GCD of Many Integers
(Extended Abstract)

Gene Cooperman[1,⋆], Sandra Feisel[2], Joachim von zur Gathen[2], and George Havas[3,⋆⋆]

[1] College of Computer Science, Northeastern University
Boston, MA 02115, USA
gene@ccs.neu.edu
[2] FB Mathematik-Informatik, Universität–GH Paderborn
33095 Paderborn, Germany
{feisel,gathen}@uni-paderborn.de
[3] Centre for Discrete Mathematics and Computing
Department of Computer Science and Electrical Engineering
The University of Queensland, Queensland 4072, Australia
havas@csee.uq.edu.au

Abstract. A probabilistic algorithm is exhibited that calculates the gcd of many integers using gcds of pairs of integers; the expected number of pairwise gcds required is less than two.

1 Introduction

In many algorithms for polynomials in $\mathbb{Z}[x]$, e.g., computation of gcds or factorization, one is interested in making a polynomial primitive, i.e., computing and removing its content. The primitive polynomial remainder sequence needs such gcd computations but is not used in practice: Ho and Yap [7] state that "the primitive PRS has the smallest possible coefficients, but it is not efficient because content computation is relatively expensive". Geddes, Czapor and Labahn [4, Chapter 7], write about the primitive polynomial remainder sequence: "The problem with this method, however, is that each step requires a significant number of GCD operations in the coefficient domain. ... The extra cost is prohibitive when working over coefficient domains of multivariate polynomials."

We remedy this sorry situation for $\mathbb{Z}[x]$ by computing the gcd of random linear combinations of the inputs. More precisely, we solve the following problem.

> Given m positive integers $a_1, \ldots, a_m$ with $m > 1$, compute $\gcd(a_1, \ldots, a_m)$ with a small number of pairwise gcds, i.e., gcds of two integers each.

Main Theorem. *This problem can be solved by taking an expected number of less than two pairwise* gcds *of random linear combinations of the input.*

⋆ Supported in part by NSF Grant CCR-9509783
⋆⋆ Supported in part by the Australian Research Council

T. Asano et al. (Eds.): COCOON'99, LNCS 1627, pp. 310–317, 1999.

We stress that we are not doing an *average-case analysis*, as is done in [11, Note 8.2], where the inputs $a_1, \ldots, a_m$ are randomly chosen according to some distribution, but rather a *worst-case analysis*: we are looking for probabilistic algorithms for which we can prove a good bound on the expected number of gcds that is true no matter what the inputs are. A different approach to solving this problem is implicit in [5, Proof of Lemma 1.2] while a brief and less precise treatment appears in [1]. We provide a refined algorithm together with a rigorous analysis and some experimental results.

The use of random linear combinations is readily suggested by the success of that method for polynomials but we do not know how to prove that the naive implementation of this idea works. Our main algorithmic contribution is a clever choice of the range for the random coefficients. There is a natural (heuristic) upper bound on the success probability of any algorithm; with our choice of the range, we can prove a lower bound that is reasonably close to that upper bound. In fact, the lower bound can be moved arbitrarily close to the upper bound, but at the expense of requiring very large coefficients, which limits the practical usefulness.

However, we believe that our method, free of any heuristic or distributional assumption, makes content computation for integer polynomials eminently practical.

2 Iterative gcd computation

By the associativity of the gcd

$$\begin{aligned}\gcd(a_1, \ldots, a_m) &= \gcd(\gcd(a_1, a_2), a_3, \ldots, a_m) \\ &= \gcd(\ldots(\gcd(\gcd(a_1, a_2), a_3), \ldots, a_m),\end{aligned}$$

the most obvious way to solve our problem is to compute the pairwise gcds successively and to stop whenever a result is 1. This iterative computation of pairwise gcds will work well for random inputs, but there are "nasty" inputs on which this method will need $m - 1$ pairwise gcd computations.

If the inputs are randomly chosen integers this naive algorithm usually does not need all the $m - 1$ steps:

Fact 1. *Let two integers a and b be chosen uniformly at random from the positive integers up to N. Then for large N*

$$\operatorname{prob}(\gcd(a, b) = 1) \to \zeta(2)^{-1} = \frac{6}{\pi^2} \sim 0.60793.$$

For a proof, see [8], Section 4.5.2.

In this case, the probability that a_1 and a_2 are already coprime is about 0.6, and for randomly chosen $a_1, \ldots, a_m$, we can therefore expect that the naive algorithm will already stop after about two computations of a pairwise gcd.

Heuristic reasoning says that a prime p divides a random integer a with probability p^{-1}, and m random integers with probability p^{-m}. Assuming the independence of these events, we have

$$\text{prob}(\gcd(a_1, \ldots, a_m) = 1) = \prod_p (1 - p^{-m}) = \zeta(m)^{-1}.$$

This can be made into a rigorous argument showing that the probability that m random integers, as in Fact 1, have trivial gcd tends (with $N \to \infty$) to $\zeta(m)^{-1}$. The first few values are:

m	$\zeta(m)^{-1}$
2	0.608
3	0.832
4	0.924
5	0.964
6	0.983
7	0.992
8	0.996
9	0.998
10	0.999

Although this strategy is quite successful for randomly chosen inputs, there exist "nasty" sequences, in which the $m - 1$ steps in the above algorithm are really necessary.

Example 1. Let $p_1, \ldots, p_m$ be the first m primes, $A = p_1 \cdots p_m$, and $a_i = A/p_i$ for each i. Then $\gcd(a_1, \ldots, a_m) = 1$ but for any proper subset S of $\{1, \ldots, m\}$, $\gcd(\{a_i : i \in S\})$ is non-trivial. So the successive computation of pairwise gcds does not give the right output until $m - 1$ steps in the above algorithm have been performed.

We do not address the question of representing the gcd as a linear combination of the inputs. This problem is considered in [9,6].

3 Random linear combinations

In the case of polynomials over a field F, the use of random linear combinations is known to be successful.

Fact 2. *Let F be a field, $a_1, \ldots, a_m \in F[x]$ be nonzero polynomials of degree at most d, $h = \gcd(a_1, \ldots, a_m)$, $A \subseteq F$ finite, $x_3, \ldots, x_m \in A$ be randomly chosen elements, and $g = a_2 + \sum_{3 \le i \le m} x_i a_i \in F[x]$. Then h divides* $\gcd(a_1, g)$, *and*

$$\text{prob}(h = \gcd(a_1, g)) \ge 1 - d/\#A.$$

This is based, of course, on resultant theory ([2], [3]), and also works for multivariate polynomials. So for the polynomial case, we expect that with only about one calculation of a pairwise gcd of two polynomials we find the true gcd of many polynomials, provided that A is chosen large enough.

We try to solve the corresponding integer problem by using a similar approach, and consider therefore the following algorithm.

Algorithm 1. *Probabilistic gcd of many integers.*

Input: m positive integers $a_1, \ldots, a_m$ with $a_i \leq N$ for all i, and a positive integer M.
Output: Probably $\gcd(a_1, \ldots, a_m)$.

1. Pick $2m$ uniformly distributed random integers $x_1, \ldots, x_m$ and $y_1, \ldots, y_m$ in $\{1, \ldots, M\}$, and compute $x = \sum_{1 \leq i \leq m} x_i a_i$ and $y = \sum_{1 \leq i \leq m} y_i a_i$.
2. Return $\gcd(x, y)$.

Then $g = \gcd(a_1, \ldots, a_m)$ divides $\gcd(x, y)$, and we can easily check equality by trial divisions by $\gcd(x, y)$. This is a Monte-Carlo algorithm and we want to prove a lower bound for its success probability, i.e., for the probability that $\gcd(x, y)$ equals g.

Due to Fact 1 we cannot expect this probability to be as high as in the polynomial case, but tests show that this strategy seems to have roughly the expected success probability of $6/\pi^2$. For two lists we have tested (with 5000 independent tests each) whether two random linear combinations of the list elements have the same gcd as the list elements. The list "nasty" has 100 elements as described in Example 1, whose largest entry has 220 decimal digits, and "random" is a random list with 100 elements of up to 220 decimal digits.

Table 1. Using random linear combinations

M	nasty list success rate	random list success rate
2	0.7542	0.6066
10	0.6522	0.6016
100	0.6146	0.6078
1000	0.6020	0.6098
30030	0.5952	0.6038

Table 1 shows that one gcd computation of two random linear combinations of the a_i gives the right answer in about 60% of the cases, i.e., in most of the cases, and this seems to be somewhat independent of the size of M. (The higher success rate for the nasty list using low M is readily explained.)

Our task is to *prove* that the probability that $\gcd(x, y) = g$ is high.

Remembering Fact 1, the solution of our problem would be easy if x and y were random numbers, but although the results in Table 1 show that they behave approximatively as if they were uniformly distributed, this is not literally true. So we have to find another way to bound the probability that $g \neq \gcd(x, y)$.

4 A probabilistic estimate

It would, of course, be sufficient to show that our random linear combination x behaves like a random integer in its range. Then the bound $\zeta(s)^{-1}$ would hold for the gcd of s random linear combinations. In any case, we cannot reasonably hope to have a better success probability than this bound. However, we want a (small) bound on M, possibly in terms of the inputs. Then, if $a_1, \ldots, a_m \leq N$, we have

$$x = \sum_{1 \leq i \leq m} x_i a_i \leq mMN = B,$$

but x is clearly not uniformly distributed in $\{1, \ldots, B\}$. Even if it were, the probability that a prime p divides x would not be the exactly desired $1/p$, but only close to it, since B is not necessarily a multiple of p. We circumvent this obstacle by choosing M to be the product of the first r primes. For $r = 6$, we have $M = 2 \cdot 3 \cdot 5 \cdot 7 \cdot 11 \cdot 13 = 30030$. (In fact, for our purposes a multiple of this number suffices.) Then for all primes p up to the rth prime p_r, p divides a random linear combination with probability exactly $1/p$. For our probability estimate, the main contribution becomes

$$\prod_{p \leq p_r} (1 - p^{-s}),$$

just as for the unavoidable $\zeta(s)^{-1} = \prod_p (1 - p^{-s})$. We only use this idea for $s = 2$, and abbreviate

$$\eta_r = \prod_{p \leq p_r} (1 - p^{-2}).$$

We fix the following notation. For $m \geq 2$ and $r \geq 1$, let $a_1, \ldots, a_m$ be positive integers, all at most N; let M be a multiple of $\Pi_r = p_1 \cdots p_r$. Let $x_1, \ldots, x_m$, $y_1, \ldots, y_m$ be uniformly distributed random integers between 1 and M; and let $x = \sum_{1 \leq i \leq m} x_i a_i$ and $y = \sum_{1 \leq i \leq m} y_i a_i$.

For the probability estimate we need the following lemma:

Lemma 1. *If* $\gcd(a_1, \ldots, a_m) = 1$*, then the events that* p_i *divides* x *for* $1 \leq i \leq r$ *are independent.*

Proof. Let $1 \leq k \leq r$. Since $\gcd(a_1, \ldots, a_m) = 1$, there exists an index j such that $p_k \nmid a_j$. Then for any $x_1, \ldots x_{j-1}, x_{j+1}, \ldots, x_m \in \mathbb{Z}$ the congruence

$$x_j a_j \equiv -\sum_{i \neq j} x_i a_i \bmod p_k$$

has exactly one solution x_j modulo p_k. Hence, there are p_k^{m-1} solutions with $1 \le x_1, \ldots, x_m \le p_k$ and

$$\sum_{1 \le i \le m} x_i a_i \equiv 0 \bmod p_k.$$

Now let $I \subseteq \{1, \ldots, r\}$ and $q = \prod_{k \in I} p_k$. By the Chinese Remainder Theorem, we have $\prod_{k \in I} p_k^{m-1} = q^{m-1}$ solutions modulo q giving $x \equiv 0 \bmod q$. Since $q \mid M$, this congruence has $q^{m-1} \cdot (M/q)^m$ solutions $1 \le x_1, \ldots, x_m \le M$. Hence,

$$\text{prob}\{q \mid x\} = \frac{q^{m-1} \cdot \left(\frac{M}{q}\right)^m}{M^m} = \frac{1}{q}. \tag{1}$$

In particular, $\text{prob}\{p_i \mid x\} = 1/p_i$ for each $i \le r$, and since (1) holds for each subset of $\{1, \ldots, r\}$, the events are independent. □

Theorem 1.
With the above notation, let P be the probability that $\gcd(a_1, \ldots, a_m) = \gcd(x, y)$. *Then*

$$P > \eta_r - \frac{2.04}{p_r \ln p_r} - \frac{2}{M}\left(\ln\ln M + \frac{1}{\ln^2 M} + \frac{1}{2\ln^2 p_r} - \ln\ln p_r\right) - \frac{mN}{M(\ln(mMN) - 3/2)} + \frac{r}{M^2}.$$

The proof of this theorem is by detailed analysis of various cases and uses several bounds from [10].

Corollary 1. *For any $\epsilon > 0$ one can choose r and M such that $P > \zeta(2)^{-1} - \epsilon$.*

Corollary 2. *Let $r \ge 6$. Then*

$$P > 0.5466 - \frac{mN}{M(\ln(mMN) - 3/2)}.$$

Proof. We have $\eta_r > 1/\zeta(2)$ for all $r \ge 1$, and

$$\frac{2}{M}\left(\ln\ln M + \frac{1}{\ln^2 M} + \frac{1}{2\ln^2 p_r} - \ln\ln p_r\right) < 0.0000983402$$

for $r = 6$. The left hand side expression decreases strictly for increasing r. Hence,

$$\begin{aligned} P &> 0.6079271016 - \frac{2.04}{13\ln 13} - 0.0000983402 - \frac{mN}{M(\ln(mMN) - 3/2)} \\ &> 0.5466489662 - \frac{mN}{M(\ln(mMN) - 3/2)}. \quad \square \end{aligned}$$

Corollary 3.
Let $r \geq 6$, and $M \geq 3mN$ be a multiple of $p_1 \cdots p_r$. Then $P > 51\%$.

Proof. Let $M \geq 3mN$. Indeed $M \geq 30030$ and we can also assume $m \geq 3$ and $N \geq 2$ (otherwise everything is trivial). Then

$$\ln(mMN) - 3/2 \geq \ln(3 \times 30030 \times 2) - 3/2 > 10,$$

so

$$\begin{aligned} P &> 0.5466489662 - \frac{mN}{M(\ln(mMN) - 3/2)} \\ &> 0.5466489662 - \frac{mN}{10M} \\ &> 0.5466489662 - 1/30 \\ &> 0.51. \quad \square \end{aligned}$$

So, if M is chosen large enough, Algorithm 1 has a success probability greater than 0.51. The results in Table 1 and other experiments suggest that P is even somewhat larger and rather independent both of the size of M and of its choice as a multiple of the small primes.

Thus we have a Monte-Carlo algorithm for which the result is correct in more than 50% of the cases. Hence after two independent executions of the algorithm we expect that the smaller value of g equals $\gcd(a_1, \ldots, a_m)$.

Acknowledgements

We are grateful to Mark Giesbrecht, Boaz Patt-Shamir and Igor Shparlinski for helpful discussions.

References

1. Gene Cooperman and George Havas, Elementary Algebra Revisited: Randomized Algorithms. In *Randomization Methods in Algorithm Design*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science **43** (1999) 37–44.
2. Angel Díaz and Erich Kaltofen, On computing greatest common divisors with polynomials given by black boxes for their evaluations. In *ISSAC'95* (Proc. Internat. Sympos. Symbolic and Algebraic Computation), ACM Press (1995) 232–239.
3. Joachim von zur Gathen, Marek Karpinski and Igor Shparlinski, Counting curves and their projections. *Computational complexity* **6** (1996), 64–99. (Extended Abstract in Proc. 25th ACM Sympos. Theory of Computing.)
4. K. O. Geddes, S. R. Czapor and G. Labahn, *Algorithms for Computer Algebra*. Kluwer Academic Publishers, 1992.
5. Mark Giesbrecht, Fast computation of the Smith normal form of an integer matrix, In *ISSAC'95* (Proc. Internat. Sympos. Symbolic and Algebraic Computation) ACM Press (1995) 110–118.

6. George Havas, Bohdan S. Majewski and Keith R. Matthews, Extended gcd and Hermite normal form algorithms via lattice basis reduction. *Experimental Mathematics* **7** (1998) 125–136.
7. Chung-Yen Ho and Chee Keng Yap, The Habicht approach to subresultants. *Journal of Symbolic Computation* **21** (1996), 1–14.
8. Donald E. Knuth, *The Art of Computer Programming, Vol.2, Seminumerical Algorithms*. Addison-Wesley, Reading MA, 3rd edition, 1997.
9. Bohdan S. Majewski and George Havas, A solution to the extended gcd problem. In *ISSAC'95* (Proc. Internat. Sympos. Symbolic and Algebraic Computation) ACM Press (1995) 248–253.
10. J. Barkley Rosser and Lowell Schoenfeld, Approximate formulas for some functions of prime numbers. *Illinois J. Math.* **6** (1962), 64–94.
11. Richard Zippel, *Effective polynomial computation*. Kluwer Academic Publishers, 1993.