applying the same procedure for I_2 instead of I_1, we can compute the annihilating ideal I_2 of the integrand f_2 and the integration ideal K_2 of I_2 with respect to x_1.

By (4) and (5), we obtain

$$P^{(1)} \cdot P \cdot F = P^{(1)} \cdot F_1 + P^{(1)} \cdot F_2,$$

and can compute the first term of the right hand side. To compute the second term $P^{(1)} \cdot F_2$, we compute $K_2 : P^{(1)}$ and take an element $P^{(2)}$ in this ideal. Since $P^{(2)}P^{(1)} \in K_2$, we can compute $P^{(2)}P^{(1)} \cdot F_2$. Finally, we can obtain an inhomogeneous differential equation

$$P^{(2)}P^{(1)}P \cdot F = P^{(2)}P^{(1)} \cdot F_1 + P^{(2)}P^{(1)} \cdot F_2.$$

Remark 1. Let

$$\ell_1 \cdot F = g_1, \cdots, \ell_p \cdot F = g_p \quad (\ell_i \in D',\ g_i \text{ is a holonomic function })$$

be a system of inhomogeneous differential equations. When $\langle \ell_1, \ldots, \ell_p \rangle$ generates the left holonomic ideal in D', we call the system *inhomogeneous holonomic*. When $m = 1$, the output of the algorithm in Theorem 2 is inhomogeneous holonomic. Although the algorithm outputs a lot of inhomogeneous differential equations when P runs over the ideal J, it is an open question whether the output of the algorithm is inhomogeneous holonomic when $m > 1$. However, since the Oaku-Shiraki-Takayama algorithm gives holonomic output (see [7], section 4.3), we can obtain inhomogeneous holonomic differential equations by the following algorithm.

Algorithm 3

Input: Generators of a holonomic left ideal annihilating $f(x_1, \ldots, x_n)$.
Output: Generators of an inhomogeneous holonomic system for (3).

1. Apply the algorithm in Theorem 2.
2. Apply the Oaku-Shiraki-Takayama algorithm if the system obtained in the step 1 is not inhomogeneous holonomic.
3. Merge the outputs of the step 1 and the step 2.

4 Comparison of Our Algorithm with Other Algorithms

4.1 The Almkvist-Zeilberger Algorithm

The Almkvist-Zeilberger algorithm (AZ algorithm, [1], [10], [2]) is very fast, but works for hyperexponential functions. Our algorithm works for holonomic functions. The AZ algorithm is based on the method of undetermined coefficients and Gosper's algorithm, and our algorithm is based on the Gröbner basis method in D.

4.2 The Chyzak Algorithm

The Chyzak algorithm ([3], [4], [5]) is based on the method of undetermined coefficients and the Gröbner basis method in the Ore algebra. By using the Ore algebra, the Chyzak algorithm can compute various summations and integrals like summations of holonomic sequences, integrals of holonomic functions and its q-analogues. For the ring of differential operators with rational function coefficients $K(x)\langle\partial\rangle$, the Chyzak algorithm is a generalization of the AZ algorithm and works for holonomic functions. The algorithm is often faster than our algorithm. But, when the algorithm returns higher order differential equations or the number of variables are many, our algorithm is sometimes faster. Here, we show only one example. We present these examples at `http://www.math.kobe-u.ac.jp/OpenXM/Math/i-hg/nk_restriction_ex.html`

Example 3 ($F(x,y) = \int_a^b \frac{1}{xt+y+t^{10}} dt$).
We set

$$F(x,y) = \int_a^b \frac{1}{xt+y+t^{10}} dt.$$

We will compute differential equations for the integral $F(x,y)$. The following output is computed by our algorithm. It takes about 1.3 seconds.

```
[2345] load("nk_restriction.rr");
[2545] F=x*t+y+t^10$
[2546] Ann=ann(F)$ /* annihilating ideal of F^s */
0.052sec(0.0485sec)
[2547] Id=map(subst, Ann, s, -1)$ /* substitute s=-1 in Ann */
0sec(4.411e-05sec)
[1569] nk_restriction.integration_ideal(Id,[t,x,y],[dt,dx,dy],[1,0,0]
|inhomo=1);
-- nd_weyl_gr :0.012sec + gc : 0.008001sec(0.02009sec)
-- weyl_minipoly :0sec(0.001189sec)
-- generic_bfct_and_gr :0.016sec + gc : 0.008001sec(0.02358sec)
generic bfct : [[1,1],[s,1],[s-9,1]]
S0 : 9
B_{S0} length : 10
-- fctr(BF) + base :0.044sec + gc : 0.024sec(0.0674sec)
-- integration_ideal_internal :0.8321sec + gc : 0.236sec(1.071sec)
[[9*x*dx+10*y*dy+9,-10*dx^9-x*dy^9,-9*dx^10+y*dy^10+9*dy^9],
[[[[dt,-t]],1],[[[dt,-dy^8]],1],[[[dt,-t*dy^9]],1]]]
0.9081sec + gc : 0.28sec(1.19sec)
```

The following output is computed by the Chyzak algorithm (package `Mgfun` [12]) on Maple12. It takes about 50 seconds.

```
with(Mgfun):
f:=1/(x*t+y+t^10):
ts:=time():
creative_telescoping(f,[x::diff,y::diff], t::diff):
time()-ts;
                                    49.583
```

These computational experiments are executed on a Linux machine with Intel Xeon5450 (3.00GHz) and 32 GB memory.

4.3 The Oaku-Shiraki-Takayama Algorithm

Although our algorithm gives inhomogeneous differential equations for definite integrals, the Oaku-Shiraki-Takayama algorithm (OST algorithm, [7]) is for computing homogeneous differential equations annihilating definite integrals by using the Heaviside function and the integration algorithm. Since outputs are different, they are different methods. However, in most examples, outputs of our algorithm can be easily transformed to homogeneous systems. Thus, it will be worth making comparison between our method and the OST method.

Let $u(t,x)$ be a smooth function defined on an open neighborhood of $[a,b]\times U$ where U is an open set of $\mathbf{R}^{n-1}$. The Heaviside function $Y(t)$ defined by $Y(t) = 0\ (t<0), Y(t) = 1\ (t \geq 0)$. Then we can regard the integral of $u(t,x)$ over $[a,b]$ as that of $Y(t-a)Y(b-t)u(t,x)$ over $(-\infty,\infty)$, and the following holds.

$$\int_a^b u(t,x)dt = \int_{-\infty}^{\infty} Y(t-a)Y(b-t)u(t,x)dt$$

Thus we can apply Algorithm 1 to obtain homogeneous differential equations. The paper [7] proposes the two methods

(a) Method of using properties of the Heaviside function
(b) Method of using tensor product in D-module

to obtain differential equations annihilating the integrand of the right hand side. In the former case, the computation finishes without a heavy part because the procedure is only multiplication of polynomials. However, it is not known whether the output is holonomic. In the latter case, when an input is holonomic, an output is also holonomic. However, the computation is often heavy. We call the former OST algorithm (a) and the latter OST algorithm (b) in this paper. See [7, Chap 5] for details.

Let us show a relation of the outputs of OST algorithm and our algorithm. We consider $v(x) = \int_0^\infty e^{(-t^3+t)x}dt$. OST algorithm (a) or (b) return the following ideal

$$\langle -27x^3\partial_x^3 - 54x^2\partial_x^2 + (4x^3+3x)\partial_x + 4x^2 - 3,$$
$$27x^2\partial_x^4 + 135x\partial_x^3 + (-4x^2+105)\partial_x^2 - 16x\partial_x - 8\rangle.$$

On the other hand, Algorithm 2 returns the following ideal generated by P and its inhomogeneous part Q:

$$\langle P\rangle = \langle -27x^2\partial_x^2 - 27x\partial_x + 4x^2 + 3\rangle,$$
$$Q = \partial_t(-9tx\partial_x + (-6t^2+4)x + 3t).$$

This output yields

$$P\cdot v(x) = \left[(-9tx\partial_x + (-6t^2+4)x + 3t)\cdot e^{(-t^3+t)x}\right]_{t=0}^{t=\infty} = -4x. \tag{6}$$

Since the annihilating ideal of $-4x$ is $\langle x\partial_x - 1, \partial_x^2\rangle$, operators $(x\partial_x - 1)P$ and $\partial_x^2 P$ annihilate $v(x)$. Although results of these algorithms are not coincide in general, these operators coincide outputs of OST algorithm (a) and (b) in this case. However, it seems that it is difficult to compute the right hand side of (6) from the output of OST algorithm. Moreover, in our algorithm we have only to do substitution process to compute for the integrals which has same integrand and another integration domain because our algorithm does not depend on the integration domain.

Table 1 shows the computing time of each part of Algorithm 2 and OST algorithm (a), (b). The entries with parentheses for inputs $\bar{v}_k$ mean that results for v_k were reused. For a comparison we show the computing time of Algorithm 1. The experiments were done on a Linux machine with Intel Xeon X5570 (2.93GHz) and 48 GB memory.

From the viewpoint of the computational efficiency, the computation time of Algorithm 2 increases more than that of Algorithm 1 for computing inhomogeneous parts. That of OST algorithm increases because the input data of the integration algorithm becomes bigger differential operators by procedure (a) or (b). It seems that Algorithm 2 is faster than OST algorithm, since the computation of inhomogeneous parts can be done by multiplication and summation of

Table 1. The comparison of the computing time (seconds)

	Alg 2			OST (a)	OST (b)			Alg 1
Input	Alg 2	Ann	Total	Total	(b)	Alg 1	Total	Total
v_1	0.0042	0.0014	0.0056	0.0062	0.11	0.012	0.12	0.0039
v_2	0.15	0.019	0.17	0.25	5.10	0.16	5.26	0.075
v_3	19.91	0.45	20.36	96.14	24.54	95.24	119.8	13.58
v_4	26724	28.33	26752	>1 day	1726	>1 day	—	24003
$\bar{v}_1$	(0.0042)	0.0015	0.0057	0.0071	0.47	0.0050	0.48	n/a
$\bar{v}_2$	(0.15)	0.027	0.18	1.56	18230	1.19	18231	n/a
$\bar{v}_3$	(19.91)	1.62	21.53	3769	848	2802	3650	n/a
$\bar{v}_4$	(26724)	294	27018	>1 day	16231	>1 day	—	n/a

$$v_k(x) = \int_0^\infty u_k(t,x)dt,\ \bar{v}_k(x) = \int_0^1 u_k(t,x)dt \text{ where } u_k(t,x) = \exp\left(-tx\prod_{i=1}^{k}(t^2-i^2)\right)$$

differential operators. However, to obtain homogeneous equation corresponding to OST algorithm output, we must compute annihilating ideals of inhomogeneous parts.

Acknowledgement

We would like to thank Prof. Takayama for fruitful discussions and encouragements. We also would like to thank anonymous referees for many useful comments.

References

1. Almkvist, G., Zeilberger, D.: The method of differentiating under the integral sign. Journal of Symbolic Computation 10, 571–591 (1990)
2. Apagodu, M., Zeilberger, D.: Multi-variable Zeilberger and Almkvist-Zeilberger algorithms and the sharpening of Wilf-Zeilberger theory. Advances in Applied Mathematics 37, 139–152 (2006)
3. Chyzak, F.: Gröbner Bases, Symbolic Summation and Symbolic Integration. London Mathematics Lecture Notes Series, vol. 251, pp. 32–60 (1998)
4. Chyzak, F.: An Extension of Zeilberger's Fast Algorithm to General Holonomic Functions. Discrete Mathematics 217, 115–134 (2000)
5. Chyzak, F., Salvy, B.: Non-commutative Elimination in Ore Algebras Proves Multivariate Holonomic Identities. Journal of Symbolic Computation 26, 187–227 (1998)
6. Oaku, T.: Algorithms for b-functions, restrictions, and algebraic local cohomology groups of D-modules. Advances in Applied Mathematics 19, 61–105 (1997)
7. Oaku, T., Shiraki, Y., Takayama, N.: Algebraic Algorithm for D-modules and numerical analysis, Computer mathematics. In: Proceedings of ASCM 2003. Lecture Notes Ser. Comput., vol. 10, pp. 23–39. World Sci. Publ., River Edge (2003)
8. Saito, M., Sturmfels, B., Takayama, N.: Gröbner Deformations of Hypergeometric Differential Equations. Springer, Heidelberg (2000)
9. Takayama, N.: An Approach to the Zero Recognition Problem by Buchberger Algorithm. Journal of Symbolic Computation 14, 265–282 (1992)
10. Tefera, A.: MultInt, a Maple package for multiple integration by the WZ method. Journal of Symbolic Computation 34, 329–353 (2002)
11. Noro, M., et al.: Risa/Asir, http://www.math.kobe-u.ac.jp/Asir
12. Chyzak, F.: Mgfun, http://algo.inria.fr/chyzak/mgfun.html
13. Koutschan, C.: HolonomicFunctions, http://www.risc.jku.at/research/combinat/software/HolonomicFunctions/
14. Nakayama, H., Nishiyama, K.: nk_restriction.rr, http://www.math.kobe-u.ac.jp/~nakayama/nk_restriction.rr

New Algorithms for Computing Primary Decomposition of Polynomial Ideals

Masayuki Noro

Department of Mathematics, Graduate School of Science,
Kobe University JST CREST
noro@math.kobe-u.ac.jp

Abstract. We propose a new algorithm and its variant for computing a primary decomposition of a polynomial ideal. The algorithms are based on the Shimoyama-Yokoyama algorithm [17] in the sense that all the isolated primary components $Q_1, \ldots, Q_r$ of an ideal I are first computed from the minimal associated primes of I. In order to extract the remaining primary components we use $I : Q$ where $Q = Q_1 \cap \cdots \cap Q_r$. Our experiment shows that the new algorithms can efficiently decompose some ideals which are hard to be decomposed by any of known algorithms.

1 Introduction

Primary decomposition of a polynomial ideal is a fundamental task in algebraic geometry and commutative algebra. In order to decompose an algebraic variety of an ideal I, it is sufficient to compute the set of minimal associated primes of I. But we often need a primary decomposition of an ideal to get more precise information on the ideal. For example, several algorithms concerned with b-function (Bernstein-Sato polynomial) were presented recently [5] [16] and these algorithms need primary decomposition over $\mathbf{Q}$. Shioda [18] showed that there exists a relation between integral sections in an elliptic surface and primary decomposition of an ideal, based on the theory of Mordell-Weil lattice. Diaconis et al. [7] showed that primary decompositions of a certain kind of binomial ideals are useful in algebraic statistics (cf. Section 4.2).

Several algorithms for computing a primary decomposition of a polynomial ideal are known. Here we briefly overview them. Let I be an ideal in a polynomial ring $\mathbf{Q}[x] = \mathbf{Q}[x_1, \ldots, x_n]$ over the rationals.

– The Gianni-Trager-Zacharias (GTZ) algorithm [10]
The GTZ algorithm extracts some of maximal dimensional primary components $Q_1, \ldots, Q_k$ of I via a reduction to a zero-dimensional case. As a by-product of this operation, one obtains an element $f^s \notin I$ such that

$$I = (I : f^s) \cap (I + f^s), \quad I : f^s = I : f^\infty = Q_1 \cap \cdots \cap Q_k. \tag{1}$$

Then this procedure is applied to $I + f^s$ to obtain the remaining primary components of I.

K. Fukuda et al. (Eds.): ICMS 2010, LNCS 6327, pp. 233–244, 2010.

– The Shimoyama-Yokoyama (SY) algorithm [17]
The SY algorithm first computes the set of all minimal associated primes $\{P_1, \ldots, P_l\}$ of I. By using them, ideals $\tilde{Q}_1 \ldots, \tilde{Q}_l$ and elements $f_1, \ldots, f_l$ satisfying $\sqrt{\tilde{Q}_i} = P_i$ and

$$I = (\tilde{Q}_1 \cap \cdots \cap \tilde{Q}_l) \cap (I + \langle f_1, \ldots, f_l \rangle), \quad \dim(I + \langle f_1, \ldots, f_l \rangle) < \dim I \quad (2)$$

are computed. Each $\tilde{Q}_i$ contains only one isolated primary component Q_i of I and we can compute an ideal I' such that $\tilde{Q}_i = Q_i \cap I'$ and $\dim I' < \dim I$. Then this procedure is applied to I' and $I'' = I + \langle f_1, \ldots, f_l \rangle$ to obtain remaining primary components of I. Each $\tilde{Q}_i$ is called a *pseudo primary component* and (2) is called a *pseudo primary decomposition* of I.
– The Eisenbud-Huneke-Vasconcelos (EHV) algorithm [8]
The EHV algorithm first computes the set of all associated primes of I via homological algebra. Then the primary components of I are computed by localization. This method is distinguished from the above ones because it does not use the generic projection (reduction to a zero-dimensional case).

These algorithms have been implemented in various computer algebra systems [1] [2] [4]. We cited these three implementations because the source codes of primary decomposition are written in the user languages and the details of implementation are easily examined. It is empirically known (cf. [6]) that it is hard to predict which algorithm is efficient for a particular input ideal and we usually try several methods to compute a primary decomposition of a given ideal. Recently we found a number of examples which are hard to be decomposed by any of these algorithms (cf. [14]). We analyzed the reason of the difficulty and found a method to resolve it. The new method may not necessarily be efficient for the other types of input ideals than our examples. However it will be practically important and useful to provide a new method for primary decomposition.

Our new method is based on the SY algorithm in the sense that all the isolated primary components of an ideal are computed from its minimal associated primes. The differences between the SY algorithm and the new method are described as follows. In this paper we call an ideal I' *a remaining ideal* of a decomposition of an ideal I if $I = (Q_1 \cap \cdots \cap Q_l) \cap I'$ holds for some primary ideals $Q_1, \ldots, Q_l$.

1. One remaining ideal at a step
The strategy of the new method is to keep only primary components of I until their intersection coincides with the input ideal I. At each step, we first compute all the isolated primary components $Q_1, \ldots, Q_l$ of I via the minimal associated primes of I. Then we compute an ideal J such that $I = Q \cap (I + J)$, where $Q = Q_1 \cap \cdots \cap Q_l$. Then setting $I = I + J$, we continue this procedure until the intersection of all the obtained primary components coincides with the input ideal I.
2. Computation of the remaining ideal via ideal quotient
The ideal J is computed as a subset of $I : Q$ to make the remaining ideal $I + J$ as large as possible.

This modification is based on the following observation and consideration. In each step of the SY algorithm, remaining ideals I' for each pseudo primary component $\tilde{Q}_i$ and I'' are produced and they have to be decomposed recursively. If I has embedded primary components, they appear either in I' or in I''. In the former case $I' = I + \langle f^s \rangle$ for some $f \in R$. We found that $I + \langle f^s \rangle$ is often hard to decompose because of newly introduced unnecessary components. A simple example is I_1 in Section 4.1. We found that we can avoid such difficulties by enlarging I', that is by adding more elements to I, keeping the relation $\tilde{Q} = Q \cap I'$ for a pseudo primary ideal $\tilde{Q}$. To apply this strategy we need some method to make remaining ideals as large as possible. Furthermore we want to detect the termination of the computation as early as possible by computing the intersection of all the obtained primary components and comparing it with I. For these purposes we keep only the primary components $Q_1, \ldots, Q_l$ and use a subset of $I : Q$ to produce a remaining ideal.

2 New Algorithms

First of all we prepare several propositions. In this section we fix a polynomial ring $k[x] = k[x_1, \ldots, x_n]$ over a field k and denote it by R. The following lemma is well-known.

Lemma 1. *Let I, J and Q be ideals in R. If $I \subset Q$, then $I = Q \cap (I + J) \Leftrightarrow Q \cap J \subset I$.*

Lemma 2. *Let I, J and Q be ideals in R. If $I = Q \cap (I + J)$ then $J \subset I : Q$.*

Proof. If $I = Q \cap (I + J)$ then we have $QJ \subset Q \cap J \subset Q \cap (I + J) = I$, which implies $J \subset I : Q$.

Lemma 2 means that we have to find an ideal $J \subset I : Q$ such that $J \not\subset I$ in order to obtain a non-trivial decomposition $I = Q \cap (I + J)$. Proposition 3.48 in [20] suggests that we can use $J = (I : Q)^m$ with sufficiently large m for that purpose.

Proposition 3. *There exists an integer $m > 0$ such that $(I : Q)^m \cap Q \subset I$.*

Proof. By Artin-Rees Lemma, there exists an integer $m > 0$ such that $(I : Q)^i \cap Q = (I : Q)^{i-m}((I : Q)^m \cap Q)$ for any integer $i > m$. If $i > m$, $(I : Q)^{i-m} \subset I : Q$ and we have $(I : Q)^{i-m}((I : Q)^m \cap Q) \subset (I : Q)Q \subset I$.

Corollary 4. *For any $f \in I : Q$ there exists an integer $m > 0$ such that $\langle f^m \rangle \cap Q \subset I$.*

Proposition 5. Let I be an ideal in R and $\dim I = d$. Suppose that $I = Q_1 \cap \cdots \cap Q_r \cap Q_{r+1} \cap \cdots \cap Q_s$ $(r < s)$ be an irredundant primary decomposition of I and that all isolated primary components appear in $\{Q_1, \ldots, Q_r\}$. Let $Q = Q_1 \cap \cdots \cap Q_r$. Then

1. For any integer $m > 0$ $\dim(I : Q)^m = \dim I : Q < d$.
2. For sufficiently large integer m, $I = Q \cap (I + (I : Q)^m)$ and $\dim(I + (I : Q)^m) < d$, which gives a non-trivial decomposition of I.

Proof. Since $I : Q = (Q : Q) \cap (Q_{r+1} : Q) \cap \cdots \cap (Q_s : Q) = (Q_{r+1} : Q) \cap \cdots \cap (Q_s : Q)$ and $\dim(Q_i : Q) \leq \dim Q_i < d$ for $i > r$, $\dim(I : Q) < d$ holds. By Proposition 3 there exists an integer $m > 0$ such that $(I : Q)^m \cap Q \subset I$. For such m, $I = Q \cap (I + (I : Q)^m)$ by Lemma 1 and $\dim(I + (I : Q)^m) \leq \dim(I : Q)^m = \dim I : Q < d$. In particular I is a proper subset of $I + (I : Q)^m$. Since I has an embedded component, I is a proper subset of Q, which implies that $I + (I : Q)^m$ is a proper ideal because $I + (I : Q)^m \subset I + (I : Q) \subset (I : Q) \neq R$. Thus $I = Q \cap (I + (I : Q)^m)$ is a non-trivial decomposition of I.

Definition 6. We call an ideal J satisfying

$$J \not\subset I, \quad I + J \neq R \quad \text{and} \quad I = Q \cap (I + J) \tag{3}$$

a separating ideal for (I, Q).

By using the notion of separating ideal, we propose an algorithm for primary decomposition.

Algorithm 7
Input : an ideal $I \subset R$
Output : an irredundant primary decomposition of I
$QL \leftarrow \emptyset$; $Q \leftarrow R$; $I_t \leftarrow I$
do
  if $I_t = R$ goto LAST
  $PL_t \leftarrow MinimalAssociatedPrimes(I_t)$
  $QL_t \leftarrow IsolatedPrimaryComponents(I_t, PL_t)$
  $Q_t \leftarrow \bigcap_{J \in QL_t} J$
  if $Q \not\subset Q_t$ then { $Q \leftarrow Q \cap Q_t$; $QL \leftarrow QL \cup QL_t$ }
  if $Q_t = I_t$ or $Q = I$ goto LAST
  $J_t \leftarrow SeparatingIdeal(I_t, Q_t, (I_t : Q_t))$
  $I_t \leftarrow I_t + J_t$
end do
LAST: $QL \leftarrow RemoveRedundancy(QL)$
return QL

In this algorithm, $MinimalAssociatedPrimes(I)$ returns the set of all minimal associated primes of an ideal I. $IsolatedPrimaryComponents(I, PL)$ ($PL = \{P_1, \ldots, P_k\}$) computes the set of all isolated primary components $\{Q_1, \ldots, Q_k\}$ of an ideal I, where PL is the set of all minimal associated primes of I and P_i is the associated prime of Q_i (cf. [17]). $SeparatingIdeal(I, Q, C)$ ($C = I : Q$) finds a separating ideal J for (I, Q). Finally $RemoveRedundancy(QL)$ combines primary components with the same associated prime and removes unnecessary components. We omit the details of these sub-procedures except for $SeparatingIdeal$.

Theorem 8. Algorithm 7 is correct.

Proof. I_t is strictly increasing because $J_t \not\subset I_t$ is added to I_t, which ensures the termination. During the execution QL consists of primary ideals and $I = (\cap_{J \in QL} J) \cap I_t$ holds. Therefore, after removing redundancy the output becomes an irredundant primary decomposition of I.

By Proposition 5, we know that $(I : Q)^m$ for sufficient large m is available as a separating ideal, but this is not efficient from the practical point of view, because m often has to be raised to a large value and the cost for computing $(I : Q)^m$ is high. Therefore we propose several practical methods for finding a separating ideal J.

Algorithm 9
$SeparatingIdeal1(I, Q, C)$
Input : an ideal $I \subset R$, an ideal $Q = \cap_{i=1}^{r} Q_i$, where all isolated primary components of I appear in $\{Q_1, \ldots, Q_r\}$, $C = I : Q$
Output : a separating ideal J for (I, Q)
$i \leftarrow 1$ do
$G \leftarrow$ a Gröbner basis of C^i
$H \leftarrow G \setminus Q$
if $(I + \langle H \rangle) \cap Q = I$ then return $\langle H \rangle$
else $i \leftarrow i + 1$
end do

By Lemma 1, if a subset J of $I : Q$ is a separating ideal, then J cannot contain any element in $Q \setminus I$. In Algorithm 9, $H = G \setminus Q$ does not meet Q and it is expected that $\langle H \rangle$ becomes a separating ideal with a smaller m than $(I : Q)^m$ itself.

Algorithm 10
$SeparatingIdeal2(I, Q, C)$
Input : an ideal $I \subset R$, an ideal $Q = \cap_{i=1}^{r} Q_i$, where all isolated primary components of I appear in $\{Q_1, \ldots, Q_r\}$, $C = I : Q$
Output : a separating ideal J for (I, Q)
$G \leftarrow$ a Gröbner basis of $I : Q$
$H = \{h_1, \ldots, h_k\} \leftarrow G \setminus \sqrt{I}$
$S \leftarrow \{h_i^{m_i} (i = 1, \ldots, k) \mid (I + \langle h_i^{m_i - 1} \rangle) \cap Q \neq I, (I + \langle h_i^{m_i} \rangle) \cap Q = I\}$
$S_0 \leftarrow$ a subset of S satisfying $(I + \langle S_0 \rangle) \cap Q = I$
return $\langle S_0 \rangle$

For each $s \in I : Q$, Corollary 4 ensures that there exists m such that $(I + \langle s^m \rangle) \cap Q = I$. Furthermore, if $s \notin \sqrt{I}$ then $s^m \notin I$ for all m and $J = \langle s^m \rangle$ is a separating ideal for (I, Q). $\dim I : Q < \dim I$ implies $I : Q \not\subset \sqrt{I}$ and $H \neq \emptyset$ in Algorithm 10.

Remark 11. *Algorithm 9 and Algorithm 10 depend on a choice of a term order used for computing a Gröbner basis of* $(I : Q)$ *or* $(I : Q)^m$. *In fact our experiment shows that there are cases where the choice critically affects the efficiency of the*

whole procedure. In Algorithm 10, we have to specify a method for choosing S_0. *This will be discussed later.*

In Algorithm 7, a separating ideal J_t is computed from the ideal quotient $I_t : Q_t$. We found that this construction often makes the subsequent computation hard. Instead of using $I_t : Q_t$ we can use $I : Q$ for computing a separating ideal, which leads to another version of primary decomposition algorithm.

Algorithm 12
Input : an ideal $I_{in} \subset R$
Output : an irredundant primary decomposition of I_{in}
$QL_{in} \leftarrow \emptyset$; $Q_{in} \leftarrow R$; $I_t \leftarrow I_{in}$
RESTART: $QL \leftarrow \emptyset$; $Q \leftarrow R$; $I \leftarrow I_t$; $C = \{0\}$
do
(1) if $I_t = R$ goto LAST
$PL_t \leftarrow MinimalAssociatedPrimes(I_t)$
$QL_t \leftarrow IsolatedPrimaryComponents(I_t, PL_t)$
$Q_t \leftarrow \bigcap_{J \in QL_t} J$
(2) if $Q \subset Q_t$ goto RESTART
else $Q \leftarrow Q \cap Q_t$
if $Q_{in} \not\subset Q_t$ then { $QL \leftarrow QL \cup QL_t$; $Q_{in} \leftarrow Q_{in} \cap Q_t$; $QL_{in} \leftarrow QL_{in} \cup QL_t$ }
if $Q_t = I_t$ or $Q = I$ or $Q_{in} = I_{in}$ goto LAST
else $C_t \leftarrow I : Q$
(3) if $C_t = C$ goto RESTART
else $C \leftarrow C_t$
$J_t \leftarrow SeparatingIdeal(I, Q, C)$
$I_t \leftarrow I + J_t$
end do
LAST: $QL_{in} \leftarrow RemoveRedundancy(QL_{in})$
return QL_{in}

Theorem 13. Algorithm 12 is correct.

Proof. In Algorithm 12, I is the current target ideal to be decomposed and its primary components are extracted from the current remaining ideal I_t and added to QL, the current list of primary components of I. $I_{in} = Q_{in} \cap I$ and $I = Q \cap I_t$ hold at (1). Since Q is decreasing, $C = I : Q$ is strictly increasing unless $C_t = C$ holds at (3). If $C_t = C$ holds at (3), then we reset the computation with $I = I_t$. The check of $Q \subset Q_t$ at (2) is to detect $C_t = C$ at (3) in advance. Thus we will reach RESTART or LAST in finite steps. At RESTART, after the replacement of I, I is strictly larger than previous I. Therefore Algorithm 12 terminates and outputs an irredundant primary decomposition of I_{in}.

3 Implementation Issues

We implemented the new algorithms in Risa/Asir. For an efficient implementation of a primary decomposition algorithm, we need efficient implementations of many sub-procedures. In this section we explain several ones among them.

3.1 Computation of Minimal Associated Primes

For realizing an efficient implantation of an SY-like algorithm, an efficient implementation of minimal associated prime computation is necessary. This is an issue that should be considered independently and we only show a rough sketch of our current implementation.

1. The SL algorithm
   The SL algorithm [12] is an algorithm for computing the set of all minimal associated primes of an ideal. It works as follows. Let $P_1, \ldots, P_l$ be minimal primes of an ideal I and suppose that $\sqrt{I}$ is a proper subset of $J = P_1 \cap \cdots \cap P_l$. Then we can find an element $f \in J \setminus \sqrt{I}$, and we can compute new minimal primes from $I : f^\infty$ via a reduction to a zero-dimensional case.
2. Zero-dimensional intermediate decomposition
   For a zero-dimensional ideal $I \subset K[x_1, \ldots, x_n]$, before searching a linear combination of variables in a normal position, we compute $I \cap K[x_i] = \langle m_i(x_i) \rangle$ for each variable. If $m_i = m_{i1}^{n_1} \cdots m_{ij}^{n_j}$ then $\sqrt{I} = \sqrt{\langle I, m_{i1} \rangle} \cap \cdots \cap \sqrt{\langle I, m_{ij} \rangle}$. We repeat this for all variables and obtain an intermediate decomposition $I = I_1 \cap \cdots \cap I_m$. In this decomposition, the minimal polynomials of all the variables modulo I_i are all irreducible.
3. Zero-dimensional complete decomposition
   During an execution of the intermediate decomposition, some components are known to be prime ideals, because I_i obtained in Step 2 is prime if $\dim_K K[x_1, \ldots, x_n]/I_i = \deg(m_j(x_j))$ for some j, where $m_j(x_j)$ is the minimal polynomial of x_j module I_i. After the intermediate decomposition, we have to decompose the remaining intermediate components completely. This is done by generating a linear combination of the variables which is in a normal position.

3.2 Computation of a Separating Ideal

The termination of Algorithm 7 and 12 does not depend on a choice of a separating ideal, but the efficiency heavily depends on it. Our preliminary experiment shows that Algorithm 9 is impractical and Algorithm 10 is preferable. We show two methods for computing S_0 in Algorithm 10.

– Partial search

$S_0 \leftarrow \emptyset$
for $i = 1$ to k do
  $m \leftarrow$ an integer satisfying $(I + \langle h_i^{m-1} \rangle) \cap Q \neq I$ and $(I + \langle h_i^m \rangle) \cap Q = I$
  if $(I + \langle S_0 \cup \{h_i^m\} \rangle) \cap Q = I$ then $S_0 \leftarrow S_0 \cup \{h_i^m\}$
  else exit this loop
end do

– Full search

$S \leftarrow \{h_i^{m_i}(i=1,\ldots,k) \mid (I+\langle h_i^{m_i-1}\rangle)\cap Q \neq I, (I+\langle h_i^{m_i}\rangle)\cap Q = I\}$
$l \leftarrow$ the largest index such that $(I+\{h_i^{m_i}(i=1,\ldots,l)\})\cap Q = I$
(The index l can be searched by a binary search.)
$S_0 \leftarrow \{h_i^{m_i}(i=1,\ldots,l)\}$
for $i = l+1$ to k do
  if $(I+\langle S_0 \cup \{h_i^m\}\rangle)\cap Q = I$ then $S_0 \leftarrow S_0 \cup \{h_i^m\}$
end do

3.3 Gröbner Basis Computation

The efficiency of Gröbner basis computation is crucial for all part of the implementation. We have already made much effort to make Gröbner basis computation efficient for a wide variety of input ideals. Here we only present two newly introduced features.

1. Incremental computation
   In a Gröbner basis computation, if a part of an input ideal is known to be a Gröbner basis then we can omit the computation of S-polynomials constructed from that part. This feature has been implemented in the current version of Risa/Asir and used in computations of saturation and ideal intersection.
2. Competitive computation
   There are several cases where it is difficult to decide a strategy of Gröbner basis computation. For example, the homogenization is often useful to suppress intermediate coefficient swells in a Gröbner basis computation, but it may make the computation inefficient by another reason. Another example is a Gröbner basis computation over $K(x)$, a field of rational functions, for which there are two choices : computation over $K(x)$ itself, or over K with some elimination order. If we fix one of them in our implementation, a primary decomposition procedure may get stuck in a Gröbner basis computation. In order to avoid such difficulties a competitive computation by OpenXM protocol[13] has been implemented. When this feature is activated, two methods are executed on remote OpenXM servers, with the result returned first used. Then the remaining server is reset for the subsequent requests.

4 Experiments

In this section we show some results of primary decomposition by Algorithm 7 and 12. It is clear that these algorithms are not superior to existing algorithms for ideals without embedded components. Therefore we focus on ideals with embedded components and hard to be decomposed by existing implementations.

4.1 Ideals Related to Computation of Local b-Functions

Let $D = \mathbf{Q}\langle x, \partial_x\rangle$ be an n-dimensional Weyl algebra over $\mathbf{Q}$. For a polynomial $f(x) = f(x_1,\ldots,x_n) \in \mathbf{Q}[x]$, we define an ideal $D\langle t, \partial_t\rangle$:

$$I_f = \langle \partial_{x_1} + \frac{\partial f}{\partial x_1}\partial_t, \ldots, \partial_{x_n} + \frac{\partial f}{\partial x_n}\partial_t \rangle.$$

We set $w = (1, 0, \ldots, 0) \in \mathbf{Z}^{n+1}$ and consider a weight vector $(-w, w)$ for variables $(t, x_1, \ldots, x_n, \partial_t, \partial_{x_1}, \ldots, \partial_{x_n})$. Then an ideal $J_f = \mathrm{in}_{(-w,w)}(I_f) \cap \mathbf{Q}[t\partial_t]$ can be regarded as an ideal in a commutative polynomial ring $\mathbf{Q}[x_1, \ldots, x_n, s]$ with $s = t\partial_t$. An algorithm for computing local b-functions of f at all points was presented by Oaku [15], in which a primary decomposition of J_f is needed. For f with complicated non-isolated singularities a primary decomposition of J_f is often very hard and it is interesting to try such primary decompositions by our new algorithm.

In [14] we considered examples from singularity theory.

- A_k singularity
  For $f_k(x, u_1, \ldots, u_k) = x^{k+1} + u_1 + u_2 x + \cdots + u_k x^{k-1} (k \geq 1)$, we consider its discriminant $\mathrm{disc}(f_k) = \mathrm{resultant}_x(f_k(x, u), f_k'(x, u)) \in \mathbf{Q}[u_1, \ldots, u_k]$.
- D_k singularity
  For $f_k(x, y, u_1, \ldots, u_k) = x^2 y - y^{k-1} + u_1 + u_2 x + u_3 x^2 + u_4 y + \cdots + u_k y^{k-3} (k \geq 4)$, there exists $g_k \in \mathbf{Q}[u_1, \ldots, u_k]$ such that $\langle f_k, \frac{\partial f_k}{\partial x}, \frac{\partial f_k}{\partial y} \rangle \cap \mathbf{C}[u_1, \ldots, u_k] = \langle g_k \rangle$.

Here we try primary decomposition of $J_{\mathrm{disc}(f_k)}$, $J_{g_k} \subset \mathbf{Q}[u_1, \ldots, u_k, s]$. In general J_f contains a polynomial $b(s)$ (the global b-function) and J_f is decomposed as $J_f = (J_f + \langle (s - s_1)^{m_1} \rangle) \cap \cdots \cap (J_f + \langle (s - s_l)^{m_l} \rangle)$ according to the irreducible factorization of $b(s)$ over $\mathbf{Q}$, $b(s) = (s - s_1)^{m_1} \cdots (s - s_l)^{m_l}$. Then our task is to compute a primary decomposition of each $J_f + \langle (s - s_i)^{m_i} \rangle$. If we try these computations, we notice that it is hard when $m_i > 1$. We show some results obtained by our new algorithms. In the following Q is the unique isolated prime component and R_i are embedded primary components.

$$I_1 = J_{\mathrm{disc}(f_4)} + \langle s^2 \rangle = Q \cap R_1,$$
$$I_2 = J_{\mathrm{disc}(f_5)} + \langle s^3 \rangle = Q \cap R_1 \cap R_2,$$
$$I_3 = J_{g_5} + \langle s^4 \rangle = Q \cap R_1 \cap R_2 \cap R_3 \cap R_4.$$

All the existing implementations fail to decompose I_2 and I_3. It may seem that I_1 is easy to decompose, but only GTZ in Singular and EHV in Macaulay2 can decompose it. The difficulty in the other implementations is caused by the remaining ideal $I' = I + \langle f \rangle$ which appears just after Q is obtained. I_1, Q and R_1 have the following generators:

$$I_1 = \langle h_1, sh_2, \ldots, sh_9, s^2 \rangle,\ Q = \langle h_1, s \rangle,\ R_1 = \langle h_2, \ldots, h_9, s^2 \rangle$$

where $h_1 = \mathrm{disc}(f_4)$, $h_2, \ldots, h_9 \in \mathbf{Q}[u_1, u_2, u_3, u_4]$. We can confirm that $R_1 = I_1 + J$ for $J = \langle h_2, \ldots, h_9 \rangle$ and $I_1 : Q = \langle S \rangle$, $S = \{h_2, \ldots, h_9, s\}$. We observed that $J = \langle S \setminus Q \rangle$ and it was the first step toward our new algorithm.

4.2 Ideals of Adjacent Minors

For an $m \times n$ matrix $X = (x_{ij})_{i=1,\ldots,m,j=1,\ldots,n}$ where $x_{i,j}$ are indeterminates, we consider the ideal $A_{k,m,n}$ in $\mathbf{Q}[X]$ generated by adjacent $k \times k$-minors of X. Primary decompositions of $A_{k,m,n}$ are known for several (k,m,n). For example, irreduntant primary decompositions of $A_{2,4,4}$ and $A_{2,3,5}$ via cellular decomposition [9] are given in [7] [19] and [11] respectively. We could compute primary decompositions of $A_{2,4,4}$, $A_{2,4,5}$, $A_{2,3,k}$ $(k \leq 8)$ by our new algorithm. The results are:

$$\begin{aligned}
A_{2,3,4} &= Q_1 \cap \cdots \cap Q_6 \cap R_1 \cap R_2 \cap R_3,\\
A_{2,3,5} &= Q_1 \cap \cdots \cap Q_{10} \cap R_1 \cap \cdots \cap R_9,\\
A_{2,3,6} &= Q_1 \cap \cdots \cap Q_{18} \cap R_1 \cap \cdots \cap R_{23},\\
A_{2,3,7} &= Q_1 \cap \cdots \cap Q_{32} \cap R_1 \cap \cdots \cap R_{56},\\
A_{2,3,8} &= Q_1 \cap \cdots \cap Q_{57} \cap R_1 \cap \cdots \cap R_{131},\\
A_{2,4,5} &= Q_1 \cap \cdots \cap Q_{15} \cap R_1 \cap \cdots \cap R_{17},\\
A_{2,4,5} &= Q_1 \cap \cdots \cap Q_{35} \cap R_1 \cap \cdots \cap R_{61},
\end{aligned}$$

where Q_i are isolated prime components and R_i are embedded primary components.

4.3 Timings

We show timing data for I_1, I_2, I_3, I_4 and $A_{2,4,4}$, $A_{2,4,5}$, $A_{2,3,k}$ $(4 \leq k \leq 8)$ where I_4 is an ideal by Huneke, which is the hardest problem in [6]:

$$I_4 = \langle s^{15}, t^{15}, u^{15}, u^5 - s^3tx + s^2t^2x + s^2t^2y - st^3y \rangle \subset \mathbf{Q}[x,y,z,t,u].$$

We presented Algorithm 7 and Algorithm 12. We apply Algorithm 10 for computing a separating ideal. For computing S_0 in Algorithm 10 we implemented the both methods in Section 3.2. Based on preliminary experiments we decided to apply the full search for adjacent minors and the partial search for the others.

In Table 1 and Table 2, Total, Colon and Sep show the total time, the time for computing $I : Q$, and the time for computing separating ideals respectively and they are given in seconds. #Iso and #Emb show the number of isolated components and the number of embedded components respectively. In #Emb, the number in parentheses is the number of embedded components before executing *RemoveRedundancy*(). In all computations the competitive computation is not used. Timings were measured on a 64-bit Linux machine with Intel Xeon X5570, 2.93GHz. The following functions are available in `noro_pd.rr`, which is written in Asir user language and is contained in the OpenXM package [3]. It runs on Asir version 20100526 or on the later version.

– `noro_pd.syc_dec(Ideal,Vars)`
`Ideal` is a list of polynomials with variables `Vars`. This function executes Algorithm 7 for `Ideal` and returns a pair of lists $[Iso, Emb]$, where $Iso = [[Q_1, \sqrt{Q_1}], \ldots]$ and $Emb = [[R_1, \sqrt{R_1}], \ldots]$ are isolated and embedded components respectively.

– `noro_pd.syca_dec(Ideal,Vars)`
  This function executes Algorithm 12. The arguments and the output are the same as above.

An option `sepideal=`n specifies an algorithm for computing a separating ideal: Algorithm 9 for $n = 0$, Algorithm 10 with the partial search of S_0 for $n = 1$ (default) and Algorithm 10 with the full search of S_0 for $n = 2$.

Table 1. Algorithm 7

Ideal	Total	Colon	Sep	#Iso	#Emb
I_1	0.05	0	0.01	1	1(1)
I_2	0.6	0.03	0.1	1	2(2)
I_3	17	0.4	1.5	1	4(12)
I_4	470	210	180	1	4(4)
$A_{2,3,4}$	13	5	5	6	3(23)
$A_{2,3,5}$	> 20h	–	–	–	–

Table 2. Algorithm 12

Ideal	Total	Colon	Sep	#Iso	#Emb
I_1	0.08	0.004	0.004	1	1 (1)
I_2	0.9	0.03	0.1	1	2 (2)
I_3	17	0.6	1.7	1	4 (8)
I_4	108	9.8	38	1	4 (4)
$A_{2,3,4}$	0.5	0.03	0.1	6	3(3)
$A_{2,3,5}$	5	0.2	2.3	10	9(9)
$A_{2,3,6}$	133	2.8	48	18	23(23)
$A_{2,3,7}$	3540	25	2090	32	56(56)
$A_{2,3,8}$	146h	284	62h	57	131(131)
$A_{2,4,4}$	31	1.8	15	15	17(21)
$A_{2,4,5}$	12700	102	7800	35	61(68)

Although the number of ideals tested here is very small, except for I_1 and $A_{2,3,4}$, they are all very hard or practically impossible to be decomposed by existing implementations. For example, it takes 21 hours to compute a primary decomposition of $A_{2,3,5}$ by SY in Macaulay2, and EHV in Macaulay2 abnormally terminates for the same input. Both Algorithm 7 and Algorithm 12 can decompose many of them. We note that we have not yet implemented any technique to avoid redundant components except for the early termination by keeping the intersection of obtained components during an execution. In spite of this, Table 1 and Table 2 show that the number of redundant components is small. We could say that this is a benefit of using $I : Q$ for constructing a large remaining ideal. In particular Algorithm 12 succeeds in computing the decompositions of $A_{k,m,n}$ with a very small number of redundant components, while many of them cannot be decomposed by Algorithm 7. It is a future work to analyze the precise reason of the success. Furthermore, there are many sub-procedures to be improved. Currently the most time-consuming part in the whole procedure is the computation of a separating ideal. Many sub-procedures including this depend on a function to compute the intersection of ideals and it is an important task to improve its efficiency.

References

1. Risa/Asir, A.: computer algebra system, `http://www.math.kobe-u.ac.jp/Asir/asir.html`
2. Macaulay 2 home page, `http://www.math.uiuc.edu/Macaulay2/`

3. OpenXM committers, OpenXM, a project to integrate mathematical software systems (1998-2010), http://www.openxm.org
4. Decker, W., Greuel, G.-M., Pfister, G., Schönemann, H.: Singular 3-1-1 — A computer algebra system for polynomial computations, http://www.singular.uni-kl.de/
5. Bahloul, R., Oaku, T.: Local Bernstein-Sato ideals: algorithm and examples. J. Symb. Comp. 45, 46–59 (2010)
6. Decker, W., Greuel, G.-M., Pfister, G.: Primary decomposition: Algorithms and comparisons. In: Algorithmic Algebra and Number Theory, pp. 187–220. Springer, Heidelberg (1998)
7. Diaconis, P., Eisenbud, D., Sturmfels, B.: Lattice walks and primary decomposition. In: Sagan, B., Stanley, R. (eds.) Mathematical Essays in Honor of Gian-Carlo Rota, pp. 173–194. Birkhäuser, Basel (1998)
8. Eisenbud, D., Huneke, C., Vasconcelos, W.: Direct methods for primary decomposition. Invent. Math. 110, 207–235 (1992)
9. Eisenbud, D., Sturmfels, B.: Binomial Ideals. Duke Math. J. 84, 1–45 (1996)
10. Gianni, P., Trager, B., Zacharias, G.: Gröbner basis and primary decomposition of polynomial ideals. J. Symb. Comp. 6, 149–167 (1988)
11. Hoşten, S., Shapiro, J.: Primary Decomposition of Lattice Basis Ideals. J. Symb. Comp. 29, 625–639 (2000)
12. Laplagne, S.: An Algorithm for the Computation of the Radical of an Ideal. In: Proc. ISSAC 2006, pp. 191–195. ACM Press, New York (2006)
13. Maekawa, M., Noro, M., Ohara, K., Takayama, N., Tamura, Y.: The Design and Implementation of OpenXM-RFC 100 and 101. In: Proc. ASCM 2001, pp. 102–111. World Scientific, Singapore (2001)
14. Nishiyama, K., Noro, M.: Stratification associated with local b-functions. J. Symb. Comp. 45, 462–480 (2010)
15. Oaku, T.: Algorithms for b-Functions, Restrictions, and Algebraic Local Cohomology Groups of D-Modules. Advances in Applied Mathematics 19, 61–105 (1997)
16. Shibuta, T.: An algorithm for computing multiplier ideals (2010) (preprint), arXiv:0807.4302v6
17. Shimoyama, T., Yokoyama, K.: Localization and Primary Decomposition of Polynomial ideals. J. Symb. Comp. 22, 247–277 (1996)
18. Shioda, T.: Gröbner basis, Mordell-Weill lattices and deformation of singularities, II. Proc. Japan Acad. Ser. A Math. Sci. 86(2), 27–32 (2010)
19. Sturmfels, B.: Solving Systems of Polynomial Equations. In: CBMS Regional Conference Series in Mathematics, vol. 97. AMS, Providence (2002)
20. Vasconcelos, W.: Computational Methods in Commutative Algebra and Algebraic Geometry. In: Algorithms and Computation in Mathematics, vol. 2. Springer, Heidelberg (1998)

An Automated Confluence Proof for an Infinite Rewrite System Parametrized over an Integro-Differential Algebra

Loredana Tec[1,⋆], Georg Regensburger[2], Markus Rosenkranz[3], and Bruno Buchberger[1]

[1] Research Institute for Symbolic Computation, Johannes Kepler University, 4032 Castle of Hagenberg, Austria
[2] Johann Radon Institute for Computational and Applied Mathematics, Austrian Academy of Sciences, Altenberger Str. 69, 4040 Linz, Austria
[3] School of Mathematics, Statistics and Actuarial Science, University of Kent, Canterbury CT2 7NF, United Kingdom

1 Introduction

In our symbolic approach to boundary problems for linear ordinary differential equations we use the algebra of *integro-differential operators* as an algebraic analogue of differential, integral and boundary operators (Section 2). They allow to express the problem statement (differential equation and boundary conditions) as well as the solution operator (an integral operator called "Green's operator"), and they are the basis for operations on boundary problems like solving and factoring [14,17]. A survey of the implementation is given in [18].

The integro-differential operators are realized by a noetherian and confluent rewrite system [17]. From a ring-theoretic point of view, this rewrite system constitutes a basis for the ideal of relations among the fundamental operators, and confluence means we have a noncommutative *Gröbner basis* [3,4,2,9]. However, since the relation ideal is infinitely generated in a polynomial ring with infinitely many indeterminates, none of the known implementations [13] is applicable.

This is why the *confluence proof* is somewhat subtle (Section 3). The generators for the relation ideal are parametrized over a given integro-differential algebra, and the reduction of S-polynomials must incorporate the computational laws of the latter. The automated proof in [15] has achieved this in an ad-hoc manner for the special case of what was called "analytic algebras" there. In our new proof, the computational laws of integro-differential algebras are internalized by using so-called integro-differential polynomials [16] in the formation of the S-polynomials. We also refer to [19] for a detailed presentation of the new automated proof and the corresponding integro-differential structures.

We use a prototype *implementation* of integro-differential polynomials and reduction rings, based on Theorema and available at `www.theorema.org`. The Theorema system was designed by B. Buchberger as an integrated environment for proving, solving and computing in various domains [6]. Implemented on top

⋆ Recipient of a DOC-fFORTE-fellowship of the Austrian Academy of Sciences.

K. Fukuda et al. (Eds.): ICMS 2010, LNCS 6327, pp. 245–248, 2010.

of Mathematica, its core language is higher-order predicate logic and contains a natural programming language such that algorithms can be coded and verified in a unified formal frame, using the powerful tool of functors for building up a hierarchy of parametrized domains; for more details and references see [8].

2 Integro-Differential Polynomials and Operators

We need an algebraic structure having *differentiation along with integration*. In the following definition [17], one may think of the standard example $\mathcal{F} = C^\infty(\mathbb{R})$, where $\partial = {}'$ is the usual derivation and $\int$ the integral operator $f \mapsto \int_a^x f(\xi)\,d\xi$ for $a \in \mathbb{R}$. The section axiom corresponds to the Fundamental Theorem of Calculus, the differential Baxter axiom to Integration by Parts. Scalars are over a field K. For the similar notion of differential Rota-Baxter algebras, we refer to [10].

Definition 1. *An* integro-differential algebra $(\mathcal{F}, \partial, \int)$ *is a commutative differential K-algebra $(\mathcal{F}, \partial)$ with a K-linear section $\int$ of ∂, meaning $(\int f)' = f$, such that the differential Baxter axiom $(\int f')(\int g') + \int (fg)' = (\int f')g + f(\int g')$ holds.*

Let $(\mathcal{F}, \partial, \int)$ be an integro-differential algebra of "coefficients". Then the *integro-differential operators* $\mathcal{F}[\partial, \int]$, introduced in [17] as a generalization of the "Green's polynomials" of [15], are defined as the quotient—modulo the rewrite rules from the table below—of the noncommutative polynomial ring over K in the following indeterminates: ∂ and $\int$, the "functions" $f \in \mathcal{F}$, and the multiplicative "functionals" φ. The functions f range over a basis of $\mathcal{F}$; the multiplicative functionals (or characters) $\varphi\colon \mathcal{F} \to K$ are typically point evaluations, and they must include the *evaluation* $\mathtt{E} = 1 - \int \partial$, which is $\mathtt{E}(f) = f(a)$ in the above example. In the rewrite rules, we use f and g range over functions, φ and ψ over multiplicative functionals.

$fg \to f \cdot g$	$\partial f \to \partial \cdot f + f\partial$	$\int f \int \to (\int \cdot f) \int - \int (\int \cdot f)$
$\varphi\psi \to \psi$	$\partial\varphi \to 0$	$\int f \partial \to f - \int (\partial \cdot f) - (\mathtt{E} \cdot f)\,\mathtt{E}$
$\varphi f \to (\varphi \cdot f)\,\varphi$	$\partial \int \to 1$	$\int f \varphi \to (\int \cdot f)\,\varphi$

Theorem 1. *The above rewrite system is noetherian and confluent.*

As explained before, one may find an outline of a manual proof for this theorem in [17], but the purpose of the present paper is to sketch a new automated proof based on the algebra of *integro-differential polynomials*. The precise definition as an instance of the universal polynomial construction [12,7,1] is tedious [16], but the underlying intuition is perfectly clear since one just adjoins an indeterminate function u to the given integro-differential algebra $\mathcal{F}$. The integro-differential polynomials are an extension of the usual differential polynomials [11] and in analogy we denote them by $\mathcal{F}\{u\}$. A proof of the following theorem can be found in [19].

Theorem 2. *The integro-differential polynomials $\mathcal{F}\{u\}$ constitute an integro-differential algebra with an algorithmic canonical simplifier.*

Unlike the integro-differential operators, $\mathcal{F}\{u\}$ is thus a commutative integro-differential algebra, and its multiplication is realized by the so-called shuffle product. While the definition of the derivation is straightforward and similar to differential polynomials, the integral must be defined by a careful case distinction on the differential exponents [16,19]. Note that integro-differential polynomials act as nonlinear differential and integral operators on $\mathcal{F}$. A typical integro-differential polynomial for $\mathcal{F} = K[x]$ is given by $4u(0)^4u^2\int u'^3 + \int(x^6uu''^5\int(x^2e^{4x}u^3u'^2\int u^4))$. For computational purposes, we have implemented a *canonical simplifier*, identifying different expressions that denote the same integro-differential polynomial.

3 An Automated Confluence Proof

As announced in the Introduction, the integro-differential polynomials are used for proving the confluence of the above rewrite rules defining the relations for $\mathcal{F}[\partial,\int]$. Equivalently, we show that the noncommutative polynomials given by the difference between the left and right sides of the rules form a noncommutative Gröbner basis. For handling parametrized polynomial reduction and S-polynomials, we use a noncommutative adaption of *reduction rings*, i.e. rings with so-called reduction multipliers in the sense of [5]. As usual, we show that all S-polynomials reduce to zero.

Since the rewrite rules contain two generic functions f and g, one can view the corresponding *S-polynomials* as elements of $\widetilde{\mathcal{F}}[\partial,\int]$ with $\widetilde{\mathcal{F}} = \mathcal{F}\{u,v\}$. Here $\mathcal{F}\{u,v\} = (\mathcal{F}\{u\})\{v\}$ denotes the integro-differential polynomials in two indeterminates. More precisely, we reason as follows: If we know that an S-polynomial reduces to zero as such, the same is true after substituting the functions $f,g \in \mathcal{F}$ for u,v. Note the subtle shift between object and meta level when we use the instance of the rewrite system for the integro-differential operators $\widetilde{\mathcal{F}}[\partial,\int]$ over integro-differential polynomials for proving the confluence of the rewrite rules for integro-differential operators over arbitrary integro-differential algebras—this proof needs only rewriting not confluence! (Actually one should also treat the functionals φ,ψ in analogy to the functions f,g, but the former are much simpler than the latter.) We refer again to [19] for further details.

We can now use Theorema for checking whether an S-polynomial reduces to zero. All S-polynomials are generated algorithmically, but as a concrete example we check the self-overlap $\int u \int$ and $\int v \int$ of the Baxter rule.

```
TS_In[554]:= ReducePol[((∫ · u⟨1⟩) ∫ · v⟨1⟩ ∫ - ∫ (∫ · u⟨1⟩) v⟨1⟩ ∫) -
                 (∫ u⟨1⟩ (∫ · v⟨1⟩) ∫ - ∫ u⟨1⟩ ∫ (∫ · v⟨1⟩))]

TS_Out[554]=
0
```

It turns out that there are 72 S-polynomials, and indeed all of them reduce to zero. Hence we conclude that the rewrite system for $\mathcal{F}[\partial,\int]$ is confluent.

References

1. Aichinger, E., Pilz, G.F.: A survey on polynomials and polynomial and compatible functions. In: Proceedings of the Third International Algebra Conference, pp. 1–16. Kluwer Acad. Publ., Dordrecht (2003)
2. Bergman, G.M.: The diamond lemma for ring theory. Adv. in Math. 29(2), 178–218 (1978)
3. Buchberger, B.: An algorithm for finding the bases elements of the residue class ring modulo a zero dimensional polynomial ideal (German). PhD thesis, Univ. of Innsbruck (1965); English Translation J. Symbolic Comput. 41(3-4), 475–511 (2006)
4. Buchberger, B.: Introduction to Gröbner bases. In: Buchberger, B., Winkler, F. (eds.) Gröbner Bases and Applications, pp. 3–31. Cambridge Univ. Press, Cambridge (1998)
5. Buchberger, B.: Gröbner rings and modules. In: Proceedings of SYNASC 2001, pp. 22–25 (2001)
6. Buchberger, B., et al.: Theorema: Towards computer-aided mathematical theory exploration. J. Appl. Log. 4(4), 359–652 (2006)
7. Buchberger, B., Loos, R.: Algebraic simplification. In: Computer Algebra, pp. 11–43. Springer, Vienna (1983)
8. Buchberger, B., Regensburger, G., Rosenkranz, M., Tec, L.: General polynomial reduction with Theorema functors: Applications to integro-differential operators and polynomials. ACM Commun. Comput. Algebra 42(3), 135–137 (2008)
9. Bueso, J., Gómez-Torrecillas, J., Verschoren, A.: Algorithmic methods in non-commutative algebra. Kluwer Academic Publishers, Dordrecht (2003)
10. Guo, L., Keigher, W.: On differential Rota-Baxter algebras. J. Pure Appl. Algebra 212(3), 522–540 (2008)
11. Kolchin, E.: Differential algebra and algebraic groups. Pure and Applied Mathematics, vol. 54. Academic Press, New York (1973)
12. Lausch, H., Nöbauer, W.: Algebra of polynomials. North-Holland Publishing Co., Amsterdam (1973)
13. Levandovskyy, V.: PLURAL, a non-commutative extension of SINGULAR: past, present and future. In: Iglesias, A., Takayama, N. (eds.) ICMS 2006. LNCS, vol. 4151, pp. 144–157. Springer, Heidelberg (2006)
14. Regensburger, G., Rosenkranz, M.: An algebraic foundation for factoring linear boundary problems. Ann. Mat. Pura. Appl. (4) 188(1), 123–151 (2009)
15. Rosenkranz, M.: A new symbolic method for solving linear two-point boundary value problems on the level of operators. J. Symbolic Comput. 39(2), 171–199 (2005)
16. Rosenkranz, M., Regensburger, G.: Integro-differential polynomials and operators. In: Proceedings of ISSAC 2008, pp. 261–268. ACM, New York (2008)
17. Rosenkranz, M., Regensburger, G.: Solving and factoring boundary problems for linear ordinary differential equations in differential algebras. J. Symbolic Comput. 43(8), 515–544 (2008)
18. Rosenkranz, M., Regensburger, G., Tec, L., Buchberger, B.: A symbolic framework for operations on linear boundary problems. In: Gerdt, V.P., Mayr, E.W., Vorozhtsov, E.V. (eds.) CASC 2009. LNCS, vol. 5743, pp. 269–283. Springer, Heidelberg (2009)
19. Rosenkranz, M., Regensburger, G., Tec, L., Buchberger, B.: Symbolic analysis for boundary problems: From rewriting to parametrized Gröbner bases. Technical Report 2010-05, RICAM (2010)

Operadic Gröbner Bases: An Implementation

Vladimir Dotsenko[1,⋆] and Mikael Vejdemo-Johansson[2,⋆⋆]

[1] Dublin Institute for Advanced Studies and School of Mathematics, Trinity College Dublin, Ireland
[2] Department of Mathematics, Stanford University
mik@stanford.edu

1 Introduction

In an upcoming paper [1], the first author and Anton Khoroshkin define the concept of a Gröbner basis for finitely presented operads, prove the diamond lemma for these Gröbner bases, and demonstrate that having a quadratic Gröbner basis is equivalent to the existence of a Poincaré-Birkhoff-Witt basis. As demonstrated by Eric Hoffbeck [2], an operad with a PBW basis is Koszul. Thus, out of this emerges an entirely computational framework for proving Koszulness, as well as the possibility to build tools for exploration of operads by means of explicit calculation.

The authors have, in [3], provided a computer implementation of the algorithms specified by Dotsenko and Khoroshkin. At the core of the paper [1] lies the recognition that every symmetric operad can be thought of as a shuffle operad, and forgetting "unnecessary" symmetries does not affect relevant results of linear and homological algebra for operads; therefore, the respective computation may be restricted to the simpler category of shuffle operads without restricting any conclusions drawn.

1.1 Shuffle Operads

For exact definitions, we refer the reader to [1,3]. For the purpose of this short communication, we shall concentrate on a less precise definition of symmetric, and shuffle operads.

Definition 1. *A* symmetric (resp. shuffle) operad *is a collection $O(n)$ of vector spaces, one for each n, together with linear maps*

$$\circ_\sigma : O(n) \times O(m_1) \times \cdots \times O(m_n) \to O(m_1 + \cdots + m_n)$$

called composition maps. The composition maps are parametrized by arbitrary permutations in $S_{m_1+\cdots+m_n}$ (resp. by shuffle permutations of type $(m_1, \ldots, m_n)$) that provide symmetry actions to the operad. These maps are required to fulfill associativity conditions and allow for a unit for the composition.

⋆ The first author was supported by an IRCSET research fellowship.
⋆⋆ The second author was supported by the Office of Naval Research, through grant N00014-08-1-0931.

K. Fukuda et al. (Eds.): ICMS 2010, LNCS 6327, pp. 249–252, 2010.

As with so many other things, it is the identification of the *free* objects that we gain both a mental and a computational model for the objects at hand. For operads, the free objects consist of *decorated trees*, with a (rooted) tree being a non-empty connected directed graph T of topological genus 0, with each vertex being equipped with at least one incoming edge and exactly one outgoing edge. We allow for some edges to only have one vertex – the other being ignored for our purposes – call such edges external. Each tree has exactly one external edge: the *root* or *output*, and some number of ingoing edges, called *leaves*.

Given a collection M, we can consider the collection of all trees *decorated by* M, by which we mean trees such that each vertex with n outgoing edges is equipped with some element of $M(n)$. Such a decorated tree, we call a *tree monomial* over M. The collection of the vector spaces spanned by all such tree monomials with exactly n leaves is denoted by $\mathcal{F}_M(n)$, and they form a collection denoted by $\mathcal{F}_M$.

Composition in the free operad can be defined on basis elements through gluing leaves to roots, and and re-arranging the leaves in an admissible way. For symmetric operads, all permutations are allowed, while shuffle operads have a smaller class of admissible permutations: the shuffle permutations, as described in [1]. The rearranging can either be imagined by allowing branches to cross as the tree is drawn at; or by requiring a planar drawing of each tree, but instead decorating all leaves with integers, and allowing the permutations to act on these integers.

1.2 Gröbner Bases for Operads

We may define divisibility for tree monomials by saying that α divides β if there is some sequence of compositions that starts with α and ends with β. We can produce the equivalent operations to S-polynomials and reductions for the Buchberger algorithm by finding such a dividing sequence $m_{\alpha\beta}$ for a pair of leading tree monomials, according to some tree monomial ordering, and applying $m_{\alpha\beta}$ to all tree monomials in some given free operad element.

This allows us to reproduce the Buchberger algorithm, with trees and non-linear compositional structures instead of the linear monomials known from commutative and non-commutative Gröbner bases. Once we are able to define greatest common divisors and least common multiples for tree monomials, using these $m_{\alpha\beta}$-operations, the resulting algorithms look very familiar to those working with computational algebra.

Our interest in these Gröbner bases lie in part in their power for the theory of operads: providing computational proof for Koszulness of specific operads, and aiding in the computational exploration of the theory – but also in the way that Gröbner bases and the computational theory of operads encompasses all Gröbner-like theories in one single framework. Commutative and non-commutative polynomial ring Gröbner bases occur as operadic Gröbner bases concentrated in degree 1.

2 An Example

To illustrate the concepts and techniques, consider the algebraic theory that stipulates a single binary operation $*$, and requires of this the rules $(a*b)*c = -a*(b*c)$

and $(a * c) * b = a * (b * c)$. This universal algebra is captured by the operad defined as a quotient of the free shuffle operad on a single generator $\vee$ by the ideal generated by the two tree polynomials on the left. These give rise to an S-polynomial (using the PathPerm ordering described in [3]) – the one consisting of only the monomial on the right; which will not reduce further.

```
% ghci -cpp Math.Operad
*Math.Operad> let v = corolla 2 [1,2]
*Math.Operad> let [g1t1,g1t2,g2t2] =
    [shuffleCompose 1 [1,2,3] v v,
     shuffleCompose 2 [1,2,3] v v,
     shuffleCompose 1 [1,3,2] v v]
*Math.Operad> let ac =
    [(oet g1t1) + (oet g1t2), (oet g2t2) - (oet g1t2)]
    :: [OperadElement Integer Rational PathPerm]
*Math.Operad> let acGB = operadicBuchberger ac
*Math.Operad> length acGB
3
*Math.Operad> putStrLn $ pp acGB
[
+1 % 1*m2(m2(1,3),2)
+(-1) % 1*m2(1,m2(2,3)),

+1 % 1*m2(m2(1,2),3)
+1 % 1*m2(1,m2(2,3)),

+2 % 1*m2(1,m2(2,m2(3,4))),
]
```

Fig. 1. Example session: computing the Gröbner basis of the operad that controls anti-commuting associative algebras: $m(m(a,b),c) = -m(a,m(b,c))$ and $m(m(a,c),b) = m(a,m(b,c))$. The symbol % denotes an element of $\mathbb{Q}$, so that $a\%b = \frac{a}{b}$. The first two clusters in the output are the original generators, and the third is the non-trivial S-polynomial added by the computation.

3 Implementing Gröbner Bases for Operads

For the implementation of the Buchberger algorithm, we chose to work in the programming language Haskell.[1] [4] For a variety of reasons, including the relative ease with which new datatypes can be constructed, and mathematical thoughts can be all but transliterated into the programming language itself, this choice made the production of a first implementation easy and even pleasant.

[1] A working version of our software package can be found at the Hackage repository `http://hackage.haskell.org/package/Operads`

However, there are plenty of issues with the implementation, and its platform. The platform choice makes the package uninviting to the casual user, and hard to integrate into other mathematical software systems. The relatively specialized programming paradigm is unusual enough that the language itself forms an additional barrier to entry, and both optimization, debugging and code analysis are made more difficult by the sometimes highly unintuitive way that the declarative programs transform into machine code.

At the core of the representation of an operad is the tree; and thus the ease with which Haskell represents trees helped making the implementation work easy; with a definition somewhat similar to

```
data Tree = Leaf | Node [Tree]
```

equipped with appropriate functionality and decorations, implementing the arithmetic of free operads became straightforward. However, it is with the trees that the core of the difficulty implementing these algorithms lies as well: in our profiling experiments, the real deep time sinks have invariably been functions that traverse the trees in order to determine their value – primarily the monomial ordering functions that get called repeatedly by every single operation that modifies a tree polynomial. Introducing caching to the polynomial storage type helps, but even so, tree traversals take up most of the time.

Hence, any other implementation will have to be carefully done to capture the tree structures in a way that is amenable to manipulation while still efficient in handling.

We would like to call for further implementations; for the modification of this implementation into something more accessible, more usable, faster, leaner, and better.

References

1. Dotsenko, V., Khoroshkin, A.: Gröbner bases for operads. Duke Math. Journal 153(2), 363–396 (2010)
2. Hoffbeck, E.: A Poincaré–Birkhoff–Witt criterion for Koszul operads. Manuscripta Mathematica 131, 87–110 (2010)
3. Dotsenko, V., Vejdemo-Johansson, M.: Implementing Gröbner bases for operads. Séminaires et Congrès (2009) (to appear)
4. Jones, S.P. (ed.): Haskell 98 language and libraries: the revised report. Cambridge Univ. Pr., Cambridge (2003)

Magma - A Tool for Number Theory

John Cannon, Steve Donnelly, Claus Fieker, and Mark Watkins

School of Mathematics and Statistics, University of Sydney
{john,donnelly,claus,watkins}@maths.usyd.edu.au

Magma [1,2,5] is a computer algebra system developed by the group of John Cannon at the University of Sydney, together with many collaborators around the world, and was first released in 1994. Based on experience obtained from the group theory system Cayley (1975–2005), also developed by Cannon et al, Magma is designed to be a general algebra system with an strong emphasis on the structural aspects of algebra. The goal is to provide a framework for implementing algorithms at a much higher level of abstraction than CA systems such as Maple and Mathematica. Magma currently provides support for most of classical algebra (groups, rings, fields), algebraic geometry, algebraic combinatorics and coding theory (this list is not exhaustive.)

A key aspect of the Magma philosophy is close integration of the various mathematical modules. While other more specialized computer algebra systems may sometimes be more efficient, Magma makes it easy to perform computations that involve the close interaction of tools from many different areas of mathematics.

This close interaction is the key reason for many of the more recent advances:

- The availability of high performance machinery for group theory, local fields, invariant theory and number fields made it possible to compute Galois groups of polynomials of degree greater than 30 for the first time.
- Access to class field theory of function fields is a key requirement in the construction of geometric codes.
- The wide range of techniques for computing Mordell-Weil groups of elliptic curves and hyperelliptic Jacobians rely on computations in number field, local fields, geometry, and lattice reduction.
- Quaternion algebras over global fields are built on module theory developed for relative number fields; together with lattice techniques, this enables efficient solution of key arithmetic problems (concerning ideals, units etc).
- Hilbert modular forms are computed using the machinery for quaternion algebras.

Although Magma is not a specialist system designed for a single area such as number theory, the performance of Magma is highly competitive with the more specialist systems. While special stand-alone software is sometimes faster, Magma typically performs within a small constant factor of the fastest known implementation.

The uniform interface to each type of arithmetic field (finite fields, local fields, number fields, function fields, extensions) allows users to develop applications that will work without change for a wide variety of rings and fields.

K. Fukuda et al. (Eds.): ICMS 2010, LNCS 6327, pp. 253–255, 2010.

In the context of number theory, Magma provides tools for determining all classical invariants of number fields such as the ring of integers, the unit group, the ideal class group and the Galois group of arbitrary finite extensions of the rational number field. Similar functionality is available for univariate function fields and hence for plane curves. Most of the classic algorithms were originally developed by the KANT group of Professor Pohst in Berlin [3,4]. This functionality for arithmetic fields is the foundation for the construction in Magma of extensive machinery for many other important areas of number theory. These include:

- *Class field theory:* Magma can compute defining equations for Abelian extensions of global fields (and local fields of characteristic 0) parametrised by suitable ray class groups.
- *Hecke characters*, as the dual groups of ray class groups, and their L-functions
- *Diophantine equations* such as norm, unit, Thue and index-form equations.

Magma's extensive packages for topics in "arithmetic geometry" rely in many cases on number field/function field machinery, but also on other core features such as efficient linear algebra, fast Groebner bases, and lattice reduction/enumeration. These topic include:-

- Solving (or deciding solubility of) *conics* over global fields; *general quadratic forms* (genus theory, isotropic subspaces).
- Elliptic curves over global fields: an unrivalled array of techniques for computing Mordell-Weil groups.
- Hyperelliptic curves over number fields: the Mordell-Weil group of the Jacobian may be found via 2-descent, height machinery and search tools. Points on the curve can be determined using Chabauty and Mordell-Weil-sieve techniques, or by "2-descent on the curve".
- *Low genus curves over finite fields:* Tools for cryptographic applications such as point counting and various attacks such as discrete logarithm and GHS attacks.
- *Modular forms* for the standard congruence subgroups and characters.
- *Hilbert modular forms* for general totally real fields, with general levels and weights.
- *L-series* attached to many arithmetic objects.

Just as the availablity of tools from many different areas in Magma has led to new computational tools and applications in number theory, methods from number theory have played a key role in the development of computational methods in other branches of algebra. For example recent progress in the construction of ordinary irreducible representations of large finite groups was only possible because of progress in effective Galois cohomology of number fields (in particular access to relative Brauer groups), which in turn is based on general cohomology for groups.

References

1. Bosma, W., Cannon, J. (eds.): Discovering Mathematics with Magma. Algorithms and Computations in Mathematics, vol. 19. Springer, Heidelberg (2006)
2. Bosma, W., Cannon, J., Playoust, C.: The Magma algebra system. I. The user language. J. Symbolic Comp. 24(3-4), 235–265 (1997)
3. Daberkow, M., Fieker, C., Klüners, J., Pohst, M., Roegner, K., Wildanger, K.: KANT V4. J. Symbolic Comp. 24, 267–283 (1997)
4. Kant/KaSH: KaSH - the KANT Shell, http://www.math.tu-berlin.de/~kant
5. Magma: The Magma computational algebra system for algebra, number theory and geometry, http://magma.maths.usyd.edu.au/magma/

Enumerating Galois Representations in Sage

Craig Citro[1,2] and Alexandru Ghitza[3]

[1] Google, Seattle WA
[2] Department of Mathematics, University of Washington,
Box 354350, Seattle WA 98195-4350, USA
craigcitro@gmail.com
[3] Department of Mathematics and Statistics, University of Melbourne,
Parkville VIC 3010, Australia
aghitza@alum.mit.edu

Abstract. We present an algorithm for enumerating all odd semisimple two-dimensional mod p Galois representations unramified outside p. We also discuss the implementation of this algorithm in Sage and give a summary of the results we obtained[1].

Keywords: Galois representations, Sage, modular forms.

1 Introduction

A great deal of arithmetic questions have found natural interpretations (and often, answers) within the realm of Galois representations and modular forms: such applications include Diophantine equations, quadratic forms, or the study of combinatorial-arithmetic objects such as partitions. In this context, it is of interest to dispose of computational tools for working with modular forms and Galois representations.

In this note, we focus on two-dimensional *Galois representations mod p*, i.e. continuous group homomorphisms

$$\rho\colon \mathrm{Gal}(\overline{\mathbb{Q}}/\mathbb{Q}) \longrightarrow \mathrm{GL}_2(\overline{\mathbb{F}}_p) \ .$$

(More precisely, we consider such representations which are semisimple, unramified outside p, and odd. For the theoretical background, we refer the reader to Khare's survey [3] or to Edixhoven's paper [2].)

By Serre's conjecture, now a theorem of Khare-Wintenberger (see [4], [5]), these representations are closely related to modular forms (mod p) of level 1 which are eigenvectors for all the Hecke operators. If f is such a form, of weight k and eigenvalues (a_ℓ), then for all primes $\ell \neq p$ we have

$$\mathrm{charpoly}(\rho(\mathrm{Frob}_\ell)) = X^2 - a_\ell X + \ell^{k-1} \ ,$$

where Frob_ℓ is a Frobenius element at ℓ inside $\mathrm{Gal}(\overline{\mathbb{Q}}/\mathbb{Q})$.

[1] The authors wish to thank Kevin Buzzard for providing several corrections and a significant improvement to Theorem 1, and the referees for suggesting improvements to the exposition.

K. Fukuda et al. (Eds.): ICMS 2010, LNCS 6327, pp. 256–259, 2010.

The *(Hecke) eigensystem* corresponding to a mod p eigenform f is the sequence (a_ℓ) of eigenvalues indexed by all primes $\ell \neq p$. The i-th *twist* of (a_ℓ) is by definition the eigensystem $(\ell^i a_\ell)$. We write $[a_\ell]$ for a finite truncation of (a_ℓ), where the cutoff point will be clear from the context.

Inspired by a remark of Khare[2], we have set out to enumerate all odd semisimple mod p representations which are unramified outside p. This corresponds to enumerating all the Hecke eigensystems which occur in spaces of level 1 modular forms mod p.

2 Description of the Algorithm

The starting point is a classical result in the theory of modular forms mod p (see Theorem 3.4 in [2]): every Hecke eigensystem occurs, up to twist, in weights less than or equal to $p+1$. Therefore it suffices to generate the spaces M_k for weights $4 \leq k \leq p+1$ and find all the eigenforms in them, which will produce all the Hecke eigensystems up to twist. This list may however contain duplicates; we investigate this question in detail in [1], where we prove

Theorem 1

(a) Let f_1 and f_2 be eigenforms of weights $k_1, k_2 \leq p+1$. If f_1 and f_2 have the same eigensystem up to twist, then $k_1 + k_2 = p+1$ or $k_1 + k_2 = p+3$.
(b) Let f_1 and f_2 be eigenforms of weights related in one of the ways described in (a). If f_1 and f_2 do not have the same eigensystem up to twist, then this is detected by a prime $\ell \neq p$ satisfying $\ell \leq (p+1)/6$.

In the process of proving Theorem 1, we obtained the following lower bound, which improves the best known lower bound (due to Serre, see Sect. 8 in [3]) by a factor of two:

Theorem 2. *Let $p > 19$ be prime. The number of odd semisimple 2-dimensional Galois representations mod p which are unramified outside p is bounded below by $p(p-1)/2$.*

Algorithm: Enumerate Galois Representations Mod p Up to Twist

1. For $4 \leq k \leq p+1$:
   (a) Compute a basis for the space M_k.
   (b) Decompose the space into Hecke eigenspaces.
   (c) For each eigenform, compute the eigenvalue a_ℓ of T_ℓ for primes ℓ up to the bound from Theorem 1. Store $(k, [a_\ell])$.
2. Remove duplicates: given $(k_1, [a_\ell])$ and $(k_2, [b_\ell])$ such that $k_1 + k_2 = p+1$ or $p+3$, check whether $[b_\ell]$ is a twist of $[a_\ell]$.

This creates the list of equivalence classes (up to twist) of Hecke eigensystems mod p. It is now straightforward to apply the twist operation to each list element and generate the list of all Hecke eigensystems.

[2] From Sect. 8 of [3]: "[...] there are only finitely many semisimple 2-dimensional mod p representations of $\mathrm{Gal}(\overline{\mathbb{Q}}/\mathbb{Q})$ of bounded (prime-to-p Artin) conductor. *It will be of interest to get quantitative refinements of this.*"

3 Sage Implementation and Results

Our task requires computing the action of Hecke operators on spaces of modular forms of high weight. Sage [8] offers several implementations of these spaces for arbitrary levels. We have initially used modular symbols over finite fields for generating the lists of eigenforms, but this method becomes quite slow as the weight increases. Restricting to level 1 allows us to take advantage of a much faster way of working with these spaces: the Victor Miller basis (see Sect. 2.3 in [7] for the properties and the algorithm Sage uses to compute this basis).

We then use one Hecke operator T_ℓ at a time to decompose the space M_k into eigenspaces. This requires (at most) the first $k/12$ primes ℓ (see the appendix of [6]).

We have run the Sage implementation of our algorithm for all primes up to 211 (see Table 1). Apart from keeping track of the number of equivalence classes of eigensystems and the total number of eigensystems, we save the list of equivalence classes; given this it is very easy to take twists and generate the entire list.

Table 1. Number of Galois representations mod p

p	number	p	number	p	number	p	number	p	number	p	number
2	1	23	264	59	4234	97	19200	137	53992	179	119705
3	1	29	532	61	4800	101	21600	139	55752	181	124020
5	4	31	630	67	6237	103	22797	149	69264	191	145445
7	9	37	1044	71	7420	107	25546	151	71700	193	150144
11	35	41	1480	73	8136	109	27216	157	80340	197	160132
13	48	43	1701	79	10257	113	30240	163	90397	199	164637
17	112	47	2185	83	12054	127	42903	167	97276	211	196560
19	153	53	3172	89	14784	131	46735	173	108016		

Khare guesses in [3] that the number of Galois representations of the type we are considering should be asymptotic to $p^3/48$. There are two phenomena that can contribute to the actual number being smaller than the guess: (i) the existence of "companion forms", which in our context appear as duplicate equivalence classes of eigenforms; (ii) the failure of "multiplicity one" for Hecke eigenvalues mod p, which results in some spaces M_k not contributing their dimension's worth of eigenforms. In the range of our computations, the actual number of representations stays very close to the best known upper bound[3], suggesting that the two phenomena are indeed quite rare. We expect this trend to be confirmed by further computations.

References

1. Citro, C., Ghitza, A.: Computing level 1 Hecke eigensystems (mod p) (preprint)
2. Edixhoven, B.: The weight in Serre's conjectures on modular forms. Invent. Math. 109(3), 563–594 (1992)

[3] For instance, for $p = 211$ the quotient between the actual number (196560) and the upper bound (196665) is about 0.9995.

3. Khare, C.: Modularity of Galois representations and motives with good reduction properties. J. Ramanujan Math. Soc. 22(1), 75–100 (2007)
4. Khare, C., Wintenberger, J.P.: Serre's modularity conjecture. I. Invent. Math. 178(3), 485–504 (2009), http://dx.doi.org/10.1007/s00222-009-0205-7
5. Khare, C., Wintenberger, J.P.: Serre's modularity conjecture. II. Invent. Math. 178(3), 505–586 (2009), http://dx.doi.org/10.1007/s00222-009-0206-6
6. Lario, J.C., Schoof, R.: Some computations with Hecke rings and deformation rings. Experiment. Math. 11(2), 303–311 (2002), http://projecteuclid.org/getRecord?id=euclid.em/1062621223; with an appendix by Amod Agashe and William Stein
7. Stein, W.: Modular forms, a computational approach. In: Graduate Studies in Mathematics, vol. 79. American Mathematical Society, Providence (2007); With an appendix by Paul E. Gunnells
8. Stein, W., et al.: Sage Mathematics Software (Version 4.4.1). The Sage Development Team (2010), http://www.sagemath.org

NZMATH 1.0

Satoru Tanaka, Naoki Ogura, Ken Nakamula,
Tetsushi Matsui, and Shigenori Uchiyama

Department of Mathematics and Information Sciences
Tokyo Metropolitan University
1-1 Minami Osawa, Hachioji, Tokyo, 192-0397 Japan
{satoru,naoki,nakamula,tetsushi,uchiyama}@tnt.math.metro-u.ac.jp

Abstract. This is an announcement of the first official release (version 1.0) of the system NZMATH for number theory by Python [18]. We review all functions in NZMATH 1.0, show its main properties added after the report [11] about NZMATH 0.5.0, and describe new features for stable development. The most important point of the release is that we can now treat number fields. The second major change is that new types of polynomial programs are provided. Elliptic curve primality proving and its related programs are also available, where we partly use a library outside NZMATH as an advantage of writing the system only by Python. A new feature is that NZMATH is registered on SourceForge [19] as an open source project in order to ensure continuous development of the project. This is a unique among existing systems for number theory.

1 Introduction

The purpose of this article is to announce and describe the first official release (version 1.0) of NZMATH, which is a system for number theory written completely as a library package of the Python language.

In ICMS 2006, we reported the visions of our development together with the status and plan at that time. The key visions of the NZMATH development are [11]:

- user / developer fusion.
- speed of development.

The first vision means that ideally there is no distinction between users and developers. From the user's view point, users should be able to become developers easily. Their programs should be merged into the system without difficulties. From the developer's view point, developers should concentrate on implementing mathematical concepts, especially algorithms for number theory, on the system. The second vision means that we put emphasis on the speed of programming, rather than that of execution. It is a paraphrasing of a commonly accepted principle "premature optimization is the root of all evil" [8, p.268]. In order to realize such visions, a scripting language is suitable. We chose the Python language as the implementation language. Here, we explicitly add the third key vision:

- open source and outsourcing.

K. Fukuda et al. (Eds.): ICMS 2010, LNCS 6327, pp. 260–269, 2010.

Although this vision has been implicitly included from the beginning, we find it necessary to state it clearly in order to continue stable development and maintenance of the system. We are distributing NZMATH with the BSD license [11, §3.3]. This time, we decided to use SourceForge [19] in order to be recognized by a wider audience. In §3.2, there will be a more detailed discussion about the third vision.

We would like to mention SAGE [20], which is a project by Python for mathematics and has gained great success, is the project gathering existing systems into one system. It uses Python as a glue language, and the concept is completely different to that of NZMATH.

We will summarize the features of NZMATH 1.0 in §2 mainly changes from NZMATH 0.5.0. Then, we will discuss our principle of NZMATH in §3, especially open source and outsourcing. Finally in §4, we will present future work.

2 Features

We have been developing NZMATH for six years. Various modules are included in the NZMATH package. Note that "module" means a file consisting of parts of program codes and "package" means a directory including modules. In this section, we explain features of NZMATH in each module or package.

2.1 Overview

In this subsection, we provide a brief summary of entire modules or packages.

Basic Utilities. NZMATH provides some modules for supplementing Python features. For example, we can determine whether NZMATH may assume correctness of the generalized Riemann hypothesis or not.

bigrandom. The module provides a random number generator for big numbers. The module was written for handling the fact that some functions of the Python standard module **random** cannot deal with long integers. For example, if you call the function **randrange** with long integers given as arguments, it raises OverflowError. Note that this bug is modified in Python 2.5.1 or higher.

bigrange. The module provides range-like generator functions. For example, the Python standard function **range** cannot deal with many components, while we can use the function **range** of bigrange for many components since the type of its outputs is generator instead of list.

compatibility. The module is for keeping compatibility between Python versions.

config. The module is for NZMATH configurations.

Functions. NZMATH provides various functions for mathematical computations.

arith1. The module provides various useful functions about integers. For example, it includes a function which computes the integer part of square root (**floorsqrt**), m-adic expansion (**expand**), etc.

arygcd. The module provides functions which compute the greatest common divisor over Gaussian integers or Eisenstein integers.

combinatorial. The module provides combinatorial theoretic functions. For example, we can compute a factorial, a binomial coefficient, a Bernoulli number, a Bell number, a Stirling number, a partition number, etc. with it.

cubic_root. The module provides a function of computing cubic roots over a finite prime field.

ecpp. The module provides the elliptic curve primality proving function. Also we can compute the numerical values of the Dedekind η function or the Hilbert class polynomials. These values and polynomial are used to compute the value of j-function of curves.

equation. The module provides functions of solving polynomial equations. It includes algebraic methods of solving linear, quadratic and cubic equations over the real number field or finite prime fields. Also, we have a function for computing approximate values of a root of polynomials over the real number field by Newton's method.

gcd. The module provides functions for the greatest common divisors, least common multiples, etc.

multiplicative. The module provides multiplicative arithmetic functions such as the Euler totient function, the Möbius function, etc.

prime. The module provides various functions on primes such as prime generating functions by using the Eratosthenes sieve. Especially, we have various functions for primality testing: the strong pseudo–prime test, Miller–Rabin pseudo–primality test, Lucas test, Frobenius test, Adleman–Pomerance–Rumery primality test.

quad. The module provides functions for imaginary quadratic fields, especially, functions for computing ideal class numbers.

squarefree. The module provides a function for square-free detection over the rational integers.

factor (package). The package provides functions for factorization of integers. It provides the elliptic curve method (ecm), multi–polynomial quadratic sieve (mpqs), $p-1$ method and Pollard's ρ method.

Classes. NZMATH provides various classes for handling mathematical objects.

algfield. The module provides classes for algebraic number fields.

elliptic. The module provides classes for elliptic curves.

finitefield. The module provides classes for finite fields.

group. The module provides classes for groups. We can define a new group from (an instance) of specific classes.

imaginary. The module provides classes for the complex number fields.

intresidue. The module provides classes for integer residue classes.

lattice. The module provides classes for lattices in the real vector space of any associated inner product. The function **LLL** of the module computes LLL-reduced basis by the LLL algorithm.

matrix. The module provides classes for linear algebraic objects, especially, matrices.

permute. The module provides classes for symmetric groups.

rational. The module provides classes for the rational integer ring and the rational number field.

real. The module provides classes for the real number field.

ring. The module provides classes for algebraic structures such as rings, fields, quotient fields, etc.

vector. The module provides classes for vectors.

poly (package). The module provides classes for univariate/multivariate polynomials.

2.2 Recent Changes

In this subsection, we describe the features different from those of NZMATH 0.5.0 [11].

algfield. The most remarkable new feature in NZMATH 1.0 is the algfield module. The module algfield includes algebraic number field and algebraic number as data type. An algebraic number is one of the most important objects in number theory. Although the module quad already provided some computation methods about imaginary quadratic fields, the module algfield now brings some operations in general algebraic number fields. This enables us to solve some problems in algebraic number theory with NZMATH. For a given algebraic number field, we can compute its discriminant, signature, and so on. Also we can obtain an integral basis by the module round2. The method for decomposing prime ideals will be released pretty soon.

poly. We provide a new package poly for polynomial data type instead of the former module polynomial. That is partly due to the inconvenience of defining symbols of indeterminates for a polynomial. Although it makes display of polynomials visually satisfactory, it makes programs on polynomial objects complex. The calculation of indeterminates wastes time if we need only the coefficients of the polynomials. The second reason we create the module poly is insufficient support for coefficient rings. Some methods, for example, GCD (Greatest Common Divisor), do not check types of coefficient rings (Euclidean domain, unique factorization domain, field, etc.). This causes errors for some methods depending on the former module polynomial. Then, we implement a new package poly. Modules in the package poly can be divided into two submodule groups. One is a group of submodules for basic definitions for polynomial, the other is for user's utilities of creating polynomials or defining convenient methods. Also, we give methods on polynomials depending on coefficient rings. For example, in the class **UniqueFactorizationDomainPolynomial** GCD is calculated by using successive pseudodivision. The method for computing Gröbner basis will be released soon.

ecpp. The module ecpp is for elliptic curve primality proving (ECPP). The method in ecpp uses an algorithm proposed by Atkin and Morain [3]. Also the module enables us to deal with some tasks associated to ECPP, for example, computation of the numerical values of the Dedekind η function, the Hilbert class polynomial, etc. The module ecpp is different from other modules in two ways. Firstly, some methods do not call other modules directly but rework these into simple forms. For example, in the module elliptic there are some optional calculations such as j-invariant. In ECPP, the computatations which use an elliptic curve explicitly are scalar multiplications. However, some parameters such as j-variant are not used for the computation of scalar multiplications. In the module ecpp, these inessential codes are abbreviated for efficiency. Note that extracting codes from other modules may promote speeding up programming. Secondary, the module ecpp uses a third party software mpmath. The software mpmath is a Python module for floating-point operation [13]. Although NZMATH includes modules for real numbers and complex numbers, that is, real and imaginary, respectively, internal calculations are implemented with approximate rational numbers. Then, the computation speed is too slow compared with normal floating operation. The software mpmath provides high performance computation with floating-point numbers, so we decided to use mpmath for improvement of efficiency. The module ecpp is the first module which uses modules outside NZMATH.

matrix. The module matrix had a similar problem to the module polynomial. That is, handling of coefficient rings is not enough. So we divide classes into new four classes depending on forms of matrix and coefficient rings. For example, **RingSquareMatrix** is for square matrix whose coefficient ring is a (general) ring while **FieldSquareMatrix** is for square matrix whose coefficient ring is a field. Various commutative rings such as the rational integer ring or the complex number field can be handled as coefficient rings in the module matrix. Then,

we can change algorithms due to coefficient rings. For instance, the method determinant uses the Gauss-Bareiss method [4] in **RingSquareMatrix** while the elementary Gaussian elimination in **FieldSquareMatrix**.

Miscellaneous Changes. NZMATH 1.0 has many changes. We explain those which are not mentioned above.

The module factor.ecm is for factoring integers with the elliptic curve method. Some families of curves are available for using factor.ecm. Many methods for calculations of combinatorial theoretical objects including Bell numbers, Stirling numbers and Euler numbers are added in the module combinatorial. Especially, we are capable of computing various partitions of natural numbers. The method **LLL** in the module lattice is rewritten as a function instead of a method. The LLL algorithm has various applications such as factoring rational-polynomials, cryptanalysis of RSA cryptosystem under some conditions or knapsack cryptosystems, and so forth. To obtain more information about recent changes, please check Release Notes corresponding to files in each release of NZMATH at `http://sourceforge.net/projects/nzmath/files/`.

2.3 Auxiliary Components

There are various files distributed in NZMATH packages. These include documents or installers to help users to introduce NZMATH easily.

Installers. Recently we have provided an installer for Windows users. Although NZMATH can be installed in each machine by typing a few commands at a command line interpreter like command prompt (cmd.exe), it is hard to call NZMATH. Thus, we offer a tool for approaching NZMATH with only three clicks. This installer has graphical user interfaces similar to many other Windows programs. We expect this installer will reduce users' time and effort.

Documents. In NZMATH 1.0, we prepare documents for each modules as a PDF file. Previously, NZMATH documents have been maintained on wiki as described in [11], and HTML files generated from it have been bundled in the distribution. However, since NZMATH is a mathematical program, its explanation needs many mathematical notations, which are hard to express with HTML or wiki notation. Thus, we switch from wiki to LaTeX, which is more suitable for mathematical documents. It is useful for representing mathematical objects, and enables developers to describe modules with a unified format. Though it is still possible to produce HTML files from LaTeX source files, we prefer to create a PDF file from them for readability and portability. During this transitional work, we also enhance the content of the document, augmenting it with many examples for each modules. These documents are expected to help users to understand NZMATH modules.

3 Principle of NZMATH

3.1 Good Use of NZMATH

We can recommend NZMATH to anyone who does not have much programming experience, especially to those who are newly beginning to implement number

theoretic algorithms. We explain the reason here. NZMATH is a pure Python library for number theory. Python is one of the scripting languages, so it is easy to read and write code. By using Python, both implementation speed of the system itself and programming speed of developers can be faster. In addition, by using the same language for implementing the system and writing programs, it becomes easier to take users' programs into the system, so it is helpful for implementation speed again. We know that execution speed of programs written alone in Python is slower than that in C or Java. On the other hand, we can master Python much faster than C or Java. We also notice that other number theoretic calculation systems which can compute faster than NZMATH require two or more languages to develop them. Therefore, we should learn at least two languages to develop them. But, Python suffices for NZMATH. In this sense, we recommend NZMATH as the best possible choice to those who are going to implement number theoretic algorithms for the first time.

We provide NZMATH with no user interface. In other words, users have to choose a good interface for NZMATH by themselves. We may not be satisfied with the editing environment of a classical Python console interface. Then, if we use IPython [6] instead, we can make a substantial improvement of our environment. We can also use an integrated develop environment for Python. For instance, with Pydev [2] — a plug-in of Eclipse IDE [5] — we are able to work a sequence of development, editing, execution testing and debugging in an integrated environment.

3.2 Open Source and Outsourcing

We describe here our third vision.

As is explained in [11, §3.3], we are distributing NZMATH with the BSD license which permits free use, free redistribution and free modification. We prefer that restriction of license is not so strong as to interfere with advanced research in number theory. Therefore we can freely modify NZMATH source code for our own algorithms. For example, we added to NZMATH 1.0 as in §2.2 the module ecpp with redesigned functions and classes of elliptic curves from the module elliptic. To improve efficiency of computation, we recreated a class of elliptic curves omitting some features of the original module elliptic. The BSD license permits such modifications for all users.

There is some background concerning why we put emphasis on outsourcing. Users of systems for number theory, like those for other specialized topics in mathematics, are essentially not so many as those of general purpose software. Consequently, developers of the systems are still few and valuable. We are aware that some good systems, which have been maintained by servers of development groups, such as SIMATH [21] or LiDIA [9] have stopped developing. Other major systems PARI [17], KANT [7] and MAGMA [10] are also maintained by servers of the development groups. As is described in [11, §4.4], employing a site hosting free / open source software with version control software, mailing list and web pages, so we can reduce our daily routines as a side effect. Hence, we decided to choose outsourcing in order to continue stable development and maintenance of

the system. It is a unique attempt among existing systems for number theory. There may be a different view point that software for specialized topics in mathematics, like that for number theory, should be kept under control of researchers. We are not against this, but we are going to proceed by another approach.

As we planned, we registered NZMATH on SourceForge [19], one of the famous repositories for open source software, in order to be recognized by wider audience. We have experienced several merits from the registration of our project. Firstly, in the process of registration, we can automatically decide the role of each developer on the web by a project management system. As a result, we are now ready to welcome new members as developers of the project. Secondly, the project is verified by many people since the records of our activities are also available from the bug tracking system and the forums on the project page. This is helpful to maintain the quality of our development. Finally, there is of course a great deal of improvement of information sharing. We can provide an integrated service from SourceForge. For example, the statistics of the projects, like access counts of the project web, commit intervals and download counts are available now.

We changed the version control system of NZMATH from CVS to Mercurial [12] on SourceForge. Mercurial is one of the distributed version control systems, so each working copy of Mercurial effectively functions as a remote backup of the repository, namely all the source codes and the history of changes. In order to minimize the risk of losing codebase, we decided to use Mercurial. In addition, Mercurial enables developers to exchange their works directly with each other without connecting to the main repository in SourceForge.

4 Future Works

The development of NZMATH will continue further. We will discuss some notable term perspectives in the subsections.

4.1 Cloud Computing and Databases

There has been no interface for NZMATH on the web although we are able to choose many candidates for the desktop environment. Previously we introduced our plan to make a web user interface [11, §4.2]. There are no systems to take advantage of cloud computing yet. We discuss how to use the cloud computing for mathematical software.

In fact, we tried to implement NZMATH/JSON [16], NZMATH on Google App Engine, but only a few modules are available. Especially, by the restriction of resources of this service, it is too hard to compute a huge data or a huge numbers of small data.

On the other hand, if we need data which spends a large amount of resources to compute, it is effective to call a database on the cloud to pick up precomputed data. In this release of 1.0, we designed a system of providing data of Hilbert class polynomials on Google App Engine [14]. The ECPP algorithm requires the computation of the Hilbert class polynomials. However, it takes much time to

compute a Hilbert class polynomial for the ECPP algorithm. So, we provided algorithms for ecpp modules with the precomputed Hilbert class polynomials. If we have the Internet connection, ecpp module can use the database and execute the ECPP algorithm faster.

So, we continue to implement new algorithms as before, and we will also provide an algorithm using precomputed data that is computable after a great deal of time.

4.2 Long Term Plans

Let us now discuss our long term plans.

As the main stream of development, we continue to improve modules of algebraic numbers fields, polynomials, finite fields and elliptic curves. Since there are few methods on elliptic curves over the rational number field, it is one of the most important tasks to implement the algorithms for them.

We should improve the methods for some topics which stopped further refinement, for example, integer factoring by the multi polynomial quadratic sieve or by the elliptic curve method. To optimize these algorithms, we should reconsider their parameters.

It is also important to implement the AKS algorithm [1]. Even though there are already many fast primality tests, there is no reason to hesitate to implement this unconditional deterministic polynomial time algorithm on NZMATH. We expect that the implementation of the AKS algorithm will advance research in primality test and proving.

5 Conclusion

We announced the first official release version 1.0 of NZMATH.

In this version, we provided algebraic number fields as one of the most important new features. We also released the module for elliptic curve primality proving with a new concept for development. Since we follow up mathematical notation on manuals, we started to provide new manuals with LaTeX.

We added our third vision, open source and outsourcing, explicitly. In order to continue stable development and maintenance of the system, we registered NZMATH on SourceForge and have experienced several merits. As a result, we are now ready to welcome new members as developers of the project.

We believe that this user-developer-friendly system will be accepted by wider range of people, from number theorists to students of other variety of areas.

References

1. Agrawal, M., Kayal, N., Saxena, N.: PRIMES is in P. Annals of Mathematics 160(2), 781–793 (2004)
2. Aptana Inc.: Pydev; http://pydev.org/
3. Atkin, A., Morain, F.: Elliptic curves and primality proving. Mathematics of Computation 61, 29–68 (1993)

4. Bareiss, E.: Sylvester's identity and multistep integer-preserving Gaussian elimination. Mathematics of Computation 22, 565–578 (1968)
5. Eclipse Foundation: Eclipse IDE, http://www.eclipse.org/
6. IPython, http://ipython.scipy.org/
7. The KANT Project: KANT / KASH, http://www.math.tu-berlin.de/%7Ekant/kash.html
8. Knuth, D.: Structured Programming with go to Statements. ACM Journal Computing Surveys 6(4), 261–301 (1974)
9. LiDIA group: News, http://www.cdc.informatik.tu-darmstadt.de/TI/LiDIA/#news
10. MAGMA group: MAGMA, http://magma.maths.usyd.edu.au/magma/
11. Matsui, T.: Development of NZMATH. In: Iglesias, A., Takayama, N. (eds.) ICMS 2006. LNCS, vol. 4151, pp. 158–169. Springer, Heidelberg (2006)
12. Matt Mackall: Mercurial, http://mercurial.selenic.com/
13. mpmath, http://code.google.com/p/mpmath/
14. NZMATH development group: Hilbert Class Polynomial, http://hilbert-class-polynomial.appspot.com/
15. NZMATH development group: NZMATH, http://tnt.math.metro-u.ac.jp/nzmath/
16. NZMATH development group: NZMATH / JSON, http://nzmath-json.appspot.com/
17. The PARI group: PARI/GP Development Headquarter, http://pari.math.u-bordeaux.fr/
18. van Rossum, G.: Foreword. In: Programming Python, 1st edn. O'Reilly, Sebastopol (May 1996)
19. SourceForge.net, http://sourceforge.net/
20. Stein, W.: Software for Algebra and Geometry Experimentation, http://modular.fas.harvard.edu/SAGE/
21. The SIMATH center: SIMATH, http://tnt.math.metro-u.ac.jp/simath/

Removing Redundant Quadratic Constraints

David Adjiashvili, Michel Baes, and Philipp Rostalski

ETH Zürich, Raemistrasse 101, 8092 Zurich, Switzerland
{david.adjiashvili,michel.baes}@ifor.math.ethz.ch,
philipp.rostalski@control.ee.ethz.ch

Abstract. Determining whether an ellipsoid contains the intersection of many concentric ellipsoids is an NP-hard problem. In this paper, we study various convex relaxations of this problem, namely two semidefinite relaxations and a second-order cone relaxation. We establish some links between these relaxations and perform extensive numerical testings to verify their exactness, their computational load and their stability. As an application of this problem, we study an issue emerging from an aircraft wing design problem: how can we simplify the description of a feasible loads region?

Keywords: Semidefinite Optimization, Semidefinite Relaxation, Aircraft Design.

1 Introduction and Motivating Example

The structural design of large mechanical architectures leads to huge optimization problems, with a massive amount of constraints. It is usually hopeless to find an exact solution to these problems, and an impressive fauna of heuristic strategies has been developed over the years to deal directly with them with more or less success. In this paper, we take an indirect approach: we try to identify some constraints that are provably redundant. These constraints are subsequently removed, yielding a simpler optimization problem. If every constraint is linear, that is, if the feasible set is a convex polyhedron, this task can be performed efficiently using the software `cdd+` written by Komei Fukuda and David Avis [6]. However, if the constraints of the problem are convex and quadratic, the machinery of Avis and Fukuda cannot be applied. The purpose of this paper is to develop the necessary tools to deal with this class of nonlinear constraints.

Convex quadratic constraints arise naturally in a number of applications, from optimal portfolio selection (see e.g. [11]) to robust linear programming (see e.g. [2]) and, among others, design of mechanical structures [1]. More applications can be found in [9,3,4] and in references therein.

A motivating application for our study originates from the field of aeronautic design, kindly given to us by S. McGuinness and Prof. C. Armstrong from the Queen's University of Belfast.

Typically, the procedure for designing large parts of an airplane, such as a wing in the example below, is divided in four stages. First, a large set of load

K. Fukuda et al. (Eds.): ICMS 2010, LNCS 6327, pp. 270–281, 2010.

scenarios is formed, corresponding to every kind of extreme solicitations that the wing has to withstand. Then these loads are converted into stresses, momentums and torques on each voxel of an appropriate finite element model of the wing. Thirdly, the constraints that these stresses, momentums, and torques generate are translated into design and size requirements for the wing. And finally, the particular design is studied thoroughly in order to understand where and with which external solicitation a damage can be caused. This step is computationally extremely demanding: we need to describe a feasible region determined by the constraints on each voxel, many of which are nonlinear. This description allows us to identify those constraints that can be tight, and therefore spot all the locations on the wing that are prone to damage given the set of load scenarios.

Of particular interest are the so-called *Von Mises yield criterion* constraints, which indicate a threshold above which the deformation of a material is not elastic anymore, and results in a permanent change of shape, or *plastic deformation*. Mathematically, this criterion can be written for a two-dimensional object at a voxel z as:

$$\sigma_1^2(z) + \sigma_2^2(z) + 3\sigma_{12}(z) - \sigma_1(z)\sigma_2(z) \leq \sigma_{\max}(z),$$

where $\begin{pmatrix} \sigma_1(z) & \sigma_{12}(z) \\ \sigma_{12}(z) & \sigma_2(z) \end{pmatrix}$ is the stress tensor at z and $\sigma_{\max}(z)$ the *Von Mises threshold*, which depends on the material used at z and on its dimensions. In our wing, these stress tensors can be modeled as linear combinations of some components of the external load, so that if x represents a particular external load, the components of the stress vector can be expressed as:

$$(\sigma_1(z), \sigma_2(z), \sigma_{12}(z))^T = A(z)x,$$

The matrix weight $A(z)$ of these combinations is deduced from the load scenarios formed at the first stage of the design process. Therefore, each Von Mises yield criterion constraint takes the form

$$x^T E(z) x \leq \sigma_{\max}, \text{ where } E(z) := A^T(z) \begin{pmatrix} 1 & -0.5 & 0 \\ -0.5 & 1 & 0 \\ 0 & 0 & 3 \end{pmatrix} A(z). \tag{1}$$

Note that the above constraints represent degenerated ellipsoids with matrices of rank 3, all centered in 0. Finally, we can assume without loss of generality that $\sigma_{\max} = 1$.

2 Ellipsoidal Constraint Removal Problem

Let

$$\mathcal{E}_i := \{x \in \mathbb{R}^n : x^T Q_i x \leq 1\},\ 1 \leq i \leq m$$

be a set of possibly degenerate ellipsoids centered in 0, where Q_i is a symmetric $n \times n$ positive semidefinite matrix of rank r_i. In order to circumvent numerical problems as much as possible, a linear change of coordinates $x \mapsto \tilde{x}$ is performed

with $x = B \cdot \tilde{x}$, such that the ellipses in the new coordinates are better conditioned. Many different strategies exist, e.g. minimizing the asphericity of the intersection of all ellipsoids, or select a random ellipsoid and making it spherical. We have used a third approach, which proved to be the most satisfactory, and which corresponds to some join spherification of all the ellipsoids. This approach uses explicitly the fact that the matrices Q_i are given in our test problem in the form $Q_i = A_i^T A_i$, where the matrices A_i might be of low rank. We compute a singular value decomposition

$$\mathcal{A} = U \begin{bmatrix} \Sigma \\ 0 \end{bmatrix} V^T$$

of the matrix

$$\mathcal{A} = \begin{bmatrix} A_1 \\ A_2 \\ \vdots \end{bmatrix}$$

obtained by stacking together all matrices A_i. Note that Σ is a square diagonal matrix and U, V are orthonormal. The matrix B is now chosen as

$$B = V \Sigma^{-1}$$

which yields the new ellipses

$$\tilde{\mathcal{E}}_i := \left\{ \tilde{x} \in \mathbb{R}^n : \tilde{x}^T B^T Q_i B \tilde{x} \leq g \right\}, \ 1 \leq i \leq m.$$

For notational simplicity, we assume that the matrices Q_i are scaled as above, and we write $\mathcal{E}_i$ for $\tilde{\mathcal{E}}_i$.

Let S be the intersection of these ellipsoids, and let $J := \{1, \ldots, m\}$. We can assume without loss of generality that $Q := \sum_{i=1}^m Q_i$ is invertible, or equivalently that S is bounded. If this was not the case, it would suffice to project every ellipsoids on $[\ker(Q)]^\perp$.

We are interested in finding *redundant ellipsoids* $\mathcal{E}_i$, for which $S = \cap_{j \neq i} \mathcal{E}_j$. We define:

$$\begin{aligned} \mu_i^* := \max_x \quad & x^T Q_i x \\ \text{subj. to} \quad & x^T Q_j x \leq 1, \ j \neq i, j \in J. \end{aligned} \tag{2}$$

The number μ_i^* indicates by how much one must inflate the ellipsoid $\mathcal{E}_i$ until it contains the intersection of all the other ellipsoids. Clearly, when the above problem is unfeasible, $\mathcal{E}_i$ can be removed. If $\mu_i^* \leq 1$, every point of $\mathcal{E}_i$ belongs to the intersection of all the other ellipsoids, and the ith inequality is redundant. Note that for $\mu_i^* = 1$, there exists a point in the boundary of S that belongs to $\mathcal{E}_i$, albeit $\mathcal{E}_i$ is not indispensable to describe S. Finally, if $1 < \mu_i^* \leq +\infty$, the ellipsoid $\mathcal{E}_i$ is not redundant.

2.1 Immediate Eliminations/Preservations

A very cheap test allows us to determine ellipsoids that have to be discarded. Let R_i be the smallest rectangle containing ellipsoid $\mathcal{E}_i$ that has its sides parallel to the basis vectors. The side lengths of R_i can be easily computed. Also, the intersection R of all the R_i's contains S, and is itself a rectangle with sides parallel to the basis vectors. We take the positive vertex v of R, and we check whether $v^T|Q_i|v \leq 1$, where $|Q_i|$is the matrix Q_i with each component replaced by its absolute value. If this inequality is satisfied, the ellipsoid $\mathcal{E}_i$ contains every extreme point of R. Therefore $\mathcal{E}_i \supseteq R \supseteq S$. Note that the problem of determining if R is contained in a given ellipsoid is NP-hard, as it corresponds to the maximization of a uniform quadratic form over the unit cube (see [12]).

Conversely, there is a simple test for determining ellipsoids that cannot be eliminated. From the diagonal element $[Q_i]_{kk}$ of Q_i, we can immediately determine the intersection point of $\mathcal{E}_i$ with the kth basis vector as $e_k[Q_i]_{kk}^{-1/2}$. Therefore, the ellipsoid $\mathcal{E}_{j_k}$ having the largest element $[Q_i]_{kk}$ cannot be discarded.

2.2 Semidefinite Relaxation

The problem (2) is NP-hard, as a particular case of it consists in maximizing a homogeneous quadratic form over the unit cube (see [12]). A possible relaxation strategy, known in the literature as *SDP relaxation*, and that we call indifferently in this paper *tailored first order moment relaxation* for a reason that will be explained at the end of this section, consists in replacing $x^T Q_i x = \mathrm{Tr}(Q_i \cdot xx^T)$ by $\mathrm{Tr}(Q_i \cdot X)$, where X is a positive semidefinite matrix of dimensions $n \times n$, and to use the components of X as new variables of our problem instead of x. The notation $\mathrm{Tr}(\cdot)$ refers to the trace of its matrix argument.

Of course, if the above matrix X has rank one, one can easily recompute a corresponding x for which $X = xx^T$ (up to a sign, which does not hurt in view of the symmetry of the problem). Thus the original problem (2) is completely equivalent to:

$$\begin{aligned} \mu_i^* := \max_X \quad & \mathrm{Tr}(Q_i \cdot X) \\ \text{subj. to} \quad & \mathrm{Tr}(Q_j \cdot X) \leq 1,\ j \neq i, j \in J \\ & X \in \mathbb{S}_+^n, \\ & \mathrm{rank}(X) = 1. \end{aligned}$$

Here $\mathbb{S}_+^n$ denotes the set of $n \times n$ positive semidefinite matrices and J the indices set of remaining ellipsoids.

Now, the tailored moment relaxation consists in suppressing the non-convex constraint $\mathrm{rank}(X) = 1$ and solve:

$$\begin{aligned} \mu_i^{\mathrm{SDP}} := \max_X \quad & \mathrm{Tr}(Q_i \cdot X) \\ \text{subj. to} \quad & \mathrm{Tr}(Q_j \cdot X) \leq 1,\ j \neq i, j \in J \\ & X \in \mathbb{S}_+^n. \end{aligned} \tag{3}$$

Obviously, $\mu_i^{\mathrm{SDP}} \geq \mu_i^*$. Moreover, as the feasible set is bounded, Nemirovski, Roos, and Terlaky, [12] show that:

$$2\ln(2|J-1|r)\mu_i^* \geq \mu_i^{\mathrm{SDP}},$$

where $r = \min\{|J-1|, \max_{1\leq i\leq m} r_i\}$ and r_i is the rank of the matrix Q_i.

In our tests, we compute successively μ_1^{SDP} up to a given accuracy of ϵ. If $\mu_1^{\mathrm{SDP}} < 1-\epsilon$, we remove 1 from J and proceed to compute μ_2^* and so on.

A possible reformulation of this approach is given by the so-called *1st moment relaxation* (not tailored), where the variables of the relaxed problem are not only the X_{ij}'s, which represent $x_i x_j$ in the unrelaxed version of the problem, but also the x_j's themselves. Although completely equivalent in the present case, this extended reformulation allows us to consider ellipsoids that are not necessarily centered in 0. We have:

$$\begin{aligned} \mu_i^{\mathrm{SDP}} := \max_{Z} \quad & \mathrm{Tr}\left(\begin{pmatrix} 0 & 0 \\ 0 & Q_i \end{pmatrix} \cdot \begin{pmatrix} 1 & x^T \\ x & X \end{pmatrix}\right) \\ \text{subj. to} \quad & \mathrm{Tr}\left(\begin{pmatrix} 0 & 0 \\ 0 & Q_j \end{pmatrix} \cdot \begin{pmatrix} 1 & x^T \\ x & X \end{pmatrix}\right) \leq 1, \quad j \neq i \\ & \begin{pmatrix} 1 & x^T \\ x & X \end{pmatrix} \in \mathbb{S}_+^{n+1}. \end{aligned}$$

In order to ensure the non-degeneracy of this problem, one can add the constraint:

$$\mathrm{Tr}\left(\begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} 1 & x^T \\ x & X \end{pmatrix}\right) \leq 1.$$

Note that, according to Schur's Lemma, the SDP constraint is equivalent to:

$$X - xx^T \in \mathbb{S}_+^n.$$

2.3 S-Lemma

The value μ_i^* can be computed efficiently and accurately when $|J| = 2$. Let, for every $i \neq j$,

$$\begin{aligned} \nu_{ij}^* := \max_{x} \quad & x^T Q_i x \\ \text{subj. to} \quad & x^T Q_j x \leq 1 \end{aligned} \tag{4}$$

The S-Lemma [14] states that the set:

$$\{x \in \mathbb{R}^n : x^T Q_i x > 1,\ x^T Q_j x \leq 1\}$$

is empty (or equivalently, that the ellipsoid $\mathcal{E}_i$ is contained in $\mathcal{E}_j$) if and only if there exists a nonnegative real number y such that

$$1 - x^T Q_i x + y(x^T Q_j x - 1) \geq 0 \ \ \forall x \in \mathbb{R}^n.$$

The latter condition is easily seen to be equivalent to a semidefinite feasibility problem:

$$\exists\, 1 \geq y \geq 0 \text{ such that } y \cdot Q_j - Q_i \in \mathbb{S}^n_+,$$

and can be casted as the following optimization problem:

$$\begin{array}{ll} \min\limits_{y} & y \\ \text{subj. to} & yQ_j - Q_i \in \mathbb{S}^n_+. \end{array}$$

Observe that, since $Q_i \in \mathbb{S}^n_+$, non-positive y's are not feasible for this problem. The optimal objective value is smaller than 1 if and only if $\mathcal{E}_i$ is not entirely contained in $\mathcal{E}_j$. Alternatively, the above optimization problem can be written in the form:

$$\begin{array}{ll} \min\limits_{y} & y \\ \text{subj. to} & \begin{pmatrix} 1 & 0 \\ 0 & -Q_i \end{pmatrix} - y \begin{pmatrix} 1 & 0 \\ 0 & -Q_j \end{pmatrix} \in \mathbb{S}^{n+1}_+ \\ & y \geq 0. \end{array}$$

If this problem is feasible and if its minimum is nonnegative, then $\mathcal{E}_i \supseteq \mathcal{E}_j$. The S-Lemma can be useful to detect whether two ellipsoids are identical (up to numerical inaccuracies). Therefore we simply apply the S-Lemma in both directions i.e. as above but also with swapped i and j. If a nonnegative scalar can be found for both directions, the ellipses are considered equivalent up to the chosen accuracy in the solver parameters.

In our numerical experiments, we perform the S-Lemma check after every other elimination procedure, because its worst-case complexity is proportional to the square of the remaining ellipsoids, given that each pair of different ellipsoid must be individually tested.

2.4 S-Procedure

The S-Lemma is restrictive as it only considers two constraints at a time. A generalization of the S-Lemma, known as the S-procedure, proceeds as follows. We select a set J of ellipsoid indices and an index $i \in J$. Then we compute:

$$\begin{array}{ll} \min\limits_{y} & \sum\limits_{j \in J_i} y_j \\ \text{subj. to} & \begin{pmatrix} 1 & 0 \\ 0 & -Q_i \end{pmatrix} - \sum\limits_{j \in J_i} y_j \begin{pmatrix} 1 & 0 \\ 0 & -Q_j \end{pmatrix} \in \mathbb{S}^{n+1}_+, \;\; y \in \mathbb{R}^{|J_i|}_+, \end{array} \tag{5}$$

where $J_i := J \backslash \{i\}$.

If the above problem is feasible, then $\mathcal{E}_i$ contains the intersection $\cap_{j \in S} \mathcal{E}_j$ and can be eliminated. As shown in the next proposition, the S-procedure is not necessary in the task of testing redundancy of ellipsoidal constraints, as it does essentially the same test as the tailored first moment relaxation.

Proposition 1. *The problem (5) is dual to (3). If the set* $\cap_{j\in J_i}\mathcal{E}_j$ *has a nonempty interior, the duality gap is null.*

Proof. Following Chapter 5 in [4], we can determine the dual of (5) mechanically:

$$\begin{aligned} \max_{Z} \quad & \operatorname{Tr}\left(\begin{pmatrix} -1 & 0 \\ 0 & Q_i \end{pmatrix} \cdot \begin{pmatrix} \alpha & x^T \\ x & X \end{pmatrix}\right) \\ \text{subj. to} \quad & \operatorname{Tr}\left(\begin{pmatrix} -1 & 0 \\ 0 & Q_j \end{pmatrix} \cdot \begin{pmatrix} \alpha & x^T \\ x & X \end{pmatrix}\right) \leq 1, \quad j \in J \\ & Z := \begin{pmatrix} \alpha & x^T \\ x & X \end{pmatrix} \in \mathbb{S}_+^{n+1}. \end{aligned}$$

After a few trivial simplifications, we get:

$$\begin{aligned} \max_{X} \quad & \operatorname{Tr}\left(Q_i \cdot X\right) \\ \text{subj. to} \quad & \operatorname{Tr}\left(Q_j \cdot X\right) \leq 1, \quad j \in J \\ & X \in \mathbb{S}_+^n, \end{aligned}$$

which is (3). □

Actually, we have chosen in our numerical tests to use the above dual version of the SDP relaxation (3), as it contains less constraints, a desirable feature for an SDP solver (see e.g. Section 6.3 in [13]).

2.5 SOCP-Relaxation

Following [7], an Second Order Cone Programming (or SOCP) relaxation can be easily derived from the SDP relaxation by replacing the constraint $X \in \mathbb{S}_+^n$ by the condition that every 2×2 principal minors of X are positive:

$$[X]_{k,k}[X]_{l,l} - [X]_{k,l}^2 \geq 0, \quad 1 \leq k < l \leq n.$$

This condition can be rewritten as the Second Order Cone constraints:

$$\left\| \begin{matrix} [X]_{i,i} - [X]_{j,j} \\ 2[X]_{i,j} \end{matrix} \right\|_2 \leq [X]_{i,i} + [X]_{j,j}\,. \tag{6}$$

for all $0 \leq i \leq j \leq n$ and the linear constraints $[X]_{i,i} \geq 0$ for all $0 \leq i \leq n$. As these conditions are only necessary for the positive semidefiniteness of X, the solution of the SOCP relaxation:

$$\begin{aligned} \mu_i^{SOCP} := \max_{X} \quad & \operatorname{Tr}(Q_i \cdot X) \\ \text{subj. to} \quad & \operatorname{Tr}(Q_j \cdot X) \leq 1,\ j \neq i \\ & X_{i,i} \geq 0 \text{ and } (6) \end{aligned} \tag{7}$$

yields an even more conservative upper bound on the maximal value for the optimization problem (2), but it is also computationally less expensive than the SDP relaxation (3).

3 Computational Results

For a first set of tests, we generate collections of dense matrices of same size and rank, each of them representing an ellipsoid. In order to render some of the ellipsoids redundant, we have scaled them so that the first ones are more likely to be larger than the others.

For our second set of tests, we have used a data set corresponding to the airplane wing design problem sketched in Section 1. This set has been kindly given to us by S. McGuinness and C. Armstrong from the Queen's University of Belfast. It consists of 200 matrices $A(z_1), \ldots, A(z_{200})$ of dimensions 3×20, from which we form the ellipsoid matrices Q_i as in (1). All these matrices where scaled using a coordinate change based on the the singular value decomposition of $\mathcal{A}$ (see Section 2). This scaling decreases the condition number (defined here as $\lambda_{\max}(A(z_i))/\lambda_{\max-3}(A(z_i))$ due to the fact that $A(z_i)$ is of rank 3) of the matrices A_i on average by one order of magnitude. The average conditioning number for the new matrices is 2.8602e+009, while the average conditioning number for the old matrices was 3.1848e+010.

3.1 Solvers and Setup

Tests were performed with the solver MOSEK (which can only deal with SOCP) and SeDuMi (for SOCP and SDP). Both these software run on MatLab, albeit they are implemented in C. Reasonable results are achieved with both solvers, except for the SOCP method with SeDuMi, which seems far less stable than the software of choice MOSEK. However, the expected benefit of the SOCP relaxation — its fastness — does not seem to overcome its drawbacks — its inaccuracy — especially in high dimension.

3.2 SOCP and SDP Relaxations

In order to decide on the redundancy of constraints, the following two sets of experiments have been performed.

SOCP relaxation. Immediate eliminations, followed by Standard SOCP relaxations (7) with solver MOSEK, using a relative duality gap tolerance of 1e-8. Once an ellipsoid has been removed, it is not taken into account in subsequent computations.

SDP relaxation. Immediate eliminations, followed by Dual SDP relaxations (5) with solver SeDuMi-1.21, using an accuracy of 1e-8, followed by S-Lemma Elimination. During the Dual SDP relaxation, we do not consider ellipsoid that have previously been removed in the description of the feasible set of subsequent problems. In the S-Lemma check, the order by which we consider pair is not insignificant. Informally speaking, it is more profitable to consider first pairs made of a small ellipsoid and a big one, because it might lead to faster eliminations, and therefore less ellipsoid pairs to deal with. "Small ellipsoids" would in our case be those that are detected as indispensable by the trivial check sketched in Subsection 2.1. In the strategy we adopted, we

order the remaining ellipsoids that are not indispensable according to their value μ_i^{SDP}, as a small value for μ_i^{SDP} is likely to indicate a large ellipsoid. Now, each time the S-Lemma detects an inclusion, we immediately check the reverse inclusion, and we eliminate the appropriate ellipsoid. Of course, every S-Lemma test concerning the eliminated ellipsoid is canceled.

For taking into account the absolute accuracy of the solver, we declare constraints to be redundant if the optimal cost is smaller than $1 - \epsilon$.

We report on Table 1 the results for the first set of experiments. The second column of the table indicates the dimension of the ellipsoid, the rank of their matrix, and their number. We have observed that the immediate eliminations procedure could not eliminate a single ellipsoid in all our tests. We can see that the SDP

Table 1. Experiment results on randomly generated instances

Randomly generated instances					
Dim / Rk / # ell.	Method	# removed ellipsoids	CPU time (s)	# numerical warnings	# numerical failures
2/2/50	SOCP	46/50	33.8964	14	0
	S-procedure	46/50	1.5470	0	0
2/2/100	SOCP	96/100	56.3070	66	0
	S-procedure	96/100	4.2660	0	0
2/2/500	SOCP	493/500	332.2311	457	0
	S-procedure	493/500	266.1870	0	0
2/2/1000	SOCP	994/1000	1257.9	945	0
	S-procedure	997/1000	2606.7	0	0
	S-Lemma	0/3	0.125	0	0
5/5/50	SOCP	11/50	30.4870	0	0
	S-procedure	34/50	2.4070	0	0
5/5/100	SOCP	25/100	63.6918	0	0
	S-procedure	74/100	6.5940	0	0
5/5/500	SOCP	391/500	442.2373	455	0
	S-procedure	438/500	386.5620	0	0
	S-Lemma	0/62	89.0620	0	0
10/2/50	SOCP	0/10	31.5869	0	0
	S-procedure	15/50	4.1250	0	0
10/2/100	SOCP	1/100	74.8320	0	0
	S-procedure	42/100	12.3750	0	0
10/2/500	SOCP	98/500	1895.0	206	0
	S-procedure	305/500	813.6410	0	0
	S-Lemma	0/195	1872.9	0	0
10/10/50	SOCP	0/50	31.6607	0	0
	S-procedure	23/50	4.3120	0	0
10/10/100	SOCP	0/100	75.0116	0	0
	S-procedure	57/100	12.5630	0	0
10/10/500	SOCP	63/500	2181.7	62	0
	S-procedure	338/500	689.1560	0	0
	S-Lemma	0/162	1238.0	0	0

Table 2. Experiment results on the airplane wing design problem

Airplane wing design problem				
Method	# removed ell. ellipsoids	CPU time (s)	# numerical warnings	# numerical failures
SOCP	100/200	1033.2	2	0
S-procedure	115/200	154.3280	0	0
S-Lemma	38/85	737.8120	114	0

relaxations perform reasonably fast for problems involving up to 500 ellipsoids of dimension 10, with virtually no numerical problems. Not surprisingly, perfect containment of an ellipsoid into another that has not been previously eliminated by the S-procedure is an extremely rare event, which we have not witnessed in our randomly generated data. Therefore, the S-Lemma does not succeed in eliminating a single ellipsoid. In order to get an idea of its computational cost, we report its computation results or the largest instances only.

In contrast, MOSEK generates a lot of numerical warning, albeit the ellipsoids it eliminates are consistently eliminated by SDP techniques. Obviously, for 2-dimensional problems, the SOCP relaxation is identical to the SDP relaxation in (3), and we should have the same elimination results. However, when the dimension increases, the quality of the SOCP relaxation deteriorates quite fast. Finally, and probably due to the heaviness of the software, MOSEK is considerably slower than the SDP solver for small instances.

The airplane wing design problem (see Table 2) shows an interesting characteristic: in contrast with completely randomly generated data, the S-Lemma

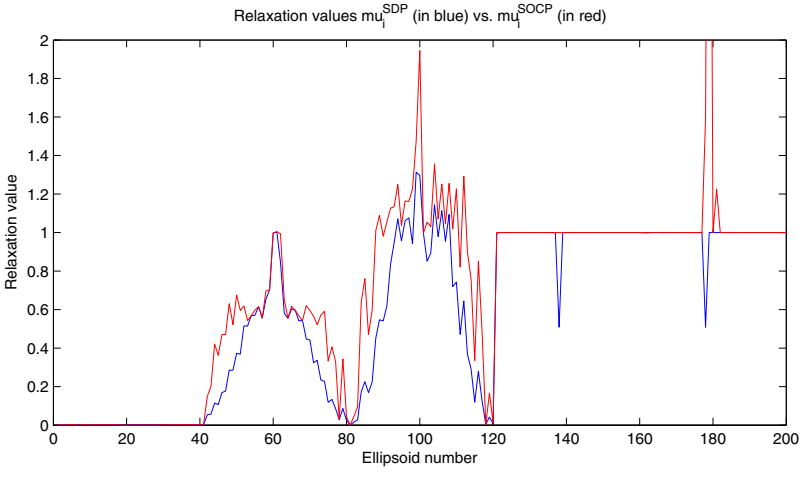


Fig. 1. Comparison of SOCP and SDP relaxation values. In accordance to the theory, the SOCP relaxation value is higher than the SDP relaxation value. In the printed version, the red curve appears slightly lighter than the blue one.

does eliminate some ellipsoids. Interestingly, all the matrices detected as redundant by the S-Lemma were detected as identical to some others. This is a first indication that the SDP relaxation of the original NP-hard problem might not be so bad, because the exact test of S-Lemma is useful only on this somewhat extreme case. As the total number of S-Lemma tests we had to perform equals 3255 (it is less than 85×84 because we do not consider ellipsoids that have already been eliminated by the S-Lemma), the number of numerical warnings shows that 96.5% of the tests happened without any numerical problems.

It is interesting to compare the computed value μ_i^{SDP} and its lower approximation μ_i^{SOCP}. Figure 1 displays the results for the second data set. As we can see, the SOCP relaxation performs not so poorly with 20×20 ellipsoids, in spite of the fact that the approximation of the positive semidefinite cone of 20×20 matrices by quadratic functions is very crude.

Acknowledgments. The authors would like to thank Prof. Komei Fukuda, Shaun McGuinness and Prof. Cecil Armstrong for submitting us the problem we have addressed in the paper, and for many fruitful discussions during the preparation of this paper. We gratefully acknowledge the constructive comments of the referees, which helped us to improve the paper.

References

1. Ben-Tal, A., Nemirovski, A.: Robust Truss Topology Design via Semidefinite Programming. SIAM Journal on Optimization 7, 991–1016 (1997)
2. Ben-Tal, A., Nemirovski, A.: Robust Convex Optimization. Mathematics of Operations Research 23(4), 769–805 (1998)
3. Ben-Tal, A., Nemirovski, A.: Lectures on Modern Convex Optimization: Analysis, Algorithms, and Engineering Applications. MPS/SIAM Series on Optimization. SIAM, Philadelphia (2001)
4. Boyd, S., Vandenberghe, L.: Convex Optimization. Cambridge University Press, Cambridge (2004)
5. Clarkson, K.L.: More Output-Sensitive Geometric Algorithms. In: Proceedings of 35th Annual Symposium on Foundations of Computer Science, pp. 695–702 (1994)
6. Fukuda, K.: cdd, cddplus and cddli homepage, http://www.ifor.math.ethz.ch/~fukuda/cdd_home/index.html
7. Kim, S., Kojima, M.: Exact solutions of some nonconvex quadratic optimization problems via sdp and socp relaxations. In: Research Report B-375, Dept. of Mathematical and Computing Sciences, Tokyo Institute of Technology, Oh-Okayama, Meguro-ku, Tokyo 152-8552, pp. 143–154 (2002)
8. Lasserre, J.B.: Global optimization with polynomials and the problem of moments. SIAM Journal on Optimization 11, 796–817 (2001)
9. Lobo, M., Vandenberghe, L., Boyd, S., Lebret, H.: Applications of Second-Order Cone Programming. Linear Algebra and its Applications 284, 193–228 (1998)

10. Löfberg, J.: Dualize it: software for automatic primal and dual conversions of conic programs. Optimization Methods Software 24(3), 313–325 (2009)
11. Markowitz, H.: Portfolio Selection. Journal of Finance 7(1), 77–91 (1952)
12. Nemirovski, A., Roos, C., Terlaky, T.: On maximization of quadratic form over intersection of ellipsoids with common center. Mathematical Programming 86(3), 463–473 (1999)
13. Nesterov, Y., Nemirovski, A.: Interior Point Polynomial Algorithms in Convex Programming. Studies in Applied Mathematics, vol. 13. SIAM, Philadelphia (1993)
14. Pólik, I., Terlaky, T.: A survey of the S-lemma. SIAM Review 49(3), 371–418 (2007)

Traversing Symmetric Polyhedral Fans

Anders Nedergaard Jensen*

Mathematisches Institut, Georg-August-Universität Göttingen, Bunsenstraße 3-5, D-37073 Göttingen, Germany

Abstract. We propose an algorithm for computing the facets of a pure connected polyhedral fan up to symmetry. The fan is represented by an oracle. With suitable implementations of the oracle the same algorithm can be used for computing secondary fans, Gröbner fans, tropical varieties and Minkowski sums up to symmetry. The algorithm has been implemented in the software Gfan.

Keywords: Polyhedral fans, tropical geometry, algorithms, symmetry.

1 Introduction

Polyhedral fans arise naturally in convex geometry, with the prime example being secondary fans whose cones index all combinatorial types of polyhedra with a fixed set of normals. In algebraic geometry they give rise to toric varieties and play the central role in the evolving field of tropical geometry. This paper is concerned with the problem of computing polyhedral fans up to symmetry.

Exploiting symmetries in computational geometry is not a new idea. Indeed the method we present here specializes to the well-known *adjacency decomposition method* when the fan to be traversed is full-dimensional; see [3]. In the case of secondary fans, our work can be viewed as a refinement of [11].

We use the following example as a motivation for our approach.

Example 1. Consider the family of curves in $\mathbb{C}^2$ each defined by a polynomial

$$f = a + bx + cx^2 + dx^2y + ex^2y^2 + gxy^2 + hy^2 + iy + jxy \in \mathbb{C}[x, y].$$

A point on a curve is called a *cusp* if

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial y} = 0 \text{ and } \frac{\partial^2 f}{\partial x \partial y}\frac{\partial^2 f}{\partial y \partial x} - \frac{\partial^2 f}{\partial x \partial x}\frac{\partial^2 f}{\partial y \partial y} = 0$$

in that point. Eliminating variables x and y we get an ideal I_{cusp} defining the subfamily of curves with a cusp. We will be interested in the tropical variety $T(I_{\text{cusp}})$ which is a polyhedral fan. The reduced Gröbner basis for I_{cusp} with respect to the term order given by the point in the support of $T(I_{\text{cusp}})$

$$(304, -158, -152, -206, 388, -248, -146, -128, 346) \in \mathbb{R}^9, \tag{1}$$

tie-broken reverse lexicographically, has 18608 terms in 23 polynomials.

* Supported by the German Research Foundation (Deutsche Forschungsgemeinschaft (DFG)) through the Institutional Strategy of the University of Göttingen.

K. Fukuda et al. (Eds.): ICMS 2010, LNCS 6327, pp. 282–294, 2010.

In [2] it was suggested that the Gröbner cones are represented by Gröbner bases when computing tropical varieties. The example shows that for practical purposes it is important not to store all these bases in memory. The algorithm we present keeps only one such algebraic representation stored at a time.

For clarity, the fan will be given to us by an oracle. Our main contributions are a description of a symmetry exploiting traversal algorithm with a minimal number of oracle calls, a practical method for checking orbit membership of cones and finally a description of an oracle implementation for the restriction of a Gröbner fan to a lower-dimensional polyhedral cone.

We give a few more details on the implementation in the software Gfan [10] and end the paper by computing the fan in Example 1 using this software.

2 Definitions and Notation

By a *polyhedral cone* $C \subseteq \mathbb{R}^n$ we mean a finite intersection of closed halfspaces in $\mathbb{R}^n$, or, equivalently, the non-negative span $\text{cone}(v_1, \ldots, v_m)$ of a collection of vectors $\{v_1, \ldots, v_m\} \subseteq \mathbb{R}^n$. We use $\text{rel int}(C)$ to denote the *relative interior* of C. The inclusion largest linear subspace contained in C is called the *lineality space* of C. It equals $C \cap -C$. For an $\omega \in \mathbb{R}^n$ we let $\text{face}_\omega(C)$ denote the *face* of C at which $\langle \omega, \cdot \rangle$ is maximized and use the same notation for polytopes. A finite collection $\mathcal{F}$ of cones is called a *polyhedral fan* if

- $C \in \mathcal{F}$ implies that every face of C is in $\mathcal{F}$, and
- $C, C' \in \mathcal{F}$ implies that $C \cap C'$ is a face of C.

In particular, the cones in a fan must all have the same common lineality space. An example of a fan is the set of faces $\text{faces}(C)$ of a cone C. The *common refinement* $A \wedge B := \{a \cap b : (a, b) \in A \times B\}$ of two fans A and B is a fan. The *f-vector* of $\mathcal{F}$ lists the number of cones of each dimension, starting with a 1 for the lineality space. The *support* $\text{supp}(\mathcal{F})$ of a fan $\mathcal{F}$ is the union of its cones. We will use the untraditional word *ray* to denote a cone with exactly two faces (the cone and its lineality space). For a rational ray C the intersection $(C \cap -C)^\perp \cap C$ is a one-dimensional half-line which has a unique first non-zero lattice point in $\mathbb{Z}^n$. We call this the *primitive vector* of C and denote it $\text{prim}(C)$. Let the *link* of a cone C at a point $v \in C$ be

$$\text{link}_v(C) = \{u \in \mathbb{R}^n : \exists \delta \in \mathbb{R}_{>0} : \forall \varepsilon \in (0, \delta) : v + \varepsilon u \in C\},$$

and define the link of a fan $\mathcal{F}$ at a point v in the support of $\mathcal{F}$ to be the fan

$$\text{link}_v(\mathcal{F}) = \{\text{link}_v(C) | v \in C \in \mathcal{F}\}.$$

Since any two points in the relative interior of a cone $R \in \mathcal{F}$ will give the same link, we will also denote the link $\text{link}_R(\mathcal{F})$. In the special case where R is a *facet* of C, meaning that the dimension of R is one smaller than the dimension of C, the link $\text{link}_R(C)$ is a ray and we also denote it by $\text{ray}(R, C)$.

An inclusion maximal cone in a fan is called a *facet*. We shall be mainly interested in *pure* fans which are fans whose facets all have the same dimension d. A cone of dimension $d-1$ in such a fan is called a *ridge*. A *ridge path* in $\mathcal{F}$ is a sequence of facets $F_1, \ldots, F_s$ such that $F_i \cap F_{i+1}$ is a ridge. A pure fan is *connected in codimension one* if any two facets are connected by a ridge path. The symmetric group S_n acts on $\mathbb{R}^n$ by permuting coordinates. This action extents to cones and fans in $\mathbb{R}^n$. A subgroup $G \subseteq S_n$ is said to be a *symmetry group* of a fan $\mathcal{F}$ if it is contained in the stabilizer of $\mathcal{F}$.

2.1 Gröbner Fans and Tropical Varieties

We consider the polynomial ring $k[x_1, \ldots, x_n]$ over a field k. For a vector $\omega \in \mathbb{R}^n$, the *initial form* $\text{in}_\omega(f)$ of a polynomial $\sum_i c_i x^{a_i}$ with $c_i \in k \setminus \{0\}$ and $a_i \in \mathbb{N}^n$ is defined as the sum of all terms $c_i x^{a_i}$ such that $\langle \omega, a_i \rangle$ is maximal. We define the *initial ideal* of I as $\text{in}_\omega(I) := \langle \text{in}_\omega(f) : f \in I \rangle$. Now, fix an ideal I. Two vectors $u, v \in \mathbb{R}^n$ are equivalent if $\text{in}_u(I) = \text{in}_v(I)$. The closure of an equivalence class containing $v \in \mathbb{R}^n_{>0}$ is a polyhedral cone $\mathcal{C}_v(I)$ and the collection of all cones and their faces is the *Gröbner fan* $\Sigma(I)$ of I. The maximal cones of $\Sigma(I)$ are in bijection with the *marked reduced Gröbner bases* of I. They are reduced Gröbner bases where the initial term of each polynomial has been distinguished – it has been marked. Given a term order $\prec$ we use the notation $\mathcal{G}_\prec(I)$ for the marked reduced Gröbner basis with respect to $\prec$ and $\mathcal{C}_\prec(I)$ for its cone. If I is homogeneous, then the fan is complete and we define the *tropical variety* $T(I)$ of I to be the following subfan of the Gröbner fan $\Sigma(I)$: $T(I) := \{\mathcal{C}_v(I) : \text{in}_v(I) \text{ contains no monomial}\}$. See [2] and [8] for details.

3 The Traversal Algorithm

In this section we present an algorithm for traversing the maximal cones of a pure d-dimensional, codimension-one-connected fan $\mathcal{F}$. Explaining the algorithm in great detail makes it easy for us to be precise in Section 3.1 where we will modify the algorithm to exploit symmetry. The fan $\mathcal{F}$ is known to the algorithm only through an oracle. The oracle allows two main operations:

- Given a maximal cone $C \in \mathcal{F}$ and a facet R of C we may ask for $\text{link}_R(\mathcal{F})$. The link is a list of rays $O_C^{link}(R)$.
- Given a maximal cone C, a facet R of C and a ray $v \in \text{link}_R(\mathcal{F})$ we may ask for the cone $O_C^{change}(R, v)$ in $\mathcal{F}$ having link v at R.

The subscript in our oracle notation needs more explanation. We do not allow oracle calls in arbitrary order but think of the oracle as having an internal state being a facet $C \in \mathcal{F}$ and additional information. We may only ask for $O_C^{link}(R)$ and $O_C^{change}(R, v)$ when the oracle is in state C. The oracle call $O_C^{change}(R, v)$ changes the state to C', where C' is the returned maximal cone giving rise to v in the link. In addition to the above calls, we are allowed to ask the oracle which cone it is in; $O_C^{cone}()$ will return C, but it will not reveal the complete state.

Furthermore, we will assume that the oracle is in some state at the beginning of the traversal.

The following example illustrates admissible oracle call sequences. It will be a Gröbner fan to emphasize that the state may consists of non-geometric data.

Example 2. Consider the ideal $I := \langle a^2 + bc, b^2 + ac, c^2 + ab \rangle \subseteq \mathbb{Q}[a, b, c]$. A computation reveals that $\Sigma(I)$ is a three-dimensional fan in $\mathbb{R}^3$ with f-vector $(1, 9, 9)$ and a 1-dimensional lineality space. The 9 rays of the fan are generated by the lineality space and one of the vectors

$$(1, -1, 0), (0, -1, 1), (-1, -1, 2), (-1, 0, 1), (-1, 1, 0),$$

$$(-1, 2, -1), (0, 1, -1), (1, 0, -1), (2, -1, -1),$$

which are ordered cyclically. By Gröbner basis theory the nine maximal cones are in bijection to the nine reduced Gröbner bases of I. One of these is

$$\{\underline{c^2} + ab, \underline{bc} + a^2, \underline{b^2} + ac, \underline{a^2c}, \underline{a^2b}, \underline{a^4}\}$$

corresponding to the maximal cone $C_1 := \text{cone}(\pm(1, 1, 1), (-1, 0, 1), (-1, 1, 0))$. Let R_1 be the ridge $\text{cone}(\pm(1, 1, 1), (-1, 0, 1))$. An oracle representing the Gröbner fan would have

$$\begin{aligned} O^{link}_{C_1}(R_1) = \{&\text{cone}(\pm(1, 1, 1), \pm(-1, 0, 1), (-1, 2, -1)), \\ &\text{cone}(\pm(1, 1, 1), \pm(-1, 0, 1), (1, -2, 1))\}. \end{aligned}$$

Later we shall be less strict and think of this as just a set of two vectors, but because of scaling and the non-trivial lineality space there are several possibilities for choosing these representatives.

Let $v := \text{cone}(\pm(1, 1, 1), \pm(-1, 0, 1), (1, -2, 1))$. Making the call

$$C_2 := O^{change}_{C_1}(R_1, v) = \text{cone}(\pm(1, 1, 1), (-1, -1, 2), (-1, 0, 1))$$

changes the state to C_2. Now the calls $O^{link}_{C_1}(R_1)$ and $O^{change}_{C_1}(R_1, v)$ are illegal, while $O^{link}_{C_2}(R_1)$ and $O^{change}_{C_2}(R_1, -v)$ are legal. Applying a total of nine O^{change} calls we can return to state C_1.

Since the hidden state information can be huge, see Example 1, our calling conventions for the oracle have been designed so that only one state is stored, keeping memory consumption as low as possible. We note that reconstructing the hidden state information from a polyhedral cone can be quite complicated. Indeed, the Gröbner walk [5] speeds up the process of computing a Gröbner basis with respect to a prescribed term order by making a sequence of local changes.

We cannot always, as in Example 2, think of the facets of $\mathcal{F}$ as being vertices of a graph with the ridges being edges connecting them, since a ridge may connect more than two facets if the fan is not full-dimensional. Rather we should think of a hypergraph, in which the hyperedges connect many vertices. Traversing a hypergraph by an exhaustive search is not more complicated than traversing a

graph. In fact our problem translates into traversing the bipartite graph $G_{\mathcal{F}}$ with the right hand side being the facets, the left hand side being the ridges, and two cones being connected if one is contained in the other. Having $\mathcal{F}$ connected in codimension one is equivalent to $G_{\mathcal{F}}$ being connected.

We recall a basic graph traversal algorithm for connected graphs:

Algorithm 1
Input: *A connected graph $G = (V, E)$, a vertex $v \in V$.*
Output: *All vertices V of G.*

- $(A, B, D) := (\{v\}, \emptyset, \emptyset)$;
- *while*$(A \neq \emptyset)$
  - *Choose* $u \in A$;
  - $A := A \setminus \{u\}$;
  - $B := B \ominus \{\{a, u\} : \{a, u\} \in E\}$;
  - $D := D \cup \{u\}$;
  - $A := A \cup \{a : \{a, u\} \in B\}$;
- *output* D;

Here $S \ominus T$ denotes the symmetric difference $(S \cup T) \setminus (S \cap T)$ of two sets S and T. An invariant for the algorithm is that after each step the edge set B is the boundary of the vertex set D. At the end $D = V$ and B and A are empty.

We will use a depth-first approach to traverse the bipartite graph $G_{\mathcal{F}}$. This means that the set A will work as a last-in-first-out stack. Equivalently, we may present the above algorithm as two mutually recursive procedures, with left and right hand side nodes being treated differently. Thus, in the algorithm below, the set A is stored implicitly on the recursion stack while facets are written to the output rather than stored in the set D. The set B will no longer be a collection of sets of cones from $\mathcal{F}$, but rather consist of pairs of the form (R, v), where R is a ridge of $\mathcal{F}$ and v is a ray in $\mathrm{link}_R(\mathcal{F})$.

Algorithm 2
Input: *An oracle O in state C_0 representing a codimension 1 connected fan $\mathcal{F}$.*
Output: *All facets of $\mathcal{F}$.*

- $B := \emptyset$;
- *Call* **EnumerateFacet**(C_0) *below;*

EnumerateFacet(C)

- *Output* C;
- *Compute the facets of* C;
- $T := \{(R, \mathrm{ray}(R, C)) : R \text{ is a facet of } C\}$;
- $B := B \ominus T$;
- *For every pair* $(R, v) \in T$
  - *If* $(R, v) \in B$ *then call* **EnumerateRidge**(R, C);

EnumerateRidge(R, C)

- $L := O_C^{link}(R)$;
- $T := \{(R, v) : v \in L\}$;
- $B := B \ominus T$;
- $v' := \text{ray}(R, C)$;
- *For pair* $(R, v) \in T$
  - *If* $(R, v) \in B$ *then*
    - $C := O_C^{change}(R, v)$
    - *Call* **EnumerateFacet**(C);
    - $C := O_C^{change}(R, v')$

Proof. The algorithm is a direct translation of Algorithm 1 as explained above. We note that at any time the oracle is in state C and that after calling EnumerateRidge, EnumerateFacet sets C to the original second argument value by calling the oracle. This shows that the sequence of oracle calls is valid.

We measure the efficiency of a traversal strategy by the maximal number of $O_C^{link}(R)$ and $O_C^{change}(R, v)$ oracle calls needed as functions of the number of ridges and facets in the fan, respectively. An enumeration strategy is considered optimal if these functions are minimal among all strategies.

Proposition 1. *Let r be the number of ridges in $\mathcal{F}$ and f the number of facets. Algorithm 2 makes r oracle calls of type $O_C^{link}(R)$ which is optimal. It makes $2(f-1)$ oracle calls of type $O_C^{change}(R, v)$. By postponing the last oracle call $C := O_C^{change}(R, v')$ until absolutely needed, Algorithm 2 makes at most $\max(2f-3, 0)$ oracle calls of type $O_C^{change}(C, v)$. This is optimal.*

Proof. The result follows from the fact that EnumerateRidge is called once for every ridge, and that it does two $O_C^{change}(R, v)$ calls for every facet except C_0. The number of $O_C^{link}(R)$ calls is optimal, since every ridge must be investigated. We never have to bring the oracle back to the initial state at the end. This reduces the number of oracle calls by at least one. To see that this is optimal we consider a worst case scenario of a pure connected fan with f facets on a ridge path and $f + 1$ ridges. In an unlucky case the oracle starts close to one end of the fan, moves to the other end and is forced to go back to finish the job. This gives $2f - 3$ calls.

Whether $f - 1$ is the optimal number of $O_C^{change}(C, v)$ calls *for a particular graph* $G_{\mathcal{F}}$ depends on the topology of $G_{\mathcal{F}}$ and is related to the Hamiltonian path problem. We note that $f - 1$ is optimal if we relax the oracle call order restriction, but that this would increase memory usage in practice.

For practical implementations of Algorithm 2 it can be an advantage to represent the elements in B as pairs of vectors. For example, (R, v) can be represented by a pair of deterministically computed points in rel int(R) and rel int(v). We will return to the choice of these vectors in the next section.

3.1 Exploiting Symmetry

In addition to the oracle, Algorithm 2 could be changed to take a subgroup G of symmetries under which $\mathcal{F}$ is known to be invariant as input. Our goal would then be to find all orbits of maximal cones $\mathcal{F}$ under this group action. We shall restrict ourselves to symmetries which are coordinate permutations and assume that $G \subseteq S_n$. However, this restriction will only be important when we define $p(C)$ later in this section (where $\mathbb{Z}^n$ must be preserved) and for Algorithm 3.

First we define what we mean by a *canonical representative* for the orbit of a pair of cones (R, v), where R is a ridge of $\mathcal{F}$ and $v \in \text{link}_R(\mathcal{F})$. Fix a total order $\prec$ on the set K of polyhedral cones in $\mathbb{R}^n$. We define $\text{CanRep}(R, v)$ to be the smallest element in $\{(\sigma(R), \sigma(r)) : \sigma \in G\}$, with the ordering being the lexicographic order on $K \times K$ with each K ordered by $\prec$.

We now explain how to change Algorithm 2 to compute just one facet (and one ridge) of each orbit. To be precise we will avoid calls EnumerateRidge(R,C) if the procedure has already been called for another ridge in the orbit of R. Similarly, we avoid calls EnumerateFacet(C) and the two surrounding oracle calls if the procedure has already been called for another facet in the orbit of C. Equivalently, we traverse the bipartite *quotient graph* $\overline{G_{\mathcal{F}}}$, where vertices are identified if they are in the same orbit and multi-edges are regarded as single edges.

Three kinds of changes are required:

- In both procedures of Algorithm 2 we let T consist of the canonical representatives of the orbits of the pairs with respect to G rather than the pairs themselves. This may make T smaller since only one element from each orbit can be in T.
- At the two places where we check for containment of (R, v) in B, we should instead check for containment of $\text{CanRep}(R, v)$.
- When we recursively call EnumerateRidge after having checked that B contains $\text{CanRep}(R, v)$, we need to recover (one of) the original facet(s) of C giving rise to R. That is we must find the σ we applied to get R in T. We then use $\sigma^{-1}(R)$ when calling EnumerateRidge. Similarly, when we in EnumerateRidge have verified that $\text{CanRep}(R, v)$ is in B, we must find the (or one) $v \in L$ giving rise to the $\text{CanRep}(R, v)$ in T. We will use this v when calling the oracle.

In the above description we do operations on polyhedral cones when handling symmetries, but this is not convenient in practice. Rather, for a cone C we wish to define a canonical, symmetry invariant relative interior point $p(C)$. In particular, we must have $\sigma(p(C)) = p(\sigma(C))$ for every $\sigma \in G$ and cone $C \subseteq \mathbb{R}^n$. Checking if two cones of $\mathcal{F}$ are in the same orbit can be done by checking that their points are in the same orbit. Even better, for a pair of ridge-facet incidences represented by (R, v) and (R', v') we can check if they are the same up to symmetry by checking if $(p(R), p(v))$ and $(p(R), p(v))$ are the same up to symmetry.

We notice that the vector $p(C) := \sum \text{prim}(r)$, where r runs over all rays of C, satisfies the above properties. However, this definition has the disadvantage

that computing it requires knowing the extreme rays of C. Often C is simplicial and this is not a problem, but in general an H-to-V conversion is needed. Alternatively, we may define $p(C)$ using *analytic centers* of polytopes, which can be computed in polynomial time by numerical methods. We have no practical experience with this approach.

3.2 Symmetry Algorithms

Complexity-wise, deciding if two vectors in $\mathbb{Z}^n$ are the same up to the action of a group $G \subseteq S_n$, specified by its generators, is as hard as the graph isomorphism problem of deciding if two graphs are the same up to permutation of their vertices. Indeed asking if the edge-vertex incidence matrices of the two graphs are the same up to row and column interchanges answers the question. The graph isomorphism problem is not known to be in P, the class of polynomial time solvable problems, and therefore we cannot expect the canonical representative computation to have polynomial time complexity. We discuss how to solve the problem in practice.

We will not address the problem of computing generators for our group G but rather suppose that they are given. Each generator can be represented by a permutation of the vector $(1, 2, \ldots, n)$. We start by precomputing all elements of G and store them in a *prefix tree* (or a *trie*). A prefix tree has an integer at each node (except the root), and it represents all vectors of integers we get by going from the root to a leaf, picking up integers from the nodes we pass through. Thus we will use a tree of depth n. We are seeking an algorithm with the following specification.

Algorithm 3
Input: *A subgroup $G \subseteq S_n$ stored in a prefix tree and vectors $R, v \in \mathbb{Z}^n$.*
Output: *A permutation $\sigma \in G$ such that $(R_{\sigma_1}, v_{\sigma_1}, \ldots, R_{\sigma_n}, v_{\sigma_n})$ is lexicographically smallest.*

Such an algorithm can be achieved by making a combinatorial backtracking search over the prefix tree. At a node at level i we follow those edges leading to vertices whose markings σ_i make $(R_{\sigma_i}, v_{\sigma_i})$ lexicographically smallest. We keep a vector with the optimal permutation of R and v seen so far. Using this vector branches can be pruned if they cannot lead to an optimal permutation.

If stabilizers are small, which is often the case in our setting, and the group fits into memory, then the method described here works well. We refer to the field of computational group theory for other approaches, see [3] for references.

4 Oracles

Using different terminology the oracles for traversing normal fans of Minkowski sums of polytopes, secondary fans, Gröbner fans and tropical varieties are already present in the literature, see [6], [11],[5] and [2], respectively. The topic of this section is slight variations of these. Due to the size limit for this paper, we only

discuss one of these oracles in detail, while briefly mentioning other possible variations.

We first consider the d-skeleton of the normal fan of a Minkowski sum of polytopes $P_1, \ldots, P_s$ whose vertices are given. This is a connected fan, and the link of a ridge with relative interior point ω is the d-skeleton of the normal fan of the Minkowski-sum of $\text{face}_\omega(P_1), \ldots, \text{face}_\omega(P_s)$. Modulo the lineality space the link is a collection of rays, which are the normals of the Minkowski sum of the faces. This is a general behavior; the computation becomes easier at the link – at least for $s = 2$, the Minkowski sum facets can be computed by a V-to-H conversion of the convex hull $\text{conv}((\text{face}_\omega(P_1) \times e_1) \cup \cdots \cup (\text{face}_\omega(P_s) \times e_s)) \subseteq \mathbb{R}^n \times \mathbb{R}^s$, which is also known as the *Cayley embedding* of $\text{face}_\omega(P_1), \ldots, \text{face}_\omega(P_s)$.

In the following we will explain how Gröbner fan computations can be restricted to cones of $\mathbb{R}^n$. It is important to note that a similar technique works for computing slices of secondary fans. One application of this can be found in the last paragraph of this paper.

4.1 The Gröbner Fan

Recall that the maximal cones of $\Sigma(I)$ are in bijection with the marked reduced Gröbner bases $\{\mathcal{G}_\prec(I)\}_\prec$ where $\prec$ runs through all term orders. Inequalities for the Gröbner cone of $\mathcal{G}_\prec(I)$ can be read off from the exponent vectors of $\mathcal{G}_\prec(I)$. To make a change to another cone $O_C^{change}(R, v)$ through a ridge R with relative interior point ω and normal v, we compute the Gröbner basis with respect to the ordering given by $\omega + \varepsilon v$ with $\varepsilon > 0$ small (tie-broken in any way).

While the ε-perturbation is easy to handle in theory and practice with matrix term orders, the reader familiar with Gröbner bases will know that the above description is an oversimplification. One will not compute the $\omega + \varepsilon v$ Gröbner basis from scratch, but rather use the identity

$$\text{in}_{\omega+\varepsilon v}(I) = \text{in}_v(\text{in}_\omega(I))$$

to construct a Gröbner basis for I from one of $\text{in}_\omega(I)$. As for the Minkowski sum problem, the computation at the link becomes easier. See [5] and [8] for details.

We now explain how to restrict the Gröbner fan computation to a possibly lower-dimensional cone $D \subseteq \mathbb{R}^n$. One problem that we might face if the ideal is not homogeneous is that $\Sigma(I)$ is not complete and the usual restriction $\Sigma(I) \wedge \text{faces}(D)$ is not connected in codimension one – take for example $I = \langle x_1^2 x_2 + x_1 x_2^2 + 1 \rangle$ and $D = \{\omega \in \mathbb{R}^2 : \omega_1 + \omega_2 \leq 0\}$. There are several ways to get around this problem. Here, to keep the exposition simple, we will assume that I is homogeneous, and thus $\Sigma(I)$ complete.

Definition 1. *Let $I \subseteq k[x_1, \ldots, x_n]$ be an ideal and let $D \subseteq \mathbb{R}^n$ be a polyhedral cone. We define the* restriction $\Sigma(I)_D := \Sigma(I) \wedge \text{faces}(D)$.

The support of the restriction $\Sigma(I)_D$ is D.

Definition 2. *A ridge R in $\Sigma(I)_D$ is called* flippable *if* $\text{rel int}(R) \cap \text{rel int}(D)$ *is not empty.*

Lemma 1. *The restriction $\Sigma(I)_D$ is a pure fan connected in codimension* 1*. It is connected even if we only consider flippable ridges.*

Every maximal cone in $\Sigma(I)_D$ is of the form $\mathcal{C}_\prec(I) \cap D$ and we will represent such cone by $\mathcal{G}_\prec(I)$. This representation is not unique. We have described how the internal state of the oracle is stored, and will now explain how the oracle calls can be implemented:

Cone: The cone represented by $\mathcal{G}_\prec(I)$ can be computed as the intersection $\mathcal{C}_\prec(I) \cap D$.

Link: The link of a ridge has either one or two rays. One of these rays v is already known to us as the link of C at R in the oracle call and we need to decide if $-v$ is in the link. To check if R is flippable, it suffices to check if a relative interior point ω of R is in one of the facets of D.

Change: Let ω be a positive vector in the flippable ridge. A reduced Gröbner basis representing the neighbouring cone can be gotten by computing a Gröbner basis with respect to the term order given by $\omega + \varepsilon v$, tie-broken in any way. A more efficient way is to pass to the initial ideal $\mathrm{in}_\omega(I)$ first. See [5] and [8].

We note that for a facet $C \in \Sigma(I)_D$ it is easy to recover the initial ideal with respect to relative interior points of C. This was used in [4] for a method to check if a given generating set of I is a tropical basis.

There is still one problem that we need to address. Namely, how we get started, i.e. given I and D, how we compute a reduced Gröbner basis $\mathcal{G}_\prec(I)$ such that $\mathcal{C}_\prec(I) \cap D$ is maximal in $\Sigma(I)_D$. The solution is an application of matrix term orders. First we pick a vector $c_1 \in \mathrm{rel\,int}((D + \mathcal{C}_0(I)) \cap \mathbb{R}^n_{\geq 0})$ and then extend c_1 to a basis $\{c_1, \ldots, c_d\}$ of $\mathrm{span}_{\mathbb{R}}(D)$. Then we extend this basis to a basis $\{c_1, \ldots, c_n\}$ of $\mathbb{R}^n$. The term ordering of the matrix with rows $c_1, \ldots, c_n$ gives the desired Gröbner basis. Alternatively, we consider $c = c_1 + \varepsilon c_2 + \ldots + \epsilon^{n-1} c_n$ and compute

$$\mathrm{in}_c(I) = \mathrm{in}_{c_n}(\mathrm{in}_{c_{n-1}}(\cdots \mathrm{in}_{c_1}(I) \cdots))$$

successively. This is the initial ideal for $\mathcal{G}_\prec(I)$. To construct $\mathcal{G}_\prec(I)$ we may repeatedly apply the Gröbner walk lifting procedure as it was done in [2, Algorithm 9].

Having specified the oracle, we may also apply the symmetric version of the traversal algorithm. The group G should be a symmetry group for $\Sigma(I)_D$.

It is tricky to extend Definition 1 and 2 to cover the non-homogeneous case and we shall only discuss one subtlety in this setting. Take $D = \mathbb{R}^n$. In this case, see [8], it is natural to allow only flips through facets with positive points in their interior, since this will guarantee usage of only allowable term orders at the ridge. Consider for example $I = \langle x^3 + y^3 + x^2y^2 \rangle$, which has a complete Gröbner fan with a ridge outside the strictly positive orthant. A priori, this ridge should not be considered flippable and the "flippable link" at that ridge should only consist of one ray, even though the geometric link consists of two. The only problem with this is that EnumerateRidge is called more than once for the same ridge in Algorithm 2. This does not change the correctness of the algorithm.

5 Comparison to Reverse Search

The memory-less reverse search method [1] can be used for traversing many types of full-dimensional fans – including Gröbner fans, see [8]. It works by traversing a spanning tree of the graph whose vertices are the facets and whose edges are the ridges of the fan. Symmetry can be exploited by restricting to a fundamental domain of the group action on $\mathbb{R}^n$, but still orbits whose cones touch the boundary of the fundamental domain may be computed more than ones. The reverse search has the drawback that in order to decide on the local structure of the traversal tree an O_C^{change} oracle call must be performed once for every ridge in a traversal. Therefore, for complicated oracles, it will often perform worse than Algorithm 2. On the other hand, the draw back of Algorithm 2 is that a vector in $\mathbb{Z}^{2n}$ needs to be stored for essentially every ridge-facet incidence pair and that the algorithm is not as easy to parallelize as reverse search, where interprocess communication is absent.

6 Implementation Details

6.1 Handling Geometric Data

We briefly discuss how to compute properties of a cone given to us by an inequality description. The natural order of getting these properties is as follows: lineality space, span and dimension, facets, a relative interior point, rays.

The lineality space can be computed by Gauss elimination, while linear programming is needed for the span and the facets. Knowing the span of the cone is equivalent to knowing the implied equations of the inequalities. We refer to literature on the simplex algorithm. For the facets of a Gröbner cone, the inequality description is often highly redundant and a preprocessing step is useful. A relative interior point can be computed with linear programming and the computation of the rays can be reduced to an H-to-V conversion of a polytope.

Knowing the facets and rays it is a combinatorial task to extract the face lattice. Indeed given the set of rays of C contained in face A of C, we find all facets of A by, for each facet normal of C, picking the set of rays perpendicular to the normal. This may give lower dimensional faces of A, so we need to take the inclusion maximal sets of rays. They represent the facets of A.

Rather than computing the face lattice, it might be useful to know the orbits of all cones in a fan. We suggest keeping a list of canonical representatives for the orbits seen so far. Then we may run through the facets of F, and for each of these repeatedly apply the method of the previous paragraph, but for each newly computed face checking if its canonical representative has been seen already. A fast implementation of Algorithm 3 is useful at this point.

6.2 Software

The presented algorithm has been implemented according to a generic programming / object oriented paradigm in the software Gfan [10] and replaces old

traversal strategies. Every oracle is derived from an abstract superclass. Starting from version 0.5 are features for computing restrictions of Gröbner fans and secondary fans. The oracle of Section 4.1 has been used in [4] to give a computer proof that the 4×4 minors of a 5×5 matrix are a tropical basis. In that paper also a tropical variety with a symmetry group of order 28800 was computed, explaining the need for handling symmetries. The Gfan software is written in C++, uses the libraries GMP [9] and cddlib [7] by default, and works like a Unix-style command line tool. In addition, Gfan can be linked to the floating point LP-solver SoPlex [12]. In this case LP-certificates will be lifted to $\mathbb{Q}$ using continued fractions, and checked. In case of failure, Gfan will fall back on cddlib.

Returning to Example 1, a two hour computation in Gfan gives the tropical variety $T(I_{\text{cusp}})$, exploiting a symmetry group of order 8. The 7-dimensional fan has a 3-dimensional lineality space and f-vector $(1, 1631, 7622, 11340, 5408)$. A total of 1431 ridges and 680 facets were processed. With a suitably prepared input file the computation can be done with the following command:

```
gfan_tropicaltraverse --symmetry < Icusp.startingcone
```

If the `--symmetry` option is left out the computation takes more than 15 hours. This order of speed-up is only expected for expensive oracles and symmetry groups that fit into memory. If the symmetry groups are sufficiently complicated and oracle calls are cheap, it is possible that all time saved on oracle calls is spent on computing canonical representatives.

We finish this paper by mentioning a few applications of Gfan and its algorithms to tropical geometry. In tropical geometry a natural question to ask is whether $\text{supp}(T(I_{\text{cusp}}))$ is the support of a subfan of the secondary fan $\mathcal{F}_1$ of the 2-dimensional Newton polytope of f. We may compute $\mathcal{F}_2 := \mathcal{F}_1 \wedge T(I_{\text{cusp}})$ by restricting the computation of $\mathcal{F}_1$ to each of the 680 facets. After this we pick a relative interior point from each facet of $\mathcal{F}_2$, and take the smallest subfan $\mathcal{F}_3 \subseteq \mathcal{F}_1$ containing these vectors. The question now is if $\text{supp}(\mathcal{F}_3) = \text{supp}(T(I_{\text{cusp}}))$. By construction, $\text{supp}(\mathcal{F}_3) \supseteq \text{supp}(T(I_{\text{cusp}}))$. The other inclusion can be checked by computing the restriction of $\Sigma(I_{\text{cusp}})$ to each cone of $\mathcal{F}_3$. Then we check for each corresponding initial ideal if it is monomial free. The polyhedral fan computations can be handled with Algorithm 2 and its implementation in Gfan. However, in our case none of this is needed since the vector (1) induces a regular subdivision whose secondary cone has dimension larger than the dimension of $\text{supp}(T(I_{\text{cusp}}))$.

References

1. Avis, D., Fukuda, K.: A basis enumeration algorithm for convex hulls and vertex enumeration of arrangements and polyhedra. Discrete Computational Geometry 8, 295–313 (1992)
2. Bogart, T., Jensen, A.N., Speyer, D., Sturmfels, B., Thomas, R.R.: Computing tropical varieties. J. Symbolic Comput. 42(1-2), 54–73 (2007)
3. Bremner, D., Sikiric, M.D., Schürmann, A.: Polyhedral representation conversion up to symmetries. CRM proceedings 48, 45–72 (2009)

4. Chan, M., Jensen, A., Rubei, E.: The 4 x 4 minors of a 5 x n matrix are a tropical basis, arXiv:0912.5264 (2009)
5. Collart, S., Kalkbrener, M., Mall, D.: Converting bases with the Gröbner walk. J. Symbolic Comput. 24(3-4), 465–469 (1997); Computational algebra and number theory, London (1993)
6. Fukuda, K.: From the zonotope construction to the Minkowski addition of convex polytopes. J. Symb. Comput. 38(4), 1261–1272 (2004)
7. Fukuda, K.: cddlib reference manual, cddlib Version 094b. Swiss Federal Institute of Technology, Lausanne and Zürich, Switzerland (2005), http://www.ifor.math.ethz.ch/~fukuda/cdd_home/cdd.html
8. Fukuda, K., Jensen, A., Thomas, R.: Computing Gröbner fans. Mathematics of Computation 76, 2189–2212 (2007)
9. Granlund, T., et al.: GNU multiple precision arithmetic library 4.3.1 (2009), http://gmplib.org
10. Jensen, A.N.: Gfan, a software system for Gröbner fans and tropical varieties, http://www.math.tu-berlin.de/~jensen/software/gfan/gfan.html
11. Rambau, J.: TOPCOM: Triangulations of point configurations and oriented matroids. ZIB Report, 02-17 (2002)
12. Wunderling, R.: Paralleler und objektorientierter Simplex-Algorithmus. PhD thesis, Technische Universität Berlin (1996), http://www.zib.de/Publications/abstracts/TR-96-09/

C++ Tools for Exploiting Polyhedral Symmetries

Thomas Rehn and Achill Schürmann*

[1] Institute of Applied Mathematics, Delft University of Technology, Mekelweg 4, 2628 CD Delft, The Netherlands
a.schurmann@tudelft.nl
[2] Faculty for Mathematics, Otto-von-Guericke University Magdeburg, 39106 Magdeburg, Germany
thomas@carmen76.de

Abstract. We report on the recently developed C++ tools `PermLib` and `SymPol` that are designed to support high performance work with symmetric polyhedra. The callable library `PermLib` provides basic support for permutation group algorithms and data structures. It can in particular be used for the development of optimization algorithms that combine methods from polyhedral combinatorics and computational group theory. The software `SymPol` is such an application helping to detect polyhedral symmetries and to analyze faces of polyhedra up to symmetries. It in particular provides successfully used decomposition methods for polyhedral representation conversions up to symmetries.

Keywords: polyhedral combinatorics, symmetries, permutation group algorithms, representation conversion.

Symmetric Polyhedra: From Beauty to Computational Use

Symmetric polyhedra as the Platonic and Archimedean solids have fascinated not only mathematicians since time immemorial. They occur frequently in diverse contexts of art and science. The "truncated icosahedron" for example is known as Buckminsterfullerene C60, but also as soccerball. Less known to a general audience, but of great importance to modern mathematics and its applications, are higher dimensional analogues of these familiar objects.

In mathematical optimization, symmetric polyhedra are often used to model problems of applications such as transportation logistics, machine scheduling or portfolio planning. Examples of symmetric polyhedra frequently studied in combinatorial optimization have names with prefixes like "travelling salesman", "assignment", "matching", "CUT" and others.

Although many polyhedra in optimization have symmetries, standard algorithms do not take advantage of them. Even worse, many methods, such as those used in integer programming, are known to work notoriously badly on symmetric problems. However, in recent years several authors have shown, at least for

* Both authors were supported by the Deutsche Forschungsgemeinschaft (DFG) under grant SCHU 1503/4-2.

K. Fukuda et al. (Eds.): ICMS 2010, LNCS 6327, pp. 295–298, 2010.

specific problems, that polyhedral symmetries can be exploited. For surveys of several approaches for integer linear programming and polyhedral representation conversion we refer the interested reader to [Mar09] and [BDS09].

PermLib: A C++ Library for Permutation Computations

Common to most symmetry exploiting approaches is the need for computational tools to work with permutation groups. Typical tasks are the detection of available symmetries, the computation of stabilizer groups and in-orbit tests. Computer algebra systems like [GAP] provide a lot of high-level support for these problems, and they are often used for experimental code, in particular for applications in mathematics. However, for high performance computations it is desirable to have more flexible tools available.

To support future developments of symmetry exploiting algorithms in optimization, we have recently developed the callable C++-library PermLib [PermLib]. The library provides fundamental data structures for the representation of symmetries as permutation groups, as well as implementations of algorithms to perform basic tasks. It allows in particular efficient computations with *bases and strong generating sets* (BSGS, see for example [HEO05] for details), and based on them backtrack searches, like the partition backtracking by Leon [Leo91]. These can be used to perform essential tasks, such as deciding on (non-)equivalence, obtaining stabilizers or fusing and splitting of orbits. We refer to [Reh10] for details on the implemented algorithms and their performance.

We view PermLib as a starting point for future developments of optimization algorithms, in particular for new methods that combine techniques from polyhedral combinatorics and computational group theory.

SymPol: A C++ Tool for Work with Symmetric Polyhedra

Our software package SymPol [SymPol] is a first application built on the functionality provided by PermLib. It helps in an easy-to-use way to detect *linear* symmetries of a given polyhedron and supports the user in analyzing the facial structure of a given polyhedron up to symmetries. Input and Output format are text files as they are used by [cdd] and [lrs]. Both packages are used internally for polyhedral computations. In addition to a representation of a polyhedron, the user has the possibility to provide known parts of its *combinatorial* symmetry group. As a callable library, SymPol can also be used for future developments of symmetry exploiting algorithms.

If no symmetries (or only a few) are known a priori, the user has the possibility to compute *restricted automorphisms* of a polyhedron with SymPol. Sometimes finding additional symmetries, for example in subpolyhedra, can be essential for solving a problem (see [DSV07] for an example). The obtained automorphisms may not generate the full (combinatorial) symmetry group of the polyhedron, but they can be computed without full knowledge of both polyhedral descriptions, via automorphisms of certain colored graphs. We refer to [BDS09] for further details. SymPol uses the backtrack techniques provided by PermLib to compute

these automorphisms, but also has the possibility to make use of the specialized software package `nauty` [nauty].

Besides symmetry detection, our package in particular supports polyhedral representation conversions up to symmetry. That is, it helps to convert the description of a polyhedron with linear inequalities into one with generating vertices and rays, or vice versa. This is one of the most fundamental tasks in polyhedral combinatorics and in many application it is sufficient to find inequalities or generators up to symmetries. Depending on the given polyhedron, different techniques have been shown to be successful. Using the `GAP` package [Polyhedral], in particular the *Adjacency Decomposition Method* has recently been shown to work well for huge conversion tasks. Two extreme examples are described in [DSV07] and [DSV09]. For a detailed description of *Decomposition Methods* that split a given conversion problem into a number of lower dimensional subproblems we refer to [BDS09]. Best results can be achieved by a combination of decomposition methods. This is made possible by `SymPol`, which supports the user also to choose an individual "fine tuning" of combinations, fitting best to his or her conversion problem.

Examples of Usage

Installing `SymPol` is quite easy on a Linux/UNIX system with `CMake` [CMake], `Boost` [Boost] and `GMP` [GMP] installed. For a more detailed description we refer to the `SymPol` manual.

To compute the restricted automorphism group of a polyhedron with its description contained in `input-file`, simply call

```
sympol --automorphisms-only input-file
```

`SymPol` offers the possibility to estimate the difficulty of a representation conversion, for example via the command-line option `--estimation-only`. For "difficult input" (for example with an estimation value above 40) try

```
sympol --adm 40 input-file
```

and experiment with the "ADM threshold" if necessary. This type of call may be successful on polyhedra whose representation conversion seems impossible with standard tools like `cdd` or `lrs`. Such an example is given by Kumar in [Kum10], where he uses `SymPol` to compute all the classes of elliptic divisors on a generic Jacobian Kummer surface.

But even for "easy input", `SymPol` offers some nice features to analyze the facial structure of a polyhedron up to symmetries. For example with the call

```
sympol --idm-adm-level 0 1 --adjacencies input-file
```

where `input-file` contains the 48 vertices of the 5-dimensional *Santos prismatoid* (see Table 1 in [San10]), `SymPol` returns the adjacency graph of facets up to symmetry (Figure 4 in [San10]). This graph contains the essential information (a shortest path of length 6) for the construction of Santos' counterexample to the Hirsch conjecture. We think that `SymPol` will serve as a very useful tool for such and similar computations in the future.

References

[BDS09] Bremner, D., Dutour Sikirić, M., Schürmann, A.: Polyhedral representation conversion up to symmetries. In: Bremner, D., Avis, D., Deza, A. (eds.) Proceedings of the 2006 CRM Workshop on Polyhedral Computation. CRM Proceedings & Lecture Notes, vol. 48, pp. 45–71. AMS, Providence (2009)

[DSV07] Dutour Sikirić, M., Schürmann, A., Vallentin, F.: Classification of eight dimensional perfect forms. Electron. Res. Announc. Amer. Math. Soc. 13, 21–32 (2007)

[DSV09] Dutour Sikirić, M., Schürmann, A., Vallentin, F.: The contact polytope of the Leech lattice. Discrete Comput. Geom. (to appear), preprint at `arXiv:0906.1427`

[HEO05] Holt, D.F., Eick, B., O'Brien, E.A.: Handbook of computational group theory. Chapman & Hall/CRC, Boca Raton (2005)

[Kum10] Kumar, A.: Elliptic fibrations on a generic Jacobian Kummer surface (in preparation)

[Leo91] Leon, J.S.: Permutation group algorithms based on partitions. I. Theory and algorithms. J. Symbolic Comput. 12, 533–583 (1991)

[Mar09] Margot, F.: Symmetry in integer linear programming. In: Jünger, M., Liebling, T.M., Naddef, D., Nemhauser, G.L., Pulleyblank, W.R., Reinelt, G., Rinaldi, G., Wolsey, L.A. (eds.) 50 Years of Integer Programming 1958–2008. Springer, Heidelberg (2009)

[Reh10] Rehn, T.: Fundamental Permutation Group Algorithms for Symmetry Computation, Diploma thesis (computer science), Otto von Guericke University Magdeburg (2010), `http://fma2.math.uni-magdeburg.de/~latgeo/permlib/diploma-thesis-cs-rehn.pdf`

[San10] Santos, F.: A counterexample to the Hirsch conjecture, preprint at arxiv:1006.2814

Software

[Boost] Free peer-reviewed portable C++ source libraries, `http://www.boost.org/`

[cdd] Fukuda, K.: cdd and cddplus, `http://www.ifor.math.ethz.ch/~fukuda/cdd_home/`

[CMake] Cross Platform Make, `http://www.cmake.org/`

[GAP] Groups, Algorithms, Programming - a system for computational discrete algebra, `http://www.gap-system.org/`

[GMP] The GNU Multiple Precision Arithmetic Library, `http://gmplib.org/`

[lrs] Avis, D.: `http://cgm.cs.mcgill.ca/~avis/C/lrs.html`

[nauty] McKay, B.D.: ver. 2.2., `http://cs.anu.edu.au/people/bdm/nauty/`

[PermLib] Rehn, T.: A callable C++ library for permutation computations, ver. 0.2., `http://fma2.math.uni-magdeburg.de/~latgeo/permlib/permlib.html`

[Polyhedral] Dutour Sikirić, M.: A GAP package, `http://www.liga.ens.fr/~dutour/Polyhedral/`

[SymPol] Rehn, T., Schürmann, A.: A C++ tool for the work with symmetric polyhedra, preliminary version 0.1, `http://fma2.math.uni-magdeburg.de/~latgeo/sympol/sympol.html`

isl: An Integer Set Library for the Polyhedral Model

Sven Verdoolaege

Department of Computer Science, Katholieke Universiteit Leuven, Belgium and
Team ALCHEMY, INRIA Saclay, France
Sven.Verdoolaege@{cs.kuleuven.be,inria.fr}

1 Introduction and Motivation

In compiler research, polytopes and related mathematical objects have been successfully used for several decades to represent and manipulate computer programs in an approach that has become known as the polyhedral model. The key insight is that the kernels of many compute-intensive applications are composed of loops with bounds that are affine combinations of symbolic constants and outer loop iterators. The iterations of a loop nest can then be represented as the integer points in a (parametric) polytope and manipulated as a whole, rather than as individual iterations. A similar reasoning holds for the elements of an array and for mappings between loop iterations and array elements.

For most types of program transformations, it is safe to approximate the set of integer points in a polytope by the polytope itself. Many researchers therefore use polyhedral libraries such as `PolyLib` [18] and `PPL` [1] that exploit the double description of polytopes in terms of both facets and vertices. In particular, some operations can be performed a lot more efficiently on one representation than on the other. However, the computation of one representation from the other may also be very costly, as in the worst case the size of the output may be exponential in that of the input. In practice, polyhedra that arise from compiler applications are typically close to hypercubes, i.e., they have few facets and many vertices.

A different approach is taken by the `Omega` library [16], which specifically handles sets of integer tuples satisfying affine constraints. There is also explicit support for parameters, existentially quantified variables and relations between pairs of integer tuples, making the library not only more accurate, but also more convenient to use. Note that polyhedral libraries have no need for existentially quantified variables since the projection of a rational polyhedron is again a rational polyhedron. The internal representation is based on the constraints of the sets (although vertices are implicitly constructed during the convex hull computation) and most operations are built on top of an extension of Fourier-Motzkin elimination [20] and a series of heuristics. The library is very fast on simple problems, but rather unpredictable on larger problems. Furthermore, it is not thread-safe and only supports machine precision. The library had also been left unmaintained for many years and had grown a reputation of being unreliable due to various unimplemented corner cases. Only recently have most, if not all, of these corner cases been resolved in the `Omega+` library [7].

K. Fukuda et al. (Eds.): ICMS 2010, LNCS 6327, pp. 299–302, 2010.

We present `isl`, an LGPL, thread-safe, C library for manipulating sets and relations of integer tuples bounded by affine constraints using `GMP` [13] based arbitrary precision integer arithmetic. The interface of the library draws inspiration from that of `Omega`, but the underlying implementation is completely different, favoring the use of a collection of targeted and efficient algorithms. The internal representation is also different, with `Omega` transforming sets with existentially quantified variables to unions of intersections of polyhedra and lattices in order to be able to perform some set operations, while `isl` uses a representation in terms of integer divisions inspired by the output format of `PipLib`, a library for performing parametric integer programming [11]. The `isl` library is available from `http://freshmeat.net/projects/isl/`

The `isl` library is mainly intended to be used in the polyhedral model for program analysis and transformation, but some of the many operations it supports can and have been used outside of this model. From inception, one of the primary long-term objectives has been to provide all set and polynomial manipulations required by the `barvinok` library, which, at that time, used a combination of `PolyLib`, `PipLib`, `Omega` and `GiNaC` [4]. We have already achieved the short-term objectives of replacing `PolyLib` in the loop generator `CLooG` [3], producing better code by eliminating constraints that are redundant over the integers but not over the rationals, and of forming the basis of an equivalence checker [22] of programs that can be represented in the polyhedral model.

2 Internals

The main objects of interest are sets and binary relations over tuples of integers bounded by affine constraints, which we will call polyhedral sets and maps, respectively. Each map R is a finite union of basic maps $R = \bigcup_i R_i$, each mapping a tuple of n integer parameters to a binary relation on tuples of integers, i.e., $R_i : \mathbb{Z}^n \to 2^{\mathbb{Z}^{d_1+d_2}} : \boldsymbol{s} \mapsto R_i(\boldsymbol{s})$, with

$$R_i(\boldsymbol{s}) = \{\, \boldsymbol{x}_1 \to \boldsymbol{x}_2 \in \mathbb{Z}^{d_1} \times \mathbb{Z}^{d_2} \mid \exists \boldsymbol{z} \in \mathbb{Z}^e : A_1\boldsymbol{x}_1 + A_2\boldsymbol{x}_2 + B\boldsymbol{s} + D\boldsymbol{z} + \boldsymbol{c} \geq \boldsymbol{0} \,\}$$

and $A_i \in \mathbb{Z}^{m \times d_i}$, $B \in \mathbb{Z}^{m \times n}$, $D \in \mathbb{Z}^{m \times e}$ and $\boldsymbol{c} \in \mathbb{Z}^m$. Sets are defined similarly. The difference between sets and maps lies only in their use. Maps have domains and ranges, can be composed with each other and can be applied to sets. Basic sets are essentially projections of the integer points in a polyhedron and include intersections of polyhedra and lattices as a special case. Note that in practice and for reasons of efficiency, equality constraints are represented separately. For some operations, it is convenient to have explicit representations for the existentially quantified variables. In particular, we use greatest integer parts of rational affine combinations of the parameters and the domain and range variables.

The core of the library is formed by an incremental LP solver modeled after that of `Simplify` [10]. This solver is used in practically every operation of the library. In particular, it is used in an ILP feasibility solver based on generalized basis reduction [9], which is in turn used to check the emptiness of a set, producing a sample element if not. Such sample elements are used during the

computation of the integer affine hull using the algorithm of [15], which is very useful for reducing the dimension of a set by detecting implicit equalities and for eliminating redundant existentially quantified variables. Finally, parametric integer programming [11] is built on top of these LP and ILP solvers. Parametric integer programming is used to compute the *lexicographic minimum* of a map and to compute a unique (lexicographically minimal) representation for the existentially quantified variables. The lexicographic minimum of a map R is a map R' that maps each domain element $\boldsymbol{x} \in \operatorname{dom} R$, to the unique lexicographically minimal element in its image, i.e., $R'(\boldsymbol{s}) = \{ \boldsymbol{x} \to \boldsymbol{y} \in R(\boldsymbol{s}) \mid \boldsymbol{y} = \operatorname{lexmin} R(\boldsymbol{s}, \boldsymbol{x}) \}$, with $R(\boldsymbol{s}, \boldsymbol{x}) = \{ \boldsymbol{y} \mid \boldsymbol{x} \to \boldsymbol{y} \in R(\boldsymbol{s}) \}$.

The above algorithms are used to implement the basic operations on sets and maps such as *intersection*, *union*, *difference*, *projection* and *emptiness check*. Other operations require additional algorithms, some of which are listed below.

- *convex hull*, a very "rational" operation, which therefore does not fit in very well in an integer set library and is not implemented very efficiently. Still, it is provided as it is used in `CLooG`. The algorithm is based on [14], extended to handle unbounded polyhedra. The library also provides a "*simple hull*" operation, which computes the smallest basic set that contains the input set and that can be described using only translates of the constraints of the input set. The result is an overapproximation of the convex hull, but it is much more efficient to compute.
- *set coalescing* changes the representation of a set (without changing its meaning) by replacing pairs of basic sets by a single basic set. The algorithm is based on a variation of the constraints based technique of [6], but extended to handle sets of integers. It is different from the algorithm of [2], which uses both constraints and vertices and considers only rational sets.
- the *transitive closure* of a map R is the map $R^+ = \bigcup_{k \geq 1} R^k$, with $R^1 = R$ and $R^k = R \circ R^{k-1}$ for $k \geq 2$. It is computed approximatively using an algorithm that improves upon both [17] and [5].
- *dependence analysis* [12] is a crucial operation for the polyhedral model. Given a list of write and read accesses in a program, dependence analysis determines which write instance is the last to precede a given read instance. The algorithm relies heavily on the computation of lexicographic maxima.
- *parametric vertex enumeration* computes the parametric vertices of a parametric polytope and is essential for the computation in `barvinok` of the number of elements in a polyhedral set. The algorithm for the actual vertex enumeration is essentially that of [19], but the corresponding chamber decomposition is implemented much more efficiently. Preliminary experiments on a couple of non-trivial cases show that the implementation is orders of magnitude faster than that of `PolyLib` and as fast as or slightly faster than TOPCOM [21] (version 0.16.2).
- *bounds on piecewise step-polynomials* are computed in an approximative, but usually fairly accurate, way using the algorithm of [8]. Step-polynomials are polynomial expressions in greatest integer parts of affine expressions and appear as the result of (weighted) counting problems over polyhedral sets.

References

1. Bagnara, R., Ricci, E., Zaffanella, E., Hill, P.M.: Possibly not closed convex polyhedra and the Parma Polyhedra Library. In: Hermenegildo, M.V., Puebla, G. (eds.) SAS 2002. LNCS, vol. 2477, pp. 213–229. Springer, Heidelberg (2002)
2. Bagnara, R., Hill, P., Zaffanella, E.: Exact join detection for convex polyhedra and other numerical abstractions. Comput. Geom. Theory Appl. 43(5), 453–473 (2010)
3. Bastoul, C.: Code generation in the polyhedral model is easier than you think. In: PACT 2004, pp. 7–16. IEEE Computer Society, Los Alamitos (2004)
4. Bauer, C., Frink, A., Kreckel, R.: Introduction to the GiNaC framework for symbolic computation within the C++ programming language. J. Symb. Comput. 33(1), 1–12 (2002)
5. Beletska, A., Barthou, D., Bielecki, W., Cohen, A.: Computing the transitive closure of a union of affine integer tuple relations. In: COCOA 2009, pp. 98–109. Springer, Heidelberg (2009)
6. Bemporad, A., Fukuda, K., Torrisi, F.D.: Convexity recognition of the union of polyhedra. Comput. Geom. 18(3), 141–154 (2001)
7. Chen, C.: Omega+ library (2009), http://www.cs.utah.edu/~chunchen/omega/
8. Clauss, P., Fernandez, F.J., Gabervetsky, D., Verdoolaege, S.: Symbolic polynomial maximization over convex sets and its application to memory requirement estimation. IEEE Transactions on VLSI Systems 17(8), 983–996 (2009)
9. Cook, W., Rutherford, T., Scarf, H.E., Shallcross, D.F.: An implementation of the generalized basis reduction algorithm for integer programming. Cowles Foundation Discussion Papers 990, Cowles Foundation, Yale University (August 1991)
10. Detlefs, D., Nelson, G., Saxe, J.B.: Simplify: a theorem prover for program checking. J. ACM 52(3), 365–473 (2005)
11. Feautrier, P.: Parametric integer programming. Operationnelle/Operations Research 22(3), 243–268 (1988)
12. Feautrier, P.: Dataflow analysis of array and scalar references. International Journal of Parallel Programming 20(1), 23–53 (1991)
13. Free Software Foundation, Inc.: GMP, available from ftp://ftp.gnu.org/gnu/gmp
14. Fukuda, K., Liebling, T.M., Lütolf, C.: Extended convex hull. In: Proceedings of the 12th Canadian Conference on Computational Geometry, pp. 57–63 (2000)
15. Karr, M.: Affine relationships among variables of a program. Acta Informatica 6, 133–151 (1976)
16. Kelly, W., Maslov, V., Pugh, W., Rosser, E., Shpeisman, T., Wonnacott, D.: The Omega library. Tech. rep., University of Maryland (November 1996)
17. Kelly, W., Pugh, W., Rosser, E., Shpeisman, T.: Transitive closure of infinite graphs and its applications. Int. J. Parallel Program. 24(6), 579–598 (1996)
18. Loechner, V.: PolyLib: A library for manipulating parameterized polyhedra. Tech. rep., ICPS, Université Louis Pasteur de Strasbourg, France (March 1999)
19. Loechner, V., Wilde, D.K.: Parameterized polyhedra and their vertices. International Journal of Parallel Programming 25(6), 525–549 (1997)
20. Pugh, W.: The Omega test: a fast and practical integer programming algorithm for dependence analysis. Communications of the ACM 8, 102–114 (1992)
21. Rambau, J.: TOPCOM: Triangulations of point configurations and oriented matroids. In: Cohen, A.M., Gao, X.S., Takayama, N. (eds.) ICMS 2002, pp. 330–340 (2002)
22. Verdoolaege, S., Janssens, G., Bruynooghe, M.: Equivalence checking of static affine programs using widening to handle recurrences. In: Bouajjani, A., Maler, O. (eds.) CAV 2009. LNCS, vol. 5643, pp. 599–613. Springer, Heidelberg (2009)

The Reformulation-Optimization Software Engine*

Leo Liberti[1,**], Sonia Cafieri[2], and David Savourey[1]

[1] LIX, École Polytechnique, Palaiseau, France
{liberti,savourey}@lix.polytechnique.fr
[2] Dept. Mathématiques et Informatique, ENAC, 7 av. E. Belin, 31055 Toulouse, France
sonia.cafieri@enac.fr

Abstract. Most optimization software performs *numerical* computation, in the sense that the main interest is to find numerical values to assign to the decision variables, e.g. a solution to an optimization problem. In mathematical programming, however, a considerable amount of *symbolic* transformation is essential to solving difficult optimization problems, e.g. relaxation or decomposition techniques. This step is usually carried out by hand, involves human ingenuity, and often constitutes the "theoretical contribution" of some research papers. We describe a Reformulation-Optimization Software Engine (ROSE) for performing (automatic) symbolic computation on mathematical programming formulations.

Keywords: reformulation, MINLP.

1 Introduction

The aim of this paper is to describe a new optimization software called Reformulation-Optimization Software Engine (ROSE). Its main purpose is to allow the symbolic analysis and reformulation of Mathematical Programs (MP), although ROSE can also interface with numerical solvers. In practice, ROSE is used either as a pre-processor or is called iteratively within numerical solvers; it can be used either stand-alone or as an AMPL [1] solver. ROSE addresses MPs in the following very general form:

$$\left.\begin{array}{rcl} \min f(x) & & \\ g^L \leq & g(x) & \leq g^U \\ x^L \leq & x & \leq x^U \\ \forall i \in Z & x_i & \in \mathbb{Z}, \end{array}\right\} \tag{1}$$

where x is a vector of n decision variables, $x^L, x^U \in \mathbb{R}^n$, $Z \subseteq \{1, \ldots, n\}$, $g^L, g^U \in \mathbb{R}^m$, $f : \mathbb{R}^n \to \mathbb{R}$ and $g : \mathbb{R}^n \to \mathbb{R}^m$. MPs in the form (1) are known as Mixed-Integer Nonlinear Programs (MINLP). The restriction on f, g is that they should

* Supported by grants: ANR 07-JCJC-0151 "ARS", Digiteo 2009-14D "RMNCCO", Digiteo 2009-55D "ARM". We acknowledge the contributions of Dr. C. D'Ambrosio (University of Bologna) and of Mr. P. Janes (Australian National University).

** Corresponding author.

K. Fukuda et al. (Eds.): ICMS 2010, LNCS 6327, pp. 303–314, 2010.

be representable as strings of a certain formal language (more details in Sect. 2 below).

Changing the formal description of optimization problems has an impact on the applicability and efficiency of the corresponding solution methods. Difficult problems are routinely decomposed, relaxed or transformed into simpler subproblems that we know how to solve efficiently, and which preserve some of the interesting mathematical properties of the original problem. Such transformations, called reformulations, almost always depend on the "mathematical structure" of the problem. Considering MP as a formal language, each formulation is a valid sentence in the language. A reformulation is a sequence of some basic symbolic transformations (such as add, modify or delete a variable, an objective or a constraint). In order to be useful, a reformulation must preserve some mathematical property: for example, all optima of the reformulation might be required to be also optima of the original formulation. Since the basic "atomic" reformulations (adding, modifying, deleting a formulation element) are in principle easy to conceive and implement, the absence of a generic software package for carrying out automatic reformulations in MP might come as a surprise. ROSE moves a few steps in this direction, providing a set of *reformulators* that can act on MP formulations. The roadmap for ROSE is to facilitate the implementation of a heuristically driven search for the best reformulation for a given solver.

ROSE consists of around 50Klines of GNU C++ code and is covered by the Common Public License (CPL). We are currently preparing its distribution through COIN-OR [2] and finalizing documentation and examples. At the moment the software can be obtained through `http://www.lix.polytechnique.fr/~liberti/rose.tar.gz`. This paper announces the first public distribution of ROSE, which provides symbolic (as opposed to numerical) methods for manipulating MPs. Currently, ROSE can perform basic and complex symbolic analysis and manipulation tasks on all formulation elements, including all expressions appearing in objective(s) and constraints in (1). These tasks have been put together in higher-level reformulation solvers, e.g. writing a (linear) convex relaxation of a MINLP automatically [3,4]; writing a DAG representation of an AMPL-encoded MINLP [5,4]; writing a `cdd` [6] or PORTA [7] representation of an AMPL-encoded LP. A list of applications of ROSE is given in Sect. 5.

The rest of this paper is organized as follows. We review existing work in Sect. 1.1, give two motivating examples in Sect. 1.2, survey the theory that ROSE is built on in Sect. 2, explain ROSE's architecture in Sect. 3, show how ROSE helps solving the motivating examples in Sect. 4 and discuss ROSE's main applications in Sect. 5, which concludes the paper.

1.1 Existing Work

Currently, optimization software focuses on *solvers* (implementations of solution algorithms), each of which includes the necessary layers of reformulation capabilities. For example, all spatial Branch-and-Bound (sBB) MINLP solvers are able to construct a convex relaxation automatically [8,4].

Solvers typically require their input in a non-quantified format: complex jagged arrays of variables and constraints must be transformed into flat lists thereof. This creates the need for "translators" that automatically convert quantified constraints to flat constraints. For example, $\forall i \in \{1,2,3\} \sum_{j \neq i} x_i = 1$ is converted to the flat form $x_1 + x_2 = 1 \wedge x_1 + x_3 = 1 \wedge x_2 + x_3 = 1$. Since different solvers read the flat form input according to different encodings, translators also include wrappers for most existing solvers, so that users can safely ignore the technicalities of the calling procedure. The two best known MP translators are AMPL and GAMS [9]: both optionally perform reformulations on the input MP before "flattening" it and passing it to the solver.

In general, the reformulation layers of existing solvers and translators cannot be accessed or modified by the user. Apart from ROSE we are aware of no user-accessible software for carrying out MP reformulations with such generality. Notwithstanding, at least two codes are available that perform symbolic analysis and reformulation to a certain extent. Dr. AMPL [10] is an analysis tool for MP formulations aimed to the automatic choice of an appropriate solver for the given formulation. The software described in [11] enriches the AMPL language with primitives for providing solvers with specific block-diagonal information about the problem.

1.2 Motivating Examples

The need for a generic reformulation software layer is given by the mounting complexity of optimization software needed to solve ever more difficult problems.

Subproblems in sBB. In sBB, for example, a branching procedure constructs a search tree, each node of which represents a pair of reformulations $(Q, \bar{Q})$ of the original problem P. The formulation Q is obtained from P by restricting the variable bounds; $\bar{Q}$ is a relaxation of Q where each nonlinear term is replaced by linear lower and upper bounding functions. A possible sBB implementation might wish to solve Q using a MINLP heuristic and $\bar{Q}$ using a MILP solver. In turn, the MINLP heuristic might alternate between solving a continuous Nonlinear Programming (NLP) reformulation and an auxiliary MILP reformulation of Q, whereas the MILP solver is a standard BB algorithm which needs to call MILP heuristics and a Linear Programming (LP) solver such as the simplex algorithm. Testing new ideas in this complex calling chain often requires changing the reformulation algorithms, which is impossible as long as these are hard-coded into the solver.

The Kissing Number Problem. Given positive integers D, $\bar{N}$, the KNP [12] asks for the maximum number (between 1 and $\bar{N}$) of spheres of unit radius that can be arranged in $\mathbb{R}^D$ around a unit sphere centered in the origin so that their interiors are disjoint. The MP formulation [13] is:

$$\left.\begin{array}{rl} \max & \sum_{i \leq \bar{N}} y_i \\ \forall i \leq \bar{N} & \|x_i\|^2 = 4y_i \\ \forall i < j \leq \bar{N} & \|x_i - x_j\|^2 \geq 4y_i y_j \\ \forall i \leq \bar{N} & x_i \in \mathbb{R}^D, \quad y_i \in \{0,1\}. \end{array}\right\} \tag{2}$$

Attempting to solve (2) directly with a MINLP solver such as BARON [8] or COUENNE [4] results in the trivial solution with $y = 0$ standing for incumbent (i.e. best optimum so far) for several days of computation as soon as $D \geqslant 3$ and $\bar{N} \geqslant 13$. We dispense with binary variables by transforming (2) into the corresponding decision problem: can $N \leq \bar{N}$ spheres be arranged around the central one? The MP formulation is:

$$\forall i \leq N \; \|x_i\|^2 = 4 \quad \wedge \quad \forall i < j \leq N \; \|x_i - x_j\|^2 \geq 4. \tag{3}$$

If (3) has a solution, then the instance (D, N) is a YES one. Since both BARON and COUENNE identify a feasible solution by calling a local NLP subsolver (e.g. SNOPT [14]), both are only as reliable as the subsolver. Computational experience shows that most local NLP solvers have difficulties in finding a local optimum of a heavily nonlinear MP if no feasible starting point is supplied. Again, days of computation will not yield any solution even for small instances. Inserting a tolerance to feasibility improves this situation:

$$\max_{x,\alpha \in [0,1]} \alpha \qquad \text{s.t. } \forall i \leq N \; \|x_i\|^2 = 4 \; \wedge \; \forall i < j \leq N \; \|x_i - x_j\|^2 \geq 4\alpha. \tag{4}$$

As shown in [12], (4) is computationally amenable to local NLP solution within a heuristic Global Optimization (GO) solver such as Variable Neighbourhood Search (VNS). Because of the large number of symmetric optima, however, sBB solvers are still far from finding any nontrivial solution. A study of the formulation group of (4) suggests adjoining the symmetry breaking constraints $\forall i \leq N \smallsetminus \{1\} \; x_{i-1,1} \leq x_{i1}$ to (4), yielding a reformulation for which sBB makes considerably more progress [15]. Identifying this reformulation chain, which leads to a more easily solvable MP, required considerable effort and resources. ROSE alleviates the situation by providing a uniform C++ interface to several reformulation needs. It is interesting to remark that other types of reformulations were recently instrumental in solving some high dimensional KNP instances [16].

2 Reformulations: Formal Definitions

We define MPs as valid sentences of a certain formal language. Instead of giving its syntax, i.e. the explicit grammar of this language (see the Appendix to [1] for an example), we describe the image of its semantic function, i.e. the data structure needed to encode a MP.

A *parameter* is a real number p (in its floating point computer representation). A *decision variable* is a symbol x_i indexed by some positive integer i. Consider a finite set O of operators $\{\oplus_1, \oplus_2, \ldots\}$ of given arities. An *expression* is defined recursively as follows:

1. parameters are expressions;
2. decision variables are expressions;
3. if $e_1, \ldots, e_k$ are expressions and $\oplus \in O$ has arity k, then $\oplus(e_1, \ldots, e_k)$ is an expression.

Let E be the set of all such expressions. We remark that each expression $e(p, x) \in E$ involving parameters $p = (p_1, \ldots, p_t)$ and decision variables $x = (x_1, \ldots, x_n)$

corresponds to a function $f_e(p,x)$, which associates to x the evaluation of $e(p,\cdot)$ at x. An *objective function* is a pair $(d,e) \in \{-1,1\} \times E$ where d is the *optimization direction*: $(-1,e(p,x))$ corresponds to $\min f_e(p,x)$ and $(1,e(p,x))$ to $\max f_e(p,x)$. A *constraint* is a triplet $(g^L,e,g^U) \in \mathbb{R} \times E \times \mathbb{R}$ encoding the double inequality $g^L \leq f_e(p,x) \leq g^U$. A *range constraint* is a triplet $(x_i^L,x_i,x_i^U) \in \mathbb{R} \times E \times \mathbb{R}$ encoding the restriction $x_i^L \leq x_i \leq x_i^U$. An *integrality constraint* is a positive integer i which encodes the restriction $x_i \in \mathbb{Z}$. A *mathematical program* is a 7-tuple $(p,x,E,\mathcal{O},\mathcal{C},\mathcal{B},Z)$ such that for all $e \in E$, e depends on no further parameters (resp. decision variables) than p (resp. x), $\mathcal{O}$ is a set of s objective functions (d,e) with $e \in E$, $\mathcal{C}$ is a set of m constraints (g^L,e,g^U) with $e \in E$, $\mathcal{B}$ is a set of n range constraints, and $Z \subseteq \{1,\ldots,n\}$ is a set of integrality constraints. An element of any component set in the 7-tuple is also called an *entity* of the MP. Semidefinite and multilevel programming can be dealt with by letting constant and/or variables symbols range over sets of matrices or other mathematical programs.

2.1 Flat and Structured MPs

MPs can be given either in structured form (i.e. by using quantifiers over indices) or flat form. *Flat MPs* are those corresponding to the definition of Sect. 2. We now define structured MPs.

Given a sequence $\mathcal{I} = \{I_i \subseteq \mathbb{N} \mid i \leq \alpha\}$ of finite subsets of integers and a multi-index $\mathbf{i} = (i_1,\ldots,i_\alpha)$ where $i_\beta \in I_\beta$ for all $\beta \leq \alpha$, a structured parameter p is a jagged array of (scalar) parameter symbols $p_{\mathbf{i}}$ (with $\mathbf{i} \in \mathcal{I}$) with an assigned (scalar) value $\mathsf{p}_{\mathbf{i}}$. A structured decision variable x is defined similarly for scalar variable symbols $x_{\mathbf{i}}$. Structured expressions, resting on an operator set O' enriched with the quantifier operators $\sum, \prod$, are defined recursively similarly to flat expressions, but with parameters and variables replaced by their structured versions. A structured constraint is a triplet $(g^L_{\mathbf{ij}}, f_e(p_{\mathbf{i}},x_{\mathbf{j}}), g^U_{\mathbf{ij}})$ where all multi-indices $\mathbf{i}$, $\mathbf{j}$ are universally quantified over some subsets of $\mathcal{I}$. Structured range and integrality constraints are defined similarly. A MP defined over structured entities is a *structured* MP. Given a structured MP P with multi-indices $\mathbf{i}_1,\ldots,\mathbf{i}_\gamma$ ranging over set families $\mathbf{I} = \{\mathcal{I}_1,\ldots,\mathcal{I}_\gamma\}$, and the jagged array of values $\mathbf{p}$ to be assigned to all parameter symbols, a translator (such as AMPL or GAMS) is able to write a flat MP P corresponding to the triplet $(\mathsf{P},\mathbf{I},\mathbf{p})$. In general, an operator $\oplus \in O'$ acts on structured entities in a componentwise fashion. Different operator semantics can be defined by simply adding new operators to O'. In the terminology of complexity analysis, flat MPs correspond to *instances* and structured MPs to *problems* defined as instance sets, each instance being given by the pair $(\mathbf{I},\mathbf{p})$.

2.2 Flat Reformulations

Reformulations may occur either at the flat or structured level. Because of a technical limitation of AMPL (i.e. the AMPL API only allows user access to the flat, rather than structured, MP), ROSE only performs flat reformulations; we

therefore only define these. Structured reformulations would essentially require hooking reformulation primitives at the AMPL grammar parsing level.

Let $\mathbb{MP}_F$ be the class of all flat MPs; for $P \in \mathbb{MP}_F$ we denote the feasible region of P by $\mathcal{F}(P)$, the set of local optima of P by $\mathcal{L}(P)$ and the set of global optima of P by $\mathcal{G}(P)$.

Definition 2.1
A flat reformulation is a relation $\hookrightarrow$ *on* $\mathbb{MP}_F$ *such that there exists a formula* ψ *with two free variables for which*

$$\forall P, Q \in \mathbb{MP}_F \quad (P \hookrightarrow Q \;\Rightarrow\; \psi(P, Q)). \tag{5}$$

The invariance scope of $\hookrightarrow$ *is the class* $\mathbb{S}(\hookrightarrow)$ *of all* ψ *for which* (5) *holds.*

We distinguish three remarkable types of reformulations.

1. *Exact reformulations*, denoted by $\equiv$: $\mathbb{S}(\equiv)$ contains the formula "there is a function $\varphi : \mathcal{F}(Q) \to \mathcal{F}(P)$ such that $\varphi|_{\mathcal{L}(Q)}$ is onto $\mathcal{L}(P)$ and $\varphi|_{\mathcal{G}(Q)}$ is onto $\mathcal{G}(P)$";
2. *Narrowings*, denoted by $\triangleright$: $\mathbb{S}(\triangleright)$ contains the formula "there is a function $\varphi : \mathcal{F}(Q) \to \mathcal{F}(P)$ such that $\varphi(\mathcal{G}(Q)) \subseteq \mathcal{G}(P)$";
3. *Relaxations*, denoted by $\geq$: $\mathbb{S}(\geq)$ contains the formula "$\mathcal{F}(Q) \supseteq \mathcal{F}(P)$ and $\mathcal{O}(Q) = \{(-1, e')\}$ and $\mathcal{O}(P) = \{(-1, e)\}$ and, for all $x \in \mathcal{F}(P)$, $f_{e'}(p, x) \leq f_e(p, x)$".

Theorem 2.2 ([17])
The relations $\equiv, \triangleright, \geq$ *are all transitive. Furthermore,* $\equiv \subseteq \triangleright$ *and* $\equiv \subseteq \geq$.

Thus, if $P \equiv Q_1 \triangleright Q_2$ then $P \triangleright Q_2$; if $P \equiv Q_1 \geq Q_2$ then $P \geq Q_2$. This allows the construction of reformulation chains with invariant properties. We only consider reformulations corresponding to computable relations. A taxonomy of useful flat reformulations is given in [18].

Example 2.3
The ProdBinCont *exact reformulation [18] replaces every product* xy *where* $x \in \{0, 1\}$ *and* $y \in [y^L, y^U]$ *with an added variable* w*, which is constrained by the natural extension of Fortet's inequalities [19]:* $w \leq y^U x$, $w \geq y^L x$, $w \leq y - (1 - x)y^L$, $w \geq y - (1 - x)y^U$.

3 ROSE Architecture

ROSE consists of a simple modular architecture based on two main classes (`Problem` and `Solver`) and a separate library (Ev3) for storing and manipulating expressions in E. The overall architecture is depicted graphically in Fig. 1. More detailed information about ROSE's and Ev3's architecture, capabilities and Application Programming Interface (API) can be found in [18], Sect. 5.2-5.3.

The `Problem` class contains lists of `Variable`, `Objective` and `Constraint` structures. Structures of `Variable` type include information about decision variables

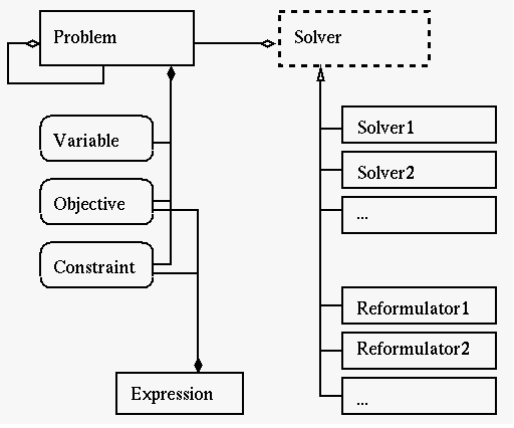


Fig. 1. ROSE architecture. Rectangles indicate `class`es (with dashed meaning virtual), rounded boxes indicates `struct`s, relation links conform to UML: void diamonds indicate aggregation (to maintain a reference of), filled diamonds indicate composition (to maintain a copy of), triangles indicate inheritance.

such as index, current and optimal value, and range and integrality constraints; `Objective`s include information about objective functions such as index, current and optimal value, corresponding expression and optimization direction; `Constraint`s include information about constraints such as index, current and optimal value, corresponding expression and bound restrictions. The `Problem` class also stores information about problem cardinalities, feasibility of a current solution with respect to the constraints, a reference to a previous `Problem` object in a reformulation chain, and other useful information. It has methods for accessing data, adding or deleting decision variables, objective functions and constraints, evaluating expressions appearing in objectives or constraints, parse a given input file (a description of an MP) into its data structures, and so on.

The `Solver` class is a virtual class whose implementations are either numerical solvers or reformulators; the latter are recognizable because their names are prefixed by `R-` (e.g. `Rprodbincont`). All `Solver` objects maintain: a pointer to the `Problem` object being solved, numerical information about current and optimal points, information about linear and nonlinear cuts and a few other items mainly used by numerical solvers. Reformulators are allowed to change the `Problem` they reference; problems can be duplicated before being changed by a reformulator by using the special `Rcopy` reformulator. Basic reformulation steps for adding or deleting problem entities are implemented in `Problem`; modification of expressions occurs via interfacing with the Ev3 expression library (Sect. 3.3). Many methods in `Problem`s and `Solver`s can be configured by user-defined parameters that passed to `Problem` and `Solver` objects via a unique object of the class `ParameterBlob`.

3.1 MP Input

ROSE can read an MP via either its own intuitive flat MP format (see [3] p. 238) or via interfacing with the AMPL interpreter [1]. Each MP entity is assigned two integer scalar indices: a unique entity ID (which is preserved across reformulations) and a local index (which is an ordinal running from 1 to the number of entities of a given type within a `Problem` object). Methods are provided for switching from ID to local indices.

3.2 MP Output

Since the AMPL API does not offer primitives for modifying the current MP, the only possibility for ROSE is to output its reformulations to a flat MP written to an AMPL formatted file. Users can then instruct AMPL to read this file. This situation is far from optimal, as it requires hard disk access, but there is no way around it — according to the AMPL authors, it is unlikely that AMPL will ever have an API which is sufficiently flexible as to allow modification of the internal data. Developers can also choose to have individual reformulators write their output to whatever syntax they wish, bypassing the default output.

3.3 Expression Tree Library

Following the recursive definition of expressions given in Sect. 2, an expression $e \in E$ is encoded in a tree data structure $T_e = (V_e, A_e)$: leaf nodes of V_e are labelled by parameters in p and by decision variables in x, and intermediate nodes are labelled by operators in $\oplus \in O$. A k-ary operator node has k subnodes in its star. An arc (u, v) is in A_e if v is a subnode of u.

An `Expression` is synonym to a `Pointer<BasicExpression>` template class. The `Pointer<T>` class is used to perform automatic memory management (i.e. automatic deallocation) from a `node` of type `T`. A `BasicExpression` inherits from `Operand` and from `Tree<BasicExpression>`. The `Operand` class simply includes information concerning a particular node (whether leaf or nonleaf, operator label, variable index, parameter value, and so on). The `Tree<T>` class includes a list of nodes of type `Pointer<T>`, and is used to represent a list of subnodes of a given node; it has methods for accessing and editing nodes. This complex architecture for expression trees makes it easy to edit, move or copy entire subtrees recursively, but floating point evaluation of the expression is slow. To circumvent this problem, expressions are also encoded in much simpler C-style tree structures (called `FastEvalTree`s) without any memory management in order to speed up evaluation. Their activation and use is completely transparent to the user.

The Ev3 library capabilities include simplification of expressions, reduction to (partial) normal form, identification of subexpressions of certain structures, conditional editing of subexpressions, recognition and separation of linear and nonlinear parts in a given expression, symbolic differentiation and many others. Since they act on trees, most methods are recursive, and consist of two functions: the "recursion start" and the "recursion step". In the case of Example 2.3, the ProdBinCont reformulator is implemented according to the pseudocode below.

```
ProdBinCont(Expression e) {
  ProdBinContRecursive(e.root);
}
ProdBinContRecursive(Expression e) {
  if (!e.leaf) {
    for(v in e.subnodes) {
      ProdBinContRecursive(v);
    }
  }
  if (e.structure == x*y && x.binary && !y.binary) {
    AddVariable(w);
    ReplaceBy(e,w);
    AdjoinConstraint(Fortet's extension inequalities);
  }
}
```

4 How ROSE Helps Solving the Motivating Examples

Subproblems in sBB. ROSE can construct a convex relaxation of (1) automatically from its Smith reformulation [20], which isolates all the nonlinearities of the problem in constraints with a simple structure; these are then replaced by appropriate convex relaxations [3]. The ROSE `Rsmith` reformulator (tasked with constructing the Smith reformulation) calls a recursive Ev3 procedure which looks for subtrees of an `Expression` e having a certain "shape" in order to replace them with an added variable w; the constraint $w = e$ is then added to the formulation. The shape of an expression is defined as an expression *schema*, i.e. an expression tree search pattern whose variable nodes are labelled by a wildcard "?" with the meaning of "any variable". Thus, for example, the tree $? \leftarrow\times\rightarrow ?$ represents a generic product of two variables, and it matches every tree $x_i \leftarrow\times\rightarrow x_j$ (for any i, j).

```
Rsmith(Problem p) {
  for (f in {p.objective, p.constraints}) {
    if (!f.linear) {
      SmithStandardForm(f);
    }
  }
}

SmithStandardForm(Expression e, vector<Expression> schemata) {
  if (!e.leaf) {
    for (v in e.subnodes) {
      SmithStandarForm(v);
    }
  }
  for (s in schemata) {
    if (e.MatchesSchema(s)) {
      AddVariable(w);
      ReplaceBy(e,w);
      AdjoinConstraint(w = e);
    }
  }
}
```

The pseudocode above shows the essential functionality of the `Rsmith` reformulator and the corresponding Ev3 recursive auxiliary function. ROSE has several relaxation reformulators (e.g. `Rconvexifier`, `RQuarticConvex`) which are chained to the `Rsmith` reformulator as per Thm. 2.2.

KNP. Solving the KNP formulation (2) requires several reformulations:

1. derive a restriction of (2) to certain neighbourhoods (in order to solve the KNP using heuristics);
2. convert the optimization problem to the corresponding decision problem for a given objective function value (3)
3. relax some constraints by means of a multiplicative tolerance;
4. adjoin an objective that maximizes the tolerance (4);
5. derive a symmetry-free narrowing and a convex relaxation of (4) (in order to solve via sBB).

The two heuristics tested on (2) are VNS and the MINLP Feasibility Pump (FP-MINLP) [21]. Both rely on certain subproblems of (2) obtained by adjoining appropriate constraints: VNS requires Local Branching type constraint [22], whilst FPMINLP requires a specific outer approximation. The reformulations listed in 2-4. above can be implemented using the basic reformulations encoded in ROSE. The symmetry-free narrowing in 5. is obtained automatically using a chain of software packages (i.e. AMPL, ROSE, `nauty` [23], GAP [24]) held together via Unix scripts. In particular, ROSE is used to analyze an AMPL flat MP and produce its Directed Acyclic Graph (DAG) encoding [4,5], which is then fed into `nauty` in order to derive its symmetry group. ROSE can also obtain a convex relaxation of (4) based on the ideas given in [20,3,4]. Computational results for sBB on (4) are reported in [15].

It may be noted that the above examples were chosen arbitrarily by a large set of ROSE applications (see Sect. 5). We believe that the first example shows how ROSE can be useful *per se*, whereas the second demonstrates ROSE's ability to interface with other tools in order to perform complex reformulating tasks.

5 ROSE's Existing Applications

ROSE's current role is to help automatize flat MP reformulations which would be too long to perform by hand, but which are necessary to implement and test research ideas. ROSE was and is instrumental to several past and current research projects: in some cases it is key to their success, in other cases it allows research teams to quickly weed out bad ideas; it is sometimes influential to other software, in that ideas found in ROSE are re-implemented (for practical reasons) in other codes.

- Fundamentals of reformulation theory [17,18], where ROSE served as a proof of concept (successful).
- Experiments on spherical cuts for Binary Linear Programs (BLPs) [25] (successful).
- An investigation of the convex relaxation of quadrilinear terms [26,27], where ROSE was used both to produce the convex relaxation and to automatically write input data to other software packages (e.g. `cdd` [6]) (successful).
- Experiments with symmetry-breaking narrowing reformulations [28,15,29,30] (successful).

- The FPMINLP heuristic [21], where ROSE is used both to analyze MINLPs (e.g. to find convex constraints), and to reformulate them, i.e. to build the Feasibility Pump subproblems (successful).
- The RECIPE MINLP heuristic [31] was implemented independently of ROSE with what was essentially ROSE code (influential).
- The conception of the Couenne [4] solver code that builds the convex relaxation was heavily influenced by ideas implemented in ROSE (influential).
- Reduced RLT-based relaxations for polynomial programs (current work, unpublished).
- A general-purpose MINLP Tabu Search heuristic based on tabu spheres (unsuccessful, unpublished).
- A general-purpose MINLP feasibility heuristic based on branching with no bounding until a feasible solution is found (unsuccessful, unpublished).

References

1. Fourer, R., Gay, D.: The AMPL Book. Duxbury Press, Pacific Grove (2002)
2. Lougee-Heimer, R.: The common optimization interface for operations research: Promoting open-source software in the operations research community. IBM Journal of Research and Development 47(1), 57–66 (2003)
3. Liberti, L.: Writing global optimization software. In: Liberti, L., Maculan, N. (eds.) Global Optimization: from Theory to Implementation, pp. 211–262. Springer, Berlin (2006)
4. Belotti, P., Lee, J., Liberti, L., Margot, F., Wächter, A.: Branching and bounds tightening techniques for non-convex MINLP. Optimization Methods and Software 24(4), 597–634 (2009)
5. Schichl, H., Neumaier, A.: Interval analysis on directed acyclic graphs for global optimization. Journal of Global Optimization 33(4), 541–562 (2005)
6. Fukuda, K., Prodon, A.: Double description method revisited. In: Deza, M., Manoussakis, I., Euler, R. (eds.) CCS 1995. LNCS, vol. 1120, pp. 91–111. Springer, Heidelberg (1996)
7. Christof, T., Löbel, A.: The porta manual page. Technical Report v. 1.4.0, ZIB, Berlin (1997)
8. Sahinidis, N., Tawarmalani, M.: BARON 7.2.5: Global Optimization of Mixed-Integer Nonlinear Programs, User's Manual (2005)
9. Brook, A., Kendrick, D., Meeraus, A.: GAMS, a user's guide. ACM SIGNUM Newsletter 23(3-4), 10–11 (1988)
10. Orban, D., Fourer, R.: Dr. AMPL: a meta solver for optimization (2004) (Presentation slides)
11. Colombo, M., Grothey, A., Hogg, J., Woodsend, K., Gondzio, J.: A structure-conveying modelling language for mathematical and stochastic programming. Mathematical Programming Computation 1(4), 223–247 (2009)
12. Kucherenko, S., Belotti, P., Liberti, L., Maculan, N.: New formulations for the kissing number problem. Discrete Applied Mathematics 155(14), 1837–1841 (2007)
13. Maculan, N., Michelon, P., MacGregor Smith, J.: Bounds on the kissing numbers in $\mathbb{R}^n$: Mathematical programming formulations. Technical report, University of Massachusetts, Amherst, USA (1996)
14. Gill, P.: User's guide for SNOPT version 7. Systems Optimization Laboratory, Stanford University, California (2006)

15. Liberti, L.: Symmetry in mathematical programming. In: Lee, J., Leyffer, S. (eds.) Mixed Integer Nonlinear Programming, vol. IMA, Springer, New York (accepted)
16. Bachoc, C., Vallentin, F.: New upper bounds for kissing numbers from semidefinite programming. Journal of the American Mathematical Society 21, 909–924 (2008)
17. Liberti, L.: Reformulations in mathematical programming: Definitions and systematics. RAIRO-RO 43(1), 55–86 (2009)
18. Liberti, L., Cafieri, S., Tarissan, F.: Reformulations in mathematical programming: A computational approach. In: Abraham, A., Hassanien, A.E., Siarry, P., Engelbrecht, A. (eds.) Foundations of Computational Intelligence. SCI, vol. 3, 203, pp. 153–234. Springer, Berlin (2009)
19. Fortet, R.: Applications de l'algèbre de Boole en recherche opérationelle. Revue Française de Recherche Opérationelle 4, 17–26 (1960)
20. Smith, E., Pantelides, C.: A symbolic reformulation/spatial branch-and-bound algorithm for the global optimisation of nonconvex MINLPs. Computers & Chemical Engineering 23, 457–478 (1999)
21. D'Ambrosio, C., Frangioni, A., Liberti, L., Lodi, A.: Experiments with a feasibility pump approach for nonconvex MINLPs. In: Festa, P. (ed.) Experimental Algorithms. LNCS, vol. 6049, pp. 350–360. Springer, Heidelberg (2010)
22. Fischetti, M., Lodi, A.: Local branching. Mathematical Programming 98, 23–37 (2005)
23. McKay, B.: Nauty User's Guide (Version 2.4). Computer Science Dept., Australian National University (2007)
24. The GAP Group: GAP – Groups, Algorithms, and Programming, Version 4.4.10 (2007)
25. Liberti, L.: Spherical cuts for integer programming problems. International Transactions in Operational Research 15, 283–294 (2008)
26. Cafieri, S., Lee, J., Liberti, L.: Comparison of convex relaxations of quadrilinear terms. In: Ma, C., Yu, L., Zhang, D., Zhou, Z. (eds.) Global Optimization: Theory, Methods and Applications I. Lecture Notes in Decision Sciences, vol. 12(B), pp. 999–1005. Global-Link Publishers, Hong Kong (2009)
27. Cafieri, S., Lee, J., Liberti, L.: On convex relaxations of quadrilinear terms. Journal of Global Optimization, doi: 10.1007/s10898-009-9484-1
28. Liberti, L.: Reformulations in mathematical programming: Automatic symmetry detection and exploitation. Mathematical Programming, doi: 10.1007/s10107-010-0351-0
29. Costa, A., Hansen, P., Liberti, L.: Formulation symmetries in circle packing. In: Mahjoub, R. (ed.) Proceedings of the International Symposium on Combinatorial Optimization. Electronic Notes in Discrete Mathematics. Elsevier, Amsterdam (accepted)
30. Costa, A., Hansen, P., Liberti, L.: Static symmetry breaking in circle packing. In: Faigle, U. (ed.) Proceedings of the 8th Cologne-Twente Workshop on Graphs and Combinatorial Optimization, University of Köln (2010)
31. Liberti, L., Mladenović, N., Nannicini, G.: A good recipe for solving MINLPs. In: Maniezzo, V., Stützle, T., Voß, S. (eds.) Hybridizing metaheuristics and mathematical programming. Annals of Information Systems, vol. 10, pp. 231–244. Springer, New York (2009)

Generating Smooth Lattice Polytopes

Christian Haase, Benjamin Lorenz*, and Andreas Paffenholz

Freie Universität Berlin, Arnimallee 3,
14195 Berlin, Germany
{chaase,benmuell,paffenholz}@math.fu-berlin.de
http://ehrhart.math.fu-berlin.de

Abstract. A lattice polytope P is the convex hull of finitely many lattice points in $\mathbb{Z}^d$. It is smooth if each cone in the normal fan is unimodular. It has recently been shown that in fixed dimension the number of lattice equivalence classes of smooth lattice polytopes in dimension d with at most N lattice points is finite. We describe an algorithm to compute a representative in each equivalence class, and report on results in dimension 2 and 3 for $N \leq 12$. Our algorithm is implemented as an extension to the software system `polymake`.

Keywords: lattice polytopes, smooth polytopes, classification, `polymake`.

1 Introduction

A lattice polytope $P \subset \mathbb{R}^d$ is the convex hull of finitely many points in the integer lattice $\mathbb{Z}^d$. Lattice polytopes have attracted increasing interest in recent years among researchers in various fields. Algebraic geometers are interested in the properties of the associated semi-group ring $\mathcal{R}_P := \mathbb{C}[C_P \cap \mathbb{Z}^{d+1}]$, where C_P is the cone over $\{1\} \times P$, and in the projective toric variety $X_P := \operatorname{Proj} \mathcal{R}_P$ [6]. In statistics, they appear as a tool to construct Markov chains for contingency tables [8]. In integer programming, an approach using Gröbner bases of toric ideals is sometimes superior to standard cutting planes techniques [1].

Although combinatorial and algebraic properties of lattice polytopes have been studied extensively, many quite basic questions are still wide open even in low dimensions. A fundamental obstacle to theoretical progress is the lack of sufficiently many, sufficiently generic examples. Some particularly interesting questions concern smooth lattice polytopes. A lattice polytope P is *smooth* if X_P is a smooth variety. In combinatorial terms, P is smooth if all maximal cones in the normal fan of P are unimodular, i.e. if the generators of each maximal cone are a lattice basis. Recently, it has been shown that for fixed dimension d and a bound N on the number of lattice points there are only finitely many lattice equivalence classes of smooth lattice polytopes [5]. Here, two lattice polytopes

* Christian Haase and Benjamin Lorenz are supported by an Emmy-Noether grant, No. HA 4383/1.

K. Fukuda et al. (Eds.): ICMS 2010, LNCS 6327, pp. 315–328, 2010.

are equivalent if they can be identified via a lattice preserving affine map. In this note we present an algorithm for their enumeration for given d and N. A list of small examples could be useful for several open questions about smooth lattice polytopes to test conjectures. We explain some in the following paragraphs.

Let P be a smooth lattice polytope and let P' be a lattice polytope whose normal fan is refined by the normal fan of P. A long standing open question by Oda [26] asks whether any lattice point in $P + P'$ is the sum of a lattice point in P and a lattice point in P'. The answer is affirmative in dimension 2 even if P is not smooth [12], but open in all higher dimensions.

Say that a lattice polytope P is *integrally closed* if for all $k \in \mathbb{N}$ each point $x \in kP \cap \mathbb{Z}^d$ is the sum of k points in $P \cap \mathbb{Z}^d$. Integrally closed polytopes are interesting in algebraic geometry as they define projectively normal embeddings of toric varieties. An important special case of Oda's problem is the question whether all smooth polytopes are integrally closed. This conjecture is the weakest in a whole hierarchy of successively stronger conjectures concerning smooth lattice polytopes [11] that has inspired researchers over the past years. The conjecture is known to hold in dimension 2, and recently Gubeladze [10] proved it in general for polytopes with long edges.

It has been asked whether ideals of smooth toric varieties are generated by quadratic binomials or admit a square-free quadratic initial ideal (e.g. see [4,28]). The latter question has a combinatorial interpretation. The ideal has a square-free initial ideal if and only if the associated polytope has a regular unimodular triangulation. Here, a triangulation is *regular*, if it is induced by a height function on the polytope. This is known to hold in dimension 2, and for some special classes in higher dimensions (e.g. see [13,27]). However, we don't even know an example of a smooth polytope without a unimodular triangulation.

Classifications of two other families of lattice polytopes have already lead to new and important results (e.g. see [3,22,23]). Say that a lattice polytope P is *reflexive* if the origin is in the interior of P and each facet has lattice distance one from the origin. Inspired by a question from physics, Kreuzer and Skarke gave an algorithm to generate all reflexive polytopes in fixed dimension up to lattice equivalence and computed a complete list up to dimension 4 [17,18]. Recently, Øbro [24] described an efficient algorithm to generate all lattice equivalence classes of smooth reflexive polytopes in fixed dimension, and produced a complete list up to dimension 8. Such polytopes correspond to smooth toric Fano varieties in algebraic geometry. Previously, Batyrev [2] has generated all 4-dimensional smooth reflexive lattice polytopes, and Kleinschmidt [16] those with few vertices.

Both classifications use the facet structure of a reflexive polytope in an essential way. So a generation of smooth polytopes needs a different approach. The algorithm we present has two main steps. We start by generating all smooth fans that appear as the normal fan of a smooth lattice polytope. For each such fan we enumerate in the second step all smooth lattice polytopes with this normal fan. We prove that both the set of fans and polytopes are finite if we fix the dimension and bound the number of lattice points. Our generation of smooth

fans uses a classification of *minimal* smooth fans as input. Such a classification is known in dimension 2, and in dimension 3 for up to 12 maximal cones [25].

The algorithm has been implemented by the second author as part of his MSc Thesis [20] based on the software system `polymake` [15]. We have used it to generate 2- and 3-dimensional smooth polytopes with up to 12 lattice points.

The paper is organized as follows. In the next section we review the necessary notions from combinatorial geometry and outline the algorithm. Sections 3 and 4 present the two main steps of our algorithm in detail. Section 5 gives details on the implementation and presents results for dimension 2 and 3. In the last section we discuss extensions and open questions.

2 Smooth Polytopes

A vector $\mathbf{a} \in \mathbb{Z}^d$ is *primitive* if the greatest common divisor of its entries is 1. Let Σ be a complete rational polyhedral fan with generators $\mathbf{a}_1, \ldots, \mathbf{a}_m$. By scaling each $\mathbf{a}_j$ appropriately we assume in the following that all generators are primitive. The set of k-dimensional faces of Σ is denoted with $\Sigma^{(k)}$. In particular, $\Sigma^{(d)}$ are the maximal cones and $\Sigma^{(1)}$ the rays of Σ. Any cone $\tau \in \Sigma^{(d-1)}$ is contained in precisely 2 maximal cones. The *combinatorial type* C_Σ of Σ is the poset of all cones ordered by inclusion. A cone $\sigma \in \Sigma^{(d)}$ is *unimodular* if its generators form a lattice basis of $\mathbb{Z}^d$, equivalently, their determinant is ± 1. A fan is *smooth* if all its maximal cones are unimodular.

A *lattice polytope* P is a polytope all whose vertices have integral coordinates. The *normal cone* of a face F of P is the cone generated by the facet normals of the facets intersecting in F. The *normal fan* $\Sigma(P)$ of P is the collection of all normal cones. P is *smooth* if its normal $\Sigma(P)$ is smooth.

Conversely, we can uniquely characterize a polytope P as a pair $(\Sigma, \mathbf{b})$ of its normal fan Σ and a vector $\mathbf{b} = (b_1, \ldots, b_m) \in \mathbb{Z}^m$ of an integer for each ray of Σ, the *ray parameters*. Namely,

$$P = \{\mathbf{x} \mid \langle \mathbf{a}_j, \mathbf{x} \rangle \leq b_j \text{ for } 1 \leq j \leq m\}\,.$$

A lattice equivalence of two lattice polytopes $P, P' \subseteq \mathbb{R}^d$ is an affine map $\varphi : \mathbb{R}^d \to \mathbb{R}^d$ that maps P to P' and is bijective on the lattice $\mathbb{Z}^d$. This is an equivalence relation. When we speak of a "polytope" P in the following then we implicitly mean the equivalence class of P with respect to this relation. Similarly, we consider smooth fans only up to lattice equivalence.

Next, we introduce a convenient representation of a smooth fan Σ with generators $\mathbf{a}_1, \ldots, \mathbf{a}_m$. Let $\tau \in \Sigma^{(d-1)}$ and let $\sigma, \sigma' \in \Sigma^{(d)}$ be the maximal cones intersecting in τ. Let $\mathbf{c}^\tau = (c_j^\tau) \in \{0,1\}^m$ with $c_j^\tau = 1$ if and only if $\mathbf{a}_j$ is a generator in $(\sigma \cup \sigma') \setminus \tau$. Smoothness of Σ implies that there is $\boldsymbol{\lambda}^\tau \in \mathbb{Z}^m$ with $\lambda_j^\tau = 0$ if $\mathbf{a}_j \notin \tau$ and such that

$$\textstyle\sum_{j=1}^m c_j^\tau \mathbf{a}_j = \sum_{j=1}^m \lambda_j^\tau \mathbf{a}_j \tag{2.1}$$

$\boldsymbol{\lambda}^\tau$ are called the *edge parameters* of τ. Σ is, up to lattice equivalence, completely determined by its combinatorial type and edge parameters for each $\tau \in \Sigma^{(d-1)}$.

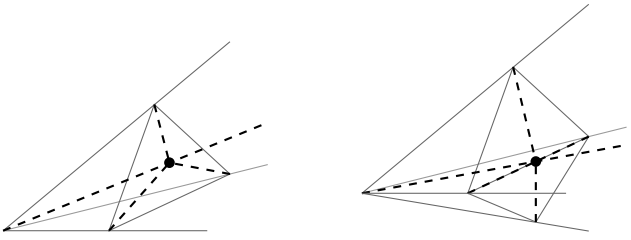


Fig. 2.1. Equivariant blowups of a smooth cone at a 2- and a 3-dimensional cone

Let $\mathcal{P}_d$ be the set of all smooth d-dimensional lattice polytopes. For a parameter $N \in \mathbb{N}$ and a smooth d-dimensional fan Σ we define

$$\begin{aligned}\mathcal{P}_\Sigma(N) &:= \{P \in \mathcal{P}_d \mid \Sigma(P) = \Sigma,\ |P \cap \mathbb{Z}^d| \leq N\}\\ \mathcal{P}_d(N) &:= \{P \in \mathcal{P}_d \mid |P \cap \mathbb{Z}^d| \leq N\}\\ \mathcal{F}_d^{\text{poly}}(N) &:= \{\Sigma \mid \Sigma \text{ is the normal fan of a polytope } P \in \mathcal{P}_d(N)\}\,.\end{aligned}$$

Up to equivalence these are finite sets by the following result.

Theorem 2.1 (Bogart et. al. 2010, [5]). *In fixed dimension there are, up to lattice equivalence, only finitely many smooth lattice polytopes with at most N lattice points.*

The original proof is constructive, but for efficiency we chose a different approach to generate $\mathcal{P}_d(N)$. Lemmas 3.1 and 4.1 below give another proof of this theorem.

Let Σ be a smooth fan and σ a cone in Σ with primitive generators $\mathbf{a}_1, \ldots, \mathbf{a}_k$. Define $\mathbf{a} := \sum \mathbf{a}_i$ and let L be the set of cones $\eta \in \Sigma$ such that $\text{cone}(\eta \cup \sigma) \in \Sigma$, but $\mathbf{a} \notin \eta$. The *equivariant blowup* of Σ at σ is

$$\Sigma_\sigma := \{\tau \in \Sigma \mid \sigma \not\subset \tau\} \cup \{\text{cone}(\tau, \mathbf{a}) \mid \tau \in L\}\,. \tag{2.2}$$

See Figure 2.1. Σ_σ is again a smooth fan. Σ is *minimal* if there is no other smooth fan Σ' such that Σ can be obtained from Σ' via equivariant blowups.

With these preparations we can give a rough outline of our algorithm to compute $\mathcal{P}_d(N)$ with the following four steps.

1. Determine all smooth minimal fans with at most N maximal cones.
2. Generate $\mathcal{F} \supset \mathcal{F}_d^{\text{poly}}(N)$ via recursive equivariant blowups.
3. For each fan $\Sigma \in \mathcal{F}_d^{\text{poly}}(N)$, compute a set $\mathcal{B}$ containing all ray parameters $\mathbf{b}$ for smooth lattice polytopes in $\mathcal{P}_\Sigma(N)$.
4. For each $\mathbf{b} \in \mathcal{B} \cap \mathbb{Z}^d$ we add $P := (\Sigma, \mathbf{b})$ to our list if $|P \cap \mathbb{Z}^d| \leq N$ and we have not already added an equivalent polytope.

The algorithm for the first two steps is explained in the next section, and the last two steps are explained in Section 4.

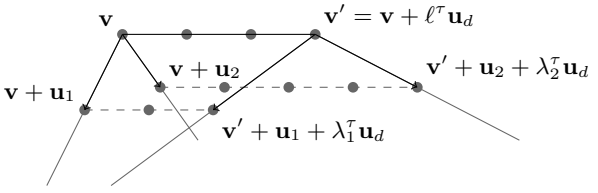


Fig. 3.1. The lattice points around an edge

3 Generation of Smooth Fans

For a given number $N \in \mathbb{N}$ there are only finitely many combinatorial types for a smooth fan with at most N maximal cones. This implies that smooth fans with at most N maximal cones come in a finite number of families that depend on the edge parameters λ_j^τ introduced in the previous section.

The combinatorial type of a blowup does not depend on the edge parameters. Hence, for our approach to generate $\mathcal{F}_d^{\text{poly}}(N)$ we don't have to consider each fan individually but can work on these families of fans and maintain a list of parameters for each. The following lemma bounds the edge parameters we have to consider for each family. In particular, $\mathcal{F}_d^{\text{poly}}(N)$ is finite.

Lemma 3.1. *Let Σ be the normal fan of a polytope in $\mathcal{P}_d(N)$ with m generators $\mathbf{a}_1, \ldots, \mathbf{a}_m$. For each cone $\tau \in \Sigma^{(d-1)}$ the edge parameters λ_i^τ, $1 \le i \le m$ satisfy*

$$\begin{aligned} \lambda_i^\tau &= 0 && \text{whenever } \mathbf{a}_i \text{ is not a generator of } \tau. \\ \lambda_i^\tau &\ge -N + d && \forall\, 1 \le i \le m \\ \textstyle\sum_{i=1}^m \lambda_i^\tau &\le N - 2d \end{aligned}$$

Proof. The first set of equations is part of the definition of edge parameters.

Let P be a smooth polytope with normal fan Σ and $\sigma, \sigma' \in \Sigma^{(d)}$ such that $\tau = \sigma \cap \sigma'$. After relabeling we can assume that $\mathbf{a}_1, \ldots, \mathbf{a}_{d-1}$ are generators of τ, and $\mathbf{a}_d, \mathbf{a}_{d+1}$ are the additional generators of σ and σ'. σ and σ' correspond to adjacent vertices $\mathbf{v}$ and $\mathbf{v}'$ of P connected by an edge e^τ of lattice length $\ell^\tau \ge 1$. P has at least $d+1$ vertices, so $\ell^\tau - 1 \le N - (d+1)$.

By assumption, $\mathbf{v}$ is in the facet defined by $\mathbf{a}_d$, so $\langle \mathbf{a}_d, \mathbf{x} \rangle \le \langle \mathbf{a}_d, \mathbf{v} \rangle$ for all $\mathbf{x} \in P$. Let $\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_d$ be the dual basis of $\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_d$. The first lattice points on the d edges adjacent to $\mathbf{v}'$ are

$$\mathbf{v}' - \mathbf{u}_d,\ \mathbf{v}' + \mathbf{u}_1 + \lambda_1^\tau \mathbf{u}_d,\ \ldots,\ \mathbf{v}' + \mathbf{u}_{d-1} + \lambda_{d-1}^\tau \mathbf{u}_d\,. \tag{3.1}$$

See Figure 3.1. Evaluating with $\mathbf{a}_d$ and using $\mathbf{v}' = \mathbf{v} + \ell^\tau \mathbf{u}_d$ proves $\lambda_i^\tau \ge -\ell^\tau$ for all $1 \le i \le d-1$.

The convex hull of $\mathbf{v}$, $\mathbf{v}'$ and the first lattice points on edges adjacent to $\mathbf{v}$ or $\mathbf{v}'$ contains $d(\ell^\tau + 1) + \sum \lambda_i^\tau$ lattice points. By assumption, this is at most N, so

$$\sum \lambda_i^\tau \le N - d(\ell^\tau + 1) \tag{3.2}$$

Using the bounds for ℓ^τ proves the lemma. □

The lemma shows that we can view $\mathcal{F}_d^{\text{poly}}(N)$ as a finite list of families of smooth fans, where each family depends on a set of edge parameters that range over a bounded set. We generate this list in two steps. In a first step we need a complete list of families of smooth *minimal* fans with at most N rays. Given this list we generate $\mathcal{F}_d^{\text{poly}}(N)$ by all possible sequences of blowups of a minimal fan.

When we started our work on this algorithm we were mainly interested in a classification of 2- and 3-dimensional smooth polytopes. In dimension 2, all families of smooth minimal fans have been classified already. In dimension 3, we know all minimal fans with up to 12 maximal cones [25]. Hence, in the present implementation we use these lists instead of generating them anew. The families of fans relevant for the application in Section 5 are shown in Figures 5.2 and 5.3. In Section 6 we discuss two approaches for an algorithm to extend these lists.

Given the minimal smooth fans we have to generate the fans in $\mathcal{F}_d^{\text{poly}}(N)$. However, deciding whether a smooth fan is the normal fan of some polytope $P \in \mathcal{P}_d(N)$ basically amounts to finding P. Hence, instead of generating $\mathcal{F}_d^{\text{poly}}(N)$ we are satisfied with a slightly larger set $\mathcal{F}$ such that $\mathcal{F}_d^{\text{poly}}(N) \subseteq \mathcal{F}$.

Any cone in $\mathcal{F}_d^{\text{poly}}(N)$ has at most N maximal cones. Hence, if $\mathcal{F}_d(N)$ is the set of families of d-dimensional smooth fans with at most N maximal cones, then $\mathcal{F}_d^{\text{poly}}(N) \subseteq \mathcal{F}_d(N)$.

$\mathcal{F}_d(N)$ can in principle be obtained with the following simple procedure. Given a smooth minimal fan $\Sigma^{\min}$ we do a depth-first search on the rooted tree of all fans Σ connected to $\Sigma^{\min}$ via a sequence of blowups. This terminates after a finite number of steps, as each blowup increases the number of maximal cones by $d-1$. Repeating this for each smooth minimal fan enumerates $\mathcal{F}_d(N)$. However, $\mathcal{F}_d(N)$ is significantly larger than $\mathcal{F}_d^{\text{poly}}(N)$. For an efficient method we need additional criteria to remove fans from the search tree.

The definition of a blowup immediately shows that cones that are incomparable in the face poset can be blown up in any order. To avoid generating the same fan again in our search tree we introduce a decision variable $v_\Sigma(\tau)$ on all cones $\tau \in \Sigma$ indicating cones that have to be considered for a blowup.

A more efficient constraint satisfied by fans in $\mathcal{F}_d^{\text{poly}}(N)$ uses the results of our algorithm dimensions $k < d$. Let $i_N(k,n)$ for $1 \le k \le d-1$ be the minimal number of interior lattice points of a smooth k-dimensional polytope with n vertices and at most N lattice points.

Lemma 3.2. *Let P be a smooth polytope with normal fan Σ, and $n_\sigma := |\{\tau \in \Sigma^{(d)} \mid \sigma \subseteq \tau\}|$. Then $|P \cap \mathbb{Z}^d| \ge \sum_{\substack{\sigma\in\Sigma \\ \dim\sigma>0}} i_N(d - \dim\sigma, n_\sigma)$.*

Proof. The combinatorics of Σ completely determines the combinatorics of P. A cone $\sigma \in \Sigma$ contained in n_σ maximal cones of Σ determines a face of dimension $k := d - \dim\sigma$ with n_σ vertices, which itself is a smooth k-polytope. Thus, summing up $i_N(k,n)$ over all cones counts the minimal number of interior lattice points in all faces and hence bounds the number of lattice points of P. □

Algorithm 3.1. BLOWUPS

Input: (class of) smooth fan(s) Σ, $v_\Sigma \in \{0,1\}^\Sigma$, $N \in \mathbb{N}$
Output: a (parametrized) set $\mathcal{F}$ with $\mathcal{F}_d^{\text{poly}}(N) \subseteq \mathcal{F} \subseteq \mathcal{F}_d(N)$.

$\mathcal{F} := \varnothing$;
if $|\Sigma^{(d)}| \le N$ **then**
  if Σ *satisfies* (3.3) **then**
    $\mathcal{F} := \mathcal{F} \cup \{\Sigma\}$;
  foreach $\sigma \in \Sigma$ *with* $\dim \sigma > 1$ *and* $v_\Sigma(\sigma) > 0$ **do**
    $\Sigma_\sigma :=$BLOWUP(Σ, σ);
    $v_{\Sigma_\sigma}(\tau) := \begin{cases} 1 & \tau \in \Sigma_\sigma \backslash \Sigma \\ 1 & \exists \rho \in \Sigma : \tau \subset \rho \wedge \sigma \subset \rho \\ v_\Sigma(\tau) & \text{otherwise} \end{cases}$;
    $\mathcal{F} := \mathcal{F} \cup$ BLOWUPS$(\Sigma_\sigma, v_{\Sigma_\sigma}, N)$;
    $v_\Sigma(\sigma) := 0$;
return $\mathcal{F}$

Note that the bound in the lemma remains true if we also allow $\dim \sigma = 0$. However, $i_N(d,n)$ refers to the interior lattice points of P itself. In our algorithm, we add a smooth fan in $\mathcal{F}_d(N)$ to $\mathcal{F}$ if

$$\sum_{\substack{\sigma \in \Sigma \\ \dim \sigma > 0}} i_N(d - \dim \sigma, n_\sigma) \le N\,. \tag{3.3}$$

Observe that this bound is independent of the edge parameters. Hence, it can also be used on a parametrized family of fans. Experiments show that this bound removes sufficiently many fans to produce a reasonable approximation of $\mathcal{F}_d^{\text{poly}}(N)$.

Algorithm 3.1 now summarizes our method. Given a finite list of parametrized smooth fans it recursively generates all possible sequences of blowups with at most N maximal cones that satisfy the bound (3.3). The actual blowup of a fan Σ at a cone σ is done by a subroutine BLOWUP(Σ, σ). This creates a new smooth fan Σ_σ according to (2.2) and updates the list of edge parameters. They are given by linear functions in the edge parameters of Σ. We give explicit formulas for dimension 2 and 3 in Section 5. Figure 5.4 shows them for all possible types of blowups in these dimensions.

4 Generation of Smooth Polytopes

Now we explain how we generate all sets of ray parameters for a given smooth fan Σ. As a lattice polytope is uniquely determined by its normal fan and an integer for each ray, this will finish the algorithm.

In the following we fix a smooth fan Σ with m rays $\mathbf{a}_1, \ldots, \mathbf{a}_m$. Up to lattice equivalence and relabeling we can assume that $\mathbf{a}_1, \ldots, \mathbf{a}_d$ are the unit basis vectors and span a maximal cone of Σ. Consider the set

$$B_\Sigma(N) := \left\{ \mathbf{b} \in \mathbb{R}^m \;\middle|\; \begin{array}{c} 1 \le \sum c_j^\tau b_j - \sum \lambda_j^\tau b_j \le \frac{1}{d}(N - \sum \lambda_j^\tau)\ \forall\, \tau \in \Sigma^{(d-1)} \\ b_j = 0 \text{ for } 1 \le j \le d \end{array} \right\},$$

Algorithm 4.1. SMOOTHPOLYTOPES

Input: smooth fan Σ, $N \in \mathbb{N}$
Output: equivalence classes of smooth polytopes with normal fan Σ and $|P \cap \mathbb{Z}^d| \leq N$.

$\mathcal{P} := \varnothing$;
E $S := B_\Sigma(N) \cap \mathbb{Z}^m$;
foreach $\mathbf{b} \in S$ **do**
  $P := (\Sigma, \mathbf{b})$;
C/I **if** $|P \cap \mathbb{Z}^d| \leq N$ *and* $P \not\sim Q$ *for all* $Q \in \mathcal{P}$ **then**
    $\mathcal{P} := \mathcal{P} \cup \{P\}$
return $\mathcal{P}$

where c_j^τ, λ_j^τ are the parameters for Σ introduced in Section 2. $B_\Sigma(N)$ is a polyhedron in $\mathbb{R}^m$.

Lemma 4.1. *$B_\Sigma(N)$ is bounded and $B_\Sigma(N) \cap \mathbb{Z}^m$ contains all sets of ray parameters for smooth polytopes with normal fan Σ and at most N lattice points.*

Proof. Each inequality in the definition of $B_\Sigma(N)$ involves the entries of $\mathbf{b}$ corresponding to the rays of two adjacent maximal cones $\sigma, \sigma' \in \Sigma$, and there is a unique ray in $\sigma \setminus \sigma'$. Therefore, if the entries of $\mathbf{b}$ are bounded for the rays in σ, then they are also bounded on σ'. Thus, fixing the entries for one maximal cone in Σ bounds all other entries, as the vertex-edge graph of P is connected. So $B_\Sigma(N)$ is bounded, i.e. a polytope.

Let $\tau \in \Sigma^{(d-1)}$ with incident maximal cones σ and σ'. σ and σ' correspond to adjacent vertices $\mathbf{v}$ and $\mathbf{v}'$ of P connected by an edge of lattice length ℓ^τ. Let $\mathbf{a}_k$ and $\mathbf{a}_{k'}$ be the generator in $\sigma \setminus \tau$ and $\sigma' \setminus \tau$, respectively. Then $c_k^\tau = c_{k'}^\tau = 1$, and $c_j^\tau = 0$ otherwise. Hence, using equation (2.1) we obtain

$$\begin{aligned}\ell^\tau = \langle \mathbf{a}_k, \mathbf{v}\rangle - \langle \mathbf{a}_k, \mathbf{v}'\rangle &= \langle \mathbf{a}_k, \mathbf{v}\rangle - \textstyle\sum c_j^\tau \langle \mathbf{a}_j, \mathbf{v}'\rangle + \langle \mathbf{a}_{k'}, \mathbf{v}'\rangle \\ &= \langle \mathbf{a}_k, \mathbf{v}\rangle + \langle \mathbf{a}_{k'}, \mathbf{v}'\rangle - \textstyle\sum \lambda_j^\tau \langle \mathbf{a}_j, \mathbf{v}'\rangle = b_k + b_{k'} - \textstyle\sum \lambda_j^\tau b_j\,.\end{aligned}$$

Using $\ell^\tau \geq 1$ and the bound in (3.2) now proves the claim. □

Note that fixing the parameters for one cone is essential, as translations of a polytope have the same normal fan. The lemma proves that $\mathcal{P}_\Sigma(N)$ is finite. Together with Lemma 3.1 this proves Theorem 2.1. We can generate $\mathcal{P}_\Sigma(N)$ now with Algorithm 4.1.

We explain some steps of the algorithm in more detail. In line (E) we have to enumerate the set $B_\Sigma(N) \cap \mathbb{Z}^m$. This amounts to enumerating the lattice points in a polytope. We have implemented the following project-and-lift method for a polytope P in `polymake`. Enumerating $P \cap \mathbb{Z}^d$ is easy if $d = 1$. Otherwise, we do a coordinate projection into $\mathbb{R}^{d-1}$ and enumerate the one-dimensional fibers over each lattice point in the projection.

We could use the same method in line (C/I) to compute $|P \cap \mathbb{Z}^d|$. However, a significantly faster algorithm for counting lattice points uses the Ehrhart function

of P. This is implemented in implemented in `LattE` [7], which can be accessed via `polymake` [15].

The equivalence check in (C/I) is done with the following algorithm. For a lattice polytope P let F_P be the matrix whose entry at position (i, j) is the lattice distance of vertex i from facet j. Two smooth polytopes P_1 and P_2 are lattice equivalent if and only if F_{P_1} and F_{P_2} are equal up to a row and column permutation. Note that this only works for smooth polytopes. Again, we have implemented this check in `polymake`.

5 Implementation and Results

The implementation of our methods is based on the software framework `polymake` (version 2.9.7) of Gawrilow and Joswig [9] and realized as an extension. Counting lattice points was done via `polymake`'s interface to `LattE` by De Loera et. al [7,15]. The code of this extension was written as part of the MSc Thesis of the second author [19].

We have applied the algorithm in dimension two and three to compute all smooth lattice polytopes with at most 12 lattice points.

Theorem 5.1. *There are 41 smooth lattice polygons and 33 smooth 3-dimensional lattice polytopes with at most 12 lattice points.*

Table 5.1 lists the numbers of polytopes obtained in dimensions 2 and 3.

In the actual implementation we maintain a list of coordinates for each ray generator of a family of fans, as we need them later to produce an explicit representative in each equivalence class. The edge parameters can be obtained via a simple computation using (2.1) from the coordinates.

The following two paragraphs give the necessary input in dimension 2 and 3 for Algorithms 3.1 and 4.1 to obtain smooth 2- and 3-dimensional lattice polytopes with at most 12 lattice.

Generation of Smooth Polygons. In dimension 2 there are only two types of parametrized smooth minimal fans, see Figure 5.2. Σ_P correspond to the complex projective plane and Σ_a to the Hirzebruch surface of degree a. Using symmetry and Lemma 3.1 we can assume $0 \leq a \leq N - d = 10$. Blowing up Σ_P at the lower cone gives Σ_1, so we omit $a = 1$. In dimension 2 the equation (3.3) is trivial, so we generate all fans in $\mathcal{F}_d(N)$.

There is only one type of edge parameter update necessary for the function BlowUp used in Algorithm 4.1. The values are given in Figure 5.4(a).

Table 5.1. Smooth polytopes in dimension 2 and 3 with at most 12 lattice points

No. of vertices	3	4	5	6	7	8	≥ 9
No. of polygons	3	30	3	4	0	1	0

(a) Dimension 2

No. of vertices	4	6	8	≥ 10
No. of polytopes	2	25	6	0

(b) Dimension 3

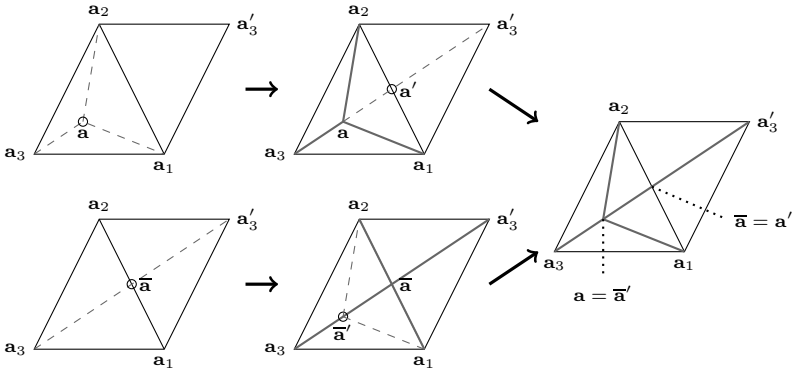


Fig. 5.1. Different series of blowups leading to the same fan

Generation of Smooth 3-Polytopes. In dimension 3, Oda [25] has classified parametrized smooth minimal fans with up to 12 maximal cones, i.e. up to 8 rays. There are 19 such fans. However, using the lattice point bound (3.3) and some similar arguments, only five of these fans or their blowups can correspond to a smooth lattice polytope with at most 12 lattice points. These fans are listed in Figure 5.3. After converting from ray description to edge parameter description we can calculate the following initial bounds with Lemma 3.1.

$$\Sigma_2(a): \quad 0 \le a \le 9 \ \text{ by symmetry}, \qquad \Sigma_3(b,c): \quad -9 \le b, c \le 9\,,$$
$$\Sigma_4(a,b,c): \quad -9 \le a, b, c \le 9\,, \qquad \Sigma_5(a): \quad -5 \le a \le 4\,.$$

The functions for the update of the edge parameters necessary in BlowUp follow from an easy computation using (2.1). There are two different types of blowups. The parameter changes for each are listed in Figure 5.4(b).

In dimension 3 we can introduce two further selection conditions for our depth-first-search in the tree of smooth fans constructed in Algorithm 3.1.

1. The two sequences of blowups shown in Figure 5.1 give the same fan.
2. In addition to the condition given in (3.3), we can skip fans that have a ray with more than 6 neighbours because there is no polygon with more than 6 vertices but strictly less than 12 lattice points.

Using the extension Following is a short example showing how to use the interactive `polymake` shell to compute all polytopes with less than 12 lattice points and normal fan $\Sigma_3(b,c)$.

```
polytope > import_extension("~/polymake-extensions/fan/");
polytope > $fan_23_2 = load("~/fans/dim3/12/23_2.pfan");
polytope > @polys_23_2 = $fan_23_2->POLYTOPES;
  Creating blowups ...
  do vertexblowup: 0-0
```

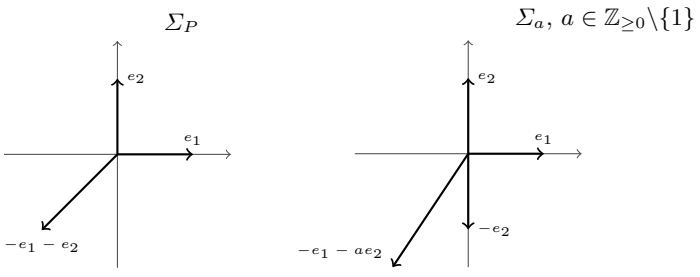


Fig. 5.2. Smooth minimal fans in dimension 2

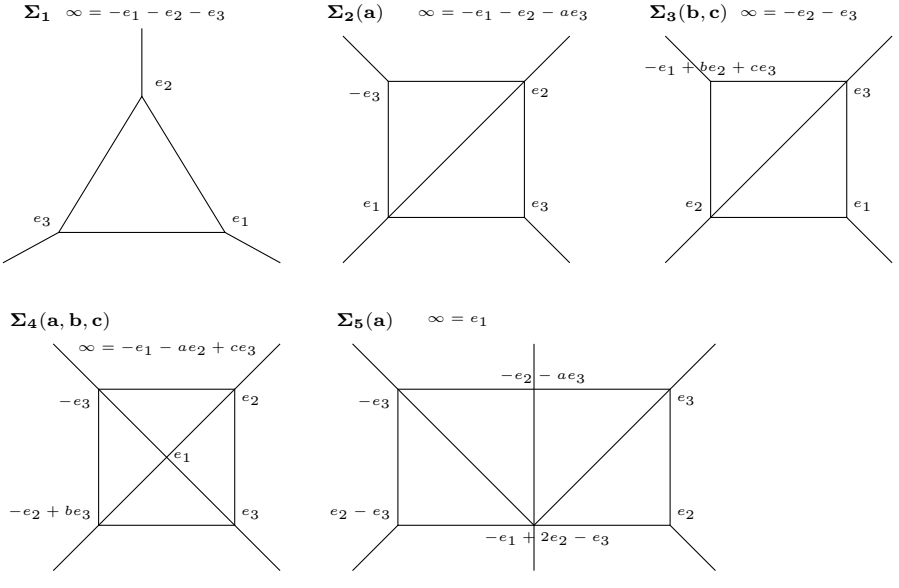
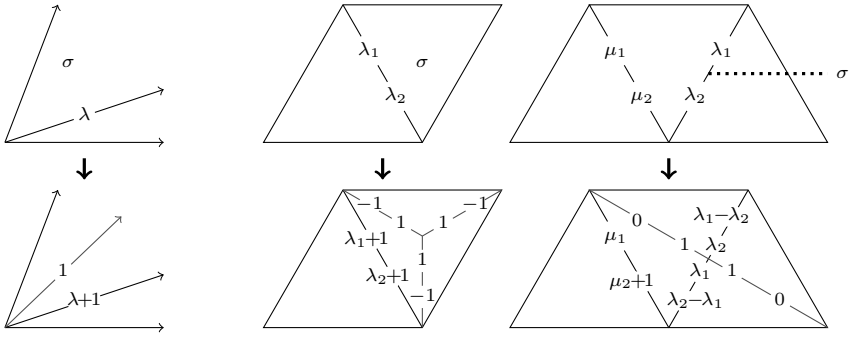


Fig. 5.3. The relevant fans in dimension 3



(a) $d = 2$ (b) $d = 3$: Blowup in a cone of codimension 0 and 1.

Fig. 5.4. The new edge parameters after a blowup at σ.

```
  do vertexblowup: 0-1
<skip>
  do edgeblowup: 0-8
  Done creating blowups.
  Generating polytopes ...
  blowup 1 of 58
        fan 1 of 361
<skip>
        fan 529 of 529
  Done generating polytopes.
  Found 91 polytopes including duplicates.
  Found 26 non-isomorphic polytopes with this normal fan.
polytope >
```

All polytopes are now stored in a perl array `@polys_23_2` as polytope objects, and can be accessed via `$polys_23_2[$i]`.

6 Conclusion and Open Problems

The given algorithm generates smooth lattice polytopes in fixed dimension. The main shortcoming of our approach is the dependence on a list of minimal smooth fans as input, which is not known in general. Still, the results in dimension 2 and 3 show that our algorithm is an efficient method to generate smooth polytopes.

We propose two approaches to remove the dependence on a classification of minimal fans. The first outlines a method to extend the classification of minimal fans, the second proposes a completely different approach to generate fans.

The weak factorization theorem [29] states that any two (not necessarily smooth) rational polyhedral fans are connected by a series of blowups and blowdowns. Hence, the graph on all rational fans with edges for blowups and blowdowns is connected. Starting from some smooth fan, we would like to do a breadth-first search to obtain a complete list of smooth fans with a bounded number of maximal cones. However, it is unknown whether the sub-graph of all these fans is connected, and there is no bound known on the path length between two nodes in the graph.

Combinatorial types of triangulations of the 2-sphere have been enumerated for a bounded number of vertices by various authors, see e.g. [21] and the references mentioned there. Using (2.1) we can generate all smooth fans with this combinatorial type. However, this is not efficient. Instead, we can use the observation that some normalized sum of the lattice points on the boundary of a polygon always adds to 12 [14]. This gives a system of linear equations in the parameters λ_j^τ for all $\tau \in \Sigma^{(d-1)}$.

Many open conjectures involving smooth lattice polytopes actually only refer to some subclass of this set, e.g. polytopes that are centrally symmetric. Restricting to such a class just adds further constraints in our algorithms.

References

1. Aardal, K., Weismantel, R., Wolsey, L.A.: Non-standard approaches to integer programming. Discrete Applied Mathematics 123(1-3), 5–74 (2002)
2. Batyrev, V.: On the classification of toric Fano 4-folds. Journal of Mathematical Sciences 94(1), 1021–1050 (1999)
3. Batyrev, V., Kreuzer, M.: Integral cohomology and mirror symmetry for Calabi-Yau 3-folds. In: Mirror symmetry. V. AMS/IP Stud. Adv. Math., vol. 38, pp. 255–270. Amer. Math. Soc., Providence (2006)
4. Beck, M., Chen, B., Fukshansky, L., Haase, C., Knutson, A., Reznick, B., Robins, S., Schürmann, A.: Problems from the Cottonwood Room. Contemporary Mathematics 374, 179–191 (2005)
5. Bogart, T., Haase, C., Hering, M., Lorenz, B., MacLagan, D., Nill, B., Paffenholz, A., Santos, F., Schenck, H.: Few smooth d-polytopes with n lattice points (May 2010) (in preparation)
6. Bruns, W., Gubeladze, J.: Polytopes, rings, and K-theory. Springer Monographs in Mathematics. Springer, Heidelberg (2009)
7. De Loera, J.A., Hemmecke, R., Yoshida, R., Tauzer, J.: lattE, `http://www.math.ucdavis.edu/~latte/`
8. Diaconis, P., Sturmfels, B.: Algebraic algorithms for sampling from conditional distributions. Annals of Statistics 26(1), 363–397 (1998)
9. Gawrilow, E., Joswig, M.: polymake, `http://www.opt.tu-darmstadt.de/polymake/`
10. Gubeladze, J.: Convex normality of rational polytopes with long edges (December 2009), arxiv.org/abs/0912.1068
11. Haase, C., Hibi, T., MacLagan, D. (eds.): Mini-Workshop: Projective normality of smooth toric varieties, Oberwolfach reports, vol. 4 (2007)
12. Haase, C., Nill, B., Paffenholz, A., Santos, F.: Lattice points in Minkowski sums. Electronic Journal of Combinatorics 15 (2008)
13. Haase, C., Paffenholz, A.: Quadratic Gröbner bases for smooth 3 x 3 transportation polytopes. Journal of Algebraic Combinatorics 30(4) (2009)
14. Haase, C., Schicho, J.: Lattice polygons and the number $2i + 7$. American Mathematical Monthly 116(2), 151–165 (2009)
15. Joswig, M., Müller, B., Paffenholz, A.: Polymake and lattice polytopes. In: DMTCS Proceedings of FPSAC (February 2009)
16. Kleinschmidt, P.: A classification of toric varieties with few generators. Aequationes Mathematicae 35(2-3), 254–266 (1988)
17. Kreuzer, M., Skarke, H.: On the classification of reflexive polyhedra. Communications in Mathematical Physics 185(2), 495–508 (1997)
18. Kreuzer, M., Skarke, H.: PALP: A package for analyzing lattice polytopes with applications to toric geometry (April 2002), `http://hep.itp.tuwien.ac.at/~kreuzer/CY/CYpalp.html`
19. Lorenz, B.: Generating smooth polytopes - extension for polymake, `http://ehrhart.math.fu-berlin.de/people/benmuell/classification.html`
20. Lorenz, B.: Classification of smooth lattice polytopes with few lattice points. Diploma thesis (2010), arxiv.org/abs/1001.0514
21. Lutz, F.: The manifold page (April 2010), `http://www.math.tu-berlin.de/diskregeom/stellar/`
22. McDuff, D.: Displacing lagrangian toric fibers via probes (April 2009), arxiv.org/abs/0904.1686

23. Nill, B., Paffenholz, A.: Examples of non-symmetric Kähler-Einstein toric Fano manifolds (May 2009), arxiv.org/abs/0905.2054
24. Øbro, M.: An algorithm for the classification of smooth Fano polytopes (April 2007), arxiv.org/abs/0704.0049
25. Oda, T.: Convex Bodies and Algebraic Geometry. In: An Introduction to the Theory of Toric Varieties. Ergebnisse der Mathematik und ihrer Grenzgebiete. Springer, Heidelberg (1987)
26. Oda, T.: Problems on Minkowski sums of convex lattice polytopes (December 2008), arxiv.org/abs/0812.1418
27. Ohsugi, H., Hibi, T.: Unimodular triangulations and coverings of configurations arising from root systems. J. Algebr. Comb. 14(3), 199–219 (2001)
28. Sturmfels, B.: Equations defining toric varieties. In: Kollár, J. (ed.) Algebraic geometry. Proc. Summer Research Institute, Santa Cruz, CA, USA, July 9-29 (1995); Proc. Symp. Pure Math., vol. 62, pp. 437–449. AMS, Providence (1997)
29. Włodarczyk, J.: Toroidal varieties and the weak factorization theorem. Inventiones Mathematicae 154(2), 223–331 (2003)

Mathemagix: Towards Large Scale Programming for Symbolic and Certified Numeric Computations

Grégoire Lecerf*

Laboratoire de Mathématiques de Versailles
UMR 8100 CNRS
Université de Versailles Saint-Quentin
45, avenue des États-Unis
78035 Versailles, France
gregoire.lecerf@math.uvsq.fr
http://www.math.uvsq.fr/~lecerf

Abstract. Coordinated by Joris van der Hoeven from the 90's, the Mathemagix project aims at the design of a scientific programming language for symbolic and certified numeric algorithms. This language can be compiled and interpreted, and it features a strong type system with classes and categories. Several C++ libraries are also being developed, mainly with Bernard Mourrain and Philippe Trébuchet, for the elementary operations with polynomials, power series and matrices, with a special care towards efficiency and numeric stability.

In my talk I will give an overview of the language, of the design and the contents of the C++ libraries, and I will illustrate possibilities offered for certified numeric computations with balls and intervals.

1 Context and Motivation

The implementation of high level algorithms for algebra or analysis is often made difficult with general purpose programming languages such as C or C++ because of the lack of the elementary data structures and operations needed in mathematics: polynomials, power series, matrices, etc. In fact, several very efficient libraries exist for low level operations, such as GMP [3] for long integers, MPFR [2] for long real and complex floating-point numbers, NTL [9] or FLINT [4] for elementary operations in number theory, etc. Unfortunately implementing an algorithm directly on the top of these libraries might reveal to be technical for non-specialists.

On the other hand computer algebra systems, such as Maple, Mathematica and Magma, provide the user with many high level functionalities, but they do not allow him to inspect the internal source code, to replace existing routines, and to compile into an efficient executable. As a good compromise between low

* This work has been partly supported by the French ANR-09-JCJC-0098-01 MaGiX project, and by the Digiteo 2009-36HD grant of the Région Ile-de-France.

K. Fukuda et al. (Eds.): ICMS 2010, LNCS 6327, pp. 329–332, 2010.

and high level software, the SAGE project [10] offers a comfortable interface to most of the existing libraries and computer algebra systems through the PYTHON interpreter. Although the PYTHON language is sophisticated and fast interpreted, it lacks a good compiler and category mechanism such as the one of AXIOM [7] and ALDOR [1] that were primarily well designed for mathematical purposes.

The MATHEMAGIX project aims at developing a new language, with its own compiler, its own interpreter, and with facilities to import functionalities from C++, in order to make possible and easy the development of efficient high level computer algebra libraries. MATHEMAGIX is freely distributed under the GPL license. It can be downloaded from `http://www.mathemagix.org`. Automatic configuration and building is ensured via the GNU AUTOTOOLS and CMAKE. At the present time only MAC OS X and LINUX platforms are supported.

The interpreter can be run in TEXMACS [5] which provides the user with a graphical front-end with high quality typesetting for mathematical formulas. In addition, MATHEMAGIX is connected to the 3D algebraic-geometric modeler AXEL [8].

2 Language, Compiler and Interpreter

The ultimate design goals of the MATHEMAGIX language are as follows:

Strong typedness. All object instances are strongly typed, and functions can be overloaded. The user can define new types and classes, and use inheritance from other classes. Usual automatic conversions are supported. Types and functions can be templated by classes with category (like a ring, a module over a ring, etc).

Runtime efficiency. The language can be compiled in order to reach the same order of efficiency as C++. The compiler, still under development, is itself written in the MATHEMAGIX language.

Fast prototyping. The language can be interpreted in order to make fast prototyping possible. The language is currently partially supported by the `mmx-light` interpreter written in C++. A complete implementation, as a backend of the compiler, is in progress.

Reusability of external libraries. Before reaching optimal performances through our compiler, we already achieved runtime efficiency thanks to an extensive use of existing libraries written in C++. Our glue mechanism is well designed for importing C++ templated types.

3 Libraries

MATHEMAGIX comes with several C++ libraries for elementary operations on polynomials and matrices:

basix contains the classical data types, such as vectors, lists, hash tables, but also the input and output streams, Posix sockets, and multi-threading facilities. In fact MATHEMAGIX does not rely on the standard C++ library (STL). This ensures us uniform interfaces throughout all our packages.

numerix is dedicated to arbitrarily long integers and floating-points numbers. It is essentially a wrapper of GMP and MPFR, but we also provide the user with intervals, balls, complex numbers, tangent numbers, and modular integers.

algebramix implements univariate polynomials, power series, p-adic numbers, and matrices. Generic templated implementations are available, but also specific variants for the usual coefficients types. Fast algorithms are implemented including the FFT and its truncated variant, the Schönhage and Strassen products. Vectors and matrices feature automatic loop unrolling, SIMD speed-ups, and also cache-friendly variants.

analyziz is devoted to polynomial and matrices over numeric types in order to ensure a good compromise between stability and efficiency.

multimix implements naive and fast algorithms for multivariate polynomials, jets and series.

realroot contains several numeric solvers for algebraic systems, mainly based on subdivision methods.

linalg is a C++ templated version of the LAPACK library, that allows the user for instance to benefit of the LAPACK algorithms on the arbitrary large floating-point numbers of MPFR.

Other packages are devoted to finite fields, symbolic expressions, asymptotic analysis, numeric homotopy continuation for polynomial system solving, polynomial factorization, lattice reduction, etc.

4 Certified Numeric Analysis

One important goal of the MATHEMAGIX project is the development of certified analysis, with goals similar to symbolic computations but with numeric types. For instance, if we consider analytic functions from the symbolic point of view, a natural representation is as an exact solution of a differential operator. Elementary arithmetic operations can be implemented directly on these operators. On the other hand, from the numeric point of view, one is interested in evaluating analytic functions at any given point as fast as possible. Symbolic processing is often better for theoretical purposes, but numeric calculation is indeed necessary to real world applications. The latter often turns out to be more efficient than the former, but the result is not usually certified. MATHEMAGIX aims at providing the user with the best of both worlds.

For instance, following [6], we can implement an analytic function as the data of: 1) an algorithm for computing finite Taylor expansions, 2) an algorithm for computing bounds on norms of a finite number of derivatives on sufficiently small balls, and 3) an algorithm for analytic continuation. MATHEMAGIX contains all the subroutines necessary to this task, with certified internal computations. In particular we implemented fast relaxed power series with interval or ball coefficients, and took a special care to minimize the overhead between long and hardware floating types. At the present time no other software offer a similar level of generality and efficiency.

References

1. Dragan, L., Huerter, S., Oancea, C., Watt, S.: Aldor programming language (2007), http://www.aldor.org
2. Fousse, L., Hanrot, G., Lefèvre, V., Pélissier, P., Zimmermann, P.: MPFR: A multiple-precision binary floating-point library with correct rounding. ACM Transactions on Mathematical Software 33(2) (June 2007), http://www.mpfr.org
3. Granlund, T., et al.: GMP, the GNU multiple precision arithmetic library (1991), http://gmplib.org
4. Hart, W.: Fast library for number theory (2008), http://www.flintlib.org
5. van der Hoeven, J.: GNU TEXMACS (1998), http://www.texmacs.org
6. van der Hoeven, J.: On effective analytic continuation. MCS 1(1), 111–175 (2007)
7. Jenks, R.D., Sutor, R.: AXIOM: the scientific computation system. Springer, New York (1992)
8. Mourrain, B., Wintz, J., Chau, S., Alberti, L.: Axel (2007), http://axel.inria.fr
9. Shoup, V.: NTL: A library for doing number theory (2005), http://shoup.net/ntl
10. Stein, W.A., et al.: Sage Mathematics Software, Version 4.2.1 (2009), http://www.sagemath.org

Complex Inclusion Functions in the CoStLy C++ Class Library

Markus Neher

Institute for Applied and Numerical Mathematics
Karlsruhe Institute of Technology
Kaiserstr, 12, 76128 Karlsruhe, Germany
markus.neher@kit.edu

Abstract. The C++ class library CoStLy for the rigorous computation of complex function values or ranges is presented. Rectangular complex interval arithmetic is used for the computations. In the CoStLy procedures, all truncation and roundoff errors are calculated during the course of the floating-point computation and enclosed into the result.

The library contains procedures for root and power functions, the exponential, trigonometric and hyperbolic functions, their inverse functions, and some auxiliary functions, such as the absolute value or the argument function.

Keywords: Interval Arithmetic, Complex Standard Functions.

1 Introduction

The Complex Standard Functions Library CoStLy has been developed as a C++ class library for the validated computation of function values and of ranges of the complex standard functions in the set

$$S_F = \{\ \exp,\ \ln,\ \arg,\ \mathrm{sqr},\ \sqrt{\ },\ \mathrm{power},\ \mathrm{pow},\ \mathrm{root},\ \cos,\ \sin,\ \tan,\ \cot,\ \cosh,\ \sinh,\ \tanh,\ \coth,\ \mathrm{acos},\ \mathrm{asin},\ \mathrm{atan},\ \mathrm{acot},\ \mathrm{acosh},\ \mathrm{asinh},\ \mathrm{atanh},\ \mathrm{acoth}\},$$

where $\mathrm{power}(z, n)$ is the power function for integer exponents, $\mathrm{pow}(z, p)$ is the power function for real or complex exponents, and $\mathrm{root}(z, n)$ denotes the nth root function.

Inclusion functions for complex standard functions were first realized in the ACRITH library by IBM [5], using algorithms developed by Braune [1] and Krämer [7]. Later, Bühler [2] implemented the same algorithms in Pascal-XSC procedures, which were then extended to a Pascal-XSC interval library of complex standard functions by Krämer and Westphal [10]. There is also a Maple implementation by Grimmer [4], but the IntpakX package contains only inclusion functions for rational functions and for the exponential function. Nevertheless, all these packages are outdated or no longer available.

Verified function values for point arguments can be obtained using the MPC library [8] by Enge, Théveny, and Zimmermann. MPC is not an interval library,

K. Fukuda et al. (Eds.): ICMS 2010, LNCS 6327, pp. 333–336, 2010.

but it provides multiprecision complex arithmetic with a wide variety of rounding modes. For point arguments, this can be equivalent to computing an interval result.

To the author's knowledge, apart from CoStLy, the only other interval package that currently contains subroutines for complex inclusion functions which also compute range bounds is INTLAB [6]. INTLAB's procedures are based on circular complex arithmetic, however, which is different from rectangular interval arithmetic so that the two packages are not directly comparable.

In the following, boldface letters are used to denote intervals. The set of all complex rectangular intervals is denoted by $\mathbb{IC}$. For a bounded subset M of $\mathbb{C}$, the *interval hull* $\Box M$ of M is the smallest rectangular interval that contains M. For $D \subseteq \mathbb{C}$, the range $\{f(z) : z \in D\}$ of a function $f : D \longrightarrow \mathbb{C}$ is denoted by $\mathrm{Rg}(f, D)$. An *inclusion function* F of a given function $f : \mathbb{C} \to \mathbb{C}$ is an interval function $F : \mathbb{IC} \to \mathbb{IC}$ that encloses the range of f on all intervals $\boldsymbol{z} \subseteq D$:

$$F(\boldsymbol{z}) \supseteq \mathrm{Rg}(f, \boldsymbol{z}) \quad \text{for all} \quad \boldsymbol{z} \subseteq D.$$

If $F(\boldsymbol{z}) = \Box\mathrm{Rg}(f, \boldsymbol{z})$ holds for all $\boldsymbol{z} \subseteq D$, then F is called *optimal*.

2 Design of the CoStLy Library

Complex Analysis and Interval Arithmetic. By definition, the real and imaginary parts of any function in S_F can be expressed as compositions of real standard functions. Optimal complex inclusion functions are obtained by determining the extremal values of these compositions [1,7,9].

For verified results, rectangular complex interval arithmetic is employed in the CoStLy library functions. Thus, rectangles in the complex plane which are guaranteed to contain the desired function values are computed. This verification applies to single function values and to ranges of functions on rectangular domains.

Interval Libraries for Real Standard Functions. For the validated computation of function values and function ranges for the *real* standard functions in S_F, there are several interval libraries available. CoStLy was developed for being used with either C-XSC [10] or filib++ [3]. Today, a version of the CoStLy library is integrated in C-XSC.

Design Philosophy. The CoStLy inclusion functions have been designed with the intention that computed range bounds must be valid in any circumstance. For a single-valued complex function f, its inclusion function $F : \mathbb{IC} \to \mathbb{IC}$, and some given rectangular complex interval $\boldsymbol{z}$, validity means that $F(\boldsymbol{z})$ must contain the bounded set $\{f(z)|z \in \boldsymbol{z}\}$. This applies to the functions exp, sqr, power, cos, sin, tan, cot, cosh, sinh, tanh, and coth, which are single-valued and analytic on their respective domain. CoStLy contains optimal inclusion functions for these functions (where optimal refers to the accuracy of the implemented algorithms, if performed in exact arithmetic).

For a multi-valued function, the meaning of a valid enclosure is less obvious. For example, the definition of $\sqrt{-1}$ depends very much on the context of the computation. Possible values include $+\imath$, $-\imath$, $\{+\imath, -\imath\}$, $\imath \cdot [-1, 1]$, or the empty set. To accommodate varied demands, three types of inclusion functions have been implemented in CoStLy:

(i) Each multi-valued function f in S_F has analytic branches on appropriate subsets of the complex plane. The CoStLy library contains an inclusion function F_p for the single-valued principal branch of f. Usually, F_p is defined on a subset of $\mathbb{IC}$. If $\boldsymbol{z}$ is not in the domain of definition of f, the computation is aborted throwing an exception and issuing a warning message.

(ii) For applications in which function values from different branches are acceptable, inclusion functions that are defined for all $\boldsymbol{z} \in \mathbb{IC}$, for which at least one branch of f is bounded on $\boldsymbol{z}$, are included. Inclusion functions of this type are denoted by F_c. Depending on the location of $\boldsymbol{z}$ in the complex plane, $F_c(\boldsymbol{z})$ returns function values belonging to different branches of f. As an immediate consequence, there are regions in $\mathbb{C}$ where inclusion isotonicity of the inclusion function is lost.

(iii) Provided that the set M of all values of a multi-valued function is bounded, an inclusion function F_a enclosing M has also been implemented in CoStLy. For example, such an inclusion function is available for roots.

A detailed description of all inclusion functions contained in CoStLy is given in [9].

3 Practical Performance

If performed in exact arithmetic, the inclusion functions of type F_p compute optimal range bounds. For the sake of accuracy, a major effort has been made in the implementation of the algorithms in floating-point arithmetic to eliminate all intermediate expressions subject to numerical overflow, underflow, or cancellation. The CoStLy library has been extensively tested for arguments with absolute values ranging from 1.0E-300 to 1.0E+300. For most arguments, the computed bounds for function values are highly accurate. In many test cases, the observed precision of the result was about 50 correct bits (out of the 53 bits available in IEEE 754 floating-point arithmetic) for point arguments.

CoStLy is distributed under the terms of the GNU General Public License. The software is currently available at the following sites:

`http://www.xsc.de`

`http://iamlasun8.mathematik.uni-karlsruhe.de/~ae16/CoStLy.html`

References

1. Braune, K.: Highly Accurate Standard Functions for Real and Complex Numbers and Intervals in Arbitrary Floating-point Number Screens. PhD thesis, Universität Karlsruhe, Germany (1987) (in German)
2. Bühler, G.: Standard Functions for Complex Intervals in the 64 Bit IEEE Data Format. Diploma thesis, Universität Karlsruhe, Germany (1993) (in German)

3. FILIB++ Interval Library, http://www.math.uni-wuppertal.de/~xsc/software/filib.html
4. Grimmer, M.: Interval Arithmetic in Maple with intpakX. PAMM 2, 442–443 (2003)
5. IBM: High-Accuracy Arithmetic Subroutine Library (ACRITH). Program Description and User's Guide, 3rd ed. SC 33-6164-02 (1986)
6. INTLAB - INTerval LABoratory, http://www.ti3.tu-harburg.de/~rump/intlab
7. Krämer, W.: Inverse Standard Functions for Real and Complex Interval Arguments with a priori Error Bounds for Arbitrary Number Formats. PhD thesis, Universität Karlsruhe, Germany (1987) (in German)
8. MPC – A Library for Multiprecision Complex Arithmetic with Exact Rounding, http://mpc.multiprecision.org
9. Neher, M.: Complex Standard Functions and Their Implementation in the CoStLy Library. ACM TOMS 33, 20–46 (2007)
10. XSC Languages (C-XSC, PASCAL-XSC), http://www.xsc.de

Standardized Interval Arithmetic and Interval Arithmetic Used in Libraries

Nathalie Revol*

INRIA
LIP (UMR 5668 CNRS - ENS de Lyon - INRIA - UCBL), Université de Lyon
École Normale Supérieure de Lyon, 46 allée d'Italie, 69007 Lyon, France
nathalie.revol@ens-lyon.fr

Abstract. The standardization of interval arithmetic is currently undertaken by the IEEE-1788 working group. Some features of the standard are detailed. The features chosen here are the ones which may be the less widely adopted in current implementations of interval arithmetic. A survey of interval-based libraries, focusing on these features, is given.

Keywords: interval arithmetic, standardization, interval-based libraries.

1 Introduction

There is no universally accepted definition of interval arithmetic, but there are specific needs and specific definitions corresponding to these needs. Libraries are usually devoted to a particular domain of applications and thus implement the version of interval arithmetic most suited to these applications. In what follows, some points of divergence will be developed, and the choices made in the current draft of the standard for interval arithmetic, as elaborated by the IEEE-1788 working group [11], will be detailed. This working group proceeds by presented motions, discussing their contents and eventually voting on the motions. Then, a list of interval-based libraries is given, along with their features with respect to these points of divergence.

2 Some Features of the Current Draft of the Standard

2.1 Arithmetic Operations

Intervals are defined as *closed connected subsets of* $\mathbb{R}$, cf. Motion 3 [11]. Arithmetic operations are defined in Motion 5 [11] in the following way: for any operation $\diamond$ that is either an addition, a subtraction, a multiplication or a division when the denominator does not contain 0, then $\mathbf{a} \diamond \mathbf{b} = \{a \diamond b,\ a \in \mathbf{a} \text{ and } b \in \mathbf{b}\}$. In the case of the division of an interval $\mathbf{a}$ by an interval $\mathbf{b}$ that contains 0, several choices are possible. What the standard defines is that $0/0 = \emptyset$, $\mathbf{a}/\mathbf{b} = \mathbb{R}$ when 0 is in the interior of $\mathbf{b}$ and one gets a semi-infinite interval when 0 is one of the endpoint of $\mathbf{b}$, for instance $[1, 2]/[0, 1] = [1, +\infty)$. The tenants of cset theory [21] advocated for getting again $\mathbb{R}$ in this case.

* This work is supported by the INRIA Direction du Développement Technologique.

K. Fukuda et al. (Eds.): ICMS 2010, LNCS 6327, pp. 337–341, 2010.

2.2 Exact Dot Product

It has been advocated that, in order to get guaranteed and tight enclosures of the results, in the case of linear algebra routines, an exact dot product is needed. An exact dot product computes the required (floating-point) dot product as if using exact arithmetic and returns the rounding of the exact result. Mandating exact dot product in the standard for interval arithmetic has been approved as Motion 9, after a thorough discussion and a close vote.

2.3 Relational Operations

Relational, or reverse, or inner, operations, correspond to the notion of "reciprocal" in algebraic structures: for instance, the subtraction is usually the reciprocal of the addition. However, this property does not hold in interval arithmetic: $(\mathbf{a}+\mathbf{b})-\mathbf{b} \supset \mathbf{a}$. So-called "inner addition and subtraction" have been adopted, in order to provide reciprocals of the addition and subtraction. However, such reciprocals are usually not functions, since they can take multiple values; rather, they indicate a relation between the operands and the result.

2.4 Exceptions Handling and Decorations

Let us exemplify this point with Schauder's theorem. This theorem states that, if f is a smooth function over a compact set K and if $f(K) \subset K$, then f has a fixed point in K. However, if $f(x) = \sqrt{x-1}$ and $K = [0, 2]$ and if $f(K)$ is computed as $f(K \cap \mathcal{D}_f)$ where $\mathcal{D}_f$ is the domain of f, then $f(K)$ returns $[0, 1] \subset K$. In such a case, the user needs to know that K has been intersected with $\mathcal{D}_f$ prior to the evaluation, otherwise he may conclude wrongly that f has a fixed point in K. More generally, a warning mechanism is needed to handle exceptions. This mechanism could be a global flag. However, global flags are not recommended when multithreading programming is used. Consequently, the standard mandates to attach an extra field, a so-called *decoration*, to the computed result.

2.5 Implementations Based on IEEE-754 Floating-Point Arithmetic

Implementations of interval arithmetic are often based on floating-point arithmetic. In particular, IEEE-754 arithmetic provides directed rounding modes: classically, the left (resp. right) endpoint is computed using rounding mode towards $-\infty$ (resp. $+\infty$). However, some libraries do not use these rounding modes. Some of them subtract/add one ulp to the endpoints, some other implement their own arithmetic, usually in higher precision.

3 Survey of Interval-Based Libraries

- **General-purpose libraries:** C-XSC [10], fi_lib [15] IntLib, used in GlobSol [12], Profil-BIAS [13], Sun Studio-C compiler [25] use double-precision floating-point arithmetic. C-XSC [10], fi_lib [15] IntpakX [14], Mathemagix [26], MPFI [22] use higher precision.

- **Libraries for linear algebra:** C-XSC [10], IntLab [23], Profil-BIAS [13].
- **Libraries for the integration of ODEs:** AWA [17], CAPD [3], COSY [1], VNODE is based either on fi_lib or Profil-BIAS [18], VSPODE [16].
- **Libraries for constraints solving:** Alias [20] is based on Profil-BIAS, Gaol [6], IBEX-Quimper [4], Realpaver [8].
- **Libraries for unconstrained or constrained global optimization:** Baron [24] is available through NEOS and its interval arithmetic is not detailed in the manual, Coconut [19] is based on fi_lib and Jail, an ancestor of Gaol, COSY-GO [1], GlobSol [12], GloptLab [5] is based on IntLab.
- **Libraries using variants of interval arithmetic:** Chebyshev models in ChebModels [2] which is based on IntpakX, Taylor models in COSY and C-XSC [1],[10], affine arithmetic in Fluctuat [7].

	intervals & arith. op.	exact dot product	inner op. (+ and -)	exceptions handling	IEEE-754	variants of int. arith.
Motions	3 & 5	9	12	8	4 (withdrawn)	
applications	all	linear algebra	constraints	Newton' constraints	all	ODE
C-XSC	✓	✓		?	✓	Taylor
fi_lib	✓			?	✓(both)	
IntLib	✓			kind of	✓	Taylor
intpakX	✓			kind of		
Mathemagix	✓				✓	
MPFI	✓			✓	✓	
Profil-BIAS	?	✓			✓	
Sun compiler	cset				✓	
IntLab	✓	✓			✓	
AWA	?	kind of	kind of			Taylor
CAPD		kind of		✓		
COSY					✓	
VSPODE	?					Taylor
Gaol	✓		✓			
Quimper	✓		✓	✓	✓	
RealPaver			✓		✓	
COSY-GO	NA					Taylor
GlobSol	cset or ✓?			kind of	✓	Taylor
Fluctuat	?		planned		✓	

4 Conclusion: Foreseen Points of Divergence

Standardizing interval arithmetic is a difficult task, because of divergent points of view and needs. Foreseen points of divergence include general relational operations, comparisons, IO, link and implementation on top of IEEE-754 floating-point arithmetic (a proposal is given in [9]), allowed and forbidden compiler optimizations. A difficulty is that this standard is not linked to any language, making it difficult to specify, for instance, how decoration values can be read.

References

1. Berz, M., Makino, K.: COSY INFINITY Version 9. Nuclear Instruments and Methods in Physics Research Section A 558(1), 346–350 (2006)
2. Brisebarre, N., Joldes, M.: Chebyshev Interpolation Polynomial-based Tools for Rigorous Computing. In: ISSAC (2010)
3. Computer Assisted Proofs in Dynamics, http://capd.ii.uj.edu.pl/
4. Chabert, G., Jaulin, L.: Contractor programming. Artificial Intelligence 173(11), 1079–1100 (2009)
5. Domes, F.: GloptLab - A configurable framework for the rigorous global solution of quadratic constraint satisfaction problems. Optimization Methods and Software 24(4-5), 727–747 (2009)
6. Goualard, F.: Gaol: NOT Just Another Interval Library (2006-2010)
7. Goubault, E., Putot, S., Baufreton, P., Gassino, J.: Static Analysis of the Accuracy in Control Systems: Principles and Experiments. In: Leue, S., Merino, P. (eds.) FMICS 2007. LNCS, vol. 4916, pp. 3–20. Springer, Heidelberg (2008)
8. Granvilliers, L., Benhamou, F.: Algorithm 852: Realpaver: An Interval Solver using Constraint Satisfaction Techniques. ACM TOMS 32(1), 138–156 (2006)
9. Hickey, T.J., Ju, Q., van Emden, M.H.: Interval arithmetic: From principles to implementation. Journal of the ACM 48(5), 1038–1068 (2001)
10. Hofschuster, W., Krämer, W., Neher, M.: C-XSC and Closely Related Software Packages. In: Cuyt, A., Krämer, W., Luther, W., Markstein, P. (eds.) Numerical Validation in Current Hardware Architectures. LNCS, vol. 5492, pp. 68–102. Springer, Heidelberg (2009)
11. IEEE Interval Standard Working Group - P1788, http://grouper.ieee.org/groups/1788/
12. Kearfott, R.B.: GlobSol user guide. Optimization Methods and Software 24(4-5), 687–708 (2009)
13. Knüppel, O.: PROFIL/BIAS A fast interval library. Computing 53(3-4), 277–287 (1994)
14. Krämer, W.: Introduction To The Maple Power Tool Intpakx. In: 12th Int. Conf. on Applications of Computer Algebra. IMI Bulgarian Academy of Sciences, vol. 1(4) (2007)
15. Lerch, M., Tischler, G., Wolff von Gudenberg, J., Hofschuster, W., Krämer, W.: filib++, a Fast Interval Library. ACM TOMS 32(2), 299–324 (2006)
16. Lin, Y., Stadtherr, M.A.: Validated solution of initial value problems for ODEs with interval parameters. In: Proc. of 2nd Reliable Engineering Computing (2006)
17. Lohner, R.: Computation of guaranteed enclosures for the solutions of ordinary initial and boundary value problems. In: Computational Ordinary Differential Equations, pp. 425–435. Clarendon Press, Oxford (1992)
18. Nedialkov, N.S.: Interval Tools for ODEs and DAEs. In: CD-Proc. of the 12th GAMM-IMACS SCAN 2006. IEEE Computer Society, Los Alamitos (2007)
19. Neumaier, A.: Complete Search in Continuous Global Optimization and Constraint Satisfaction. Acta Numerica, 271–369 (2004)
20. Papegay, Y., Daney, D., Merlet, J.-P.: Parallel Implementation of Interval Analysis for Equations Solving. In: Dongarra, J., Laforenza, D., Orlando, S. (eds.) EuroPVM/MPI 2003. LNCS, vol. 2840, pp. 555–559. Springer, Heidelberg (2003)

21. Pryce, J.D., Corliss, G.: Interval Arithmetic with Containment Sets. Computing 78(3), 251–276 (2006)
22. Revol, N., Rouillier, F.: Motivations for an Arbitrary Precision Interval Arithmetic and the MPFI Library. Reliable Computing 11(4), 275–290 (2005)
23. Rump, S.M.: INTLAB - INTerval LABoratory. In: Csendes, T. (ed.) Developments in Reliable Computing, pp. 77–104. Kluwer, Dordrecht (1999)
24. Sahinides, N.: Baron - Branch And Reduce Optimization Navigator 4.0 (2000)
25. Sun Studio C., http://docs.sun.com/source/819-3696/iapgCusing.html
26. van der Hoeven, J., Lecerf, G., Mourrain, B.: Mathemagix (2002), http://www.mathemagix.org

Efficient Evaluation of Large Polynomials

Charles E. Leiserson[1], Liyun Li[2], Marc Moreno Maza[2], and Yuzhen Xie[2]

[1] CSAIL, Massachussets Institute of Technology, Cambridge MA, USA
[2] Department of Computer Science, University of Western Ontario, London ON, Canada

Abstract. Minimizing the evaluation cost of a polynomial expression is a fundamental problem in computer science. We propose tools that, for a polynomial P given as the sum of its terms, compute a representation that permits a more efficient evaluation. Our algorithm runs in $d(nt)^{O(1)}$ bit operations plus $dt^{O(1)}$ operations in the base field where d, n and t are the total degree, number of variables and number of terms of P. Our experimental results show that our approach can handle much larger polynomials than other available software solutions. Moreover, our computed representation reduce the evaluation cost of P substantially.

Keywords: Multivariate polynomial evaluation, code optimization, Cilk++.

1 Introduction

If polynomials and matrices are the fundamental mathematical entities on which computer algebra algorithms operate, expression trees are the common data type that computer algebra systems use for all their symbolic objects. In MAPLE, by means of common subexpression elimination, an expression tree can be encoded as a directed acyclic graph (DAG) which can then be turned into a straight-line program (SLP), if required by the user. These two data-structures are well adapted when a polynomial (or a matrix depending on some variables) needs to be regarded as a function and evaluated at points which are not known in advance and whose coordinates may contain "symbolic expressions". This is a fundamental technique, for instance in the *Hensel-Newton lifting techniques* [6] which are used in many places in scientific computing.

In this work, we study and develop tools for manipulating polynomials as DAGs. The main goal is to be able to compute with polynomials that are far too large for being manipulated using standard encodings (such as lists of terms) and thus where the only hope is to represent them as DAGs. Our main tool is an algorithm that, for a polynomial P given as the sum its terms, computes a DAG representation which permits to evaluate P more efficiently in terms of work, data locality and parallelism. After introducing the related concepts in Section 2, this algorithm is presented in Section 3.

The initial motivation of this study arose from the following problem. Consider $a = a_m x^m + \cdots + a_1 x + a_0$ and $b = b_n x^n + \cdots + b_1 x + b_0$ two *generic* univariate polynomials of respective positive degrees m and n. Let $R(a, b)$ be the resultant of a and b. By generic polynomials, we mean here that $a_m, \ldots, a_1, a_0, b_n, \ldots, b_1, b_0$ are independent symbols. Suppose that $a_m, \ldots, a_1, a_0, b_n, \ldots, b_1, b_0$ are substituted to polynomials $\alpha_m, \ldots, \alpha_1, \alpha_0, \beta_n, \ldots, \beta_1, \beta_0$ in some other variables $c_1, \ldots, c_p$. Let us denote by $R(\alpha, \beta)$ the "specialized" resultant. If these α_i's and β_j's are large, then

K. Fukuda et al. (Eds.): ICMS 2010, LNCS 6327, pp. 342–353, 2010.

computing $R(\alpha, \beta)$ as a polynomial in $c_1, \ldots, c_p$, expressed as the sum of its terms, may become practically impossible. However, if $R(a, b)$ was originally computed as a DAG with $a_m, \ldots, a_1, a_0, b_n, \ldots, b_1, b_0$ as input and if the α_i's and β_j's are also given as DAGs with $c_1, \ldots, c_p$ as input, then one may still be able to manipulate $R(\alpha, \beta)$.

The techniques presented in this work do not make any assumptions about the input polynomials and, thus, they are not specific to resultant of generic polynomials. We simply use this example as an illustrative well-known problem in computer algebra.

Given an input polynomial expression, there are a number of approaches focusing on minimizing its size. Conventional common subexpression elimination techniques are typical methods to optimize an expression. However, as general-purpose applications, they are not suited for optimizing large polynomial expressions. In particular, they do not take full advantage of the algebraic properties of polynomials. Some researchers have developed special methods for making use of algebraic factorization in eliminating common subexpressions [1,7] but this is still not sufficient for minimizing the size of a polynomial expression. Indeed, such a polynomial may be irreducible. One economic and popular approach to reduce the size of polynomial expressions and facilitate their evaluation is the use of Horner's rule. This high-school trick for univariate polynomials has been extended to multivariate polynomials via different schemes [8,9,3,4]. However, it is difficult to compare these extensions and obtain an optimal scheme from any of them. Indeed, they all rely on selecting an appropriate ordering of the variables. Unfortunately, there are $n!$ possible orderings for n variables.

As shown in Section 4, our algorithm runs in polynomial time w.r.t. the number of variables, total degree and number of terms of the input polynomial expression. We have implemented our algorithm in the `Cilk++` concurrency platform. Our experimental results reported in Section 5 illustrate the effectiveness of our approach compared to other available software tools. For $2 \leq n, m \leq 7$, we have applied our techniques to the resultant $R(a, b)$ defined above. For $(n, m) = (7, 6)$, our optimized DAG representation can be evaluated sequentially 10 times faster than the input DAG representation. For that problem, none of code optimization software tools that we have tried produces a satisfactory result.

2 Syntactic Decomposition of a Polynomial

Let $\mathbb{K}$ be a field and let $x_1 > \cdots > x_n$ be n ordered variables, with $n \geq 1$. Define $X = \{x_1, \ldots, x_n\}$. We denote by $\mathbb{K}[X]$ the ring of polynomials with coefficients in $\mathbb{K}$ and with variables in X. For a non-zero polynomial $f \in \mathbb{K}[X]$, the set of its monomials is $\mathsf{mons}(f)$, thus f writes $f = \sum_{m \in \mathsf{mons}(f)} c_m\, m$, where, for all $m \in \mathsf{mons}(f)$, $c_m \in \mathbb{K}$ is the coefficient of f w.r.t. m. The set $\mathsf{terms}(f) = \{c_m\, m \mid m \in \mathsf{mons}(f)\}$ is the set of the terms of f. We use $\sharp\mathsf{terms}(f)$ to denote the number of terms in f.

Syntactic operations. Let $g, h \in \mathbb{K}[X]$. We say that gh is a *syntactic product*, and we write $g \odot h$, whenever $\sharp\mathsf{terms}(g\,h) = \sharp\mathsf{terms}(g) \cdot \sharp\mathsf{terms}(h)$ holds, that is, if no grouping of terms occurs when multiplying g and h. Similarly, we say that $g + h$ (resp. $g - h$) is a *syntactic sum* (resp. *syntactic difference*), written $g \oplus h$ (resp. $g \ominus h$), if we have $\sharp\mathsf{terms}(g+h) = \sharp\mathsf{terms}(g) + \sharp\mathsf{terms}(h)$ (resp. $\sharp\mathsf{terms}(g-h) = \sharp\mathsf{terms}(g) + \sharp\mathsf{terms}(h)$).

Syntactic factorization. For non-constant $f, g, h \in \mathbb{K}[X]$, we say that $g\,h$ is a *syntactic factorization* of f if $f = g \odot h$ holds. A syntactic factorization is said *trivial* if each factor is a single term. For a set of monomials $\mathcal{M} \subset \mathbb{K}[X]$ we say that $g\,h$ is a syntactic factorization of f *with respect to* $\mathcal{M}$ if $f = g \odot h$ and $\mathsf{mons}(g) \subseteq \mathcal{M}$ both hold.

Evaluation cost. Assume that $f \in \mathbb{K}[X]$ is non-constant. We call *evaluation cost* of f, denoted by $\mathsf{cost}(f)$, the minimum number of arithmetic operations necessary to evaluate f when $x_1, \ldots, x_n$ are replaced by actual values from $\mathbb{K}$ (or an extension field of $\mathbb{K}$). For a constant f we define $\mathsf{cost}(f) = 0$. Proposition 1 gives an obvious upper bound for $\mathsf{cost}(f)$. The proof, which is routine, is not reported here.

Proposition 1. *Let $f, g, h \in \mathbb{K}[X]$ be non-constant polynomials with total degrees d_f, d_g, d_h and numbers of terms t_f, t_g, t_h. Then, we have $\mathsf{cost}(f) \leq t_f(d_f + 1) - 1$. Moreover, if $g \odot h$ is a nontrivial syntactic factorization of f, then we have:*

$$\frac{\min(t_g, t_h)}{2}\,(1 + \mathsf{cost}(g) + \mathsf{cost}(h)) \leq t_f(d_f + 1) - 1. \tag{1}$$

Proposition 1 yields the following remark. Suppose that f is given in *expanded form*, that is, as the sum of its terms. Evaluating f, when $x_1, \ldots, x_n$ are replaced by actual values $k_1, \ldots, k_n \in \mathbb{K}$, amounts then to at most $t_f(d_f + 1) - 1$ arithmetic operations in $\mathbb{K}$. Assume $g \odot h$ is a syntactic factorization of f. Then evaluating both g and h at $k_1, \ldots, k_n$ may provide a speedup factor in the order of $\min(t_g, t_h)/2$. This observation motivates the introduction of the notions introduced in this section.

Syntactic decomposition. Let T be a binary tree whose internal nodes are the operators $+, -, \times$ and whose leaves belong to $\mathbb{K} \cup X$. Let p_T be the polynomial represented by T. We say that T is a *syntactic decomposition* of p_T if either (1), (2) or (3) holds:

(1) T consists of a single node which is p_T,
(2) if T has root $+$ (resp. $-$) with left subtree T_ℓ and right subtree T_r then we have:
  (a) T_ℓ, T_r are syntactic decompositions of two polynomials $p_{T_\ell}, p_{T_r} \in \mathbb{K}[X]$,
  (b) $p_T = p_{T_\ell} \oplus p_{T_r}$ (resp. $p_T = p_{T_\ell} \ominus p_{T_r}$) holds,
(3) if T has root $\times$, with left subtree T_ℓ and right subtree T_r then we have:
  (a) T_ℓ, T_r are syntactic decompositions of two polynomials $p_{T_\ell}, p_{T_r} \in \mathbb{K}[X]$,
  (b) $p_T = p_{T_\ell} \odot p_{T_r}$ holds.

We shall describe an algorithm that computes a syntactic decomposition of a polynomial. The design of this algorithm is guided by our objective of processing polynomials with many terms. Before presenting this algorithm, we make a few observations.

First, suppose that f admits a syntactic factorization $f = g \odot h$. Suppose also that the monomials of g and h are known, but not their coefficients. Then, one can easily deduce the coefficients of both g and h, see Proposition 3 hereafter.

Secondly, suppose that f admits a syntactic factorization $g\,h$ while nothing is known about g and h, except their numbers of terms. Then, one can set up a system of polynomial equations to compute the terms of g and h. For instance with $t_f = 4$ and $t_g = t_h = 2$, let $f = M + N + P + Q$, $g = X + Y$, $h = Z + T$. Up to renaming the terms of f, the following system must have a solution: $XZ = M$, $XT = P$, $YZ = N$ and $YT = Q$.

This implies that $M/P = N/Q$ holds. Then, one can check that $(g, g', M/g, N/g')$ is a solution for (X, Y, Z, T), where $g = \gcd(M, P)$ and $g' = \gcd(N, Q)$.

Thirdly, suppose that f admits a syntactic factorization $f = g \odot h$ while nothing is known about g, h including numbers of terms. In the worst case, all integer pairs (t_g, t_h) satisfying $t_g t_h = t_f$ need to be considered, leading to an algorithm which is exponential in t_f. This approach is too costly for our targeted large polynomials. Finally, in practice, we do not know whether f admits a syntactic factorization or not. Traversing every subset of $\mathsf{terms}(f)$ to test this property would lead to another combinatorial explosion.

3 The Hypergraph Method

Based on the previous observations, we develop the following strategy. Given a set of monomials $\mathcal{M}$, which we call *base monomial set*, we look for a polynomial p such that $\mathsf{terms}(p) \subseteq \mathsf{terms}(f)$, and p admits a syntactic factorization gh w.r.t $\mathcal{M}$. Replacing f by $f - p$ and repeating this construction would eventually produce a *partial syntactic factorization* of f, as defined below. The algorithm $\mathsf{ParSynFactorization}(f, \mathcal{M})$ states this strategy formally. We will discuss the choice and computation of the set $\mathcal{M}$ at the end of this section. The key idea of Algorithm ParSynFactorization is to consider a hypergraph $\mathsf{HG}(f, \mathcal{M})$ which detects "candidate syntactic factorizations".

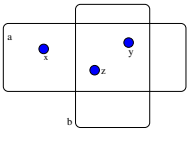
Partial syntactic factorization. A set of pairs $\{(g_1, h_1), (g_2, h_2), \ldots, (g_e, h_e)\}$ of polynomials and a polynomial r in $\mathbb{K}[x_1, \ldots, x_n]$ is a *partial syntactic factorization* of f w.r.t. $\mathcal{M}$ if the following conditions hold:

1. $\forall i = 1 \cdots e$, $\mathsf{mons}(g_i) \subseteq \mathcal{M}$,
2. no monomials in $\mathcal{M}$ divides a monomial of r,
3. $f = (g_1 \odot h_1) \oplus (g_2 \odot h_2) \oplus \cdots \oplus (g_e \odot h_e) \oplus r$ holds.

Assume that the above conditions hold. We say this partial syntactic factorization is *trivial* if each $g_i \odot h_i$ is a trivial syntactic factorization. Observe that all g_i for $1 \leq i \leq e$ and r do not admit any nontrivial partial syntactic factorization w.r.t. $\mathcal{M}$, whereas it is possible that one of h_i's admits a nontrivial partial syntactic factorization.

Hypergraph $\mathsf{HG}(f, \mathcal{M})$. Given a polynomial f and a set of monomials $\mathcal{M}$, we construct a hypergraph $\mathsf{HG}(f, \mathcal{M})$ as follows. Its vertex set is $\mathcal{V} = \mathcal{M}$ and its hyperedge set $\mathcal{E}$ consists of all nonempty sets $E_q := \{m \in \mathcal{M} \mid m\,q \in \mathsf{mons}(f)\}$, for an arbitrary monomial q. Observe that if a term of f is not the multiple of any monomials in $\mathcal{M}$, then it is not involved in the construction of $\mathsf{HG}(f, \mathcal{M})$. We call such a term *isolated*.

Example. For $f = ay + az + by + bz + ax + aw \in \mathbb{Q}[x, y, z, w, a, b]$ and $\mathcal{M} = \{x, y, z\}$, the hypergraph $\mathsf{HG}(f, \mathcal{M})$ has 3 vertices x, y, z and 2 hyperedges $E_a = \{x, y, z\}$ and $E_b = \{y, z\}$. A partial syntactic factorization of f w.r.t $\mathcal{M}$ consists of $\{(y + z, a + b), (x, a)\}$ and aw.



We observe that a straightforward algorithm computes $\mathsf{HG}(f, \mathcal{M})$ in $O(|\mathcal{M}|\, n\, t)$ bit operations. The following proposition, whose proof is immediate, suggests how $\mathsf{HG}(f, \mathcal{M})$ can be used to compute a partial syntactic factorization of f w.r.t. $\mathcal{M}$.

Proposition 2. *Let $f, g, h \in \mathbb{K}[X]$ such that $f = g \odot h$ and $\mathsf{mons}(g) \subseteq \mathcal{M}$ both hold. Then, the intersection of all E_q, for $q \in \mathsf{mons}(h)$, contains $\mathsf{mons}(g)$.*

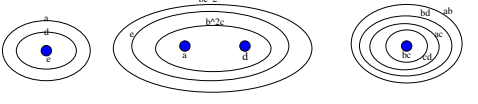
Before stating Algorithm ParSynFactorization, we make a simple observation.

Proposition 3. *Let $F_1, F_2, \ldots, F_c$ be the monomials and $f_1, f_2, \ldots, f_c$ be the coefficients of a polynomial $f \in \mathbb{K}[X]$, such that $f = \sum_{i=1}^{c} f_i F_i$. Let $a, b > 0$ be two integers such that $c = ab$. Given monomials $G_1, G_2, \ldots, G_a$ and $H_1, H_2, \ldots, H_b$ such that the products $G_i H_j$ are all in $\mathsf{mons}(f)$ and are pairwise different. Then, within $O(ab)$ operations in $\mathbb{K}$ and $O(a^2 b^2 n)$ bit operations, one can decide whether $f = g \odot h$, $\mathsf{mons}(g) = \{G_1, G_2, \ldots, G_a\}$ and $\mathsf{mons}(h) = \{H_1, H_2, \ldots, H_b\}$ all hold. Moreover, if such a syntactic factorization exists it can be computed within the same time bound.*

Proof. Define $g = \sum_{i=1}^{a} g_i G_i$ and $h = \sum_{i=1}^{b} h_i H_i$ where $g_1, \ldots, g_a$ and $h_1, \ldots, h_b$ are unknown coefficients. The system to be solved is $g_i h_j = f_{ij}$, for all $i = 1 \cdots a$ and all $j = 1 \cdots b$ where f_{ij} is the coefficient of $G_i H_j$ in p. To set up this system $g_i h_j = f_{ij}$, one needs to locate each monomial $G_i H_j$ in $\mathsf{mons}(f)$. Assuming that each exponent of a monomial is a machine word, any two monomials of $\mathbb{K}[x_1, \ldots, x_n]$ are compared within $O(n)$ bit operations. Hence, each of these ab monomials can be located in $\{F_1, F_2, \ldots, F_c\}$ within $O(cn)$ bit operations and the system is set up within $O(a^2 b^2 n)$ bit operations. We observe that if $f = g \odot h$ holds, one can freely set g_1 to 1 since the coefficients are in a field. This allows us to deduce $h_1, \ldots, h_b$ and then $g_2, \ldots, g_a$ using $a + b - 1$ equations. The remaining equations of the system should be used to check if these values of $h_1, \ldots, h_b$ and $g_2, \ldots, g_a$ lead indeed to a solution. Overall, for each of the ab equations one simply needs to perform one operation in $\mathbb{K}$.

Remark on Algorithm 1. Following the property of the hypergraph $\mathsf{HG}(f, \mathcal{M})$ given by Proposition 2, we use a greedy strategy and search for the largest hyperedge intersection in $\mathsf{HG}(f, \mathcal{M})$. Once such intersection is found, we build a candidate syntactic factorization from it. However, it is possible that the equality in Line 12 does not hold. For example, when $\mathcal{M} = Q = \{a, b\}$, we have $|N| = 3 \neq 2 \times 2 = |M| \cdot |Q|$. When the equality $|N| = |M| \cdot |Q|$ holds, there is still a possibility that the system set up as in the proof of Proposition 3 does not have solutions. For example, when $\mathcal{M} = \{a, b\}, Q = \{c, d\}$ and $p = ac + ad + bc + 2\,bd$. Nevertheless, the termination of the while loop in Line 10 is ensured by the following observation. When $|Q| = 1$, the equality $|N| = |M| \cdot |Q|$ always holds and the system set up as in the proof of Proposition 3 always has a solution. After extracting a syntactic factorization from the hypergraph $\mathsf{HG}(f, \mathcal{M})$, we update the hypergraph by removing all monomials in the set N and keep extracting syntactic factorizations from the hypergraph until no hyperedges remain.

Example. Consider $f = 3ab^2c + 5abc^2 + 2ae + 6b^2cd + 10bc^2d + 4de + s$. Our base monomial set $\mathcal{M}$ is chosen as $\{a, bc, e, d\}$. Following Algorithm 1, we first construct the hypergraph $\mathsf{HG}(f, \mathcal{M})$ w.r.t. which the term s is isolated.



Input : a polynomial f given as a sorted set $\mathsf{terms}(f)$, a monomial set $\mathcal{M}$
Output : a partial syntactic factorization of f w.r.t $\mathcal{M}$

$\mathcal{T} \leftarrow \mathsf{terms}(f)$, $\mathcal{F} \leftarrow \emptyset$;
$r \leftarrow \sum_{t\in\mathcal{I}} t$ where $\mathcal{I} = \{t \in \mathsf{terms}(f) \mid (\forall m \in \mathcal{M})\ m \nmid t\}$;
compute the hypergraph $\mathsf{HG}(f, \mathcal{M}) = (\mathcal{V}, \mathcal{E})$;
while $\mathcal{E}$ *is not empty* **do**
  if $\mathcal{E}$ *contains only one edge* E_q **then** $Q \leftarrow \{q\}$, $M \leftarrow E_q$;
  else
    find q, q' such that $E_q \cap E_{q'}$ has the maximal cardinality;
    $M \leftarrow E_q \cap E_{q'}$, $Q \leftarrow \emptyset$;
    if $|M| < 1$ **then** find the largest edge E_q, $M \leftarrow E_q$, $Q \leftarrow \{q\}$;
    else for $E_q \in \mathcal{E}$ **do if** $M \subseteq E_q$ **then** $Q \leftarrow Q \cup \{q\}$;
  while *true* **do**
    $N = \{mq \mid m \in M, q \in Q\}$;
    if $|N| = |M| \cdot |Q|$ **then**
      let p be the polynomial such that $\mathsf{mons}(p) = N$ and $\mathsf{terms}(p) \subseteq \mathcal{T}$;
      if $p = g \odot h$ *with* $\mathsf{mons}(g) = M$ *and* $\mathsf{mons}(h) = Q$ **then**
        compute g, h (Proposition 3); break;
    else randomly choose $q \in Q$, $Q \leftarrow Q \setminus \{q\}$, $M \leftarrow \cap_{q\in Q} E_q$;
  for $E_q \in \mathcal{E}$ **do**
    for $m' \in N$ **do**
      if $q \mid m'$ **then** $E_q \leftarrow E_q \setminus \{m'/q\}$;
    if $E_q = \emptyset$ **then** $\mathcal{E} \leftarrow \mathcal{E} \setminus \{E_q\}$;
  $\mathcal{T} \leftarrow \mathcal{T} \setminus \mathsf{terms}(p)$, $\mathcal{F} \leftarrow \mathcal{F} \cup \{g \odot h\}$;
return $\mathcal{F}, r$

Algorithm 1. ParSynFactorization

The largest edge intersection is $M = \{a, d\} = E_{b^2c} \cap E_{bc^2} \cap E_e$ yielding $Q = \{b^2c, bc^2, e\}$. The set N is $\{mq \mid m \in M, q \in Q\} = \{ab^2c, abc^2, ae, b^2cd, bc^2d, de\}$. The cardinality of N equals the product of the cardinalities of M and of Q. So we keep searching for a polynomial p with N as monomial set and with $\mathsf{terms}(p) \subseteq \mathsf{terms}(f)$. By scanning $\mathsf{terms}(f)$ we obtain $p = 3ab^2c + 5abc^2 + 2ae + 6b^2cd + 10bc^2d + 4de$. Now we look for polynomials g, h with respective monomial sets M, Q and such that $p = g \odot h$ holds. The following equality yields a system of equations whose unknowns are the coefficients of g and h: $(g_1a + g_2d)(h_1b^2c + h_2bc^2 + h_3e) = 3ab^2c + 5abc^2 + 2ae + 6b^2cd + 10bc^2d + 4de$. As described in Proposition 3, we can freely set g_1 to 1 and then use 4 out of the 6 equations to deduce h_1, h_2, h_3, g_2; these computed values must verify the remaining equations for $p = g \odot h$ to hold, which is the case here.

$$\begin{cases} g_1h_1 = 3 \\ g_1h_2 = 5 \\ g_1h_3 = 2 \\ g_2h_1 = 6 \end{cases} \overset{g_1=1}{\Longrightarrow} \begin{cases} g_1 = 1 \\ g_2 = 2 \\ h_1 = 3 \\ h_2 = 5 \\ h_3 = 2 \end{cases} \Rightarrow \begin{cases} g_2h_2 = 10 \\ g_2h_3 = 4 \end{cases}$$

Now we have found a syntactic factorization of p. We update each edge in the hypergraph, which, in this example, will make the hypergraph empty. After adding $(a + 2d, 3b^2c + 5bc^2 + 2e)$ to $\mathcal{F}$, the algorithm terminates with $\mathcal{F}, s$ as output.

One may notice that in Example 3, $h = 3b^2c + 5bc^2 + 2e$ also admits a nontrivial partial syntactical factorization. Computing it will produce a syntactic decomposition of f. When a polynomial which does not admit any nontrivial partial syntactical factorizations w.r.t $\mathcal{M}$ is hit, for instance, g_i or r in a partial syntactic factorization, we directly convert it to an expression tree. To this end, we assume that there is a procedure $\mathsf{ExpressionTree}(f)$ that outputs an expression tree of a given polynomial f. Algorithm 2, which we give for the only purpose of being precise, states the most straight forward way to implement $\mathsf{ExpressionTree}(f)$. Then, Algorithm 3 formally states how to produce a syntactic decomposition of a given polynomial.

Input : a polynomial f given as $\mathsf{terms}(f) = \{t_1, t_2, \ldots, t_s\}$
Output: an expression tree whose value equals f

if $\sharp\mathsf{terms}(f) = 1$ *say* $f = c \cdot x_1^{d_1} x_2^{d_2} \cdots x_k^{d_k}$ **then**
  for $i \leftarrow 1$ **to** k **do**
    $T_i \leftarrow x_i$;
    for $j \leftarrow 2$ **to** d_i **do**
      $T_{i,\ell} \leftarrow T_i, \mathsf{root}(T_i) \leftarrow \times, T_{i,r} \leftarrow x_i$;
  $T \leftarrow$ empty tree, $\mathsf{root}(T) \leftarrow \times, T_\ell \leftarrow c, T_r \leftarrow T_1$;
  for $i \leftarrow 2$ **to** k **do**
    $T_\ell \leftarrow T, \mathsf{root}(T) \leftarrow \times, T_r \leftarrow T_i$;
else
  $k \leftarrow s/2, f_1 \leftarrow \sum_{i=1}^{k} t_i, f_2 \leftarrow \sum_{i=k+1}^{s} t_i$;
  $T_1 \leftarrow \mathsf{ExpressionTree}(f_1)$;
  $T_2 \leftarrow \mathsf{ExpressionTree}(f_2)$;
  $\mathsf{root}(T) \leftarrow +, T_\ell \leftarrow T_1, T_r \leftarrow T_2$;

Algorithm 2. ExpressionTree

We have stated all the algorithms that support the construction of a syntactic decomposition except for the computation of the base monomial set $\mathcal{M}$. Note that in Algorithm 1 our main strategy is to keep extracting syntactic factorizations from the hypergraph $\mathsf{HG}(f, \mathcal{M})$. For all the syntactic factorizations $g \odot h$ computed in this manner, we have $\mathsf{mons}(g) \subseteq \mathcal{M}$. Therefore, to discover all the possible syntactic factorizations in $\mathsf{HG}(f, \mathcal{M})$, the base monomial set should be chosen so as to contain all the monomials from which a syntactic factorization may be derived. The most obvious choice is to consider the set G of all non constant gcds of any two distinct terms of f. However, $|G|$ could be quadratic in $\#\mathsf{terms}(f)$, which would be a bottleneck on large polynomials f. Our strategy is to choose for $\mathcal{M}$ as the set of the minimal elements of G for the divisibility relation. A straightforward algorithm computes this set $\mathcal{M}$ within $O(t^4 n)$ operations in $\mathbb{K}$; indeed $|\mathcal{M}|$ fits in $|G| = O(t^2)$. In practice, $\mathcal{M}$ is much smaller than G

Input : a polynomial f given as $\mathsf{terms}(f)$
Output: a syntactic decomposition of f

1 compute the base monomial set $\mathcal{M}$ for f;
2 **if** $\mathcal{M} = \emptyset$ **then** return $\mathsf{ExpressionTree}(f)$;
3 **else**
4 $\quad \mathcal{F}, r \leftarrow \mathsf{ParSynFactorization}(f, \mathcal{M})$;
5 $\quad$ **for** $i \leftarrow 1$ **to** $|\mathcal{F}|$ **do**
6 $\quad\quad (g_i, h_i) \leftarrow \mathcal{F}_i, T_i \leftarrow$ empty tree, $\mathsf{root}(T_i) \leftarrow \times$;
7 $\quad\quad T_{i,\ell} \leftarrow \mathsf{ExpressionTree}(g_i)$;
8 $\quad\quad T_{i,r} \leftarrow \mathsf{SyntacticDecomposition}(h_i)$;
9 $\quad T \leftarrow$ empty tree, $\mathsf{root}(T) \leftarrow +, T_\ell \leftarrow \mathsf{ExpressionTree}(r), T_r \leftarrow T_1$;
10 $\quad$ **for** $i \leftarrow 2$ **to** $|\mathcal{F}|$ **do**
11 $\quad\quad T_\ell \leftarrow T, \mathsf{root}(T) \leftarrow +, T_r \leftarrow T_i$;

Algorithm 3. SyntacticDecomposition

(for large dense polynomials, $\mathcal{M} = X$ holds) and this choice is very effective. However, since we aim at manipulating large polynomials, the set G can be so large that its size can be a memory bottleneck when computing $\mathcal{M}$. In [2] we address this question: we propose a divide-and-conquer algorithm which computes $\mathcal{M}$ directly from f without storing the whole set G in memory. In addition, the parallel implementation in `Cilk+` shows linear speed-up on 32 cores for sufficiently large input.

4 Complexity Estimates

Given a polynomial f of t terms with total degree d in $\mathbb{K}[X]$, we analyze the running time for Algorithm 3 to compute a syntactic decomposition of f. Assuming that each exponent in a monomial is encoded by a machine word, each operation (GCD, division) on a pair of monomials of $\mathbb{K}[X]$ requires $O(n)$ bit operations. Due to the different manners of constructing a base monomial set, we keep $\mu := |\mathcal{M}|$ as an input complexity measure. As mentioned in Section 3, $\mathsf{HG}(f, \mathcal{M})$ is constructed within $O(\mu t n)$ bit operations. This hypergraph contains μ vertices and $O(\mu t)$ hyperedges. We first proceed by analyzing Algorithm 1. To do so, we follow its steps.

- The "isolated" polynomial r can be easily computed by testing the divisibility of each term in f w.r.t each monomial in $\mathcal{M}$, i.e. in $O(\mu \cdot t \cdot n)$ bit operations.
- Each hyperedge in $\mathsf{HG}(f, \mathcal{M})$ is a subset of $\mathcal{M}$. The intersection of two hyperedges can then be computed in $\mu \cdot n$ bit operations. Thus we need $O((\mu t)^2 \cdot \mu n) = O(\mu^3 t^2 n)$ bit operations to find the largest intersection M (Line 7).
- If M is empty, we traverse all the hyperedges in $\mathsf{HG}(f, \mathcal{M})$ to find the largest one. This takes no more than $\mu t \cdot \mu n = \mu^2 t n$ bit operations (Line 9).
- If M is not empty, we traverse all the hyperedges in $\mathsf{HG}(f, \mathcal{M})$ to test if M is a subset of it. This takes at most $\mu t \cdot \mu n = \mu^2 t n$ bit operations (Line 10).
- Line 6 to Line 10 takes $O(\mu^3 t^2 n)$ bit operations.

- The set N can be computed in $\mu \cdot \mu t \cdot n$ bit operations (Line 12).
- by Proposition 3, the candidate syntactic factorization can be either computed or rejected in $O(|M|^2 \cdot |Q|^2 n) = O(\mu^4 t^2 n)$ bit operations and $O(\mu^2 t)$ operations in $\mathbb{K}$ (Lines 13 to 16).
- If $|N| \neq |M| \cdot |Q|$ or the candidate syntactic factorization is rejected, we remove one element from Q and repeat the work in Line 12 to Line 16. This while loop ends before or when $|Q| = 1$, hence it iterates at most $|Q|$ times. So the bit operations of the while loop are in $O(\mu^4 t^2 n \cdot \mu t) = O(\mu^5 t^3 n)$ while operations in $\mathbb{K}$ are within $O(\mu^2 t \cdot \mu t) = O(\mu^3 t^2)$ (Line 11 to Line 17).
- We update the hypergraph by removing the monomials in the constructed syntactic factorization. The two nested for loops in Line 18 to Line 21 take $O(|\mathcal{E}| \cdot |N| \cdot n) = O(|\mathcal{E}| \cdot |M| \cdot |Q| \cdot n) = O(\mu t \cdot \mu \cdot \mu t \cdot n) = O(\mu^3 t^2 n)$ bit operations.
- Each time a syntactic factorization is found, at least one monomial in $\mathsf{mons}(f)$ is removed from the hypergraph $\mathsf{HG}(f, \mathcal{M})$. So the while loop from Line 4 to Line 22 would terminate in $O(t)$ iterations.

Overall, Algorithm 1 takes $O(\mu^5 t^4 n)$ bit operations and $O(\mu^3 t^3)$ operations in $\mathbb{K}$. One easily checks from Algorithm 2 that an expression tree can be computed from f (where f has t terms and total degree d) within in $O(ndt)$ bit operations. In the sequel of this section, we analyze Algorithm 3. We make two preliminary observations. First, for the input polynomial f, the cost of computing a base monomial set can be covered by the cost of finding a partial syntactic factorization of f. Secondly, the expression trees of all g_i's (Line 7) and of the isolated polynomial r (Line 9) can be computed within $O(ndt)$ operations. Now, we shall establish an equation that rules the running time of Algorithm 3. Assume that $\mathcal{F}$ in Line 4 contains e syntactic factorizations. For each g_i, h_i such that $(g_i, h_i) \in \mathcal{F}$, let the number of terms in h_i be t_i and the total degree of h_i be d_i. By the specification of the partial syntactic factorization, we have $\sum_{i=1}^{e} t_i \leq t$. It is easy to show that $d_i \leq d - 1$ holds for $1 \leq i \leq e$ as total degree of each g_i is at least 1. We recursively call Algorithm 3 on all h_i's. Let $T_b(t, d, n)(T_{\mathbb{K}}(t, d, n))$ be the number of bit operations (operations in $\mathbb{K}$) performed by Algorithm 3. We have the following recurrence relation,

$$T_b(t, d, n) = \sum_{i=1}^{e} T_b(t_i, d_i, n) + O(\mu^5 t^4 n), T_{\mathbb{K}}(t, d, n) = \sum_{i=1}^{e} T_{\mathbb{K}}(t_i, d_i, n) + O(\mu^3 t^3),$$

from which we derive that $T_b(t, d, n)$ is within $O(\mu^5 t^4 n d)$ and $T_{\mathbb{K}}(t, d, n)$ is within $O(\mu^3 t^3 d)$. Next, one can verify that if the base monomial set $\mathcal{M}$ is chosen as the set of the minimal elements of all the pairwise gcd's of monomials of f, where $\mu = O(t^2)$, then a syntactic decomposition of f can be computed in $O(t^{14} n d)$ bit operations and $O(t^9 d)$ operations in $\mathbb{K}$. If the base monomial set is simply set to be $X = \{x_1, x_2, \ldots, x_n\}$, then a syntactic decomposition of f can be found in $O(t^4 n^6 d)$ bit operations and $O(t^3 n^3 d)$ operations in $\mathbb{K}$.

5 Experimental Results

In this section we discuss the performances of different software tools for reducing the evaluation cost of large polynomials. These tools are based respectively on a multivariate Horner's scheme [3], the `optimize` function with `tryhard` option provided by

the computer algebra system Maple and our algorithm presented in Section 3. As described in the introduction, we use the evaluation of resultants of generic polynomials as a driving example. We have implemented our algorithm in the `Cilk++` programming language. We report on different performance measures of our optimized DAG representations as well as those obtained with the other software tools.

Evaluation cost. Figure 1 shows the total number of internal nodes of a DAG representing the resultant $R(a, b)$ of two generic polynomials $a = a_m x^m + \cdots + a_0$ and $b = b_n x^n + \cdots + b_0$ of degrees m and n, after optimizing this DAG by different approaches. The number of internal nodes of this DAG measures the cost of evaluating $R(a, b)$ after specializing the variables $a_m, \ldots, a_0, b_n \ldots, b_0$. The first two columns of Figure 1 gives m and n. The third column indicates the number of monomials appearing in $R(a, b)$. The number of internal nodes of the input DAG, as computed by MAPLE, is given by the fourth column (Input). The fifth column (Horner) is the evaluation cost (number of internal nodes) of the DAG after MAPLE's multivariate Horner's rule is applied. The sixth column (tryhard) records the evaluation cost after MAPLE's optimize function (with the tryhard option) is applied. The last two columns reports the evaluation cost of the DAG computed by our *hypergraph method* (HG) before and after removing common subexpressions. Indeed, our hypergraph method requires this post-processing (for which we use standard techniques running in time linear w.r.t. input size) to produce better results. We note that the evaluation cost of the DAG returned by HG + CSE is less than the ones obtained with the Horner's rule and MAPLE's `optimize` functions.

m	n	#Mon	Input	Horner	tryhard	HG	HG + CSE
4	4	219	1876	977	721	899	549
5	4	549	5199	2673	1496	2211	1263
5	5	1696	18185	7779	4056	7134	3543
6	4	1233	13221	6539	3230	4853	2547
6	5	4605	54269	22779	10678	18861	8432
6	6	14869	190890	69909	31760	63492	24701
7	4	2562	30438	14948	6707	9862	4905
7	5	11380	146988	61399	27363	45546	19148
7	6	43166	601633	219341	-	179870	65770

Fig. 1. Cost to evaluate a DAG by different approaches

Figure 2 shows the timing in seconds that each approach takes to optimize the DAGs analyzed in Figure 1. The first three columns of Figure 2 have the same meaning as in Figure 1. The columns (Horner), (tryhard) show the timing of optimizing these DAGs. The last column (HG) shows the timing to produce the syntactic decompositions with our `Cilk++` implementation on multicores using 1, 4, 8 and 16 cores. All the sequential benchmarks (Horner, tryhard) were conducted on a 64bit Intel Pentium VI Quad CPU 2.40 GHZ machine with 4 MB L2 cache and 3 GB main memory. The parallel benchmarks were carried out on a 16-core machine at `SHARCNET` (www.sharcnet.ca) with 128 GB memory in total and 8×4096 KB of L2 cache (each integrated by 2 cores). All the processors are Intel Xeon E7340 @ 2.40GHz.

As the input size grows, the timing of the MAPLE `Optimize` command (with tryhard option) grows dramatically and takes more than 40 hours to optimize the resultant of two generic polynomials with degrees 6 and 6. For the generic polynomials with degree 7 and 6, it does not terminate after 5 days. For the largest input (7, 6), our algorithm completes within 5 minutes on one core. Our preliminary implementation shows a speedup around 8 when 16 cores are available. The parallelization of our algorithm is still work in progress (for instance, in the current implementation Algorithm 3 has **not** been parallelized yet). We are further improving the implementation and leave for a future paper reporting the parallelization of our algorithms.

m	n	#Mon	Horner	tryhard	HG (# cores $= 1, 4, 8, 16$)			
4	4	219	0.116	7.776	0.017	0.019	0.020	0.023
5	4	549	0.332	49.207	0.092	0.073	0.068	0.067
5	5	1696	1.276	868.118	0.499	0.344	0.280	0.250
6	4	1233	0.988	363.970	0.383	0.249	0.213	0.188
6	5	4605	4.868	8658.037	3.267	1.477	1.103	0.940
6	6	14869	24.378	145602.915	29.130	9.946	6.568	4.712
7	4	2562	4.377	1459.343	1.418	0.745	0.603	0.477
7	5	11380	24.305	98225.730	22.101	7.687	5.106	3.680
7	6	43166	108.035	>136 hours	273.963	82.497	49.067	31.722

Fig. 2. Timing to optimize large polynomials

Evaluation schedule. Let T be a syntactic decomposition of an input polynomial f. Targeting multi-core evaluation, our objective is to decompose T into p sub-DAGs, given a fixed parameter p, the number of available processors. Ideally, we want these sub-DAGs to be balanced in size such that the "span" of the intended parallel evaluation can be minimized. These sub-DAGs should also be independent to each other in the sense that the evaluation of one does not depend on the evaluation of another. In this manner, these sub-DAGs can be assigned to different processors. When p processors are available, we call "p-schedule" such a decomposition. We report on the 4 and 8-schedules generated from our syntactic decompositions. The column "T" records the size of a syntactic decomposition, counting the number of nodes. The column "#CS" indicates the number of common subexpressions. We notice that the amount of work assigned to each sub-DAG is balanced. However, scheduling the evaluation of the common subexpressions is still work in progress.

Benchmarking generated code. We generated 4-schedules of our syntactic decompositions and compared with three other methods for evaluating our test polynomials on a

m	n	T	#CS	4-schedule	8-schedule
6	5	8432	1385	1782, 1739, 1760, 1757	836, 889, 884, 881, 892, 886, 886, 869
6	6	24701	4388	4939, 5114, 5063, 5194	2436, 2498, 2496, 2606, 2535, 2615, 2552, 2555
7	5	19148	3058	3900, 4045, 4106, 4054	1999, 2049, 2078, 1904, 2044, 2019, 1974, 2020
7	6	65770	10958	13644, 13253, 14233, 13705	6710, 6449, 7117, 6802, 6938, 7025, 6807, 6968

Fig. 3. Parallel evaluation schedule

large number of uniformly generated random points over Z/pZ where $p = 2147483647$ is the largest 31-bit prime number. Our experimental data are summarized in Figure 4. Out the four different evaluation methods, the first three are sequential and are based on the following DAGs: the original MAPLE DAG (labeled as Input), the DAG computed by our hypergraph method (labeled as HG), the HG DAG further optimized by CSE (labeled as HG + CSE). The last method uses the 4-schedule generated from the DAG obtained by HG + CSE. All these evaluation schemes are automatically generated as a list of SLPs. When an SLP is generated as one procedure in a source file, the file size grows linearly with the number of lines in this SLP. We observe that gcc 4.2.4 failed to compile the resultant of generic polynomials of degree 6 and 6 (the optimization level is 2). In Figure 4, we report the timings of the four approaches to evaluate the input at $10K$ and $100K$ points. The first four data rows report timings where the gcc optimization level is 0 during the compilation, and the last row shows the timings with the optimization at level 2. We observe that the optimization level affects the evaluation time by a factor of 2, for each of the four methods. Among the four methods, the 4-schedule method is the fastest and it is about 20 times faster than the first method.

m	n	#point	Input	HG	HG+CSE	4-schedule	#point	Input	HG	HG+CSE	4-schedule
6	5	10K	14.490	2.675	1.816	0.997	100K	144.838	26.681	18.103	9.343
6	6	10K	57.853	18.618	4.281	2.851	100K	577.624	185.883	42.788	28.716
7	5	10K	46.180	11.423	4.053	2.104	100K	461.981	114.026	40.526	19.560
7	6	10K	190.397	54.552	13.896	8.479	100K	1902.813	545.569	138.656	81.270
6	5	10K	6.611	1.241	0.836	0.435	100K	66.043	12.377	8.426	4.358

Fig. 4. Timing to evaluate large polynomials

References

1. Breuer, M.A.: Generation of optimal code for expressions via factorization. ACM Commun. 12(6), 333–340 (1969)
2. Leiserson, C.E., Li, L., Moreno Maza, M., Xie, Y.: Parallel computation of the minimal elements of a poset. In: Proc. PASCO 2010. ACM Press, New York (2010)
3. Carnicer, J., Gasca, M.: Evaluation of multivariate polynomials and their derivatives. Mathematics of Computation 54(189), 231–243 (1990)
4. Ceberio, M., Kreinovich, V.: Greedy algorithms for optimizing multivariate horner schemes. SIGSAM Bull 38(1), 8–15 (2004)
5. Intel Corporation. Cilk++., http://www.cilk.com/
6. von zur Gathen, J., Gerhard, J.: Modern Computer Algebra. Cambridge Univ. Press, Cambridge (1999)
7. Hosangadi, A., Fallah, F., Kastner, R.: Factoring and eliminating common subexpressions in polynomial expressions. In: ICCAD 2004, pp. 169–174. IEEE Computer Society, Los Alamitos (2004)
8. Peña, J.M.: On the multivariate Horner scheme. SIAM J. Numer. Anal. 37(4), 1186–1197 (2000)
9. Peña, J.M., Sauer, T.: On the multivariate Horner scheme ii: running error analysis. Computing 65(4), 313–322 (2000)

Communicating Functional Expressions from *Mathematica* to C-XSC*

Evgenija D. Popova[1] and Walter Krämer[2]

[1] Institute of Mathematics & Informatics, Bulgarian Academy of Sciences, Acad. G. Bonchev str., block 8, 1113 Sofia, Bulgaria
epopova@bio.bas.bg

[2] WRSWT, Bergische Universität Wuppertal, Faculty of Mathematics and Natural Sciences, Gaußstr. 20, D-42097 Wuppertal, Germany
kraemer@math.uni-wuppertal.de

Abstract. This work focuses on a mechanism (and software) which communicates (via *MathLink* protocol) and provides compatibility between the representation of nonlinear functions specified as *Mathematica* expressions and objects of suitable classes supported by the C-XSC automatic differentiation modules. The application of the developed communication software is demonstrated by *MathLink* compatible programs embedding in *Mathematica* the C-XSC modules for automatic differentiation as packages. The design methodology, some implementation issues and the use of the developed software are discussed.

1 Introduction

Modeling, design and simulation of real-life problems often require integration of symbolic and numerical computations, visualization tools, etc. Providing interoperability between the general-purpose environments for scientific/technical computing (like *Mathematica*, Maple, etc.), which possess several features not attributable to the compiled languages, and the interval software, developed in some compiled language for efficiency reasons, is highly desirable due to the many positive consequences for both environments [10]. Computer-assisted proofs is another field that requires a collaboration of both environments [4].

Although many interval methods are brought to reliable, high-quality and fast implementations, they remain isolated software systems. The software comparing studies are also hampered by the diversity in the implementation supporting environments and in the interval data representation (see e.g. [3] and the references given therein). Communication based on file reading/writing operations to exchange ordinary text is not suitable for interval computations. The problems that have to be solved are enlisted in [3]. Due to the necessity of avoiding decimal-to-binary/binary-to-decimal input/output conversions of floating-point data, the most suitable approach when providing interoperability between interval software is that based on communication protocols [10].

* This work is partially supported by the DFG grant GZ: KR1612/7-1, AOBJ: 570029.

K. Fukuda et al. (Eds.): ICMS 2010, LNCS 6327, pp. 354–365, 2010.

In order to promote the interoperability between interval supporting environments as an alternative to building complicated systems from scratch we have initiated a concrete study. Its aim is establishing a general framework for delivering external specific arithmetic tools and implementations of interval algorithms, provided by the C++ class library C-XSC [8,9], in the general purpose environment of *Mathematica* [12] via the communication protocol *MathLink* [5]. *Mathematica* was chosen as a genera-purpose computer algebra system providing the most flexible support of interval arithmetic, e.g. supporting exact interval arithmetic. *Mathematica* also has its own high-level interface standard for interprogram communication. It is desirable to expand *Mathematica*'s interval functionality and to help new developments by connecting to external interval libraries. C-XSC is an open source library facilitating the implementation of reliable numerical methods. It supplies some arithmetic tools that are not available in other interval libraries, such as accurate dot product expressions, or complex interval arithmetic that is not supported in *Mathematica*. A lot of problem solvers delivering validated results are involved[1] in the distribution of C-XSC or provided as external software.

A preliminary research demonstrates the transparency in the communication of floating-point (interval) data between *Mathematica* and C-XSC [10]. The article contains a lot of technical details giving the reader a better feeling how easy or complicated it is to establish a connection between the two environments. However, some important problems such as isolating zeros of nonlinear functions (one- and multi-dimensional), guaranteed global optimization (one- and multi-dimensional), and automatic differentiation (AD) of interval functions, for which there are C-XSC modules, as well as many other problems, are formulated in terms of nonlinear functions. Therefore, the communication of functional expressions from *Mathematica* to C-XSC is an important step toward creating a general interface that allows using all application functions delivered with C-XSC from within *Mathematica*.

The goal of this paper is to discuss some issues related to communication of functional expressions from *Mathematica* to C-XSC and to present the software tools developed for this purpose. Since all of C-XSC problem solvers for nonlinear functions are built on top of the AD arithmetic supported by three C-XSC modules, the particular goal of the present work is to provide compatibility between the functional expressions specified in *Mathematica* and the specific data type objects used in the C-XSC AD modules. Section 2 presents shortly the C-XSC AD modules. Some aspects of C-XSC that are important for the present development are mentioned. The notion of expression in *Mathematica* is presented in the next section. Section 4 is devoted to the basic software communicating nonlinear functions, specified in *Mathematica*, to the respective C-XSC class objects. The design principles and the implementation are discussed. As a proof of concept illustrating the application of this tool, three interfacing *MathLink* programs are developed. They embed in *Mathematica* the corresponding C-XSC modules for automatic differentiation as packages. Section 5 describes their use.

[1] The former C++ Toolbox for Verified Computing [7].

2 C-XSC Background

C-XSC [8,9] contains modules for AD based on interval operations to get guaranteed enclosures for the function value and the derivatives up to order two in the one dimensional case, resp., the gradient/hessian in the n-dimensional case.

The C-XSC module `ddf_ari` implements AD arithmetic for functions $f : \mathbb{R} \to \mathbb{R}$ which are twice continuously differentiable. This module supports the class `DerivType` including operations and elementary functions for the differentiation arithmetic up to second order. An object of type `DerivType` is implemented as a triple of intervals representing the function value and the values of first and second derivatives, respectively. To get enclosures for the true values of the function and its derivatives, the differentiation arithmetic is based on interval arithmetic. That is, the three components of the `DerivType` object are replaced by the corresponding interval values, and all arithmetic operations and function evaluations are executed in interval arithmetic.

Any application program using the C-XSC module `ddf_ari` must contain a C++ function, with argument type `DerivType` and result type `DerivType`, which specifies the functional expression. For example, a function $f(x) = x(4+x)/(3-x)$ should be specified in a C++ function as follows.

```
DerivType f(const DerivType& x)
{ return x*(4.0 + x)/(3.0 - x); }
```

Let `x, fx` be declared as `DerivType` variables and `y(123.0)` be an interval variable. The following two objects of class DerivType contain a variable with value y and the representation of $f(x)$ in the point y, respectively.

```
x = DerivVar(y);        fx = f(x);
```

Then, `fValue(fx), dfValue(fx), ddfValue(fx)` get the function and the derivative values in the point interval $y = 123.0$. The C-XSC functions `fEval(), dfEval(), ddfEval()` simplify the mechanism of function and derivative evaluation. Thus, `ddfEval(f, y, fy, dfy, ddfy)` evaluates the three enclosures.

The module `hess_ari` supports classes `HessType` and `HTvector`, as well as operators and elementary functions for interval differentiation arithmetic of gradients and Hessians of scalar-valued functions $f : \mathbb{R}^n \to \mathbb{R}$. The module can also be used to compute Jacobians of vector-valued functions $f : \mathbb{R}^n \to \mathbb{R}^n$. The module `grad_ari` supports classes `GradType` and `GTvector`. These classes avoid the storage overhead for the Hessian components. The design, implementation and the application of the C-XSC AD modules are similar, see [7]. Table 1 summarizes the supported data types and the evaluation routines.

C-XSC AD modules include derivative constants and variables with both real and interval arguments, predefined arithmetic operations $\circ \in \{+, -, \times, /\}$ for any combination of real, interval and `DerivType` (or `HessType`, `GradType`, resp.), and the elementary functions enlisted in Table 2 that are predefined for the corresponding `DerivType`, `HessType` or `GradType` argument. Although the present version of C-XSC contains a lot of other mathematical functions, these

Table 1. C-XSC modules for AD: supported data types and predefined routines

C-XSC module	`ddf_ari`	`grad_ari`	`hess_ari`
scalar func.	$\mathbb{R} \to \mathbb{R}$	$\mathbb{R}^n \to \mathbb{R}$	$\mathbb{R}^n \to \mathbb{R}$
object type	`DerivType`	`GradType`	`HessType`
arg. type	`DerivType`	`GTvector`	`HTvector`
predefined functions	`fValue(DerivType)`	`fValue(GradType)`	`fValue(HessType)`
	`dfValue(DerivType)`	`gradValue(GradType)`	`gradValue(HessType)`
	`ddfValue(DerivType)`		`hessValue(HessType)`
	`fEval()`	`fEvalG()`	`fEvalH()`
	`dfEval()`	`fgEvalG()`	`fgEvalH()`
	`ddfEval()`		`fghEvalH()`
vector func.		$\mathbb{R}^n \to \mathbb{R}^n$	$\mathbb{R}^n \to \mathbb{R}^n$
object type		`GTvector`	`HTvector`
arg. type		`GTvector`	`HTvector`
predefined functions		`fValue(GTvector)`	`fValue(HTvector)`
		`JacValue(GTvector)`	`JacValue(HTvector)`
		`fEvalJ()`	`fEvalJ()`
		`fJEvalJ()`	`fJEvalJ()`

functions cannot be used explicitly in the functional expressions specifying function objects of types `DerivType`, `HessType` or `GradType`.

Some rules for getting true enclosures in connection with conversion (decimal to binary) errors (see [7, Chapters 2.5, 3.7], [11]) should also be applied when specifying functional expressions used by the C-XSC AD modules.

Table 2. Elementary functions supported by C-XSC AD arithmetic. Power function implements integer power.

exp ln sin cos tan cot asin acos atan acot
sqr power sqrt sinh cosh tanh coth asinh acosh atanh acoth

3 Expressions in *Mathematica*

At the core of *Mathematica* is the foundational idea that everything — data, programs, formulas, graphics, documents — can be represented as symbolic expressions. Although they often look very different, *Mathematica* represents all of these things in one uniform way. They are all expressions. A prototypical example of a *Mathematica* expression is `f[x,y,...]`. It is not necessary to write expressions in this form. For example, `x+y` is also an expression. When it is typed in `x+y`, *Mathematica* converts it to the standard form `Plus[x,y]`. Then, when it prints it out again, it gives it as `x+y`. All expressions – whatever they may represent – ultimately have a uniform structure. The function `FullForm` gives the full functional form of an expression, without shortened syntax.

```
In[1] := FullForm[1 + x^2 + x/y]
Out[1]//FullForm = Plus[1, Power[x, 2], Times[x,Power[y,-1]]]
```

Whenever *Mathematica* knows that a function is associative, it tries to remove parentheses (or nested invocations of the function) to get the function into a standard "flattened" form. The standard form is the one in which all the terms are in a definite order, corresponding roughly to alphabetical order [12]. A function like addition or multiplication is not only associative, but also commutative, which means that expressions like $a + c + b$ and $a + b + c$ with terms in different orders are equal. One reason for putting expressions into standard forms is that if two expressions are really in standard form, it is obvious whether or not they are equal. However, it is not always desirable that expressions be reduced to the same standard form. This is especially valid in interval arithmetic, where the machine addition and multiplication are not associative, and in range computation, where different forms of an expression can give different quality of the range enclosure, cf. [6, Chapter 3.3]. The *Mathematica* function `Hold[expr]` provides "wrappers" inside which the expressions remain unevaluated.

```
In[3] := x*Hold[b*a] //FullForm
Out[3]//FullForm = x Hold[b a]
```

When the user types in an expression, *Mathematica* automatically applies its large repertoire of rules for transforming expressions. The function `Hold` can also be used to prevent an expression from being simplified.

4 Communication of Functional Expressions

In this section we present a *MathLink* compatible basic C++ software, called `ADExpressions`, which communicates dynamically (via the *MathLink* protocol) functional expressions defined in *Mathematica* and in evaluation converts them into objects of C-XSC AD classes representing the function (and its derivatives).

4.1 Algorithm

The main goal of `ADExpressions` is to provide compatibility between the *Mathematica* representation of a functional expression and the representation used by C-XSC AD modules. Therefore, the software should read a *Mathematica* expression via appropriate *MathLink* tools, parse the expression providing compatibility between the operations and the elementary functions in both environments, and represent the parsed expression appropriately so that the evaluation generates objects of suitable C-XSC AD classes (`DerivType, GradType, GTVector, HessType, HTVector`). This work is organized in two separate main stages.

1. *Initialization.* The input expression coming from *Mathematica* via the *MathLink* protocol is parsed and stored in an appropriate C++ representation.
2. *Evaluation.* Read from *Mathematica* an interval value for each of the expression variables, substitute the expression variables (in the internal expression

representation) with objects of respective classes `DerivType, GradType, GTVector, HessType, HTVector`, where the objects are instantiated with the input values, and evaluate the expression according to its internal representation and by the respective automatic differentiation arithmetic. The result of the evaluation is an object of the appropriate C-XSC class.

The two stages are separated in order to provide re-usability of the evaluation step. In the initialization stage the expression formula is parsed and converted from the canonical *Mathematica* form to a postfix (reverse Polish) notation [1]. The postfix notation is chosen to allow a faster evaluation of the expression than the infix notation. For example, an expression having the canonical *Mathematica* form `Plus[Sin[y], x]` is stored as $\{y, Sin, x, Plus\}$, where the number of arguments for each function is known.

A nonlinear function represented as a *Mathematica* expression may contain

- numerical constants: integer, real, interval, the *Mathematica* constants `E`, `Pi`; The interval constants should be represented by the *Mathematica* object `Interval[{a, b}]`, where the interval end-points `a, b` can be integer or real numbers. Exact singletons cannot be used at the interval end-points but can be used in the other parts of a functional expression. For simplicity *Mathematica* numbers with heads `Rational` and `Complex`[2], as well as another syntax of the interval object are not admissible for the functional expression.
- variables having interval values;
  The names of the variables should follow the *Mathematica* convention for names, or be *Mathematica* indexed variables with the syntax `name[i]`.
- arithmetic operations, a set of elementary functions, and the *Mathematica* function `Hold` enlisted in Table 3.

To provide compatibility between the *Mathematica* expression and its C-XSC representation, the following transformations are performed during the parsing.

T.1 All the admissible functions in an input *Mathematica* expression have one or two arguments, except for the functions `Plus` and `Times` which are associative for exact arguments. During the parsing multi-argument functions `Plus` and `Times` are replaced by multiple two-argument functions. Therefore, we assume that all functions in the expression have one or two arguments.

T.2 All non-interval numerical constants are stored as C-XSC point intervals. Corresponding C-XSC interval constants containing the true value of the *Mathematica* constants `E` and `Pi` are used.

T.3 When all arguments of a function are constants, it is evaluated by the corresponding C-XSC function and the enclosing interval constant is stored in the internal representation.

T.4 For every two-argument function there are three internal representations corresponding to all combinations of the argument types (constant or expression). The commutative functions have two internal representations. Thus, the function `Plus`, which is commutative, is represented internally

[2] C-XSC AD modules do not support complex arithmetic.

Table 3. *Mathematica* operator symbols and functions, recognized by the expression parser, and their equivalent C-XSC representation

Mathematica operator or function	C-XSC representation	Comments
$+, -, \times, /$	$+, -, \times, /$	Sect. 3, T.1
`Hold[Subtract[x,y]]`	`x-y`	
`Hold[Divide[x,y]]`	`x/y`	
`^, Power[x, y_Integer]`	`sqr(x)`	if y=2
	`power(x, y)`	other integer `y`
`Power[E, y], Hold[Exp[x]]`	`exp(x)`	
`^, Power[x, y]`	`exp(ln(x)*y)`	otherwise
`Hold[Sqrt[x]]`	`sqrt(x)`	
`Log[x], Log[E, x]`	`ln(x)`	nat. logarithm
`Log[b,x]`	`ln(x)/ln(b)`	log to base `b`
`Sin[x],Cos[x],Tan[x],Cot[x]`	$\sin(x), \cos(x), \tan(x), \cot(x)$	
`Sinh[x],Cosh[x],Tanh[x],Coth[x]`	$\sinh(x), \cosh(x), \tanh(x), \coth(x)$	
`ArcSin[x], ArcCos[x], ArcTan[x]`	$\mathrm{asin}(x)$, $\mathrm{acos}(x)$, $\mathrm{atan}(x)$	
`ArcCot[x],ArcSinh[x],ArcCosh[x]`	$\mathrm{acot}(x)$, $\mathrm{asinh}(x)$, $\mathrm{acosh}(x)$	
`ArcTanh[x], ArcCoth[x]`	$\mathrm{atanh}(x)$, $\mathrm{acoth}(x)$	
`Csc[x], Sec[x]`	`1/sin(x), 1/cos(x)`	
`Hold[expr]`	expr	`Hold` dropped

by two classes, say `plus(expr1, expr2)` and `plus(const, expr)`. During the initialization, depending on the type of the particular arguments, it is determined which of the two classes will be used in the internal representation. In case of a constant argument, the second representation class `plus(const, expr)` is used and it is initialized with the particular constant value.

T.5 *Mathematica* functions like logarithm to a base, power function with real exponent, secant, cosecant, which are not supported by the C-XSC AD modules, are transformed to relevant C-XSC expressions, see Table 3.

The evaluation stage consists of three actions:

- Reading a numerical interval for each variable. It is assumed that an interval comes from *Mathematica* as a two-dimensional list of the interval end-points. The order $\leq$ of the interval end-points is not checked by the communication software, so a C-XSC runtime error can arise if the relation is not fulfilled.
- The variables involved in the functional expression are replaced by a variable of appropriate C-XSC class which contains the concrete interval value(s). Which C-XSC class will be used depends on the function under evaluation and the particular C-XSC automatic differentiation module, see Table 1.
- The evaluation of an expression in postfix notation is done by using a stack. The postfix expression to be evaluated is scanned from left to right. Variables or constants are pushed onto the stack. When an operator/function is encountered, the indicated action is performed (by the particular C-XSC

differentiation arithmetic) using the top elements of the stack, and the result replaces the operands on the stack.

4.2 Implementation

All functions and classes of the basic communication software `ADExpressions` are defined in the namespace `mlcxsc`[3]. Hence, it is required to qualify each identifier of the module with `mlcxsc::` or to use the directive `using namespace mlcxsc`. `ADExpressions` consists of two basic template classes:

- `MLCXSCFunctionScalar<class T>` which communicates, represents and evaluates one- or n-variate scalar functions $f(x) : \mathbb{R} \longrightarrow \mathbb{R}$ or $f(x) : \mathbb{R}^n \longrightarrow \mathbb{R}$;
- `MLCXSCFunctionVector<class T>` which communicates, represents and evaluates vector-valued functions $f(x) : \mathbb{R}^n \longrightarrow \mathbb{R}^n$.

The template parameter `T` in the class `MLCXSCFunctionScalar` can be one of the three C-XSC data types `DerivType`, `GradType`, or `HessType`, corresponding to the respective C-XSC AD modules. Vector-valued functions are treated as n functions $f_i : \mathbb{R}^n \longrightarrow \mathbb{R}$. The two template classes are intended to work with the communication protocol *MathLink* since in their initialization it is supposed that the functional expression is in the *MathLink* queue.

The two basic template classes have four public methods (member functions):

- `Init()` initializes an object, called representation list, with the functional expression which should be evaluated. In case of some error it throws an exception with specified corresponding error message.
- `operator()` evaluates the function computing its range and the ranges of its first and second derivatives for the given interval values of the variable(s). In case of some error it throws an exception.
- `IsInitialized()` returns `true` if the representation list is initialized and `false` otherwise. The representation list is initialized whenever the function `Init()` has finished successfully, then we can call `operator()`.
- `Invalidate()` makes a representation list not valid.

Copy constructor and `operator=` for the template classes `MLCXSCFunction Scalar` and `MLCXSCFunctionVector` are not implemented.

In order to facilitate catching and communicating error messages, an exception class `MLCXSCFunctionException` is defined as follows.

```
class MLCXSCFunctionException: public std::logic_error
{public:
 enum ErrorCode
 {ecIllegalArgumentError,// Init,operator called with illegal arguments
  ecNotInitializedError,// calling operator() before executing Init()
  ecExpressionError,    // error in parsing a Mma expression by Init()
  ecInternalError       // internal error that should not normally occur
```

[3] "`ml`" in the abbreviation `mlcxsc` comes from *MathLink*.

```
 };
 private:
   ErrorCode mErrorCode;
 public:
   MLCXSCFunctionException(ErrorCode aErrorCode,
              const string& aMessage):std::logic_error(aMessage)
   {      mErrorCode = aErrorCode;                }
   ErrorCode GetErrorCode() const {  return mErrorCode; }
};
```

The error messages are defined in sufficient details at the place they arise and are associated with particular error code. The error code has self-explanatory enumerated values commented in the above source code. The four values try to minimize the number of exception types that will be triggered to *Mathematica* and correspond to four main stages of the algorithm. Different error messages, regarding the same stage of the algorithm, are communicated by one error code. For example, all kind of errors that arise during the expression parsing are associated with and communicated by the `ecExpressionError` code. Some of these messages are: `Unknown variable`, `Unknown function`, `Unknown Data Type`, wrong syntax of an interval in the expression, etc.

The software `ADExpressions` is designed as an auxiliary tool. Although based on *MathLink* protocol for communicating the expressions, this basic software does not contain functions installable in *Mathematica*. Its purpose is to facilitate the development of such functions, that is, the implementation of external *MathLink* programs integrating C-XSC routines for, or routines based on, AD arithmetic. `ADExpressions` can be downloaded from

http://www.math.bas.bg/~epopova/software/ADExpressions.zip

For the sake of readability the distribution is split into three files: `types.h`, `expression.h`, `expression.cpp`. Everyone who wants to use the software shall include `expression.h` in the source file of the own *MathLink* program. Examples of installable *MathLink* programs which use `ADExpressions` are contained in the archive `ADpackages.zip` which can be downloaded from the above site. The archive contains three external *MathLink* compatible programs `ddfAri, gradAri, hessAri` embedding in *Mathematica* the corresponding C-XSC AD modules by using the `ADExpressions` software. Any of the three *MathLink* programs is an excellent example of how to use the basic communication software `ADExpressions` and provides a framework for further developments.

5 Embedding C-XSC AD Modules in *Mathematica*

MathLink [5,12] and the implemented basic software communicating functional expressions allow any external C-XSC function to be called from within *Mathematica*. Any of the *MathLink* compatible programs `ddfAri, gradAri, hessAri` presents an interesting example of extending the *Mathematica* kernel. These programs involve an interaction between the external C-XSC code, the parsing

software communicating functional expressions, the *Mathematica* package code, and the kernel. Due to the lack of space some interesting design and *MathLink* programming issues that arose in their development are discussed in [11].

Once the external *MathLink* programs are compiled, the executable files (containing the corresponding package) can be launched in any *Mathematica* session.

```
In[1] := Install["gradAri"]
Out[1] = LinkObject[./gradAri, 2, 2]
```

Table 4 enlists the *Mathematica* functions providing interface for the respective C-XSC functions from Table 1. All three packages can be installed in the same *Mathematica* session and be used simultaneously.

Table 4. *Mathematica* packages, resp. functions, interfacing the C-XSC modules for automatic differentiation `ddf_ari, hess_ari, grad_ari`

	MathLink compatible packages	
ddfAri	gradAri	hessAri
SetFunction[var, func]	SetFScalarGrad[{vars}, func]	SetFScalarHess[{vars}, func]
ReadyFunctionQ[ ]	ReadyScalarGradQ[ ]	ReadyScalarHessQ[ ]
fValue[int]	fValueScalarG[{ints}]	fValueScalarH[{ints}]
dfValue[int]	gradValueG[{ints}]	gradValueH[{ints}]
ddfValue[int]		hessValue[{ints}]
dfEval[var, fun, int]	fgEvalG[{vars}, func, {ints}]	fgEvalH[{vars}, func, {ints}]
ddfEval[var, fun, int]		fghEvalH[{vars}, func, {ints}]
	SetFVectorGrad[{vars}, {funcs}]	SetFVectorHess[{vars}, {funcs}]
	ReadyVectorGradQ[ ]	ReadyVectorHessQ[ ]
	fValueVectorG[{ints}]	fValueVectorH[{ints}]
	JacValueG[{ints}]	JacValueH[{ints}]
	fJEvalJGrad[{vars},{funcs},{ints}]	fJEvalJHess[{vars},{funcs},{ints}]

Now, we initialize a scalar function $f = 2 - \sum_{i=1}^{10} x_i^2$ in 10 variables and evaluate it in the interval $[-1, 1]$ for each variable. Since the variables are many, we use indexed variables and *Mathematica* functions for generating the functional expression and the lists of variable names and variable values.

```
In[5]:= SetFScalarGrad[Table[x[i],{i,10}], 2-Sum[x[i]^2, {i,10}]]
```

The output `Null` of the successful evaluation of `SetFScalarGrad` is not visible.

```
In[6] := ReadyScalarGradQ[ ]
Out[6] = True
In[7] := fValueScalarG[Table[{-1, 1}, {i, 10}]]
Out[7] = {-8., 2.}
```

For the evaluation of vector-valued functions and their derivatives, C-XSC modules for AD impose the restriction (which is followed by the interface) that the number of specified variables must be equal to the dimension of the function.

```
In[10] := SetFVectorGrad[{x}, {x, 1 - x, x^2}]
MLCXSCErrorGrad::illargs : Illegal arguments: Init: The number
of variables must be equal to the  dimension of the function.
Out[10] = $Failed
```

However, one variable name can be repeated many times in the list of variable names, see [11]. Dummy variables can be also used which makes possible the guaranteed evaluation of constants.

```
In[13] := fJEvalJGrad[{x}, {Hold[Sqrt[3]]}, {{1,2}}]
Out[13] = {{{1.73205, 1.73205}}, {{{0., 0.}}}}
```

The implementation of C-XSC AD modules uses the standard error handling (runtime error) of the interval library employed in case of some error during the function or the derivatives evaluation, see [7]. C-XSC runtime errors during the evaluations cannot be checked and cause closing the connection but the corresponding C-XSC message is displayed in a separate `stderr` window.

The *Mathematica* interface for C-XSC routines provides more possibilities for controlling the conversion input errors. Inexact quantities can be enclosed either by the *Mathematica* function `Interval`, or by an interval enclosure in C-XSC (using the techniques discussed in [7]) and wrapping in the `Hold` function.

```
In[16]:= SetFScalarGrad[{x}, Hold[Divide[23,Interval[{10,10}]]]*x]
        fValueScalarG[{{1, 2}}] // FullForm
Out[17]//FullForm = List[2.3‘,4.6000000000000005‘]
```

The representation of a mathematical function in different expression forms is very important for the interval computations. The symbolic interface of C-XSC enables evaluation of functions in different forms via the `Hold` function. The next simple example demonstrates the lack of inverse interval addition.

```
In[18] := fgEvalG[{x}, Hold[x-x], {{-1,1}}]
Out[18] = {{-2., 2.}, {{0., 0.}}}
```

6 Conclusion

The integration of C-XSC automatic differentiation modules in the environment of *Mathematica* via *MathLink* communication protocol brings dynamics and interactivity in the execution of C-XSC modules working with nonlinear functions. This saves the compilation time for each concrete function in the analysis of practical problems. The paper shows that it might be much easier and cheaper to connect a general-purpose computer algebra (CA) system to a sophisticated interval library than to implement interval algorithms from scratch using the programming language of the corresponding CA system (even if some interval tools are already provided by the CA system). `ADExpressions` can facilitate the implementation of any other *MathLink* program providing new numerical methods based on AD. By the interoperability approach we obtain an

integrated environment which combines the interactivity, symbolic and visualization tools of *Mathematica* with the rigorousness and the speed of C-XSC interval functionality.

While the syntactical interoperability between *Mathematica* and C-XSC is provided essentially by *MathLink* communication protocol, the major efforts should be devoted to providing semantic interoperability — the ability to automatically interpret the information exchanged, as defined by the end users, meaningfully and accurately in order to produce useful results. AD of complex-valued functions is not supported in C-XSC and therefore by `ADExpressions`. However, the present version of C-XSC contains a lot of other mathematical functions, besides those enlisted in Table 2, which can be evaluated for complex-valued real/interval variables. A separate work could provide a more generic framework allowing communication of real or complex-valued interval functions evaluated rigorously by the expanded set of C-XSC elementary functions. Other interesting directions of future investigation are the ability of both systems to exchange dot-product expressions and results, and the embedding of C-XSC complex interval arithmetic/solvers not available in *Mathematica*.

Although the current work focuses on one-way communication — the embedding of C-XSC functionality in the environment of *Mathematica*, it can help the respective work on using the *Mathematica* kernel from within C-XSC programs.

References

1. Aho, A., Lam, M., Sethi, R., Ullman, J.: Compilers: Principles, Techniques, and Tools, 2nd edn. Pearson/Addison Wesley (2007)
2. Alt, R., Frommer, A., Kearfott, R.B., Luther, W. (eds.): Numerical Software with Result Verification. LNCS, vol. 2991. Springer, Heidelberg (2004)
3. Corliss, G.F., Yu, J.: Interval Testing Strategies Applied to COSY's Interval and Taylor Model Arithmetic. In: [2], pp. 91–106
4. Frommer, A.: Proving conjectures by use of interval arithmetic. In: Kulisch, U., et al. (eds.) Perspectives on Enclosure Methods, pp. 1–13. Springer, Wien (2001)
5. Gayley, T.: A MathLink Tutorial. Wolfram Research, Champaign (2002)
6. Hansen, E.: Global Optimization Using Interval Analysis. Marcel Dekker, New York (1992)
7. Hammer, R., Hocks, M., Kulisch, U., Ratz, D.: C++ Toolbox for Verified Computing: Basic Numerical Problems. Springer, Heidelberg (1995)
8. Hofschuster, W., Krämer, W.: C-XSC 2.0: A C++ Library for Extended Scientific Computing. In: [2], pp. 15–35
9. Klatte, R., Kulisch, U., Lawo, C., Rauch, M., Wiethoff, A.: C-XSC, A C++ Class Library for Extended Scientific Computing. Springer, Heidelberg (1993)
10. Popova, E.: *Mathematica* Connectivity to Interval Libraries filib++ and C-XSC. In: Cuyt, A., Krämer, W., Luther, W., Markstein, P. (eds.) Numerical Validation in Current Hardware Architectures. LNCS, vol. 5492, pp. 117–132. Springer, Heidelberg (2009)
11. Popova, E., Krämer, W., Russev, M.: Integration of C-XSC Automatic Differentiation in Mathematica, Preprint 3/2010, IMI-BAS, Sofia (March 2010), `http://www.math.bas.bg/~epopova/papers/10-preprintAD.pdf`
12. Wolfram Research Inc.: Mathematica, Version 5.2, Champaign, IL (2005)

Author Index