

FACTORING WITH TWO LARGE PRIMES

A. K. LENSTRA AND M. S. MANASSE

ABSTRACT. We describe a modification to the well-known large prime variant of the multiple polynomial quadratic sieve factoring algorithm. In practice this leads to a speed-up factor of 2 to 2.5. We discuss several implementation-related aspects, and we include some examples. Our new variation is also of practical importance for the number field sieve factoring algorithm.

1. FACTORING WITH TWO LARGE PRIMES

Let $n > 1$ be an odd integer which is not a prime power. For each random integer x satisfying

$$(1.1) \quad x^2 \equiv 1 \pmod{n}$$

there is a probability of at least $1/2$ that $\gcd(n, x - 1)$ is a nontrivial factor of n . To factor n , it therefore suffices to construct several such x 's in a more or less random manner.

In many factoring algorithms, solutions to (1.1) are sought by collecting integers v such that

$$(1.2) \quad v^2 \equiv \prod_{p \in P} p^{e_p(v)} \pmod{n},$$

where the *factor base* P is some finite set of integers that are coprime to n , and $e_p(v) \in \mathbb{Z}$ for $p \in P$. A pair $(v, e(v))$ satisfying (1.2), with $e(v) = (e_p(v))_{p \in P} \in \mathbb{Z}^{\#P}$, is called a *relation*, and will be denoted by v for short. If V is a set of relations with $\#V > \#P$, then there exist at least $2^{\#V - \#P}$ distinct subsets W of V with $\sum_{v \in W} e(v) = (2w_p)_{p \in P}$ and $w_p \in \mathbb{Z}$; these subsets can be found using Gaussian elimination modulo 2. Each such W leads to an $x \equiv (\prod_{v \in W} v) \cdot (\prod_{p \in P} p^{-w_p}) \pmod{n}$ satisfying (1.1).

The factoring algorithm determines how P is chosen and how relations are collected. In this paper we will restrict our attention to the multiple polynomial variation of the quadratic sieve algorithm, but our method also applies to other factoring algorithms, like the continued fraction method and the number field sieve which for our purposes fit the same capsule description of factoring algorithms given in the previous paragraph [6, 8].

Received by the editor August 3, 1992 and, in revised form, March 23, 1993.

1991 *Mathematics Subject Classification*. Primary 11Y05.

Key words and phrases. Factoring algorithm.

A preliminary version of this paper appeared in [11].

In the quadratic sieve algorithm, P consists of -1 and the primes $\leq B$ with Legendre symbol $\left(\frac{n}{p}\right) = 1$, for some bound B that behaves asymptotically as $L_n[1/2, 1/2]$ for $n \rightarrow \infty$, where

$$(1.3) \quad L_z[\nu, \gamma] = \exp((\gamma + o(1))(\log z)^\nu (\log \log z)^{1-\nu}),$$

for real numbers ν and γ , and $z \rightarrow \infty$. Relations are found in a particularly efficient way, using a *sieve* [6, 15]. During the sieving step *reports* are produced, which correspond to v 's for which (1.2) is probably satisfied; for each reported v , the least absolute residue modulo n of its square is inspected by means of trial division with the elements of P . On loose heuristic grounds it is expected that it takes time $L_n[1/2, 1]$ to collect more than $\#P = L_n[1/2, 1/2]$ relations.

The large prime variation. With some small modifications and a minor loss of efficiency, the sieve can also report *near misses*: integers u for which the least absolute residue modulo n of their square can be factored using the elements of P , except for one prime factor $q(u)$ with $B < q(u) < B^2$:

$$(1.4) \quad u^2 \equiv q(u) \cdot \prod_{p \in P} p^{e_p(u)} \pmod{n}.$$

Triples $(u, q(u), e(u))$ satisfying (1.4), with $e(u) = (e_p(u))_{p \in P} \in \mathbb{Z}^{\#P}$, will be called *partial relations*, as opposed to pairs $(v, e(v))$ satisfying (1.2) which will be called *full relations* from here on. A partial relation $(u, q(u), e(u))$ will be denoted by u , and the prime $q(u)$ in u is called the *large prime*. The number of partial relations reported by the sieve is much larger than the number of full relations.

A set of partial relations might contain two relations u and \bar{u} for which $q(u) = q(\bar{u})$. With $v \equiv (u/\bar{u}) \pmod{n}$ and $e(v) = (e_p(u) - e_p(\bar{u}))_{p \in P}$, it follows that $(v, e(v))$ is a full relation, unless v is not well defined—in which case $\gcd(n, \bar{u})$ is a proper divisor of n . This alternative way of producing full relations can be combined with the direct way of finding them; it is known as the *large prime variation*. Note that if $k \geq 3$ partial relations have a large prime in common, we get at most $k - 1$ useful combinations; the other $\binom{k}{2} - k + 1$ combinations are linearly dependent on these. Finding the matches can for instance be done by sorting the partial relations with respect to their large prime, or using a hash function.

The expected asymptotic time to collect a total of more than $\#P$ full relations with the large prime variation of the quadratic sieve is still $L_n[1/2, 1]$ (cf. [15] and §3). In practice, however, it leads to savings of 50 to 60 percent, where one usually imposes a much lower upper bound on $q(u)$ than B^2 to keep the amount of data manageable. Notice that possibly composite $q(u) \geq B^2$ can be collected and combined as well (cf. [17]). This would probably lead to a minor gain in efficiency, and be much less effective than the double large prime variation.

The double large prime variation. An obvious extension of the large prime variation is to allow *two* instead of only one large prime in a partial relation. This leads to relations of the form

$$(1.5) \quad u^2 \equiv q_1(u) \cdot q_2(u) \cdot \prod_{p \in P} p^{e_p(u)} \pmod{n},$$

where the $q_i(u)$ are primes with $B < q_1(u) < q_2(u) < B^2$; the $q_i(u)$ will be called the large primes. We may exclude the possibility that $q_1(u) = q_2(u)$, because such a relation would immediately give rise to a full relation, unless $\gcd(n, q_1(u)) \neq 1$. Because quadruples $(u, q_1(u), q_2(u), e(u))$ can be regarded as near misses for partial relations, they will be called *partial-partial relations*, or *pp*'s for short. The sieve can again be adapted to report the pp's as well, but this adaptation requires substantially more work than the large prime variation (cf. §3).

Like the partial relations, the pp's can be combined into full relations. Let Q be the set of large primes occurring in some set of partial and partial-partial relations, where partial relations $(u, q(u), e(u))$ are regarded as pp's of the form $(u, 1, q(u), e(u))$, so that Q also contains the "large prime" 1. Let G be a graph which has Q as its vertex set, and with edges defined by the partial and partial-partial relations: for each relation with large primes q_1 and q_2 (where $q_1 < q_2$, with q_1 possibly equal to 1), there is an edge between the vertices associated with q_1 and q_2 . A cycle in G corresponds to a set of relations where each large prime occurs an even number of times, and therefore a full relation if n is coprime to the u 's and the large primes involved in the cycle. This alternative way of producing full relations will be referred to as the *double large prime variation*. Notice that we are only interested in a fundamental set of cycles, i.e., a set that forms a basis for the cycles in G (cf. [19]).

It is not hard to see that, given a collection of partial relations and pp's, a set of fundamental cycles can be found using some type of Gaussian elimination algorithm [8]. In practice it might, however, be more convenient to use the method sketched in §2, because it also provides a fast way to count the number of fundamental cycles before they are actually built. This is useful, because it seems to be hard to give a reasonable estimate for that number in another way (cf. §3).

Compared to the single large prime variation, the double large prime variation of the multiple polynomial quadratic sieve achieves a speed-up by approximately a factor of 2.5, for sufficiently large n , but see §3. As mentioned above, the double large prime method can also be applied to the number field sieve factoring algorithm. There it seems to lead to a much more dramatic improvement of the performance of the algorithm, cf. [8].

We originally developed the algorithms presented in §2 because we needed them to get a practical version of the number field sieve factoring algorithm. Once this software was in place, we realized that it could trivially be employed in the quadratic sieve algorithm, if we would allow an additional large prime in the ordinary large prime variation. Independently, P. L. Montgomery in 1985 [17] had proposed the use of two large primes. J. M. Pollard suggested the same idea to us after we had already begun its implementation. We cannot even begin to count the number of people who have subsequently suggested using three, or even more, large primes; we believe that such a variant would make the algorithms of the next section considerably more complicated, if not useless, and that it would most likely slow the sieving step enough to be counterproductive—the number of false reports would be immense.

The next section explains how we count the number of fundamental cycles, and how we build them. The final section discusses various other practical aspects and presents some of the factorizations we have obtained.

2. COUNTING AND FINDING CYCLES

For some set of partial and partial-partial relations, let G be the graph defined in the previous section. In this section we describe an algorithm to count the number of fundamental cycles of G and an algorithm to find a set of fundamental cycles.

Let v , e , and c be the number of vertices, edges, and connected components of G , respectively. It is well known that the number of fundamental cycles of G equals $e + c - v$, so to solve the counting problem it suffices to find v , e , and c . Unfortunately, our representation of G is not such that any of these quantities is known right away. All that is given, initially, is some collection C of putative partial and partial-partial relations. Depending on how C was obtained, it might be known that all elements of C are indeed relations, and that they are distinct, allowing e to be computed as $\#C$. It might instead be the case, as in [10], that some of the elements of C are corrupted, or that C contains duplicates. An additional practical problem is caused by the size of C : in our work C consists of several hundred megabytes of data, so that C cannot be expected to fit in memory.

To deal with this problem, we first describe a preprocessing step that determines a set of correct relations R , given some collection C of putative relations. The collection C and the set R should be thought of as input and output files, respectively, and should be sequentially accessible for reading (C) or writing (R). We assume that C contains one relation per line. Furthermore, we assume that C does not contain putative full relations, but only partial and partial-partial relations; the full relations can be treated similarly and separately. The preprocessing step below is by no means the only way in which R can be derived from C . It might, however, be helpful and contain useful suggestions for one's own implementation.

Preprocessing: counting relations. Let $T = (t_i)_{i=0}^{S-1}$ be a hash table of size S , where the t_i are randomly accessible and store four bytes of information. The size S should be chosen in such a way that $4 \cdot S$ bytes of memory, i.e., the memory needed to store T , can be allocated, and such that the hash table can accommodate $\#R$ four-byte hash values. We used $S = 2^{23}$, which allows several million relations in R —sufficient for double large prime multiple polynomial quadratic sieve factorizations of numbers up to at least 125 digits. For efficient operation, S should be at least $2 \cdot \#R$.

First of all, set e to zero, and initialize all t_i to -1 . For each line of C do the following. If the line cannot be recognized as a relation of the form $(u, q_1(u), q_2(u), e(u))$, or if this quadruple does not satisfy (1.5), then proceed to the next line. Let r be some concise, uniform representation of $(u, q_1(u), q_2(u), e(u))$ satisfying (1.5); thus, r could for instance consist of the hexadecimal representations of the large primes and u , in some fixed ordering, followed by the pairs $(e_p(u), p)$ for the nonzero $e_p(u)$'s and increasing p . Notice that r will in general not fit in four bytes, so that r itself cannot be stored in one of the t_i . Therefore, we first compute an eight-byte fingerprint $f(r)$ of r . This can for instance be done by regarding r as a polynomial over $\mathbb{Z}/2\mathbb{Z}$ and by defining $f(r)$ as r modulo some fixed irreducible polynomial of degree 64 over $\mathbb{Z}/2\mathbb{Z}$, a computation that can be carried out efficiently with some auxiliary tables.

Let h be a hash function that maps a bit string to an integer in $\{0, 1, \dots, S-1\}$. Compute the minimal $j \geq h(f(r))$ such that either $t_j = -1$ or t_j equals the first four bytes of $f(r)$ (or use another more intelligent collision resolution algorithm). Here we take the indices of the hash table modulo S , i.e., $t_j = t_{j \bmod S}$. If $t_j = -1$, then replace t_j by the first four bytes of $f(r)$, replace e by $e+1$ and R by $R \cup r$, and proceed to the next line of C . If, on the other hand, t_j is equal to the first four bytes of $f(r)$, then reject r as a duplicate, and proceed to the next line of C . Notice that r gets unjustly rejected if R already contains some other relation $\bar{r} \neq r$ for which the first four bytes of $f(\bar{r})$ and $f(r)$ are the same, and for which the corresponding hash values, after collision resolution, turn out to be the same. The probability that this happens is sufficiently low to be acceptable for our purposes. This completes the description of the preprocessing step.

After all lines of C have been processed, R consists of correct partial and partial-partial relations, without duplicates, and e is the number of edges of the graph G associated with R . If the hash data t_i are kept, the preprocessing step can also be used to incorporate the data from some newly received collection of putative relations into an already existing set R .

Counting fundamental cycles. To count the number of fundamental cycles, it remains to determine v and c , the number of vertices and components of G , respectively. For this we employed the well-known Union-Find algorithm [12], which, in the present context, can be described as follows. Let $T' = (d_i, a_i)_{i=0}^{S'-1}$ be a hash table of size S' , where each entry of the table contains eight bytes: four bytes for the data field d_i and four bytes for the ancestor field a_i . The size S' should be chosen in such a way that $8 \cdot S'$ bytes can be allocated, and such that T' is large enough to accommodate the large primes in R . The hash table T' is going to contain a representation of G that allows us to keep track of the number of components c .

Initially, set v and c to zero, and all d_i to -1 . For each relation in R do the following. Let q_1 and q_2 be the large primes in the relation, where q_1 might equal 1. First, insert the vertices q_1 and q_2 into the graph: to insert q , compute the minimal $j \geq h'(q) \in \{0, 1, \dots, S'-1\}$ for some hash function h' , such that $d_j = -1$ or $d_j = q$ (with indices modulo S'), and if $d_j = -1$ replace d_j by q , a_j by j , v by $v+1$, and c by $c+1$. As a result, we get $j_1, j_2 \in \{0, 1, \dots, S'-1\}$ such that $d_{j_i} = q_i$ for $i = 1, 2$. Next, find the roots of the components to which q_1 and q_2 belong: to find the root for q with $d_j = q$, set r equal to j , replace r by a_r as long as $a_r \neq r$, and at the end replace all a_i visited underway by the resulting r . As a result, we have that r_i is the root of the component to which q_i belongs, for $i = 1, 2$. Finally, insert the edge between q_1 and q_2 in the graph: if $r_1 \neq r_2$, replace c by $c-1$; if $d_{r_1} < d_{r_2}$, replace a_{r_2} by r_1 , and if $d_{r_2} < d_{r_1}$, replace a_{r_1} by r_2 (smaller primes occur more often than larger ones, so this usually joins the smaller component to the larger). If $d_{r_1} = d_{r_2}$, then q_1 and q_2 are in the same component, and a cycle has been found. Notice that throughout this algorithm $\bar{e} + c - v$ equals the number of fundamental cycles found after \bar{e} relations from R have been processed. This completes the description of the counting algorithm.

After all relations in R have been processed, v is the number of vertices, and c the number of components of G . The number of fundamental cycles can then

be computed as $e + c - v$, finishing the description of the counting algorithm. The number of operations needed by the algorithm is only slightly more than linear in e ; see [12] for details. This makes it relatively easy to monitor the progress of the relation-collection step, in particular if the hash data (d_i, a_i) are kept, so that the count can be updated for newly found relations.

Building a set of fundamental cycles. As soon as the number of fundamental cycles plus the number of ordinary, not combined, full relations is large enough, a set of fundamental cycles has to be constructed in terms of the partial and partial-partial relations involved in the cycles. This can be achieved using a breadth-first traversal of the same graph, where the a_i -field is used to point to the immediate predecessor instead of to a common root. Furthermore, each entry of the hash table should contain an additional four-byte field that contains information to retrieve the corresponding relation from R , and a field indicating the depth of the vertex; a single bit indicating the parity of the depth will suffice.

The counting algorithm already computed the connected components of the graph; the roots of these components will serve as the roots of our traversal. We place these roots in the hash table, at depth zero. We now repeatedly scan the list of unused edges. On each pass, some edges will be added to the graph, some will be set aside for future consideration, and some will generate cycles. As we consider an edge, we see if either of its vertices is already in the graph at the previous depth (by checking to see first if it is in the graph, and second if the parity differs from the current parity). If neither vertex is in the graph at the previous depth, we defer this edge. If one vertex of the new edge is in the graph at the previous depth, but the other vertex is neither present at the current nor the previous depth, add that edge; the a_i -field of the newly added vertex should point to the vertex that was already present, and the parity bit should be set to the current parity.

In the final case where at least one vertex of the new edge is in the graph at the previous depth, and the other vertex is present on the current or the previous depth, we have found a cycle. This cycle can be enumerated by following parent pointers from each vertex until they coincide. In our implementation, we do this by reversing the a_i -pointers from one of the vertices up to the root, and then following the path from the other vertex back to the first. We then reverse the pointers again. In this way, we avoid any storage overhead, and we traverse the vertices in an order where it is easy to see where the squared large primes come from. Furthermore, the additional field is used to retrieve the relations involved in the cycle.

Notice that the way we use the bit indicating the depth makes this traversal of the graph into a breadth-first search. Without this mechanism, we would generate cycles somewhat longer than we get now. We found it helpful first to trim the set R down to its subset of relations that occur in a cycle, before we actually build the cycles. This can again be achieved using similar methods.

Various other algorithms are known that find a set of fundamental cycles for a given graph. In [14], for instance, an algorithm is given that runs in time $O(v(e - v + 1))$, where v is the number of vertices and e the number of edges of G . The algorithm from [3] runs in time $O(v e^3)$ and has the advantage that it finds a fundamental set of cycles with the least possible number

of edges. Although this last property is useful for our purposes, because the corresponding matrix of exponent vectors is probably of relatively low weight, both these algorithms are less practical than the one presented above.

Essentially the same algorithms can be applied if q_1 and q_2 belong to distinct sets, as is the case in the number field sieve factoring algorithm. In that case, the hash table T could, for instance, be replaced by two smaller hash tables, and G would consist of the union of a bipartite graph and a vertex associated with 1.

3. EXAMPLES

In this section we discuss the following issues: predicting the yield of the single and double large prime variation, adapting the sieve to report partial-partial relations, the crossover point between the single and the double large prime variation, and choosing $\#P$, the factor base size. We conclude with a few examples.

Predicting the yield. The following lemma is a generalization of [15, Lemma 6.1].

Lemma (3.1). *Let Y be some finite set of cardinality y , let P_1, P_2, \dots, P_x be x disjoint finite sets, and let S denote the set of functions $f: Y \rightarrow \bigcup_{i=1}^x P_i$. If $f \in S$, let $M(f) = y - \sum_{i=1}^x \chi_i(f)$, where $\chi_i(f) = 0$ if $f(Y) \cap P_i = \emptyset$ and 1 otherwise. Then with the uniform distribution on S , the expected value $E(M)$ of M is*

$$y - x + \sum_{i=1}^x (1 - p_i)^y$$

and the variance is

$$\sum_{i=1}^x (1 - p_i)^y + 2 \sum_{i=1}^x \sum_{j=i+1}^x (1 - p_i - p_j)^y - \left(\sum_{i=1}^x (1 - p_i)^y \right)^2,$$

where $p_i = \#P_i / \sum_{j=1}^x \#P_j$.

Proof. The proof follows the lines of the proof of [15, Lemma 6.1]. We compute the expectation and variance of X , where $X(f) = x - \sum_{i=1}^x \chi_i(f) = x - y + M(f)$ for $f \in S$. Note that $E(M) = y - x + E(X)$ and that X and M have the same variance.

Let $\bar{\chi}_i(f) = 1 - \chi_i(f)$; then $E(\bar{\chi}_i) = (1 - p_i)^y$. Since $X = \sum_{i=1}^x \bar{\chi}_i$, we find that

$$E(X) = \sum_{i=1}^x E(\bar{\chi}_i) = \sum_{i=1}^x (1 - p_i)^y.$$

The variance $E(X^2) - E(X)^2$ follows from $E(\bar{\chi}_i^2) = E(\bar{\chi}_i)$, $E(\bar{\chi}_i \bar{\chi}_j) = (1 - p_i - p_j)^y$ for $i \neq j$, and

$$E(X^2) = \sum_{i=1}^x \sum_{j=1}^x E(\bar{\chi}_i \bar{\chi}_j).$$

This finishes the proof of (3.1). \square

Lemma (3.1) can be used to predict the number of matches in the ordinary large prime variation. Let R be a set of partial relations, and let $Q = \{q: q \text{ prime, } B < q < U, (\frac{n}{q}) = 1\}$, where $U \leq B^2$ is the upper bound that was imposed on the large primes. We assume that for each $u \in R$ we have probability

$$p_q = \frac{1/q}{\sum_{q \in Q} 1/q}$$

that $q = q(u)$, for $q \in Q$; note that if P_q is a set of cardinality $\prod_{t \in Q, t \neq q} t$, then $p_q = \#P_q / \sum_{q \in Q} \#P_q$. It follows from the lemma that the expected number of matches equals

$$(3.2) \quad \#R - \#Q + \sum_{q \in Q} (1 - p_q)^{\#R}.$$

This expression can be approximated in various ways. For instance, since $\#R$ is relatively small compared to $\#Q$, it follows from a rough approximation of (3.2) that the number of matches can initially be expected to behave as $c \cdot (\#R)^2$, where $c = (\sum_{q \in Q} p_q^2) / 2$.

Our practical observations have shown that the number of matches indeed grows quadratically with $\#R$, and that the yield of the large prime variation varies considerably, which agrees with the fact that the variance is rather large. We found, however, that the prediction given by (3.2) is consistently too high. This can be explained by the following argument, which was kindly suggested to us by a referee. In our model we assume that a particular large prime q occurs with probability inversely proportional to q , since $1/q$ of all numbers are multiples of q . This assumption neglects the fact that, after the large prime q is removed from the number, the resulting cofactor is smooth. Since smaller numbers are more likely to be smooth than larger numbers, this may make the occurrence of larger large primes more likely. This suggests an alternative definition for p_q , namely,

$$p_q = \frac{1/q^\alpha}{\sum_{q \in Q} 1/q^\alpha},$$

for some positive $\alpha < 1$, the value of which should be tried empirically. We found that for numbers in our range of interest $\alpha \in [2/3, 3/4]$ resulted in a reasonable fit.

This apparent difficulty to predict the yield in a reliable way is not a serious issue for the large prime variation, because the matchings can easily be counted and found, as remarked in §1, which makes it straightforward to monitor the progress of the relation-collection step. See [13] for an alternative analysis of the expected number of matches in the large prime variation.

The progress of a multiple polynomial quadratic sieve factorization, using the large prime variation, typically behaves as illustrated in Figure 1. Both the number of ordinary full relations and the number of partial relations behave approximately as linear functions of the time spent, which implies that the number of ordinary full relations behaves approximately as a linear function of the number of partial relations, as shown in the figure. The number of matching pairs among the partial relations and the total number of full relations, however, follow the convex curves. These data were derived from the factorization of a

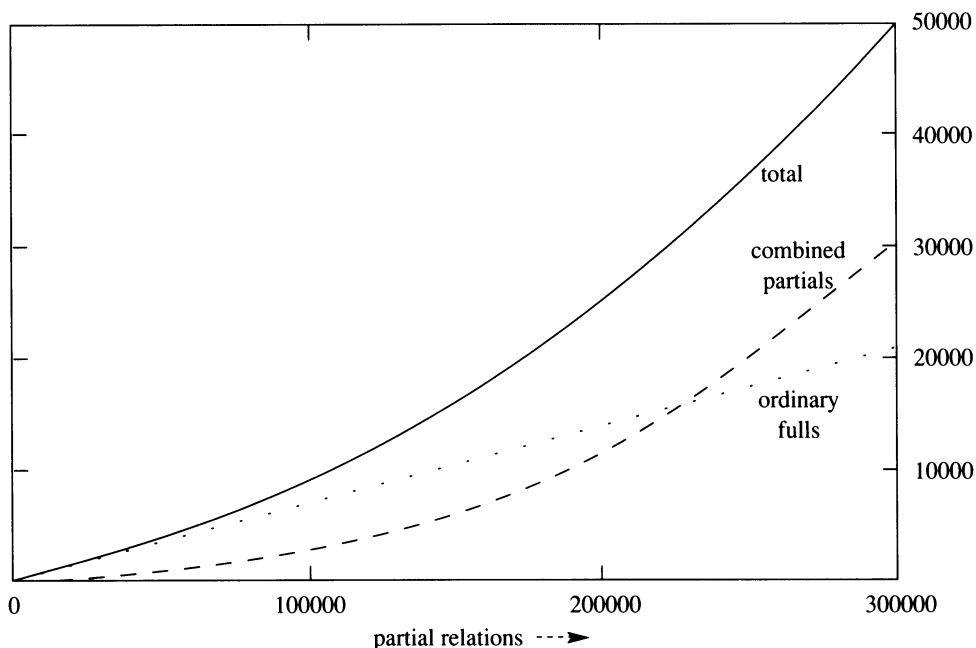


FIGURE 1

100-digit factor of $11^{104} + 1$ for which $\#P$ was set to approximately 50,000 and 10^8 was used as upper bound for the large primes in the partial relations. With the set-up as described in [10] it took us slightly more than 24 days to collect 300,000 partial relations, at which point we had 20,500 ordinary full relations and 29,500 matching pairs of partial relations (cf. [10]).

We have not attempted to carry out a theoretical analysis of the estimated yield of the double large prime variation. From our experience we know, however, that such an analysis cannot be expected to have much practical value: various times we have factored numbers of approximately the same size, with identical parameter settings, for which the same number of fundamental cycles required entirely different numbers of partial and partial-partial relations. Furthermore, the standard deviation of the yield is already quite large if only a single large prime is used; apparently the situation is even worse for the double large prime variation. For all numbers we factored, using the double large prime variation, however, we found a behavior similar to the second figure, though the combination curves might lie considerably higher or lower. Thus, once the counting algorithm has been used a few times to find the initial part of the combination curves for a particular number, its completion time can be predicted fairly accurately. As a rule of thumb: as soon as the number of fundamental cycles equals the number of ordinary full relations, the relation-collection step is half completed.

The data in Figure 2 were derived from the factorization of a 107-digit factor of $2^{401} - 2^{201} + 1$ for which $\#P$ was set to 65,500 and 10^8 was again used as upper bound for the large primes in the partial and partial-partial relations. It took us approximately 50 days to collect 165,000 partial and 760,000 partial-partial relations, at which point we had approximately 11,000 ordinary full

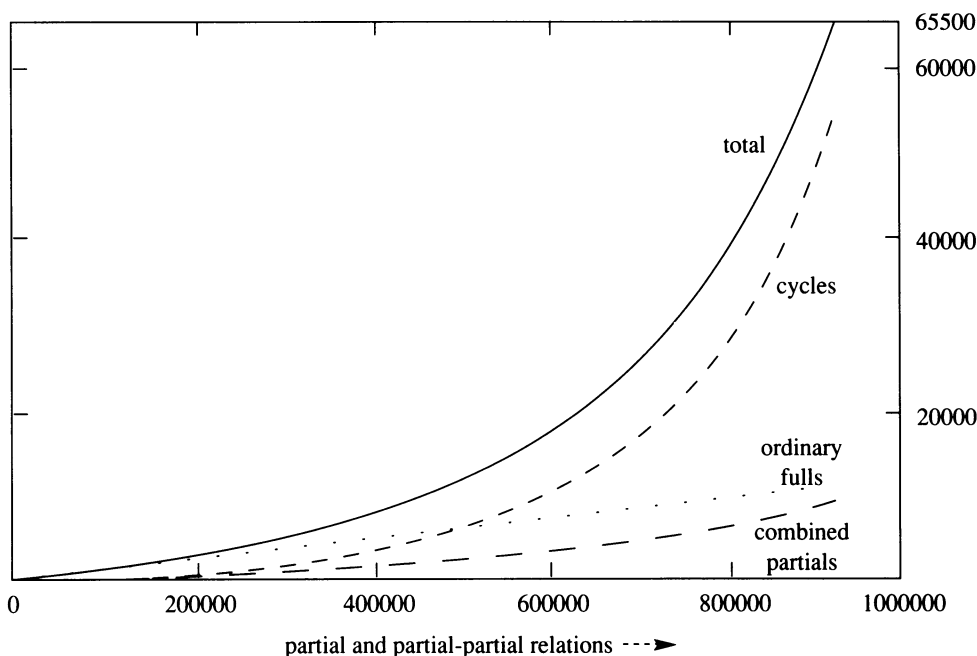


FIGURE 2

relations and 54,500 fundamental cycles. As shown in the figure, the partial relations among themselves gave rise to only 9,000 useful combinations. It might be interesting to note that the partial-partial relations among themselves gave rise to only two cycles.

Crossover between single and double large prime variation. As far as we are aware, it is always better to use the single large prime variation than it is not to use large primes at all. This follows from the way full relations are found during the sieving step: if for a candidate v the least absolute residue of v^2 modulo n fails to factor completely after trial division with the primes $\leq B$, but the unfactored part m is $< B^2$, then a partial relation has been found at no extra cost. Usually a small extra cost will be involved in practice, because the report bound during the sieving step will be set to a slightly lower value to catch more partial relations.

The situation is more complicated in the double large prime variation of the quadratic sieve. If the unfactored part m is $\geq B^2$ and $< B^3$, this either implies a partial-partial relation or a false report: a pp if m happens to be composite, and a false report otherwise. In the case that m is composite and factors as $q_1 \cdot q_2$ with $B < q_1 \leq q_2 < B^2$, it still remains to determine the q_i . Depending on the size of B , this can be done using Pollard's rho method (as we did), Shanks's "squfof", or the elliptic curve method [6]. Because it may be expected that there are far more pp's than partial relations, these factorizations and the compositeness tests to detect the false reports cause a noticeably slower performance of the relation-collection step. As a consequence, the double large prime variation of the quadratic sieve tends to get less efficient than the single large prime variation for relatively small n . Where the crossover point lies depends on many fine points, of which we mention a few.

For instance, it is advisable to impose a much lower upper bound than B^2 for q_2 and to use a relatively high report bound. We found that it also helps to use a much lower upper bound for q_1 than for q_2 : it makes relation-collection faster, and it cuts down considerably on the number of pp's without affecting the yield in the combination step by much because pp's with *both* large primes close to the same large upper bound have a relatively small probability of being useful. Good values for the various parameters in this process are best determined empirically. We found that for our implementation the double large prime variation was already more efficient than the single large prime variation for n having approximately 75 digits. For n having more than 90 digits we achieved a speed-up over the single large prime variation by a factor of approximately 2.5.

As noted by a referee, the speed-up factor is also a function of the size of the factor base that is used. The large prime variations serve to extend the factor base to primes which occur rarely in factorizations. It is more efficient to discover them by removing all the small primes first and examining the quotient, than to look for them in a more systematic way using a sieve. Owing to the restricted form of large-prime relations, we can efficiently determine when enough relations are available, and how to eliminate the large primes. Consequently, it can be argued that a relatively small factor base results in a larger speed-up factor. Notice also that, according to (3.2), for a larger B more partial relations are needed to get the same expected number of matches. Because we tend to favor small factor base sizes (see below), the speed-up factor of approximately 2.5 that we found may represent the high end of the spectrum.

For the large prime variation of the number field sieve, the problems with compositeness testing, etc. do not occur. There the sieve consists of two separate sieves, and the single large prime variation is applied to each of the two numbers to be trial-divided. Consequently, we never experienced any loss of efficiency while using the large prime variation of the number field sieve. If double large primes are allowed in either of the two numbers that are trial-divided, then the problems mentioned above return. For small numbers the first author found this last variation of the number field sieve quite slow and not competitive with the "normal" version; it might be the case that this variant becomes more practical for larger n .

Choosing the factor base size. For all variations of the quadratic sieve algorithm the bound B behaves asymptotically as $L_n[1/2, 1/2]$, with L_n as in (1.3). Notice that, because of the $o(1)$ in (1.3), the same asymptotic behavior $L_n[1/2, 1/2]$ follows for the factor base size $\#P \approx B/(2 \ln B)$; it should be clear that the optimal value for $\#P$ usually does not follow by evaluating $L_n[1/2, 1/2]$ with $o(1)$ simply set to zero. A reasonable indication of the growth rate of $\#P$, within a limited range of n 's, can however be obtained by considering the quotient $L_{\tilde{n}}[1/2, 1/2]/L_n[1/2, 1/2]$, where both $o(1)$'s are set to zero. Thus, factor base sizes can fairly reliably be chosen for \tilde{n} , once optimal choices have been made for n 's of various different sizes. These choices will depend on the algorithm, the actual implementation, and various parameter settings. Entirely different considerations may also play a role: we have often used suboptimal factor base sizes in order to keep program sizes acceptable to our many contributors, to cut down on the sizes of the resulting data files, and

in an attempt to avoid overloading the mail-handlers at DEC SRC. For actual examples, see below.

Factorizations. All factorizations reported here were obtained using the set-up as described in [10]: volunteers run our sieving program on their machines and communicate with us using electronic mail. Computing times are given in *mips-years*, where one mips-year is about $3.15 \cdot 10^{13}$ instructions. Most numbers we factored came from the list of unfactored numbers from [2] and are reported in [20].

The largest number we factored with the single large prime variation of the multiple polynomial quadratic sieve was a 106-digit factor of $2^{353} + 1$. This factorization took approximately 140 mips-years, and we used $\#P = 65,500$. As reported above, we used the same $\#P$ for the factorization of a 107-digit factor of $2^{401} - 2^{201} + 1$, which was our first factorization using the double large prime variation of the multiple polynomial quadratic sieve. This factorization took less than 60 mips-years.

Since that time, we factored several numbers in the 110+ digit range, the largest one a 116-digit factor of $10^{142} + 1$ for which we used $\#P = 120,000$. In principle, a larger value like 160,000 would have been better, but there were several reasons why we preferred the suboptimal smaller choice. In addition to the reasons mentioned above, we also had to be careful about the size of the matrix in the Gaussian elimination step; more about this later. After approximately 400 mips-years, we gathered 25,361 ordinary full relations and a total of 284,750 partial and 953,242 partial-partial relations (with 10^8 as upper bound for the large primes), which gave rise to 117,420 fundamental cycles; the graph G (cf. §2) had $e = 1,237,992$, $v = 1,286,057$, and $c = 165,485$. The number of cycles of each length is given in Table 1. There were no cycles that did not involve partial relations, and 352,872 of the 1,237,992 relations were used to build the set of fundamental cycles.

Finding subsets W that lead to solutions of (1.1) is equivalent to finding dependencies modulo 2 among the rows of the matrix consisting of the $25,361 + 117,420 = 142,781$ exponent vectors that correspond to the relations. To find these dependencies, we first applied *structured Gaussian elimination* [4,16], which reduced the sparse $142,781 \times 120,000$ bit-matrix to a dense bit-matrix consisting of 44,971 rows and 44,721 columns. So, we kept only 250

TABLE 1

cycle length	number of cycles	cycle length	number of cycles
2	22556	11	544
3	25394	12	233
4	22536	13	129
5	18402	14	48
6	12417	15	17
7	7747	16	5
8	4175	17	5
9	2150	18	1
10	1059	19	1

of the $142,781 - 120,000 > 20,000$ excess full relations. We then used a massively parallel Gaussian elimination program to find the dependencies in the dense matrix, which could easily be transformed into dependencies among the rows of the original large sparse matrix, and to solutions of (1.1). The resulting factorization, which is at the time of writing still the record general-purpose factorization, is:

$$10^{142} + 1 = 101 \cdot 569 \cdot 7669 \cdot 380623 \cdot 849488 \cdot 714809 \\ \cdot 7716926 \cdot 518833 \cdot 508778 \cdot 689508 \cdot 504941 \\ \cdot 93611 \cdot 382287 \cdot 513950 \cdot 329431 \cdot 625811 \cdot 490669 \\ \cdot 82 \cdot 519882 \cdot 659061 \cdot 966708 \cdot 762483 \cdot 486719 \\ 446639 \cdot 288430 \cdot 446081.$$

The number we factored was the product of the last three factors.

The elimination of the dense matrix was carried out on a 16K MasPar massively parallel computer and took less than half an hour. Because the entire dense matrix had to fit in core for our program, and because the MasPar had, at that time, only 1/4 GigaByte of memory, we could not have processed a much larger dense matrix. Structured Gaussian elimination on our type of matrices typically reduces the dimension by a factor between 2.5 and 3, so we expected that our choice $\#P = 120,000$ would lead to a dense matrix that we would be able to process. This indeed turned out to be the case, but only after we had generated more than 20,000 excess full relations.

Shortly after this computation, the MasPar got upgraded to a core size of one GigaByte, which makes it possible to process dense bit matrices of up to approximately 90,000 rows and columns in about two hours. This corresponds to sparse matrices consisting of approximately 250,000 rows and columns. For a description of the massively parallel Gaussian elimination see [5].

For factorizations obtained with the double large prime variation of the number field sieve, we refer to [1, 8, and 9].

ACKNOWLEDGMENTS

Suggestions by H. W. Lenstra, Jr., R. D. Silverman, S. S. Wagstaff, Jr., and an anonymous referee are gratefully acknowledged.

BIBLIOGRAPHY

1. D. J. Bernstein and A. K. Lenstra, *A general number field sieve implementation*, The Development of the Number Field Sieve, Lecture Notes in Math., vol. 1554, Springer-Verlag, Berlin and New York, 1993, pp. 103–126.
2. J. Brillhart, D. H. Lehmer, J. L. Selfridge, B. Tuckerman, and S. S. Wagstaff, Jr., *Factorizations of $b^n \pm 1$, $b = 2, 3, 5, 6, 7, 10, 11, 12$ up to high powers*, 2nd ed., Contemp. Math., no. 22, Amer. Math. Soc., Providence, RI, 1988.
3. J. D. Horton, *A polynomial-time algorithm to find the shortest cycle basis of a graph*, SIAM J. Comput. **16** (1987), 358–366.
4. B. A. LaMacchia and A. M. Odlyzko, *Solving large sparse systems over finite fields*, Advances in Cryptology, Proc. Crypto '90, Lecture Notes in Comput. Sci., vol. 537, Springer-Verlag, Berlin and New York, 1991, pp. 109–133.
5. A. K. Lenstra, *Massively parallel computing and factoring*, Proc. Latin '92, Lecture Notes in Comput. Sci., vol. 583, Springer-Verlag, Berlin and New York, 1992, pp. 344–355.

6. A. K. Lenstra and H. W. Lenstra, Jr., *Algorithms in number theory*, Handbook of Theoretical Computer Science, vol. A, Algorithms and Complexity, Elsevier, Amsterdam, 1990, Chapter 12.
7. ———, *The development of the number field sieve*, Lecture Notes in Math., vol. 1554, Springer-Verlag, Berlin and New York, 1993.
8. A. K. Lenstra, H. W. Lenstra, Jr., M. S. Manasse, and J. M. Pollard, *The number field sieve*, The Development of the Number Field Sieve, Lecture Notes in Math., vol. 1554, Springer-Verlag, Berlin and New York, 1993, pp. 11–42; Extended abstract: Proc. 22nd Annual ACM Sympos. on Theory of Computing (STOC), Baltimore, May 14–16, 1990, pp. 564–572.
9. ———, *The factorization of the ninth Fermat number*, Math. Comp. **61** (1993), 319–349.
10. A. K. Lenstra and M. S. Manasse, *Factoring by electronic mail*, Advances in Cryptology, Eurocrypt '89, Lecture Notes in Comput. Sci., vol. 434, Springer-Verlag, Berlin and New York, 1990, pp. 355–371.
11. ———, *Factoring with two large primes*, Advances in Cryptology, Eurocrypt '90, Lecture Notes in Comput. Sci., vol. 473, Springer-Verlag, Berlin and New York, 1991, pp. 72–82.
12. K. Mehlhorn and A. Tsakalidis, *Data structures*, Handbook of Theoretical Computer Science, Vol. A, Algorithms and Complexity, Elsevier, Amsterdam, 1990, Chapter 6.
13. F. Morain, *A short note on analyzing the large prime variation*, manuscript, 1991.
14. K. Paton, *An algorithm for finding a fundamental set of cycles of a graph*, Comm. ACM **12** (1969), 514–518.
15. C. Pomerance, *Analysis and comparison of some integer factoring algorithms*, Computational Methods in Number Theory (H. W. Lenstra, Jr. and R. Tijdeman, eds.), Math. Centre Tracts, no. 154/155, Mathematisch Centrum, Amsterdam, 1983, pp. 89–139.
16. C. Pomerance and J. W. Smith, *Reduction of huge, sparse matrices over finite fields via created catastrophes*, Experimental Math. **1** (1992), 90–94.
17. R. D. Silverman, private communication.
18. J. van Leeuwen (ed.), *Handbook of theoretical computer science*, Vol. A, Algorithms and complexity, Elsevier, Amsterdam, 1990.
19. J. van Leeuwen, *Graph algorithms*, Handbook of Theoretical Computer Science, Vol. A, Algorithms and Complexity, Elsevier, Amsterdam, 1990, Chapter 10.
20. S. S. Wagstaff, Jr., Update 2.4 to [2].

BELLCORE, 445 SOUTH STREET, MORRISTOWN, NEW JERSEY 07960
 E-mail address: lenstra@bellcore.com

DEC SRC, 130 LYTTON AVENUE, PALO ALTO, CALIFORNIA 94301
 E-mail address: msm@src.dec.com