# Extended gcd calculation

George Havas* and Bohdan S. Majewski†
Key Centre for Software Technology
Department of Computer Science
The University of Queensland
Queensland 4072, Australia

**Abstract**

Given an integer vector of $n$ positive numbers $a = [a_i]_{i=1}^n$ the extended gcd problem asks for an integer vector $x$ of length $n$ such that

$$xa^{\mathrm{T}} = \sum_{i=1}^{n} x_i a_i = \gcd(a_1, a_2, \ldots, a_n).$$

For many applications it is vital that some measure of $x$, $\|x\|$, is small. We have proved, however, that if we choose either the max norm or the zero metric the question of finding $x$ such that $\|x\|$ is smaller than some positive constant $K$ is NP-complete. We conjecture that the question remains NP-complete for other norms.

In the light of these results we have proposed two approximation algorithms. Their respective complexities are $O\left(n^2 \log(\max_i\{a_i\})\right)$ and $O\left(n^4 \log(\max_i\{a_i\})\right)$. Theoretical analysis of the algorithms leads to unsatisfactory bounds on the quality of the solution. Thus here we undertake a practical study of the methods, where their performance is matched against optimal solutions.

## 1 Introduction

The extended gcd problem asks that, for a given vector of $n$ positive integers $[a_i]_{i=1}^n$, we find an integer vector $x = [x_i]_{i=1}^n$ such that

$$xa^{\mathrm{T}} = \sum_{i=1}^{n} x_i a_i = \gcd(a_1, a_2, \ldots, a_n). \tag{1}$$

Sometimes we may be interested in finding an $n \times n$ unimodular matrix $P$ such that

$$Pa^{\mathrm{T}} = [g, 0, \ldots, 0]^{\mathrm{T}}, \tag{2}$$

*E-mail: havas@cs.uq.oz.au; partially supported by the Australian Research Council
†E-mail: bohdan@cs.uq.oz.au

where $g = \gcd(a_1, a_2, \ldots, a_n)$. In the latter form, the first row of $P$ corresponds to the vector $x$ and the remainder of $P$ is a basis for the null space of $a$. A benefit of finding a matrix $P$ is that we can easily obtain alternative vectors of multipliers, by adding integer linear combinations of other row vectors of $P$ to the first vector. Thus

$$x = p_1 + \sum_{i=2}^{n} t_i p_i, \tag{3}$$

where the $t_i$'s are integers and $p_i$ denotes the $i$th row of $P$.

In many applications it is profitable to have a 'small' vector $x$, where small is defined with respect to some selected norm or metric. Useful measures include: the zero metric, when a sparse vector $x$ is desired; the Euclidean norm, when a short vector is advantageous; and the max norm, when the worst case impact due to algebraic operations involving $x$ is minimized.

Computing a solution to (2), namely a matrix $P$, is easy, i.e., computable in polynomial time. However finding coefficients $t_i$ that minimize the length of $x$ is intractable [vEB81, Theorem 2]. Alternative measures do not help here, since minimizing $x$ with respect to the max norm is equally difficult [MH94, Corollary 7], likewise with respect to the zero metric [MH94, Theorem 1].

Moreover, unless bases of null spaces have some special features that allow finding a solution to the nearest vector problem in polynomial time, it follows that minimizing $x$ with respect to the Euclidean norm is NP-complete as well. A simple informal argument is as follows. Suppose that we are given a matrix $P$, and we want to know whether there are coefficients $t_i$ such that $x$ given by (3) has Euclidean length not exceeding a given integer constant $K$. To answer this question in polynomial time we use an algorithm that returns $x^*$, a shortest solution to the extended gcd problem. Knowing $x^*$ allows us to check if there exists a solution to (3) with length not exceeding $K$. To retrieve the coefficients $t_i$ we solve

$$P \begin{bmatrix} 1 \\ t_2 \\ t_3 \\ \vdots \\ t_n \end{bmatrix} = x^* \tag{4}$$

over the integers. Solvability of (4) over the integers in polynomial time is a consequence of results in [KB79, CC82].

Finally, in Section 3 we show that there is an easy way of obtaining integer linear programs for the $L_1$ and max norms. ILP is another well-known NP-complete problem [GJ79].

The above discussion indicates that in all likelihood finding short solutions to the extended gcd problem with respect to other norms is difficult. Therefore it makes sense to investigate approximation methods that are capable of finding reasonably small $x$'s. Two such methods are outlined in the following section.

# 2 The algorithms

The algorithms described in this section are presented in [MH95] and [HMM95]. We outline them here for convenient reference and mention a few enhancements that lead to improved performance.

## The sorting gcd algorithm

For a given integer vector $a = [a_i]_{i=1}^n$ we start by assigning $d_i := a_{\pi(i)}$, for $i = 1, 2, \ldots, n$, where $\pi$ is a random permutation of numbers 1 to $n$. Let $B$ be an $n \times n$ integer matrix $[[b_{ij}]_{j=1}^n]_{i=1}^n$. Initialize $B$ by assigning the $n \times n$ identity matrix, $I_n$, to it. Throughout the algorithm we have

$$\sum_{j=1}^{n} a_{\pi(j)} b_{ij} = d_i, \qquad i = 1, 2, \ldots, n.$$

During the algorithm each operation on a value $d_i$ is also performed on the corresponding row $i$ of $B$.

Select two elements $d_i$ and $d_j$, where $d_i$ is the largest element and $d_j$ is the second largest element in the sequence; resolve ties arbitrarily. If $d_j = 0$ then stop. At this stage, the $i$th row of $B$ contains a set of multipliers. Assign $b_{i\pi^{-1}(j)}$ to $x_j$ and return the vector $[x_j]_{j=1}^n$. Otherwise, subtract $d_j$ from $d_i$, $\lfloor d_i/d_j \rfloor$ times and do the same with rows $i$ and $j$ of $B$. Return to the selection step.

To improve the performance of the algorithm we may use the following techniques. Suppose that more than one element is equal to the second largest value among the $d_k$'s. Denote them by $d_{j_1}, d_{j_2}, \ldots, d_{j_p}$. If $q = \lfloor d_i/d_{j_1} \rfloor \neq 1$, instead of subtracting $qd_{j_1}$ from $d_i$ we may distribute $q$ among all the $d_{j_k}$'s, $1 \leq k \leq p$. Furthermore, if the row associated with $d_{j_1}$ is 'better' than the row associated with $d_{j_2}$, the distribution of $q$ should be adjusted appropriately, so that $d_{j_1}$ is subtracted more times from $d_i$ than $d_{j_2}$. In our practical experiments we have found that using a combination of the max norm and zero metric as the selection criterion works the best.

Also, throughout the algorithm we actively check each $d_k$ to see if its value is equal to $\pm \gcd(a_1, a_2, \ldots, a_n)$. For any such $d_k$ we compare the length of the $k$th row of $B$ with the shortest known solution, and if it is better we make the $k$th row the new best solution. Thus at the end of the algorithm instead of returning the $i$th row of $B$ we return the stored best solution.

Note that this algorithm does in fact find a solution to equation (2). The other vectors of $B$ give a basis for the null space, i.e., the rest of $P$.

## A LLL-based gcd method

Keeping entries small in integer matrix computations is closely related to finding small bases for integer lattices. The LLL method [LLL82] provides an effective algorithm for finding a short basis of a given lattice. The idea behind our LLL gcd method is to supply the LLL algorithm with a 'long' basis of a lattice constructed so that the LLL-reduced basis gives a solution to (2). It is proved in [HMM95] that one such

initial basis has the form

$$B = \begin{bmatrix} 1 & \cdots & 0 & \gamma a_1 \\ & \ddots & & \vdots \\ 0 & \cdots & 1 & \gamma a_n \end{bmatrix}$$

where $\gamma$ is a sufficiently large constant. After an application of the LLL algorithm the matrix $B$ has the form

$$B = \begin{bmatrix} b_{11} & \cdots & b_{1n} & 0 \\ & \ddots & & \vdots \\ b_{n1} & \cdots & b_{nn} & \gamma g \end{bmatrix}$$

where $g = \gcd(a_1, a_2, \ldots, a_n)$. Thus the matrix $B$, minus the last column, corresponds to the matrix $P$ in (2) and the last column of $B$ corresponds to the right hand side of (2), with the top row swapped to the bottom. The bottom vector of $B$ is usually a very short solution to the extended gcd problem.

Further, it is shown in [HMM95] that by making appropriate modifications to the LLL algorithm we can dispose of $\gamma$ entirely. This gives us an efficient version of the LLL gcd method.

In order to improve the average performance of the LLL-based algorithm we may consider splitting the vector $a$, and hence the matrix $P$, into a number of blocks. Initially the reduction is carried out within each block, without any interaction between them. Next two or more neighbouring blocks are merged together and further reduction steps are done. The purpose of this approach is to start the LLL algorithm with many (partial) solutions to the extended gcd problem. Having more diversified information may lead to better, shorter solutions, especially for dense sets, i.e., sets where the ratio $\max_i\{a_i\}/n$ is small.

# 3 Obtaining optimum solutions

In order to judge the quality of the solutions returned by the two approximation methods we need to employ potentially exponential time algorithms that are guaranteed to return the best possible solution with respect to a given measure. Each measure requires a different solution, since an optimal solution with respect to one measure may not be optimal with respect to another, as illustrated in Figure 1.

| vector | 811 | 963 | 3395 | 6942 | 9631 | 4670 | 9775 | 2001 | 7886 | 4031 | 22268 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $L_0$ metric | 19 | −16 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| $L_1$ norm | 3 | 1 | −1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| $L_2$ norm | −1 | 0 | 0 | 0 | −1 | 1 | 1 | −2 | 0 | 0 | 0 |
| $L_\infty$ norm | 1 | 1 | 1 | 1 | 1 | 1 | 1 | −1 | −1 | −1 | −1 |

Figure 1: Different optimal solutions

## Euclidean norm

The best solution with respect to the Euclidean norm may be obtained from (3) using an algorithm due to Fincke and Pohst [PZ89, p 191]. The algorithm searches for solutions in integers $t_2, \ldots, t_n$ of the inequality

$$\|p_1 - \sum_{i=2}^{n} t_i p_i\|^2 \leq \|p_1\|^2.$$

Here we use $p_i$ which are LLL-reduced rows of the matrix $P$, with $p_1$ being the multiplier vector. In its simplest version, the Fincke-Pohst algorithm returns all integer solutions of the inequality. By replacing $p_1$ with a shorter vector whenever one is found, we obtain a vector of the shortest length.

## Max norm

To minimize the maximum ($L_\infty$) norm we must minimize

$$\max_i \left| p_{1i} - \sum_{j=2}^{n} t_j p_{ji} \right|. \tag{5}$$

Chvátal [Chv83, pp 221–226] shows how to transform (5) into an ILP problem. By solving the following instance of ILP

$$\begin{array}{lll}
\text{minimize:} & z & \\
\text{subject to:} & z + \sum_{j=2}^{n} t_j p_{ji} \geq p_{1i}, & (i = 1, 2, \ldots, n) \\
& z - \sum_{j=2}^{n} t_j p_{ji} \geq -p_{1i}, & (i = 1, 2, \ldots, n) \\
& t_j \in \mathbb{Z}, & (j = 2, 3, \ldots, n)
\end{array}$$

we are guaranteed to find the best possible solution with respect to the max norm. The reason for that is as follows [Chv83, p 223].

Let $z^*, t_2^*, t_3^*, \ldots, t_n^*$ be an optimal solution, i.e., a solution with the smallest possible value of the objective function. The number $z^*$ is hit against the largest of the $2n$ lower bounds. Hence

$$z^* = \max_i \left| p_{1i} - \sum_{j=2}^{n} t_j p_{ji} \right|$$

and thus $t_2^*, t_3^*, \ldots, t_n^*$ is the best max norm approximation to a solution of (2). Note also that the following ILP solves the extended gcd problem with respect to the $L_1$ norm:

$$\begin{array}{lll}
\text{minimize:} & \sum_{i=1}^{n} e_i & \\
\text{subject to:} & e_i + \sum_{j=2}^{n} t_j p_{ji} \geq p_{1i}, & (i = 1, 2, \ldots, n) \\
& e_i - \sum_{j=2}^{n} t_j p_{ji} \geq -p_{1i}, & (i = 1, 2, \ldots, n) \\
& t_j \in \mathbb{Z}, & (j = 2, 3, \ldots, n)\ .
\end{array}$$

# 4 Comparison

Theoretical analyses of these algorithms are far from satisfactory. For example, upper bounds derivable on the basis that the rows of $P$ are LLL-reduced indicate that the length of the solution will not exceed $O\left(2^{(n-1)/2}\log(\min_i\{a_i\})\right)$. In practice we obtain solutions of much better quality. Even though the upper bound for the sophisticated LLL gcd method is worse than that given for the simple gcd tree algorithm [MH94, Theorem 9], the LLL-based algorithm gives much better solutions than the gcd tree method. (A lower bound for arbitrary $n$ on the max norm of an optimal solution is given in [MH94, Section 3]. If we insist that the numbers are distinct a lower bound is given in [HMM95].)

The aim of this study is to shed some light on the average performance of the sorting gcd method and LLL gcd algorithm. We are primarily interested in two parameters: the average value of the maximum multiplier (the average max norm) and the average length of the multiplier vector (the average Euclidean norm). We assess these against the optimum solution for the Euclidean norm given by the Fincke-Pohst algorithm.

For our analysis here we use two parameters: $n$, the number of integers in the vector $a$; and the maximum allowed integer in the vector. Vectors with 10, 15 and 20 integers selected uniformly at random from the range $[1, \lfloor e^i \rfloor]$, where $i = 7, 8, \ldots, 18$ were used as input vector $a$. For each fixed pair of parameters, 250 runs were executed and the average over those runs was calculated. We plot (on the $y$-axis) in Figures 2 to 7 the average observed Euclidean lengths and observed maximum entries of the solution vector against the chosen top of the range (on the $x$-axis).

These results show that the LLL gcd method performs exceptionally well. The average difference between the length of truly optimal solutions and those obtained by the LLL gcd method never exceeded 0.64, a remarkable feat indeed. Moreover, for 20 random numbers the LLL gcd method never had worse average max norm than that obtained by the Fincke-Pohst algorithm. Even though neither of the methods aims at minimizing the max norm explicitly, they both performed exceptionally well, with the LLL gcd method outperforming Fincke-Pohst on a number of occasions.

The sorting gcd method seems, in comparison, to perform rather badly. We however need to keep in mind that even for numbers as large as $\lfloor e^{18} \rfloor = 65659969$ the average values of the max norm were 11.76, 4.86 and 3.56 for 10, 15 and 20 numbers, respectively. Moreover, for sets where the number of integers is significant in comparison with their size, the sorting gcd method outperformed the LLL gcd algorithm (this occured for 15 numbers selected from the range $[1, 1097]$ and 20 numbers selected from the ranges $[1, 1097]$ and $[1, 2980]$). By observing the average Euclidean norms of the solutions, we can conclude that for the majority of problems within these ranges there exist one or more solutions with three units. That is where the sorting gcd method may be superior. Its minimum change strategy naturally leads to discovering first some solutions with just two nonzeros, then three nonzeros, etc. Providing that for a given set there is a significant percentage of solutions with three units, chances are that one or more of them will be discovered by the sorting gcd method. On the other hand the LLL gcd algorithm starts by computing the gcd of the first two numbers in the provided vector. Next it proceeds to compute the gcd

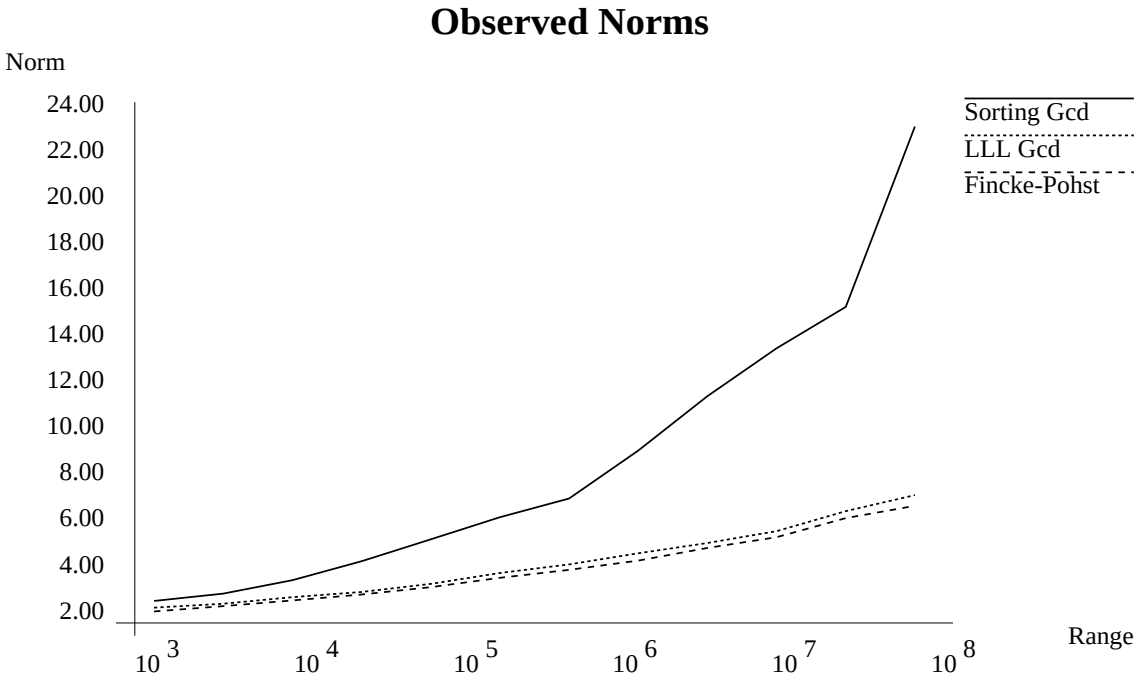


Figure 2: Lengths for 10 random numbers

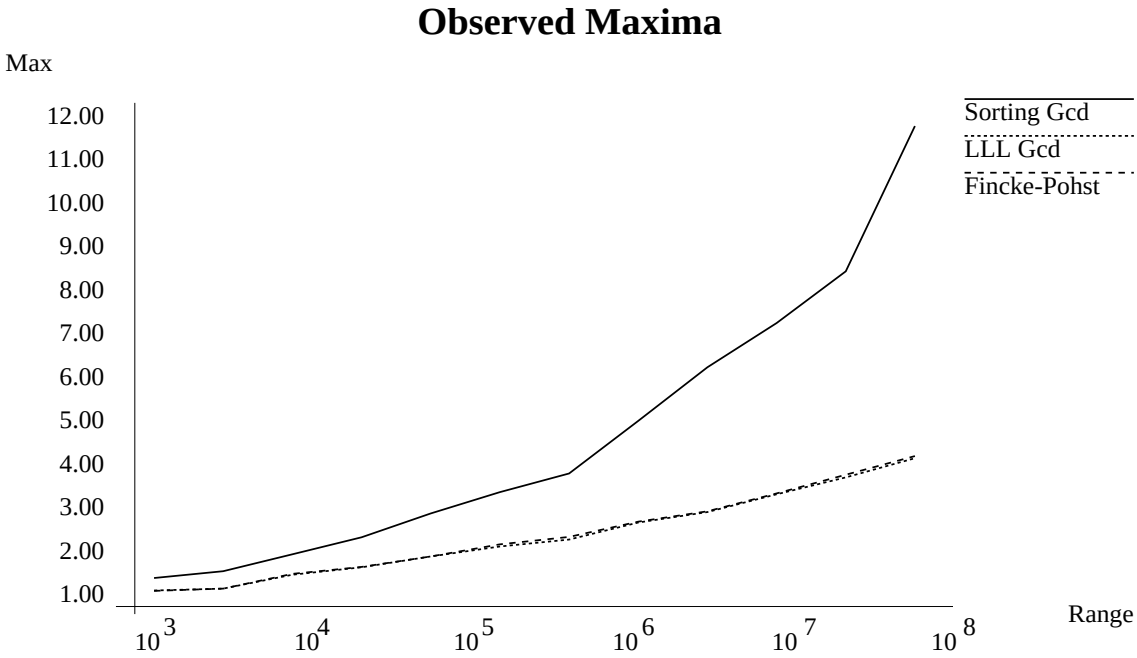


Figure 3: Maximum entries for 10 random numbers

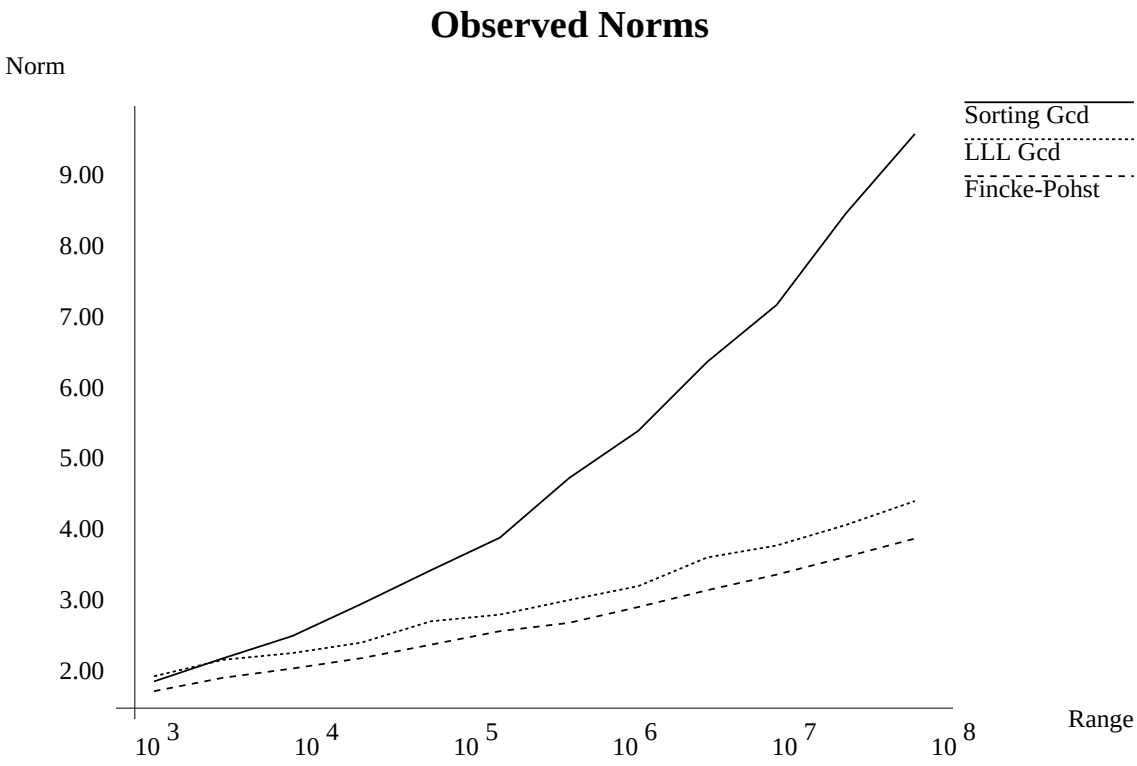


Figure 4: Lengths for 15 random numbers

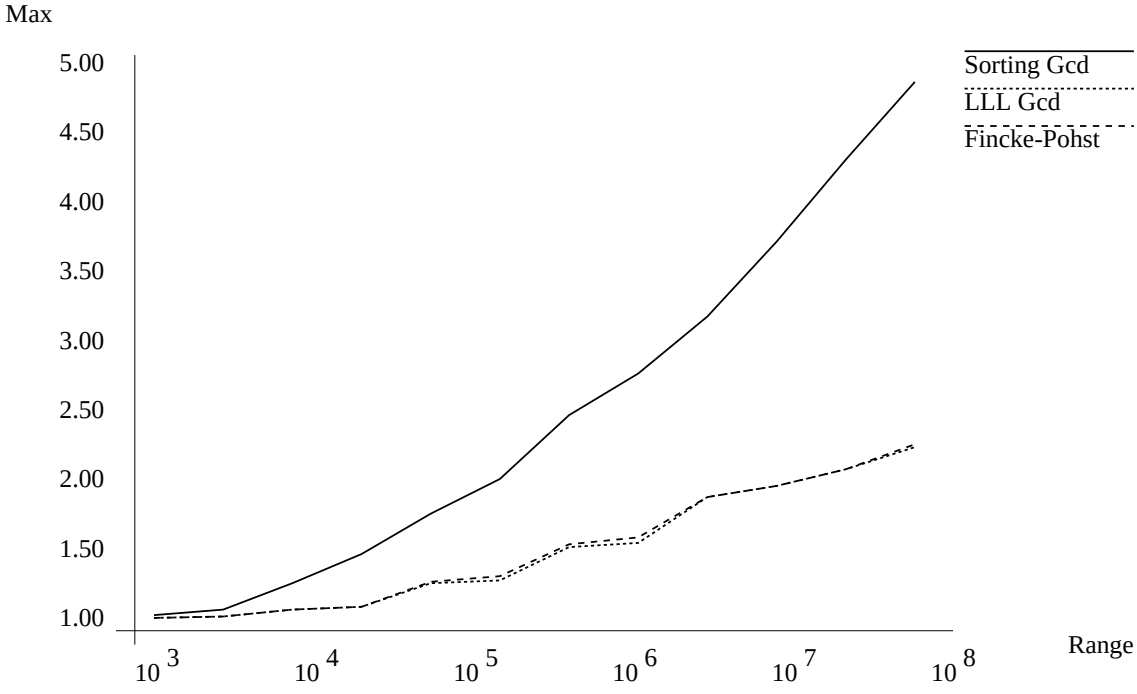


Figure 5: Maximum entries for 15 random numbers

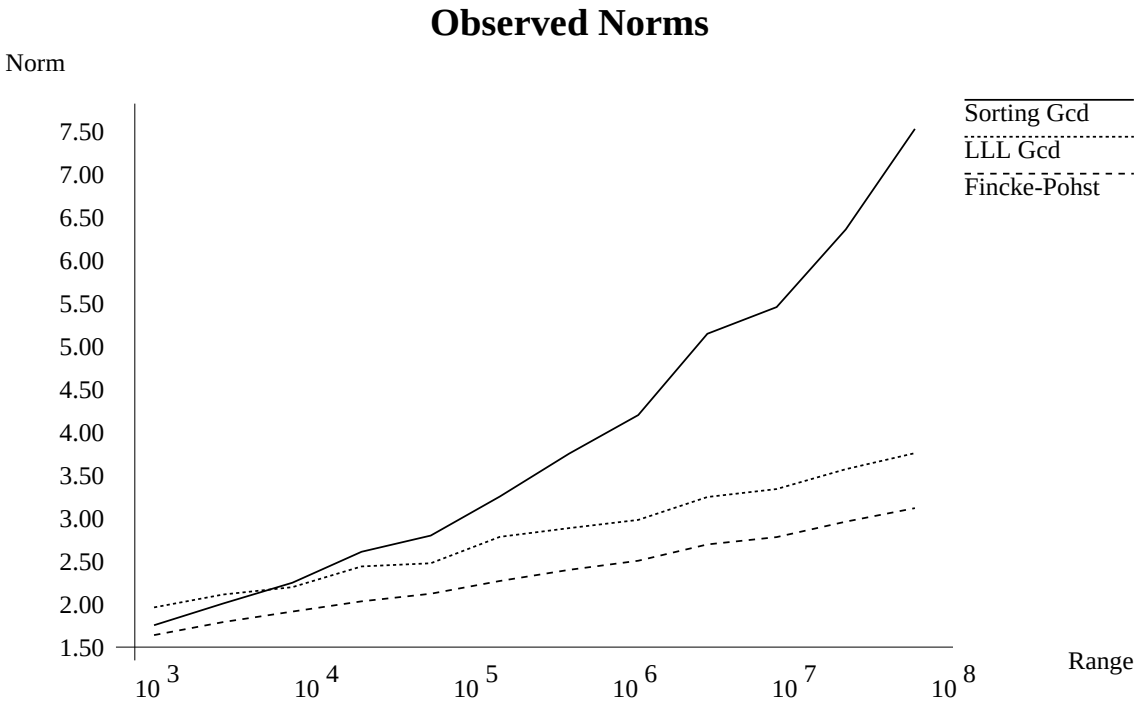


Figure 6: Lengths for 20 random numbers

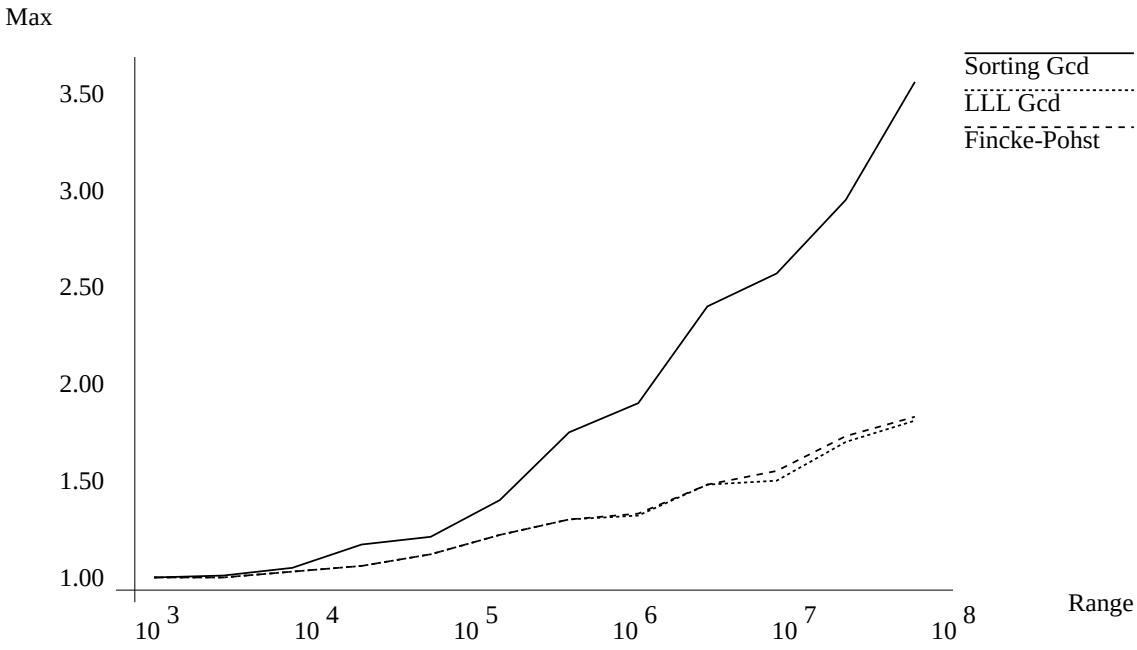


Figure 7: Maximum entries for 20 random numbers

of the first three numbers and uses the then 2 vectors from the null space to improve the quality of the obtained solution. Next the solution is extended to four numbers, etc. As it does not have to be the case that a solution with three units uses the first three numbers, it is likely that such a solution will not be discovered by LLL, and the refinement process is not exact enough to push the algorithm towards it.

As the numbers grow and more of them are necessary to obtain a good quality solution to the extended gcd problem, the sorting gcd method performs less well. This is due to the fact that partial solutions obtained in earlier steps are never improved in subsequent steps. Unlike the LLL gcd method, where each partial solution is immediately improved via existing vectors from the null space, the sorting gcd method has no such mechanism. If a partial solution is not a good one, this deficiency will be propagated in future steps. Thus the sorting gcd strategy is optimistic in the sense that, once it has obtained partial solutions, it accepts and uses them without any effort at improving them. On the other hand, the LLL gcd algorithm is highly skeptical about its partial solution, and at each step uses all available information to improve it. Note that the LLL gcd method maintains only one (partial) solution, while the sorting gcd method can and usually does obtain a number of them.

# 5 Conclusions

We have presented a practical study of two well-performing approximation methods for solving the NP-complete task of minimizing solutions to extended gcd problems. The first algorithm, the sorting gcd method, performs reasonably well. However, for large input numbers, i.e., numbers that exceed $e^{n/2}$, its performance deteriorates due to lack of corrective mechanisms. The second method, the LLL gcd algorithm, exhibits excellent performance. This comes at a cost however, as its theoretical complexity is $O(n^2)$ times that of the sorting gcd method, and this is indeed reflected in practical computation times. Both methods provide significantly better quality solutions than earlier approaches (see [MH95]).

# Acknowledgments

We are grateful to Dr K. Matthews for providing computational assistance, especially with the Fincke-Pohst algorithm.

# References

[CC82] T-W.J. Chou and G.E. Collins. Algorithms for the solution of systems of linear Diophantine equations. *SIAM J. Comput.*, 11:687–708, 1982.

[Chv83] V. Chvátal. *Linear Programming*. W.H. Freeman and Company, New York, 1983.

[GJ79] M.R. Garey and D.S. Johnson. *Computers and Intractibility: A Guide to the Theory of NP-completeness*. W.H. Freeman, San Francisco, 1979.

[HMM95] G. Havas, B.S. Majewski, and K.R. Matthews. Extended gcd algorithms. Technical Report TR0302, The University of Queensland, Brisbane, 1995.

[KB79] R. Kannan and A. Bachem. Polynomial algorithms for computing Smith and Hermite normal forms of an integer matrix. *SIAM J. Comput.*, 8:499–507, 1979.

[LLL82] A.K. Lenstra, H.W. Lenstra Jr., and L. Lovász. Factoring polynomials with rational coefficients. *Math. Ann.*, 261:515–534, 1982.

[MH94] B.S. Majewski and G. Havas. The complexity of greatest common divisor computations. In *Algorithmic Number Theory*, Lecture Notes in Computer Science 877, 184–193, 1994.

[MH95] B.S. Majewski and G. Havas. A solution to the extended gcd problem. In *Proceedings of the 1995 International Symposium on Symbolic and Algebraic Computation – ISSAC'95*, ACM Press, New York, 1995.

[PZ89] M. Pohst and H. Zassenhaus. *Algorithmic Algebraic Number Theory.* Cambridge University Press, 1989.

[vEB81] P. van Emde Boas. Another NP-complete partition problem and the complexity of computing short vectors in a lattice. Technical Report MI/UVA 81–04, The University of Amsterdam, Amsterdam, 1981.