Dense Bivariate Polynomial GCD Computation.

Matthew Gibson Michael Monagan Department of Mathematics, Simon Fraser University.

Modular GCD Algorithm

Given $A, B \in \mathbb{Z}_p[x_1, x_2, \ldots x_n]$ where A and B are dense, the goal was to compute $G = \mathrm{GCD}(A, B)$ *quickly*. The focus was on developing an efficient algorithm for the bivariate case, with the intent that it could be used in algorithms for a larger number of variables.

A modular algorithm for bivariate polynomial GCD's was developed by Brown [1] as an alternative to methods involving bivariate polynomial remainder sequences. Browns method uses the evaluation homomorphism $\phi_{y=\alpha} : \mathbb{Z}_p[x, y] \to \mathbb{Z}_p[x]$ and preforms the GCD computation on univariate images. It then interpolates the image GCD's iteratively, to recover the true bivariate GCD.

To use Brown's method, it is necessary to overcome a number of difficulties. For the correctness of the algorithm, the most significant difficulty is the presence of *unlucky evaluation points*. To see what these are, consider the simple example in the coefficient field $\mathbb{Z}_{13}$:

$$A = (x^2 + y)(x + xy + 2)$$
$$B = (x^2 + y)(3x + y)$$

In this case the GCD is $G = x^2 + y$. To find this GCD, the algorithm will apply the homomorphism on a number of points α_i.

α_i	$A_i = \phi_{\alpha_i}(A)$	$B_i = \phi_{\alpha_i}(B)$	$g_i = \mathrm{GCD}(A_i, B_i)$
0	$(x^2)(x+2)$	$3x^3$	x^2
1	$(x^2+1)(2x+2)$	$(x^2+1)(3x+1)$	x^2+1
2	$(x^2+2)(3x+2)$	$(x^2+2)(3x+2)$	$(x^2+2)(3x+2)$
3	$(x^2+3)(4x+2)$	$(x^2+3)(3x+3)$	(x^2+3)

In general, g_i is an associate of $\phi_{\alpha_i}(G)$. However, in the case $\alpha_i = 2$, the image GCD contains a portion of the input images which is *not* a part of the true GCD. Such an image is called **unlucky** and cannot be used in the interpolation of the GCD.

Through a process involving checks on the degree of the image GCD's, it is possible to ensure that either none of the the evaluation points are unlucky, or all of them are unlucky. This allows for the computation of the true GCD with very high probability. However, to ensure that not all of the evaluation points are unlucky, Lemma 1 is needed.

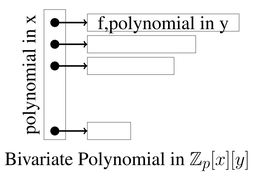
Lemma 1. *Let F be a field, $G, E \in F[x, y]$ with $G \neq 0$ and G primitive with respect to x. Let $\alpha \in F$ and let $\phi_\alpha : F[x, y] \to F[x]$ be the image homomorphism $\phi_\alpha(f(x, y)) = f(x, \alpha)$. Require that $\deg_x(\phi_\alpha(G)) = \deg_x(G)$. Then if $E \mid G$ and $\phi_\alpha(G) \mid \phi_\alpha(E)$, it follows that E and G are associates in $F[x, y]$.*

If E is the interpolation of GCD images, a divisibility check of $E \mid A$, $E \mid B$ certifies that we have the true GCD. This can either be done using the classical division method, or by continuing to preform evaluations and univariate computations up to a predetermined bound.

Computational Optimizations

Evaluations

The preformance of evaluation and interpolation can be almost doubled by choosing evaluation points in positive/negative pairs.



Bivariate Polynomial in $\mathbb{Z}_p[x][y]$

If the evaluations $y^2, y^4, \ldots$ at $y = \alpha$ are precomputed, this work can be shared across all polynomials in $\mathbb{Z}_p[y]$. Then the **accumulator method** utilizes a larger data type for computing long sums of integer products, so that a single modular reduction can be preformed at the end.

$$f = [f_0 + f_2 y^2 + f_4 y^4 + \ldots] + y \cdot [f_1 + f_3 y^2 + \ldots]$$

$$f(\pm\alpha) = [f_0 + f_2 \alpha^2 + f_4 \alpha^4 + \ldots] \pm \alpha \cdot [f_1 + f_3 \alpha^2 + \ldots]$$

Univariate GCD Optimization

Usually, each univariate GCD computation will compute a remainder sequence $R_0, R_1, R_2 \ldots$ where each new remainder always decreases in degree by one. Rather than doing polynomial division, this type of remainder sequence can be constructed by computing a new polynomial as a linear combination of the previous two, leading to optimizations. Consider the following code snippets:

```
for(;k>1;k--){
  R1[k-1] = R1[k-1]*inv % p;
  R0[k] = R0[k] - R1[k-1];
  if(R0[k] < 0) R0[k] += p;
  R0[k] = R0[k] - R1[k]*c % p;
  if(R0[k] < 0) R0[k] += p; }
```

Old Code

```
while(k>1){
  v = e*R0[k]-c*R1[k]-a*R1[k-1];
  if(v < 0) v += M;
  if(v < 0) v += M;
  R0[k] = v % p;
  k--; }
```

New Code

The old code assumes R0 to be monic, and makes R1 monic while it computes a new R0 of lesser degree. The new code requires R1 to be monic, and then computes a new R0 of lesser degree while also making R0 monic. The old code computes two modular reductions (`%p`) per term in R0, while the new code only one. As a result, the preformance of univariate GCD is almost doubled.

Root of Unity and Newton Basis

The GCD is interpolated in Newton Form. Taking into account the use of $\pm\alpha$ evaluation points, this can be written

$$\begin{aligned} G &= n_0 + n_1(y-\alpha_0) + n_2(y-\alpha_0)(y+\alpha_0) + n_3(y-\alpha_0)(y+\alpha_0)(y-\alpha_1) + \ldots \\ &= [n_0 + n_1(y-\alpha_0)] + [n_2 + n_3(y-\alpha_1)](y^2-\alpha_0^2) + [n_4 + n_5(y-\alpha_2)](y^2-\alpha_0^2)(y^2-\alpha_1^2) + \ldots \end{aligned}$$

If we think of $1, (y^2-\alpha_0^2), (y^2-\alpha_0^2)(y^2-\alpha_1^2), \ldots$ as a kind of basis, this suggests the following data structure:

$$G_{\mathcal{N}} = [g_0, g_1, g_2, g_3, \ldots] = [n_0 - \alpha_0 n_1, n_1, n_2 - \alpha_1 n_3, n_3, \ldots]$$

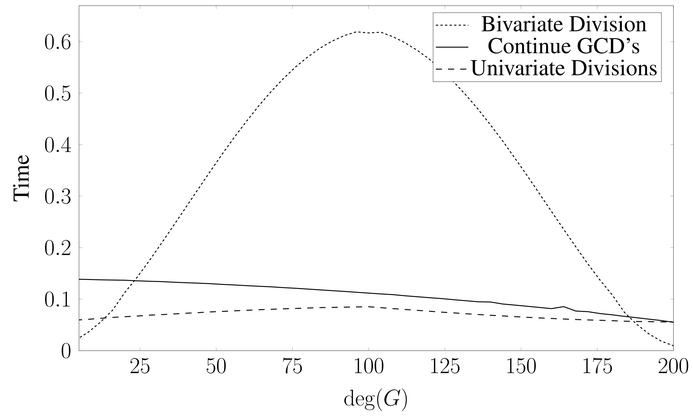
These coefficients can be used for basis conversion and for efficient evaluation. Write the polynomial G in the form:

$$G = [g_0 + g_2(y^2-\alpha_0^2) + g_4(y^2-\alpha_0^2)(y^2-\alpha_1^2) + \ldots] + y \cdot [g_1 + g_3(y^2-\alpha_0^2) + g_5(y^2-\alpha_0^2)(y^2-\alpha_1^2) + \ldots]$$

The terms $(y^2-\alpha_0^2), (y^2-\alpha_0^2)(y^2-\alpha_1^2), \ldots$ can be precomputed, and the polynomial can be evaluated using a simple loop.

Algorithmic Optimizations

Once the number of interpolated images has exceeded a pre-calculated bound, it can be proven that divisibility has been satisfied. However, it is also possible to stop interpolating before the bound and switch to another method to ensure divisibility. Several variations are compared below, for $\deg(G)$ as indicated and $\deg(A) = \deg(B) = 200$ in both x and y.



Classical bivariate trial division, which is $O(n^4)$ rather than $O(n^3)$, is much slower for all but very large or very small GCD's. Further, it is always preferable to stop computing univariate GCD's as soon as possible and instead do evaluations and trial divisions.

Parallelization for Three Variables

The dense bivariate polynomial GCD algorithm was used to solve a dense polynomial GCD in three variables. The strategy was to use Brown's algorithm in three-variables. The variable z was evaluated out and then the optimized dense bivariate method was applied to the images in $\mathbb{Z}_p[x, y]$ in parallel. Parallelization was implemented through Intel's Cilk-Plus™.

			New Algorithm				
n	Maple	Magma	1 CPUs	2 CPUs	4 CPUs	8 CPUs	16 CPUs
25	9.950	1.020	0.129	0.068	0.036	0.022	0.016
50	351.7	13.66	1.360	0.695	0.362	0.206	0.145
75	2433	65.36	5.885	2.989	1.521	0.784	0.496
100	10684	198.3	17.33	8.740	4.458	2.293	1.260

The timings are in seconds, and n is the degree of G in x,y,z. The degree of A and B in x,y,z is $2n$. The test was preformed on a Intel Xeon processor with 16 cores. For large inputs, the new algorithm almost scales to the number of threads.

References

[1] W. S. Brown. On Euclid's Algorithm and the Computation of Polynomial Greatest Common Divisors, *J. ACM* **18** (1971), pp. 476–504.

[2] M. B. Monagan and A. D. Wittkopf. On the Design and Implementation of Brown's Algorithm over the Integers and Number Fields. *Proceedings of ISSAC '2000*, ACM Press, pp. 225–233, 2000.