A solution to the extended gcd problem

Bohdan S. Majewski* and George Havas†
Department of Computer Science
University of Queensland
Queensland 4072
Australia

Abstract

An improved method for expressing the greatest common divisor of n numbers as an integer linear combination of the numbers is presented and analyzed, both theoretically and practically. The performance of this algorithm is compared with other methods, indicating substantial improvements in the size of the solution.

The results are given in the light of the current knowledge about the complexity of extended gcd computations. Thus, finding optimal sets of multipliers has been proved to be an NP-complete problem. We present a relatively efficient approximation algorithm with excellent performance.

This problem is interesting in its own right. Furthermore, it has important applications, for example in computing canonical normal forms of integer matrices.

1 Introduction

The Euclidean algorithm for computing greatest common divisors is perhaps the oldest proper algorithm known [9, Section 4.5.2]. A closely related problem, the extended gcd problem, calls for solving

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

for a given sequence of integers $A = \langle a_i \rangle_{i=1}^{n}$. Extended gcd computation is of particular interest in number theory (see [3, chapters 2 and 3]) and in computational linear algebra ([5, 6, 8]), in both of which it takes a fundamental role in basic algorithms. An overview of some of the earlier history of the extended gcd is given in [3], showing that it dates back at least to Euler, while additional references and more recent results appear in [11].

In general, finding a solution to an extended gcd problem is easy, i.e. computable in polynomial time [9, Section 4.5.2]. For just 2 integers we also can readily find the best solution [10]. However, finding "good" solutions for arbitrary n is much more difficult, as the following theorem shows.

Theorem 1 [11, Corollary 7] *Given a positive integer K and a sequence of n positive integers $A = \langle a_1, \ldots, a_n \rangle$, the task of expressing $g = \gcd(a_1, \ldots, a_n)$ in the form*

$$\sum_{i=1}^{n} x_i a_i = g$$

with $|x_i| \leq K$, is NP-complete.

This means that it is difficult to find an optimal solution with respect to the L_∞ (or max) norm. However, there are a number of algorithms that solve the extended gcd problem. The time complexity of the task has been characterized by Bradley:

Theorem 2 [2] *The number of iterations of the Euclidean algorithm for n integers is less than $n-1$ plus the logarithm of the smallest number to the base* 1.6.

This bound given by Bradley has a significant deficiency: it necessitates that any computation must start with the smallest integer. This can be too restrictive in the context of finding good solutions in terms of the multipliers. Instead we use the following definition of time and space optimality.

Definition 3 *An extended gcd algorithm is time and space optimal if, for a sequence of integers $A = \langle a_1, \ldots, a_n \rangle$, the algorithm executes at most $O(n \log(\max_i\{a_i\}))$ elementary operations and requires at most $O(n)$ words, $\log(\max_i\{a_i\}))$ bits long.*

Several attempts have been made to construct algorithms that return small multipliers. Bradley controlled the growth of multipliers x_2 to x_n, at the expense of the first multiplier.

Theorem 4 [2] *There is an algorithm which computes g_n (the gcd of any sequence of integers $a_1, a_2, \ldots, a_n$) and n integers $x_1, x_2, \ldots, x_n$ such that*

$$\sum_{i=1}^{n} x_i a_i = g_n$$

with $x_i \leq g_{i-1}/(2g_i)$, for $i \geq 2$, where $g_i = \gcd(a_1, \ldots, a_i)$.

Iliopoulos presented a method based on a binary tree of computations and proved the following bound on the size of the largest multiplier.

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

ISSAC'95 - 7/95 Montreal, Canada

Theorem 5 [8, Corollary 1.6] *There is an algorithm which computes g (the gcd of any sequence of integers $a_1, a_2, \ldots, a_n$ each of which has length not exceeding β bits) and n integers $x_1, x_2, \ldots, x_n$ such that*

$$x_1a_1 + x_2a_2 + \cdots + x_na_n = g$$

with

$$\log(\max_i\{x_i\}) = O(\beta \log n)$$

in $O(n \log \beta M(\beta))$ elementary operations. Here $M(\beta)$ is the time required to multiply two integers β bits long.

This result has the bound on the largest number growing with the number of numbers, a deficiency we have remedied. Thus, we have improved on the last result, and presented a method that guarantees a solution for arbitrary n which maintains the same bound as the solution for $n = 2$.

Theorem 6 [11, Theorem 9] *There is an optimal time and optimal space algorithm for computing the extended gcd of n integers $\langle a_1, \ldots, a_n \rangle$, which guarantees that no multiplier is larger than the largest of the numbers divided by 2.*

Without some further restriction on the sequence A, this result cannot be further improved, as is confirmed by the sequence $\langle 2, 2k+1, 2k+1, \ldots, 2k+1 \rangle$ for $k \geq 1$ [11]. On the other hand, for sequences containing a more diverse selection of numbers, we may reasonably expect better performance. The improved algorithm, described in the next section, is designed precisely with this in mind.

2 An improved method

We propose and analyze another method for solving the extended gcd problem. The method is loosely based on the algorithm for solving the extended gcd problem suggested by Blankinship [1] and is similar to a method due to Brun [3, section 3H]. The results obtained by our algorithm, which we call the sorting gcd method, are of very good quality. A description of the algorithm follows.

Input: A sequence of positive integers $A = \langle a_1, a_2, \ldots, a_n \rangle$.
Output: An integer sequence $X = \langle x_1, x_2, \ldots, x_n \rangle$, such that $\sum_{i=1}^{n} a_i x_i = \gcd(a_1, a_2, \ldots, a_n)$.

Method: Assign $d_i := a_{\pi(i)}$, for $i = 1, 2, \ldots, n$, where π 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 the following holds

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

We say that value d_i is associated with row i of B, and any operation on d_i is copied for 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 ith 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. □

The idea behind the method is to use as many elements as reasonable to reduce each individual number. Instead of reducing numbers quickly by using the smallest number, which naturally leads to relatively large multipliers, we reduce numbers by a relatively close number, which leads to somewhat slower reduction but better multipliers.

By (in effect) constructing a random row permutation of the identity matrix (which is done by randomly permuting the input sequence) we avoid creating patterns in B, which in turn avoids a snowballing effect in size of the elements of B. Various other improvements are possible (cf. [7]), however for analysis purposes we restrict our attention to this relatively simple version of the algorithm.

3 Theoretical analysis

In this section we present an analysis of the algorithm. While an exact analysis must depend on the distribution and values of the numbers in the sequence A, we can obtain a good understanding of the algorithm by assuming a random uniform distribution of the input numbers. In a sense, the analysis obtained this way may be regarded as optimistic, since in practical computations we can expect somewhat skewed distributions. On the other hand, our experience indicates that it still provides a fairly accurate picture.

Theorem 7 *The complexity of the sorting gcd method is $O\big(n^2 \log(\max_i\{a_i\})\big)$.*

Proof The length of any number d_i decreases by at least 1 after d_i takes part in a row operation. (For any two positive integers $b \leq a$ we have $\log(a \bmod b) \leq \log(a) - 1$, that is, $a \bmod b < a/2$.) Hence any number d_i, and the row associated with it, cannot be modified more than $\log d_i$ times. Multiplying the number of rows by the complexity of a single row operation and the number of times such an operation can be performed proves the theorem. (The cost of selection is dominated by the complexity of the row operations.) □

In the following paragraphs we establish the expected density of the entries in the unimodular matrix B for the sorting gcd method. In the analysis we make a number of simplifications. We assume that nonzeros are distributed uniformly and randomly throughout each row of B. Furthermore, in order to obtain a closed form for the expected number of nonzeros in B we presume that a pair of rows that take part in a reduction is randomly and uniformly selected from all possible distinct pairs. This way we overestimate the rate at which B fills in. However the analysis shows the general behavior of the fill-in and allows some predictions to be made.

Let t_i^r be the number of nonzeros in row i of an $n \times n$ integer matrix B, and let $f_r(B)$ be the total number of nonzeros in B after r row operations. Thus $f_r(B)$ is the fill-in factor of B, $f_r(B) = \sum_{i=1}^{n} t_i^r$. For brevity we denote $f_r(B)$ by f_r, when B is understood. The initial condition is assumed to be $f_0(B) \equiv f_0$. Notice that the fill-in, as defined by Rose and Tarjan [13], can be easily calculated as $f_{r+1} - f_r$.

In the computations we may ignore or include fortunate cancellations. We are only interested in the number of zeros in a matrix, not in the size of the elements stored. Consequently we may simplify the description by treating elements of B as binary, i.e. $\{0, 1\}$, variables. Thus t_i^r counts the number of 1's in row i, that is, $t_i^r = \sum_j b_{ij}$.

When fortunate cancellations are disregarded, a row operation (which we call a *reduction* due to its impact on the associated value of d_i), between rows i and j is defined by

$$b_{ik} := \begin{cases} 1 & \text{if } b_{ik} = 1 \\ b_{jk} & \text{otherwise} \end{cases}$$

for $k = 1, 2, \ldots, n$. To take fortunate cancellations into account we introduce an additional parameter, p_0. If both b_{ik} and b_{ij} are 1 we throw a biased coin that comes up heads with probability p_0. If the coin comes up heads, b_{ik} is reset to 0, and remains 1 otherwise. Notice that fortunate cancellations do not affect either 1's in row i that are aligned against 0's in row j or 0's in row i that are aligned against 1's in row j.

In the uniform model t_i^r 1's in each row are distributed randomly among all of the columns. Thus matrix B, regardless of its sparsity, has no preexisting patterns of nonzeros. Under such an assumption we have the following result.

Theorem 8 *Let B be a binary matrix with the reduction operation as defined above. Denote by p_{ij} the probability that row i is reduced by row j. Then the expected number of nonzero elements in B after r reductions is*

$$f_{r+1} = f_r + \sum_i \sum_j t_j^r p_{ij} - \frac{1}{n} \sum_i t_i^r \sum_j t_j^r p_{ij} \qquad (1)$$

Proof Consider a reduction operation between row i and j. Let $h(i,j)$ be the number of 1's from row j that are aligned against 1's in row i. With the assumed random distribution of nonzeros in both rows, the number of ways this can be arranged is $\binom{t_i^r}{h(i,j)}\binom{n-t_i^r}{t_j^r-h(i,j)}$. The probability that $h(i,j) = k$ can be expressed as

$$\Pr(h(i,j) = k) = \frac{\binom{t_i^r}{k}\binom{n-t_i^r}{t_j^r-k}}{\binom{n}{t_j^r}}.$$

To compute the expected value of $h(i,j)$, which we write $\tilde{h}(i,j)$, we apply the following transformations

$$\begin{aligned} \tilde{h}(i,j) &= \sum_k k \frac{\binom{t_i^r}{k}\binom{n-t_i^r}{t_j^r-k}}{\binom{n}{t_j^r}} \\ &= \frac{t_i^r}{\binom{n}{t_j^r}} \sum_k \binom{t_i^r - 1}{k-1}\binom{n-t_i^r}{t_j^r-k} \\ &= \frac{t_i^r}{\binom{n}{t_j^r}} \binom{n-1}{t_j^r-1} \\ &= \frac{t_i^r t_j^r}{n}. \end{aligned}$$

To see that the last result is intuitively correct, consider a few simple cases. If $t_i^r = t_j^r = 1$, we have a 1 in n chance that the ones are in the same column, thus $\tilde{h}(i,j) = 1/n$, which is correct. If either t_i^r or t_j^r is equal to n, the number of collisions must be equal to the number of ones in the other row, as indicated by $\tilde{h}(i,j)$.

The expected number of new 1's in the matrix, after a single reduction operation between two rows i and j, is $t_j^r - \tilde{h}(i,j)$. By adding the expected number of new 1's to the number of existing 1's, for all possible choices of i and j, taking into account that each choice has probability p_{ij} of occurring, we get:

$$\begin{aligned} f_{r+1} &= \sum_{\substack{1 \le i,j \le n \\ j \ne i}} p_{ij} \left(f_r + t_j^r - \frac{t_i^r t_j^r}{n} \right) \\ &= f_r \times \sum_{\substack{1 \le i,j \le n \\ j \ne i}} p_{ij} \\ &\quad + \sum_{i=1}^n \left[\sum_{j=1}^n \left(1 - \frac{t_i^r}{n}\right) t_j^r p_{ij} - (t_i^r - h(i,i)) p_{ii} \right] \\ &= f_r + \sum_{i=1}^n \left(1 - \frac{t_i^r}{n}\right) \sum_{j=1}^n t_j^r p_{ij} \end{aligned}$$

The last maneuver is legitimate because $p_{ii} = 0$ and $h(i,i) = t_i^r$. Expanding the sum over i proves the theorem. □

In a pair of rows used in a reduction, one row is the operator and the other is being reduced. The probability p_{ij} may be defined so that it grades the operator row but is indifferent to the reduced row. In such a case, equation (1) is easy to interpret. The first double sum represents the average density of the operator row, with the selection criterion defined by p_{ij}. The second double sum is the average collision rate between the operator and the reduced row. The difference between these two gives the average number of new ones in matrix B after one reduction.

We now consider a random, or "don't care", approach, where each member of the pair of rows for the next reduction is picked totally at random. Before we state the theorem that describes the fill-in for this case, we introduce some notation from Graham, Knuth and Patashnik [4]. For any boolean statement S, the bracketed notation $[S]$ stands for 1 if S is true, 0 otherwise. In particular $[i = j]$ takes value 1 if and only if indexes i and j are equal.

Theorem 9 *Let the probability of selecting a pair (i,j) for the next reduction operation be equal for all pairs (i,j). Thus*

$$p_{ij} = \frac{1}{n(n-1)} - \frac{[i=j]}{n(n-1)}.$$

Clearly $\sum_{i,j} p_{ij} = 1$. It follows that the expected fill-in after r operations is characterized by the following recurrence

$$f_{r+1} = f_r \left(1 + \frac{1}{n}\right) - \frac{f_r^2 - \sum_i (t_i^r)^2}{n^2(n-1)}.$$

Proof Straightforward. □

The recurrence obtained above is interesting in its own right. Thus, we now study the asymptotic behavior of $\langle f_r \rangle$ for increasing r. While, for the purposes of the algorithm, we are only interested in f_r such that in the $(r-1)$th step there are at least two nonzero d_i's, here we consider a more general situation. In addition, as f_r can increase by fractional quantities, it makes sense to test the limits for r tending to infinity, without requiring n to be arbitrarily large. We begin by proving a convergence lemma.

Lemma 10 *For the initial condition $f_0 \le n^2$ the value of f_r converges to n^2, that is*

$$\lim_{r \to \infty} f_r = n^2.$$

Proof Let $f_0 < n^2$ be the initial condition. Then f_{r+1} may be expressed as $f_r + \Delta_r/(n^3 - n^2)$, where Δ_r is given by

$$\Delta_r = n(n-1)f_r + \sum_i (t_i^r)^2 - f_r^2.$$

As long as $t_i^r \leq n-1$, for $i = 1, 2, \ldots, n$, Δ_r is positive, that is

$$\begin{aligned} \Delta_r &= n(n-1)f_r + \sum_i (t_i^r)^2 - f_r \sum_i t_i^r \\ &> n(n-1)f_r - f_r \sum_i (n-1) = 0. \end{aligned}$$

Consider the process of adding a single 1 to some row in B, say row i. The new Δ_{r+1} is equal to

$$\Delta_{r+1} = \Delta_r + n(n-1) + 2t_i^r - 2f_r.$$

For $f_r < n(n-1) + 2t_i^r$, the new increment, Δ_{r+1}, is greater than the old one. However once $f_r > \frac{1}{2}n(n+1)$, $\Delta_{r+1} < \Delta_r$. From that point on, adding new 1's to B decreases the value of Δ. Hence to find the minimum Δ we need to add as many 1's to B as possible, without violating the constraint $f_r < n^2$. This we achieve, as we are dealing with a continuous model, for $t_i^r = n$, for $i = 1, 2 \ldots, j-1, j+1, \ldots, n$ and $t_j^r = n - \epsilon$, $\epsilon > 0$. Thus

$$\Delta_r \geq n(n-1)(n^2-\epsilon)+n^3-2\epsilon n+\epsilon-(n^2-\epsilon)^2 = (n^2-n+1-\epsilon)\epsilon.$$

Therefore, for $f_r < n^2$,

$$f_{r+1} \geq f_r + \epsilon/n + (\epsilon - \epsilon^2)/(n^3 - n^2) > f_r.$$

Finally, for $f_r = n^2$,

$$f_{r+1} = n^2 + n - (n^4 - n^3)/(n^3 - n^2) = n^2.$$

□

Corollary 11 *If the number of nonzeros per row is roughly the same and is $t_i^r = f_r/n$, for $i = 1, 2, \ldots, n$, the expected value of the fill-in of matrix B after r reductions is given by*

$$f_{r+1} = f_r \left(1 + \frac{1}{n}\right) - \frac{f_r^2}{n^3}. \tag{2}$$

We give an accurate approximation for f_{r+1}, since there seems to be no closed form for it.

Lemma 12 *Let $\langle f_r \rangle$ be the sequence specified by equation (2), with the initial condition f_0. Then*

$$f_r = \frac{f_0 n^2}{f_0 + \left(1 - \frac{1}{n}\right)^r (n^2 - f_0)} + O(1). \tag{3}$$

Proof The recurrence (2) may be rewritten as

$$\frac{f_{r+1} - f_r}{h} = \frac{f_r}{nh}\left(1 - \frac{f_r}{n^2}\right).$$

From this we obtain the differential equation:

$$\frac{df(r)}{dr} = \frac{f(r)}{nh}\left(1 - \frac{f(r)}{n^2}\right)$$

with initial condition $f(0) = f_0$. Solving it, and replacing h by 1 on the right hand side gives

$$f_r = f(r) = n^2 e^{r/n} f_0 \left(e^{r/n} f_0 + n^2 - f_0\right)^{-1}.$$

The factor $\exp(r/n)$ is simply $(1 + 1/n)^r$. Thus dividing the last expression by $\exp(r/n)$ and replacing $\exp(-r/n)$ with $(1 - 1/n)^r$ gives us the desired approximation. Next we apply the technique called bootstrapping [4, Section 9.4]. By substituting the estimate back into the recurrence we express f_{r+1} in terms of the estimate plus some error. In our case this results in

$$f_{r+1} = \frac{f_0 n^2}{f_0 + \left(1 - \frac{1}{n}\right)^{r+1}(n^2 - f_0)} \times \left(1 + O(n^{-2})\right).$$

As $f_r \leq n^2$ we obtain the result stated in the lemma. □

While an error term of constant size may appear to be large, we need to remember that for all realistic cases $f_r \geq n$. Thus, relative to f_r, the error is $\Omega(n)$ times smaller than the value of f_r. We note that for $r = 0$ we have $f_r = f_0$, as desired, and for $r \to \infty$ we have $f_r = n^2$, as indicated by Lemma 10.

If not for the quadratic factor, f_r would reach equilibrium in $n \ln n$ steps for $f_0 = n$, and in $2n \ln n$ steps for $f_0 = 1$. Surprisingly, even with the last term curbing the rate of growth, f_r reaches a close to terminal value in $O(n \ln n)$ steps.

Lemma 13 *Let the initial condition be $f_0 = cn$, for some constant c. The value of r for which f_r reaches $n^2 - n$ can be approximated by*

$$n \ln(n-1) + n \ln(n-c) - n \ln c.$$

Proof By solving $f_r = n^2 - n$ using equation (3) we get

$$r \approx -n \ln\left(\frac{f_0}{(n-1)(n^2 - f_0)}\right).$$

□

We have disregarded lucky cancellations so far. In their presence, when the distribution characterized in Corollary 11 is assumed, we have the following result.

Theorem 14 *If the probability of lucky cancellations is $p_0 \in [0, 1]$, each row has roughly the same number of nonzeros and each pair is equally likely to be selected for a reduction, then the expected number of nonzero entries in matrix B after r random reductions are performed is given by*

$$f_r \simeq \frac{f_0 n^2}{(1 + p_0) f_0 + e^{-r/n}(n^2 - (1 + p_0) f_0)} \tag{4}$$

Proof The lucky cancellations as described earlier in this section are a slightly disguised version of $h(i, j)$ independent Bernoulli trials, with $h(i, j)$ being the number of colliding 1's in rows i and j. Define as "success" the event that the coin comes heads up. The number of new 1's is clearly t_j^r minus the number of collisions and the number of 1's converted back to 0. The average number of successes in $h(i, j)$ independent Bernoulli trials is $p_0 h(i, j)$. Taking the average over all possible configurations of 1's in rows i and j gives us $t_j^r - \tilde{h}(i, j) - p_0 \tilde{h}(i, j)$ as the number of newly created 1's. The rest follows exactly the same pattern as in Theorem 9 and Corollary 11. □

Thus for p_0 varying from 0 to 1 we obtain a complete spectrum of fill-in functions (see Figure 1), with

$$\lim_{r\to\infty} f_r(p_0) = \frac{n^2}{1+p_0}.$$

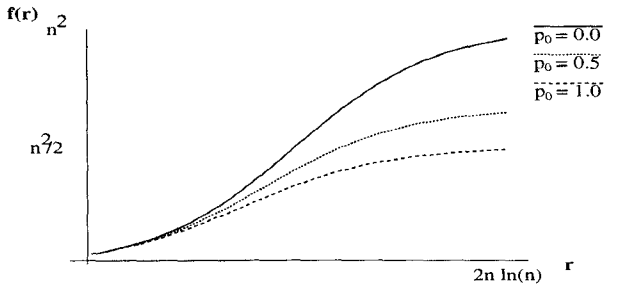

Figure 1: The asymptotic behavior of fill-in for different probabilities of lucky cancellations

In actual extended gcd computations the value of p_0 hovers around $1/2u$, where u is the average absolute value of the integers currently in the array. Thus $p_0 = n^{-k}$ is a realistic estimate of the probability of lucky cancellation. Clearly, the effect of fortunate cancellation on the rate of fill-in, for such a p_0, is negligible.

4 Experimental results and analysis

We have performed more than 12,000 experiments to determine the expected size and number of multipliers generated by this method. (Further information on practical experiments with this and other extended gcd algorithms appears in [12].)

For our analysis here we used two parameters: n, the number of integers in the sequence A, and the maximum allowed integer. We tested the algorithm for sequences of length 10, 15, 20 and 25 integers selected uniformly at random from the range $[1, \lfloor e^i \rfloor]$, where $i = 7, 8, \ldots, 18$. For each fixed pair of parameters, 250 runs were executed and the average over those runs was calculated. The results are presented in Tables 1 and 2, where $Emax = E(\max_i\{|x_i|\})$ (the average size of the multiplier with largest magnitude) and $Enonz = E(|\{x_i : x_i \neq 0\}|)$ (the average number of nonzero multipliers).

i	Top of range	$n = 10$ *Emax*	$n = 15$ *Emax*	$n = 20$ *Emax*	$n = 25$ *Emax*
7	1097	1.216	1.020	1.000	1.004
8	2980	1.544	1.064	1.004	1.000
9	8103	1.860	1.152	1.016	1.000
10	22026	2.204	1.456	1.064	1.000
11	59874	2.700	1.660	1.224	1.032
12	162754	3.256	1.988	1.400	1.164
13	442413	4.032	2.300	1.720	1.380
14	1202604	4.788	2.644	1.988	1.636
15	3269017	5.832	2.948	2.220	1.756
16	8886111	7.220	3.560	2.488	2.032
17	24154953	8.592	4.136	2.892	2.392
18	65659969	10.980	4.900	3.332	2.552

Table 1: The average size of the largest multiplier

i	Top of range	$n = 10$ *Enonz*	$n = 15$ *Enonz*	$n = 20$ *Enonz*	$n = 25$ *Enonz*
7	1097	5.592	6.848	7.868	8.208
8	2980	6.060	7.300	8.480	9.296
9	8103	6.592	7.712	9.176	10.104
10	22026	7.036	8.700	9.300	10.692
11	59874	7.536	9.240	9.932	11.108
12	162754	8.004	9.948	10.676	11.544
13	442413	8.312	10.588	12.356	12.744
14	1202604	8.408	10.940	12.260	13.896
15	3269017	8.828	11.356	13.212	14.848
16	8886111	8.920	11.916	13.992	16.224
17	24154953	9.084	12.328	14.904	17.424
18	65659969	9.220	12.668	15.320	17.780

Table 2: The average number of nonzero multipliers

The theoretical and practical results combine to indicate that the algorithm goes through a number of phases, with three of them deserving special consideration. The total number of row reductions performed by the algorithm is a function of both n and $\max_i\{a_i\}$ which can be estimated as $O(n \log(\max_i\{a_i\}))$. Theorem 9 shows the direct influence that the size of the numbers in the sequence has on the density of the matrix B. Using the estimate obtained in Lemma 12 we may split the "life" of matrix B into three major components.

Firstly, when $\max_i\{a_i\} < e^n$ the algorithm spends most of its time filling matrix B with small constants. Occasionally, the algorithm may slip, producing an entry as large as $\ln(\max_i\{a_i\})$. However, in practice usually more than one solution to the extended gcd problem is created. By sorting the rows of B we ensure that a bad solution, even if present, has a negligible chance of becoming the returned answer. Observe that for $n = 25$, as long as the a_i's were selected from the range [1, 22026], the algorithm, with one exception in a thousand experiments, found a solution with only zero and unit multipliers. Such an answer solves WEAK UNEVEN PARTITION, which is an NP-complete problem [11].

The second phase in the life of the sorting gcd method starts when $\max_i\{a_i\} = O(e^n)$. (Then $r = O(n \log n)$ and, by Lemma 13, matrix B becomes dense.) Occasional slips now have much greater chances of surfacing, as matrix B is dense, and hence the likelihood of large elements in B starting to accumulate, becomes significant. This can be observed in the results in Table 1, where the initially slow increase in size of the multipliers accelerates for larger and larger a_i's. The increase is hampered by the variation of signs of elements of B, which may cause lucky cancellations or decreases in size of the largest element which are not catered for in our analysis.

The third phase begins with numbers as large as $O(e^{n \ln n})$, which is $O(n^n)$. Then matrix B is dense for so long that the algorithm is almost certain to produce entries as large as $O(a_i)$. To understand this phase better we need more analysis than presented in this paper.

5 Conclusions

The multipliers that we have obtained with the sorting gcd method are much better than those obtained by previous methods. For random sequences of numbers of bounded size and length the multipliers are effectively constant, a significant improvement on previous algorithms which gave

multipliers bounded, at best, by the input numbers. This is in accord with both the theoretical and practical analyses presented here. The multipliers obtained by such extended gcd calculations play a critical role in algorithms for computing Smith and Hermite normal forms of integer matrices [5, 6, 8]. The next step in this research is the application of the sorting gcd algorithm to such calculations.

References

[1] W.A. Blankinship. A new version of the Euclidean algorithm. *Amer. Math. Mon.*, 70:742–745, 1963.

[2] G.H. Bradley. Algorithm and bound for the greatest common divisor of n integers. *Communications of the ACM*, 13:433–436, 1970.

[3] A.J. Brentjes, *Multi-dimensional continued fraction algorithms*, Mathematisch Centrum, Amsterdam 1981.

[4] R.L. Graham, D.E. Knuth, and O. Patashnik. *Concrete Mathematics: A Foundation of Computer Science.* Addison-Wesley, Reading, Mass., 1989.

[5] G. Havas and B.S. Majewski. Integer matrix diagonalization. Technical Report TR0277, The University of Queensland, Brisbane, 1993.

[6] G. Havas and B.S. Majewski. Hermite normal form computation for integer matrices. *Congressus Numerantium*, 105:184–193, 1994.

[7] G. Havas, B.S. Majewski, and K.R. Matthews. Extended gcd algorithms. Technical Report TR0302, The University of Queensland, Brisbane, 1994.

[8] C.S. Iliopoulos. Worst case complexity bounds on algorithms for computing the canonical structure of finite abelian groups and the Hermite and Smith normal forms of an integer matrix. *SIAM J. Computing*, 18:658–669, 1989.

[9] D.E. Knuth. *The Art of Computer Programming, Vol. 2: Seminumerical Algorithms.* Addison-Wesley, Reading, Mass., 2nd edition, 1973.

[10] R.J. Levit. A minimum solution to a diophantine equation. *American Math. Mon.*, 63:647–651, 1956.

[11] 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.

[12] B.S. Majewski and G. Havas. Extended gcd calculation. Technical Report TR0325, The University of Queensland, Brisbane, 1995.

[13] D.J. Rose and R.E. Tarjan. Algorithmic aspects of vertex elimination on directed graphs. *SIAM J. Appl. Math.*, 34:176–197, 1978.