On the complexity of computing certain resultants

Romain Lebreton
LIX, École polytechnique, France
`lebreton@lix.polytechnique.fr`
Esmaeil Mehrabi and Éric Schost
Computer Science Department, Western University, Canada
`emehrab@uwo.ca`, `eschost@uwo.ca`

Computing resultants is a fundamental algorithmic question, at the heart of higher-level algorithms for solving systems of equations, computational topology, etc. However, in many situations, the best known algorithms are still sub-optimal. The following table summarizes the best results known to us (from [3]), using soft-Oh notation to omit logarithmic factors. In all cases, we assume that f, g have coefficients in a field k, and that their partial degrees in all variables is at most d. The partial degree in all remaining variables of their resultant $r = \mathrm{res}(f, g, x_1)$ is then at most $2d^2$. In this note, the cost of an algorithm is the number of arithmetic operations in k it performs.

f, g are in ...	$k[x_1]$	$k[t, x_1]$	$k[t, x_0, x_1]$
number of terms in f, g	$\Theta(d)$	$\Theta(d^2)$	$\Theta(d^3)$
number of terms in $r = \mathrm{res}(f, g, x_1)$	1	$\Theta(d^2)$	$\Theta(d^4)$
cost of computing r (best known bound)	$O\tilde{~}(d)$	$O\tilde{~}(d^3)$	$O\tilde{~}(d^5)$
optimal?	yes, up to log factors	no	no

In the last cases, one can replace the ring $k[t]$ by $\mathbb{Z}$ and consider the bit-complexity of computing resultants of polynomials in $\mathbb{Z}[x_1]$ or $\mathbb{Z}[x_0, x_1]$. We do not give details, but most results carry over, mutatis mutandis. One can also consider polynomials in $\mathbb{Z}[t, x_0, x_1]$, etc; we do not discuss this.

Main result. Our contribution is on the third case, with f, g in $k[t, x_0, x_1]$. We make the following assumptions (which hold for generic f, g):

- $\mathbf{A}_1$. The reduced Groebner basis of the ideal $\langle f, g\rangle$ in $k(t)[x_0, x_1]$ for the lexicographic order $x_1 > x_0$ has the form $\langle R(x_0), x_1 - S(x_0)\rangle$, with R, S in $k(t)[x_0]$ and R monic.
- $\mathbf{A}_2$. All solutions of the system $f = g = 0$ in $\overline{k(t)}$ have multiplicity 1.

Our algorithm uses matrix multiplication; we let $2 < \omega \le 3$ be such that one can multiply $n \times n$ matrices over k in n^ω operations. The best known result [5] is $\omega \simeq 2.37$.

Theorem 1 *Let f, g be in $k[t, x_0, x_1]$, with degree at most d in all variables and that satisfy $\mathbf{A}_1, \mathbf{A}_2$. Suppose that k has cardinality at least $12d^4$. There exists a probabilistic algorithm that computes $r = \mathrm{res}(f, g, x_1)$ using $O(d^{\frac{\omega+7}{2}})$ operations in k and success probability at least $1/2$.*

Since we have $2 < \omega \le 3$, the exponent ρ in our running time satisfies $4.5 < \rho \le 5$. This improves on the best previous results, getting us closer to an optimal $O\tilde{~}(d^4)$. Even under assumptions $\mathbf{A}_1, \mathbf{A}_2$, and allowing probabilistic algorithms, we are not aware of any previous improvements over $O\tilde{~}(d^5)$.

Sketch of our algorithm. The polynomial R introduced in $\mathbf{A}_1$ and the resultant r of f and g are related by the equality $R = r/\mathrm{LeadingCoefficient}(r, x_0)$. We describe how to compute R, since finding the proportionality factor that gives r is straightforward.

The algorithm uses Newton / Hensel lifting techniques. We choose a random expansion point τ for t in k. This is the source of the probabilistic aspect of the algorithm: we expect that no denominator in R or S vanishes at τ, and that the solutions of the system $f(\tau, x_0, x_1) = g(\tau, x_0, x_1) = 0$ in $\overline{k}$ still have multiplicity 1; the analysis in [4, Prop. 3] and our assumption on the cardinality of k show that at least half the points in k are "lucky" for this random choice. Below, we take $\tau = 0$ for simplicity; then, by assumption, for $\kappa \geq 1$, $R_\kappa = R \bmod t^\kappa$ and $S_\kappa = S \bmod t^\kappa$ are well-defined; they lie in $A_\kappa[x_0]$, with $A_\kappa = k[t]/t^\kappa$.

We first compute $R_1 = R \bmod t$ and $S_1 = S \bmod t$, using Reischert's algorithm in $k[x_0, x_1]$; this costs $O\tilde{~}(d^3)$. Then, we compute R_κ and S_κ for some $\kappa \geq 4d^2$ using lifting techniques: the successive lifting steps compute $(R_2, S_2), (R_4, S_4), \ldots, (R_{2^\ell}, S_{2^\ell}), \ldots$. The assumption that the system $f(\tau, x_0, x_1) = g(\tau, x_0, x_1) = 0$ has simple roots makes this step well-defined; we analyze its cost below. Finally, we get R by applying rational reconstruction to all coefficients of R_κ in time $O\tilde{~}(d^4)$.

The key subroutine. The above process is hardly new: the references [2, 4] give details on such lifting algorithms, in more general contexts; however, as explained now, a direct application of these results performs poorly in our context.

Given (R_κ, S_κ), the algorithm of [2, 4] computes $(R_{2\kappa}, S_{2\kappa})$ as follows. Let $B_\kappa = A_{2\kappa}[x_0, x_1]/\langle R_\kappa, x_1 - S_\kappa\rangle = k[t, x_0, x_1]/\langle t^{2\kappa}, R_\kappa, x_1 - S_\kappa\rangle$. First, compute the normal form of (f, g), and of their Jacobian matrix J, in B_κ; then, deduce the vector

$$\begin{bmatrix} \delta_R \\ \delta_S \end{bmatrix} = \begin{bmatrix} R'_\kappa & 0 \\ -S'_\kappa & 1 \end{bmatrix} J^{-1} \begin{bmatrix} f \\ g \end{bmatrix} \in B_k^{2\times 1}.$$

Taking canonical preimages of δ_R and δ_S in $A_{2\kappa}[x_0]$, we have $R_{2\kappa} = R_\kappa + \delta_R$ and $S_{2\kappa} = S_\kappa + \delta_S$. The bottleneck is the computation of the normal form of f, g and J: the algorithm in [2, 4] does $O(d)$ operations in B_κ, for a total of $O\tilde{~}(\kappa d^3)$ operations in k. Summing over all lifting steps, with $\kappa = 1, 2, 4, 8, \ldots$ up to about $\kappa \simeq d^2$ leads to the bound $O\tilde{~}(d^5)$, which is no better than Reischert's algorithm.

We now sketch how to compute e.g. the normal form of f in B_k more efficiently, using a baby-steps / giant-steps approach inspired by Brent and Kung's algorithm [1].

1. Seeing f as a polynomial of degree d in x_1, with coefficients $f_i \in A_{2\kappa}[x_0]$ of degree d in x_0, build the $\sqrt{d+1} \times \sqrt{d+1}$ matrix $M_1 = (f_{(\sqrt{d+1}-i)\sqrt{d+1}+j-1})$ with entries in $A_{2\kappa}[x_0]$.
2. Compute $\sigma_0 = S_\kappa^0, \sigma_1 = S_\kappa^1, \cdots, \sigma_{\sqrt{d+1}-1} = S_\kappa^{\sqrt{d+1}-1}$ in B_k (baby steps).
3. Cut all σ_i into slices, writing $\sigma_i = \sum_{j=0}^{d-1} \sigma_{i,j} x_0^{dj}$, with $\sigma_{i,j} \in A_{2\kappa}[x_0]$ of degree less than d in x_0. Build the $\sqrt{d+1} \times d$ matrix $M_2 = (\sigma_{i,j})$ and compute $M = (m_{ij}) = M_1 M_2$.
4. Using the $m_{i,j}$, reconstruct $f \bmod \langle R_\kappa, x_1 - S_\kappa\rangle$ using Horner's scheme (giant steps).

As in Brent and Kung's algorithm, the dominant cost is matrix multiplication (Step 3). We do matrix multiplication in sizes $\sqrt{d+1} \times \sqrt{d+1}$ and $\sqrt{d+1} \times d$, with entries in $k[t, x_0]$, of degrees at most 2κ in t and d in x_0. The cost is thus $O\tilde{~}(\kappa d^{\frac{\omega+3}{2}})$ operations in k.

Summing over all lifting steps, using the fact that we take $\kappa = 1, 2, 4, 8, \ldots$ (powers of 2) until approximately d^2, the total cost is $O\tilde{~}(d^{\frac{\omega+7}{2}})$, as claimed.

References

[1] R. P. Brent and H. T. Kung. Fast algorithms for manipulating formal power series. *Journal of the ACM*, 25(4):581–595, 1978.

[2] M. Giusti, G. Lecerf, and B. Salvy. A Gröbner free alternative for polynomial system solving. *Journal of Complexity*, 17(1):154–211, 2001.

[3] D. Reischert. Asymptotically fast computation of subresultants. In *ISSAC'97*, pages 233–240. ACM, 1997.

[4] É. Schost. Computing parametric geometric resolutions. *Appl. Algebra Engrg. Comm. Comput.*, 13(5):349–393, 2003.

[5] V. Vassilevska Williams. Breaking the Coppersmith-Winograd barrier. 2011.