Centrum voor Wiskunde en Informatica
**REPORT***RAPPORT*

# Strategies in Filtering in the Number Field Sieve

S. Cavallar

Modelling, Analysis and Simulation (MAS)

**MAS-R0012 May 31, 2000**

Report MAS-R0012
ISSN 1386-3703

CWI
P.O. Box 94079
1090 GB Amsterdam
The Netherlands

# Strategies in Filtering in the Number Field Sieve

Stefania Cavallar
*CWI*
*P.O. Box 94079, 1090 GB Amsterdam, The Netherlands*
*Stefania.Cavallar@cwi.nl*

ABSTRACT
A critical step when factoring large integers by the Number Field Sieve [8] consists of finding dependencies in a huge sparse matrix over the field $\mathbb{F}_2$, using a Block Lanczos algorithm. Both size and weight (the number of non-zero elements) of the matrix critically affect the running time of Block Lanczos. In order to keep size and weight small the relations coming out of the siever do not flow directly into the matrix, but are filtered first in order to reduce the matrix size. This paper discusses several possible filter strategies and their use in the recent record factorizations of RSA-140, R211 and RSA-155.

*2000 Mathematics Subject Classification:* Primary 11Y05. Secondary 11A51.
*1999 ACM Computing Classification System:* F.2.1.
*Keywords and Phrases:* Number Field Sieve, factoring, filtering, Structured Gaussian elimination, Block Lanczos, RSA.
*Note:* Work carried out under project MAS2.2 "Computational number theory and data security".
This report will appear in the proceedings of ANTS IV, Leiden, The Netherlands, July 2–7, 2000.

## Introduction

The Number Field Sieve (NFS) is the asymptotically fastest algorithm known for factoring large integers. It holds the records in factoring special numbers (R211 [3]) as well as general numbers (RSA-140 [4] and RSA-155 [5]). One disadvantage is that it produces considerably larger matrices than other methods, such as the Quadratic Sieve [1]. Therefore it is more and more important to find ways to limit the matrix size. This can be achieved by using good sieving parameters and by "intelligent" filtering.

In this paper we describe the extended version of the program `filter` which we implemented following ideas of Peter L. Montgomery. Its goal is to speed up Block Lanczos's running time by reducing the matrix size but still keeping the weight under control.

A previous implementation of the program `filter` [8, section 7] did 2- and 3-way merges. When using Block Lanczos, higher-way merges were commonly banned from the filter step in order to limit the matrix weight. For instance, also James Cowie et al. [6, section Cycles] explicitly avoided merges higher than 3 for the factorization of RSA-130.

The most important new ingredients of the present `filter` implementation are an algorithm to discard excess relations and "controlled" higher-way merges. We determine arithmetically which merges reduce Block Lanczos's running time.

For the factorization of RSA-140 only 2- and 3-way merges were performed which led to a matrix of 4.7 million columns. With the present filter strategy we could have saved up to 33% of linear algebra time by reducing the size to 3.3 million columns. For the factorization of R211 we already used an intermediate `filter` version which did 4- and 5-way merges, but we could still get an improved matrix after the factorization. For RSA-155, we could take full advantage of the present version and did "controlled" merges up to prime ideal frequency 8 which led to a matrix of 6.7 million columns

and an average of 62 entries per column which was used to factor the number. Afterwards, we were able to reduce this size to 6.3 million columns.

First, we give a brief description of the NFS. Secondly, the `filter` implementation will be described with special focus on the new features. In section 3 we will describe other filter strategies we came across in the literature and compare it with our approach. Finally, experimental results for RSA-140, R211 and RSA-155 are listed and interpreted.

## 1. Brief description of NFS

We briefly describe the NFS factoring method here, skipping parts which are not relevant for the understanding of this paper such as the sieving step itself.

By $N$ we denote the composite number we would like to factor. We select an integer $M$ and two irreducible polynomials $f(x)$ and $g(x) \in \mathbb{Z}[x]$ with $\mathrm{cont}(f) = \mathrm{cont}(g) = 1$ and $f \neq \pm g$ such that $f(M) \equiv g(M) \equiv 0 \bmod N$. By $\alpha, \beta \in \mathbb{C}$ we denote roots of $f(x)$ and $g(x)$, respectively.

The goal is to construct a non-empty set $S$ of co-prime integer pairs $(a, b)$ for which both $\prod_{(a,b)\in S}(a - b\alpha)$ and $\prod_{(a,b)\in S}(a - b\beta)$ are squares, say, $\gamma^2 \in \mathbb{Z}[\alpha]$ and $\delta^2 \in \mathbb{Z}[\beta]$, respectively. Once we have found $S$, the two natural ring homomorphisms $\phi_1 : \mathbb{Z}[\alpha] \to \mathbb{Z}/N\mathbb{Z}$ mapping $\alpha$ to $M$ and $\phi_2 : \mathbb{Z}[\beta] \to \mathbb{Z}/N\mathbb{Z}$ mapping $\beta$ to $M$ as well, yield the congruence

$$\phi_1(\gamma)^2 \equiv \phi_1(\gamma^2) \equiv \prod_{(a,b)\in S} (a - bM) \equiv \phi_2(\delta^2) \equiv \phi_2(\delta)^2 \bmod N.$$

which has the desired form $X^2 \equiv Y^2 \bmod N$. By computing $\gcd(X - Y, N)$ we may find a divisor of $N$. The major obstruction in this series of congruences is that we need to find $\gamma \in \mathbb{Q}(\alpha)$ from $\gamma^2$ (and $\delta$ from $\delta^2$, respectively). See Montgomery's [15] or Phong Nguyen's [17] papers for a description of their square root algorithms.

How to find the set $S$? We write

$$F(x, y) = f(x/y)y^{\deg(f)} \quad \text{and} \quad G(x, y) = g(x/y)y^{\deg(g)}$$

for the homogeneous form of $f(x)$ and $g(x)$, respectively. Consider $a - b\alpha \in \mathbb{Q}(\alpha)$ and $a - b\beta \in \mathbb{Q}(\beta)$. The minus sign is chosen in order to have

$$N_{\mathbb{Q}(\alpha)/\mathbb{Q}}(a - b\alpha) = F(a, b)/c_1 \quad \text{and} \quad N_{\mathbb{Q}(\beta)/\mathbb{Q}}(a - b\beta) = G(a, b)/c_2,$$

where the $c_i$'s are the respective leading coefficients of $f(x)$ and $g(x)$.

After the sieving we are left with many pairs $(a, b)$ such that $\gcd(a, b) = 1$ and both $F(a, b)$ and $G(a, b)$ are products of primes smaller than the large prime bounds $L_1$ and $L_2$, respectively, which were chosen by the user before the sieving. The pairs $(a, b)$ are commonly denoted as *relations*. A necessary condition for

$$\prod_{(a,b)\in S} (a - b\alpha) \quad \text{and} \quad \prod_{(a,b)\in S} (a - b\beta)$$

to be squares is that the norms

$$N_{\mathbb{Q}(\alpha)/\mathbb{Q}}\left(\prod_{(a,b)\in S} (a - b\alpha)\right) \quad \text{and} \quad N_{\mathbb{Q}(\beta)/\mathbb{Q}}\left(\prod_{(a,b)\in S} (a - b\beta)\right)$$

are squares. Therefore we require $S$ to have even cardinality and

$$\prod_{(a,b)\in S} F(a, b) \quad \text{and} \quad \prod_{(a,b)\in S} G(a, b)$$

to be squares. The condition is not sufficient because elements having the same norm may differ from each other (not only by units!). Let $p$ be a prime divisor of $F(a,b) = f(a/b)b^{\deg(f)}$. We distinguish two cases:

- $p \mid f(a/b)$. This means that $a/b \equiv q \bmod p$ with $0 \le q < p$ is a root of $f(x)$ modulo $p$. In the sequel such a $p$ is referred to as $p, q$.
- $p \mid b$. Since $\gcd(a,b) = 1$ it follows that $p \nmid a$ and therefore $p \mid c_1$. This can happen for a small set of primes only, since the leading coefficient is of limited size. These roots are called *projective roots* and denoted as $p, \infty$.

We will call the couples $p, q$, where $q$ is allowed to be $\infty$, *prime ideals*, since they are in bijective correspondence with the first degree prime ideals of the ring $\mathbb{Z}[\alpha] \cap \mathbb{Z}[\alpha^{-1}]$. See [2, Section 12.6].

Consequently, we write

$$|F(a,b)| = \prod_{p,q} p^{e_1(a,b,p,q)} \quad \text{and} \quad |G(a,b)| = \prod_{p,q} p^{e_2(a,b,p,q)}.$$

In order for $\prod_{(a,b)\in S} F(a,b)$ and $\prod_{(a,b)\in S} G(a,b)$ to be squares in $\mathbb{Q}(\alpha)$ and $\mathbb{Q}(\beta)$, respectively, we require all the exponents in

$$\prod_{(a,b)\in S} |F(a,b)| = \prod_{p,q} p^{\sum_S e_1(a,b,p,q)} \quad \text{and} \quad \prod_{(a,b)\in S} |G(a,b)| = \prod_{p,q} p^{\sum_S e_2(a,b,p,q)}$$

to be even. This condition can be stated in terms of the field $\mathbb{F}_2$ as well. We just think of a relation $(a,b)$ as a vector in $\mathbb{F}_2$ whose first entry is 1 (in order to control the parity of $S$) and the following entries are given by the exponents $e_1(a,b,p,r)$ and $e_2(a,b,p,r)$ modulo 2. A 1 signals the occurrence of an uneven power of a prime ideal. The task of finding some suitable sets $S$ translates now into finding dependencies modulo 2 between the columns of a matrix which is built up with the relation vectors given by the siever. We need to have enough relations to guarantee that the matrix provides enough dependencies.

Alas, not every dependency yields a set $S$ such that $\prod_{(a,b)\in S}(a - b\alpha)$ and $\prod_{(a,b)\in S}(a - b\beta)$ are squares, but we can make the method practical by producing several dependencies and doing quadratic character tests [2, Section 8].

The filter stage occurs between the sieving step and the linear algebra step of the NFS. It is a preliminary linear algebra process since it corresponds to dropping columns (*pruning*) and adding up columns modulo 2 (*merging*).

## 2. Description of the new filter tasks

We distinguish 19 merge levels: level 0 and 1 fall into pruning, level 2 through 18 within merging.

We shall say that a prime ideal $p, q$ is *(un)balanced* in a relation $(a,b)$ if it appears to an (un)even number in $F(a,b)$ or $G(a,b)$*. We distinguish between prime ideals of norm below and above a user determined bound `filtmin`. Accordingly, we speak about *small* and *large* prime ideals. We will denote prime ideals $p, q$ by $I$. We write a relation $r = r(a,b)$ as the collection of its unbalanced large prime ideals, $r : I_1, I_2, \ldots, I_k$. Merging means combining relations which have a common prime ideal in order to balance it. For example, if $I$ appears only in $r_1 : I_{10} = I, I_{11}, \ldots, I_{1k_1}$ and $r_2 : I_{20} = I, I_{21}, \ldots, I_{2k_2}$, we can combine the two relations into $r_1 + r_2 : I_{11}, \ldots, I_{1k_1}, I_{21}, \ldots, I_{2k_2}$ with the result that $I$ is balanced in $r_1 + r_2$. More generally, a *$k$-way merge* is the procedure of combining $k$ relations with a common prime ideal $I$ into $k-1$ relation pairs without $I$. By a *relation-set* we mean a single relation, or a collection of two or more relations generated by a merge. We do merges up to prime ideal frequency 18. The parameter `mergelevel` $l$ means that $k$-way merges with $k \le l$ may be performed. The weight of a relation-set $r$, i.e., the number of unbalanced prime ideals in it, is denoted by $w(r)$.

*In very rare cases ($p$ divides the polynomial resultant) we can have the same $p, q$ appearing in both $F$ and $G$. Recall that they are *not* the same, since they correspond to ideals in different rings. We abstain from labeling the ideals accordingly, for the sake of simplicity.

*2.1 Pruning*

As the verb "pruning" suggests, this part of the program removes unnecessary relations from the given data, that is *duplicates* and *singletons* and, if the user wants to, also excess relations. Duplicates are obviously superfluous and singletons cannot be part of a winning set $S$ since they contain a prime ideal which does not occur in any other relation and can subsequently not be combined to form a square. If the difference between the number of relations and the number of large prime ideals outnumbers a user-chosen bound (`keep`), the *clique* algorithm selects relations to delete.

`mergelevel` 0 only removes duplicates and can be used to merge several sieving outputs to a single file, possibly before sieving completes. `mergelevel` 1 will only be performed if the full set of relations is available and covers algorithms for the removal of duplicates, singletons and excess relations.

*Duplicates.* First we want to eliminate duplicate relations. They may arise for various reasons. Most commonly they come from sieving jobs that were stopped and later restarted. In case of a *line-by-line siever* [8, section 6] the resumed jobs start with the last $b$ sieved by the previous job; this is the only way that duplicates arise. In case of a *lattice siever* [18] the job starts with the special prime ideal $I$ sieved last, and will generate duplicates, or it can do so because a relation may contain, apart from its own special $I$, other prime ideals that are used as special prime ideals as well. The simultaneous use of line-by-line and lattice siever also causes overlap.

Duplicates are tracked down by hashing [12]. Since it is easier and cheaper to use a number instead of a relation as a hash table entry, we "identify" a relation with a number. The user specifies how many relations he expects to be in the input file(s) (`maxrelsinp`). This figure is used to choose the size of the in-memory tables needed during the pruning algorithm. The program reads in relation after relation. In order to detect duplicates, the program maps each relation $(a, b)$ to an integer between 0 and $2^{64} - 1$. The mapping function, $h = h(a, b)$, should be nearly injective since relations mapped to the same value will be treated as duplicates. It is rather easy to construct such a function, since even a huge amount of relations, say 200 million (for RSA-155 we had to handle 124.7 million relations), is small compared to the $2^{64}$ possible function values. With 64 bits for the function value we expect about

$$\frac{\binom{2 \cdot 10^8}{2}}{2^{64}} \approx 0.0011$$

false duplicates, which means that there will hardly be any false duplicates. With 32 bits only, this number would amount to about $4.7 \cdot 10^6$, which is a fair proportion of all relations.

The function $h(a, b)$ is defined as follows. It takes values of $a$ and $b$ up to $2^{53}$. Put $\Pi = \lfloor \pi \cdot 10^{17} \rfloor$ and $E = \lfloor e \cdot 10^{17} \rfloor$. We have $\gcd(\Pi, E) = 1$. Define

$$H(a, b) = \Pi a + Eb.$$

If $H(a_1, b_1) = H(a_2, b_2)$ and $(a_1, b_1) \neq (a_2, b_2)$ we have

$$\frac{a_1 - a_2}{b_1 - b_2} = -\frac{E}{\Pi}$$

which is impossible, since $|a|$ and $|b|$ are known to be much smaller than $\Pi/2$ and $E/2$, and $\gcd(\Pi, E) = 1$. Define $h(a, b) = H(a, b) \bmod 2^{64}$. Since $H$ is injective, false duplicates for $h$ can only come from the truncation modulo $2^{64}$.

The function values of $h$ again are mapped by a hash function into a hash table. If the user has specified `mergelevel` 0, the non-duplicates are written to the output file whereas, if the user has chosen `mergelevel` 1, the non-duplicate relations are memorized in a table for further processing, while considering only the large prime ideals. In the sequel, we shall call this table the *relation table*.

*Singletons.* If both polynomials $f$ and $g$ split completely into distinct linear factors modulo a prime $p$ which does not divide the leading coefficients, we get a so-called free relation corresponding to the prime ideal factorization of the elements $p = p - 0\alpha$ and $p = p - 0\beta$ of norm $N_{\mathbb{Q}(\alpha)/\mathbb{Q}}(p) = F(0,p)/c_1 = p^{\deg(f)}$ and $N_{\mathbb{Q}(\beta)/\mathbb{Q}}(p) = G(0,p)/c_2 = p^{\deg(g)}$, respectively. Approximately $1/(g_f \cdot g_g)$ of the primes offer a free relation, where $g_f$ and $g_g$ are the orders of the Galois groups of the polynomials $f$ and $g$, respectively [10]. The free relation $(p, 0)$ is added to the relation table only if all prime ideals of norm $p$ appear in the relation table.

Next, a frequency table is built for all occurring prime ideals which is adjusted as the relation table changes. The relation table is then scanned circularly and relations containing an ideal of frequency 1 (singletons) are removed from it. The program executes as many passes through the table as is needed to remove all singletons.

At the end of the pruning algorithm we would like the remaining number of relations to be larger than the total number of prime ideals. Therefore we need to reserve a surplus of relations for the small prime ideals: Per polynomial, the number of prime ideals below `filtmin` is approximately $\pi(\texttt{filtmin})$, i.e., the number of primes below `filtmin`, see [14]. Consequently, we require a surplus of approximately $(2 - (g_f \cdot g_g)^{-1}) \cdot \pi(\texttt{filtmin})$ relations. If the required surplus is not reached we need to sieve more relations.

*Clique algorithm.* If there are sufficiently many more relations than ideals, the user may want to specify how many more relations than large ideals to retain after the pruning stage (`keep`).

In [19, step 3] Pomerance and Smith eject excess relations by simply deleting the heaviest relations. However, as an alternative, they suggest to delete relations which contain many primes of frequency 2. Our approach is similar to this alternative. The algorithm we use is called *clique algorithm*, since it deletes relations that stick together.

Consider the graph with the relations from the relation table as nodes. We connect two nodes if the corresponding relations would be merged in a 2-way merge. The components of the graph are called cliques. The relations in a clique are close to each other in the sense that if one of them is removed, the others will become singletons after some steps and are therefore useless.

The clique algorithm determines all the cliques, evaluates them with the help of a metric and at each step keeps up to a prescribed number of them in a priority heap [12, page 144], ordered by the size of a metric value. The metric being used weighs the contribution from the small prime ideals by adding 1 for each relation in the clique and 0.5 for each free relation. The large prime ideals which occur more than twice in the relation table contribute $0.5^{f-2}$ where $f$ is the prime ideal's frequency. This way we "penalize" ideals with low frequency. Relation-sets containing many ideals with low frequencies are more likely to be deleted than those containing mainly high frequency ideals. By deleting these low-frequency relation-sets we hope to reduce especially low frequencies even more and get new merge candidates.

Finally, the relations belonging to cliques in the heap are deleted from the relation table. When deleting relations we decrease the ideal frequencies of the primes involved. Singletons may arise and we therefore continue with the singleton processing step. The clique algorithm may be repeated if the number of excess relations does not approximate `keep` sufficiently.

After duplication, singleton and possibly clique processing the relations are read again and only the non-free relations[†] appearing in the relation table are written to the output file. If the input files have grown in the meantime, the new relations are discarded.

### *2.2 Merging*

First, we have a closer look at how merging works, which parameters can be given and at how to minimize the weight increase during a $k$-way merge. Next, we give details about the implementation

[†] Free relations will be generated during the merge stage again.

of the "controlled" merges. Finally we study the influence of merging on Block Lanczos's running time.

Merging aims at reducing the matrix size by combining relations. Throughout this section we give figures about weight changes in the matrix. These figures do not take account of possible other primes that may have been balanced incidentally during the same merge.

*Parameters* `mergelevel`, `maxpass`, `maxrels` *and* `maxdiscard`. With the parameter `mergelevel` the user specifies the highest $k$ for which $k$-way merges are allowed to be executed. The user fixes the maximum number (`maxpass`) of *shrinkage passes* to execute. During a shrinkage pass, all large primes are checked once and possibly merged, see [8, section 7] for more details.

The simplest case is the so-called 2-way merge. A prime ideal $I$ is unbalanced in exactly two relations, $r_1$ and $r_2$, and we combine the relations into the relation-set $r_1 + r_2$. As a result, we have one fewer column ($r_1$ and $r_2$ disappear, $r_1 + r_2$ enters) as well as one fewer row (prime ideal $I$) and the total weight has thereby decreased by 2.

In general, if a prime ideal $I$ is unbalanced in exactly $k$ relations ($k \geq 2$)[‡], we can choose $k-1$ independent relation pairs out of the possible $\binom{k}{2}$ pairs. For example, if $k = 3$, there are 3 possible ways to combine the 3 relations involved, $r_1$, $r_2$ and $r_3$, to a couple, namely $r_1+r_2$, $r_2+r_3$ and $r_1+r_3$. Each one can be obtained from the other two, for instance $r_1 + r_3 = (r_1 + r_2) + (r_2 + r_3)$ as all the prime ideals of $r_2$ are balanced since $r_2$ appears twice.

After the merge, the prime ideal $I$ is balanced. Its corresponding row has disappeared from the matrix. The total gain of every merge consists in fact in one fewer column and one fewer row. The drawback of merging is, of course, matrix fill-in. A 2-way merge causes no fill-in at all, we even have 2 entries fewer in the matrix. However, a $k$-way merge, $k \geq 3$, causes the matrix to be heavier by about the weight of $k-2$ relations minus the $2(k-1)$ entries that disappeared.

If the matrix is going to be "lopsided", i.e., if it has many more relations than ideals, it is useful to drop heavy relation-sets. The program therefore discards the ones which contain more relations than the user-determined bound `maxrels`.[§] The user may specify `maxdiscard`, that is, the maximum number of relation-sets to be dropped during one `filter` run. Once `maxdiscard` has been reached, $k$-way merges, $k \geq 3$, are inhibited.

*Minimizing the weight increase of a $k$-way merge.* Which $k-1$ of the possible $\binom{k}{2}$ relation pairs should be chosen in order to achieve the lowest weight increase? First of all, each relation has to appear in at least one relation couple, that is, we need to form independent relation sets, in order not to loose data. Secondly, we focus on minimizing the weight increase. In the beginning, when all relations are true single relations, we usually achieve the lowest weight increase by choosing the lightest relation (*pivot*) and combining it with the remaining $k-1$ relations. We call this *pivoting*. More precisely, this happens always when no additional prime ideals except for the prime ideal $I$ become balanced in any of the candidate relation couples. If we assume the pivot relation to be $r_k$, the weight increase $\Delta w$ will be exactly

$$\Delta w = (k-2)w(r_k) - 2(k-1). \tag{2.1}$$

The choice becomes more complicated, when additional prime ideals get balanced, especially when we are merging already combined relation-sets. For example, consider the following 5 relations, which

[‡] The case $k = 1$ denotes a singleton which would be deleted.

[§] We weigh a free relation less than 1 (we used 0.5), because, even if it may have several large primes, it should have less total weight.

are candidates for two 3-way merges with the prime ideals $I$ and $J$:

$r_1$ : $I$ and $v-1$ other prime ideals
$r_2$ : $I$ and $v-1$ other prime ideals
$r_3$ : $I, J$ and $v-2$ other prime ideals
$r_4$ : $J$ and $v-1$ other prime ideals
$r_5$ : $J$ and $v-1$ other prime ideals

For the sake of simplicity, we assume that all the relations have the same weight $v$ and do not share other primes except for $I$ and $J$. Imagine, $r_3$ is used as a pivot relation to eliminate $I$. We get

$r_1 + r_3$ : $J$ and $2v-3$ other prime ideals
$r_2 + r_3$ : $J$ and $2v-3$ other prime ideals
$r_4$ : $J$ and $v-1$ other prime ideals
$r_5$ : $J$ and $v-1$ other prime ideals

Now $J$ appears 4 times, so we need a 4-way merge to balance it. For the elimination of $J$ the two relations $r_4$ and $r_5$ seem the best pivot candidates in a 4-way merge, since they have lowest weight. However, pivoting with $r_5$ results into

$(r_1 + r_3) + r_5$ : $3v-4$ prime ideals
$(r_2 + r_3) + r_5$ : $3v-4$ prime ideals
$r_4 + r_5$ : $2v-2$ prime ideals

with total weight $8v-10$, whereas

$(r_1 + r_3) + (r_2 + r_3)$ : $2v-2$ prime ideals
$(r_1 + r_3) + r_5$ : $3v-4$ prime ideals
$r_4 + r_5$ : $2v-2$ prime ideals

ends with weight $7v-8$¶. When $v > 2$ we have $8v-10 > 7v-8$ which indicates that we should not stick to pivoting for all the merges.

The problem of minimizing the weight increase can be stated using graphs. The vertices are given by the $k$ relations which are candidates for a $k$-way merge and the $\binom{k}{2}$ edges between them represent possible merges. The edge between two nodes $r_j$ and $r_j$ has weight $w(r_i + r_j)$. Given this weighted graph we wish to select a tree with minimum total weight. The solution is called a *minimum spanning tree* [11, page 460]. This problem is a well-known problem of combinatorial optimization. In order to solve it we use the algorithm as formulated by Jarník [9, pages 46–47].

*Implementation of "controlled" merges.* We limit the weight increase of a single merge by requiring that a merge should not add more than a prescribed number, $m_{max}$, of original relations to the matrix. We give all the initial relations the same weight (except for free relations that weigh one half), which is reasonable since the relations are the factorizations of numbers of about the same size.

Let us consider $k$ relation-sets which are candidates for a $k$-way merge. The individual relation-sets may contain several original relations. Suppose the lightest candidate relation-set has $j$ relations, where free relations count for 0.5. Let $c$ be the number of relation-sets with exactly this minimum number $j$ of relations. Shrinkage pass 1 starts with $m = 1$ and we subsequently augment $m$ up until $m_{max}$ and allow for the $k$-way merge when $(k-2)j \leq m-(c-1)/2$. The $m$ gives the maximum weight increase (in number of relations) allowed during a merge. We introduced $c$ in order to postpone some merges and do the ones where the best way to merge is clear cut first. Since we are still interested

¶ The latter situation is also achieved when first using $r_1$ as a pivot and then doing a 3-way merge with pivot relation $r_5$.

| $k$ | $\left\lfloor \frac{m_{max}}{k-2} 2 \right\rfloor /2$ | |
|---|---|---|
| | $m_{max} = 7$ | $m_{max} = 8$ |
| 3 | 7 | 8 |
| 4 | 3.5 | 4 |
| 5 | 2 | 2.5 |
| 6 | 1.5 | 2 |
| 7 | 1 | 1.5 |
| 8–9 | 1 | 1 |
| 10 | 0.5 | 1 |
| 11–16 | 0.5 | 0.5 |
| 17–18 | — | 0.5 |

Table 1: Allowed number of relations in pivot relation-set for $k$-way merge

in doing lower weight merges before higher weight merges we increase $m$ only every other shrinkage pass and set $c = 1$ during these shrinkage passes. In most of the runs we had $m_{max} = 7$, but we tried $m_{max} = 8$ as well. Solving the inequality $(k-2)j \leq m_{max}$ for $k$ gives $k \leq \frac{m_{max}}{j} + 2$. It follows that, with $m_{max} = 7$, merges with ordinary relations ($j = 1$) are limited to prime ideal frequency 9 whereas free relations ($j = 0.5$) can be used in merges up to prime ideal frequency 16. For the factorization of RSA-155 we performed merges up to prime ideal frequency 8.

Table 1 shows the maximum number of relations a pivot relation-set may consist of, for $m_{max} = 7$ and 8. Even if we are not pivoting, we ask at least one relation not to contain more relations than this bound.

*Influence of merging on Block Lanczos's running time.* Given an $m \times n$ matrix, $n > m$, of total weight $w$, the running time estimate of Block Lanczos is given by $\mathcal{O}(wn) + \mathcal{O}(n^2)$ [16]. Both terms grow with $n$, so we will focus on reducing $n$. If we manage to reduce $n$ by a certain factor while $w$ does not grow by more than this factor, we will get a running time reduction, independently of the constants in the two terms. Moreover, we predict the constant in the $\mathcal{O}(n^2)$ term to be the larger one. Therefore, it is natural to write the running time as

$$\mathcal{O}((w + Cn)n) \tag{2.2}$$

with $C \geq 1$. Since we do not need absolute running times, we drop the $\mathcal{O}$-sign and use the function $t(n, w) = (w + Cn)n$. The larger the constant $C$, the more it will be convenient to reduce the matrix size. The constant depends on the implementation, for example on the number of bits per vector element ($K$) used[‖]. Montgomery (personal communication) at first estimated the constant $C$ to be about 50. For some approximate values of $C$ see Table 7 or Table 2.

Let us determine a bound for the weight increase $\Delta w$ such that a merge causing an increase below this bound still is beneficial to the running time. The condition for $\Delta w$ becomes

$$t(n-1, w + \Delta w) - t(n, w) < 0. \tag{2.3}$$

Inequality (2.3) is equivalent to

$$0 > n\left((1-2n)C - w + (n-1)\Delta w\right) = (n-1)(-2Cn - w + n\Delta w) - w - Cn.$$

[‖] Montgomery [16] gives the formula $\mathcal{O}(wn/K) + \mathcal{O}(n^2)$ for the running time.

The inequality is satisfied if $\Delta w < 2C + \frac{w}{n}$. It follows that the allowed weight increase grows with $C$ and the average column weight $\frac{w}{n}$. That means that denser matrices allow heavier merges than sparser matrices do.

Let us calculate a limit for the pivot relation weight $j$ of a general $k$-way merge, $k \geq 3$. According to equation (2.1) we require

$$\Delta w = (k-2)j - 2(k-1) < 2C + \frac{w}{n}.$$

which results into

$$j < \frac{2C + \frac{w}{n} + 2(k-1)}{k-2}. \tag{2.4}$$

In Table 2 we report the allowed pivot relation weights for merges up to prime ideal frequency 10. We chose $\frac{w}{n} = 30$ (typical after applying only 2- and 3-way merges) and $\frac{w}{n} = 50$ (typical $\frac{w}{n}$ of many of our final matrices). The horizontal lines divide between above and below $\frac{w}{n}$.

| $k$ | $\left\lceil \frac{2C + \frac{w}{n} + 2(k-1)}{k-2} \right\rceil - 1$ | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | $\frac{w}{n} = 30$ | | | | $\frac{w}{n} = 50$ | | | |
| | $C = 49$ | $C = 37$ | $C = 14$ | $C = 1$ | $C = 49$ | $C = 37$ | $C = 14$ | $C = 1$ |
| 3 | 131 | 107 | 61 | 35 | 151 | 127 | 81 | 55 |
| 4 | 66 | 54 | 31 | 18 | 76 | 64 | 41 | 28 |
| 5 | 45 | 37 | 21 | 13 | 51 | 43 | 28 | 19 |
| 6 | 34 | 28 | 16 | 10 | 39 | 33 | 21 | 15 |
| 7 | 27 | 23 | 13 | 8 | 31 | 27 | 17 | 12 |
| 8 | 23 | 19 | 11 | 7 | 26 | 22 | 15 | 10 |
| 9 | 20 | 17 | 10 | 6 | 23 | 19 | 13 | 9 |
| 10 | 18 | 15 | 9 | 6 | 20 | 17 | 11 | 8 |

Table 2: Allowed pivot relation weights for $k$-way merge

From Table 2 we can see that 3-way merges can be done with rather heavy pivot relations; even for $C = 1$ and $\frac{w}{n} = 50$ the allowed weight exceeds $\frac{w}{n}$. Denser matrices allow also for denser pivot relations.

By substituting $\frac{w}{n}$ for $j$ in (2.4) we can derive a condition for when to do $k$-way merges for $k > 3$ with an average weighing pivot relation:

$$\frac{w}{n} < \frac{2C + 2(k-1)}{k-3} \tag{2.5}$$

The analysis for $k = 3$ has to be done separately, we require (2.3) for $\Delta w = \frac{w}{n} - 4$ By reorganizing the terms we get $-4\,(n-1) - \frac{w}{n} - C\,(2n-1) < 0$ which is always satisfied. This means that 3-way merges with an average weight pivot relation are always profitable, independently from the density of the matrix or the constant $C$.

Table 3 gives the allowed average weights when merging with an average weight pivot relation. If we assume $C < 50$ and we apply the merges in ascending order of prime ideal frequency, 6-way merges with average weighing pivot relations will not be worthwhile because after the 5-way merges we have seen in practice $\frac{w}{n}$ to be around 50, which is higher than the maximum value of 35.

| $k$ | $\left\lceil \frac{2C+2(k-1)}{k-3} \right\rceil - 1$ | | | |
|---|---|---|---|---|
| | $C = 49$ | $C = 37$ | $C = 14$ | $C = 1$ |
| 4 | 103 | 79 | 33 | 7 |
| 5 | 52 | 40 | 17 | 4 |
| 6 | 35 | 27 | 12 | 3 |
| 7 | 27 | 21 | 9 | 3 |
| 8 | 22 | 17 | 8 | 3 |
| 9 | 18 | 14 | 7 | 2 |
| 10 | 16 | 13 | 6 | 2 |

Table 3: Allowed average weights for $k$-way merge

## 3. Other methods in the literature

We would like to mention two articles about similar filter strategies. These are "Solving Large Sparse Linear Systems Over Finite Fields" of LaMacchia and Odlyzko from 1990[13] and "Reduction of Huge, Sparse Matrices over Finite Fields Via Created Catastrophes" of Pomerance and Smith from 1992[19]. Their strategies are similar to each other but differ in some points. Both were designed to reduce the initial data to a substantially smaller matrix. This matrix was allowed to be fairly dense since it was going to be processed by Gaussian elimination afterwards. In contrast, the purpose of our method is to reduce the matrix size but still keep it sparse in order to take advantage of the Block Lanczos method. They were dealing with matrices of size up to 300K, we with matrices of size up to 7M. Each reflects the maximum size that could be handled at the time.

Both other methods executed their operations on the matrix itself whereas we dealt with the raw relations. We identified relations with columns in the final matrix whereas they identified relations with rows. Nevertheless, for an easier comparison, we will stick to identify relations with columns in the present description.

They operate only on part of the matrix (active rows) where no fill-in takes place. The operations must be memorized in order to be repeated on the complete matrix afterwards. LaMacchia and Odlyzko store the history in core, whereas Pomerance and Smith keep a history file.

We will distinguish between the pruning and merging step, as in the description of our method. The weight they look at is only the weight of the active primes at that moment.

The pruning step does differ from our approach only in how to delete excess relations. Duplicates and singletons are removed as soon as possible, as in our approach. Pomerance and Smith choose to remove the excess immediately, whereas LaMacchia and Odlyzko remove the excess just before the "collapse" or "catastrophe" during the merge step. Both decide to drop the heaviest relations, but Pomerance and Smith indicate that one might try other strategies (as we did).

In the beginning of the merge stage, a small number of rows (the heaviest, which correspond to small primes) are declared inactive. Merges are done by pivoting with columns that have only one 1 in the active part. There is no fixed limit for the prime ideal frequency up to which to merge. Once all possible merges have been done and there are still 1's in the active part, more rows (again the heaviest) are declared inactive and the merge step is repeated. This is repeated until the active part collapses. This procedure leads to very heavy matrices. To overcome this, LaMacchia and Odlyzko for example, extend the inactive part considerably after it has reached a certain critical size. This way fewer merges can be executed and the fill-in is confined. Nevertheless, the matrices still have high column weights: the lightest example given by LaMacchia and Odlyzko has an average of 115 entries per column for a $6.0 \cdot 10^4$ columns matrix which is much denser than our densest matrix, the $6.3 \cdot 10^6$

columns matrix from Table 11 having an average 81 entries per column*.

Initially, for a sparse matrix, merges are done with very light columns, since the inactive part is small and cannot contain many 1's. Further on, pivot relations can be very heavy: very probably, the single 1 in the increasingly smaller active part mostly represents a large prime and goes together with many small prime factors, since all polynomial values are about the same size (Pomerance and Smith try to overcome this by also allowing merges with pivot columns having two 1's in the active part of the matrix.). Moreover, they do not make a distinction between "original" pivot relations and already merged ones, which can be substantially heavier.

In our merge procedure we also merge with already merged relations, but this happens in a controlled way. We limit the number of original relations which can be added during a single merge. We also minimize the fill-in per merge by using a minimum spanning tree algorithm instead of the simpler pivoting, see Section 2.2. But here we also have to say, that we cannot guarantee to always get the cheapest merge, because we count the contribution from the large prime ideals but only *estimate* the contribution from the small prime ideals.

In 1995, Thomas Denny proposed a Structured Gaussian elimination preliminary step for Block Lanczos [7]. He estimated $C = 1$ for his own Block Lanczos program. We therefore also included $C = 1$ in Tables 2 and 3.

## 4. Experimental results

The experiments were done with two versions of our program `filter`. Both of them include pruning facilities.

The first version was capable of doing merges up to prime ideal frequency 5 and corresponded to the old program [8, section 7] if invoked with `mergelevel` 2 or 3. With the first version the user needed to specify when to start with the 4- and 5-way merges. For example, in the tables about filter runs (Tables 5, 8 and 10) the notation 4(x) in column `mergelevel` means that 4-way merges started $x$ shrinkage passes after 3-way merges started. 5(x-y) means that 4-way merges started $x$ shrinkage passes after 3-way merges did, and 5-way merges started $y$ shrinkage passes later than 3-way merges.

The present `filter` version does not need this information any more. It can do merges up to prime ideal frequency 18. The merges are done in order of weight increase (measured in numbers of original relations). All runs except RSA-155's B6 had $m_{max} = 7$.

Table 4 gives an overview of all pruning activities in our experiments for RSA-140, R211 and RSA-155. All the figures are in units of a million. With prime ideals we mean prime ideals above 10M; we need to reserve an excess of 1.3M relations for the small prime ideals. The non-duplicate relation counts differ so much due to the use of different large prime bounds. Apparent errors are due to rounding values to units of one million.

The figures in Tables 5–11 are given in units of a million (M) or a thousand (K). We labeled the experiments with capital letters. All experiments with the same letter started with the same `mergelevel` 1 run.

In Tables 5, 8 and 10, columns 2–6 are input parameters. Column 7–10 are results: column "sets" gives the number of relation-sets remaining after the run, column "discarded" gives the total number of relation-sets which were discarded during the run. "excess" gives how many more relations than the approximate total number of ideals we retained. It indicates how many more relations we might still throw away in a further run. "not merged" gives the number of large prime ideals of frequency smaller or equal to `mergelevel` among the output relations. For the runs with the new version we also report the number of output relation-sets made of one single relation since among those could be candidates for future high-way merges.

The Block Lanczos code typically finds almost $K$ dependencies [16], where $K$ is the number of bits per vector element. This enables us to drop the heaviest rows which leads to substantially

*The column weight 70 given in Table 11 corresponds to the matrix obtained when dropping the prime ideals of norm below 40.

| number being factored | | RSA-140 | | R211 | | RSA-155 | | | |
|---|---|---|---|---|---|---|---|---|---|
| experiment | | A | B | A | B | A | B | C | D |
| raw relations | (1) | 65.7 | 68.5 | 57.6 | | | 130.8 | | |
| duplicates | (2) | 10.6 | 11.9 | 10.6 | | | 45.3 | | |
| non-duplicates | (3)=(1)−(2) | 55.1 | 56.6 | 47.0 | | | 85.5 | | |
| free relations | (4) | 0.1 | 0.1 | 0.8 | | | 0.2 | | |
| prime ideals | (5) | 54.2 | 54.7 | 49.5 | | | 78.8 | | |
| excess | (6)=(3)+(4)−(5) | 1.1 | 2.0 | −1.7 | | | 6.9 | | |
| singletons | (7) | 28.5 | 28.2 | 26.5 | | | 32.5 | | |
| relations left | (8)=(3)+(4)−(7) | 26.8 | 28.5 | 21.3 | | | 53.2 | | |
| prime ideals left | (9) | 21.5 | 22.6 | 18.5 | | | 42.6 | | |
| excess | (10)=(8)−(9) | 5.2 | 6.0 | 2.8 | | | 10.6 | | |
| clique relations | (11) | 17.6 | 18.7 | 7.4 | 0 | 34.1 | 33.0 | 29.6 | 22.9 |
| relations left | (12)=(8)−(11) | 9.2 | 9.8 | 13.9 | 21.3 | 19.1 | 20.2 | 23.6 | 30.3 |
| prime ideals left | (13) | 7.8 | 8.1 | 12.2 | 18.5 | 17.4 | 18.2 | 20.6 | 25.3 |
| excess (=`keep`) | (14)=(12)−(13) | 1.4 | 1.7 | 1.7 | 2.8 | 1.7 | 2.0 | 3.0 | 5.0 |

Table 4: summary of `mergelevel` 0 and 1 runs

lighter matrices*. We dropped the rows corresponding to prime ideals of norm smaller than 50 for R211, whereas for RSA-140 and RSA-155, which have both exceptionally many small prime ideals, we omitted the prime ideals of norm smaller than 40†. In addition, the Block Lanczos code truncates every $m \times n$ matrix by default to $m \times (m + K + 100)$.

The tables featuring matrix data (Tables 6, 9 and 11) are made of two parts. In the first part we state the real size ($m \times n$), weight ($w$) and average column weight ($\frac{w}{n}$) of the matrices built. The numbers between two lines express the changes in size (number of columns) and weight from one matrix to the smaller one as percentages. Note that a $i$% decrease in matrix size makes the term $wn$ shrink as long as the weight does not increase by more than $\frac{100i}{100-i}$% which is slightly larger than $i$%. The second part shows the effective weight ($w_{eff}$) after truncating the matrix to size $m \times (m+K+100)$, the effective average column weight ($\frac{w_{eff}}{m+K+100}$) and the Block Lanczos timings from a Cray C90 and a Silicon Graphics Origin 2000. The timings can vary substantially according to the load on the machines (other jobs interacting with ours): time differences of 20% are not unusual. Aiming at a fair comparison we tried to run the matrices at times with comparable load. In our tables, comparable timings are written in the same column. Only one Block Lanczos job per number was completely executed. All times in the tables are extrapolations: we did a short run, took the time of the fastest iteration and multiplied it by the number of iterations $(m + K + 100)/(K - 0.76)$, see [16].

*RSA-140*

This 140-digit number was factored on February 2, 1999. The experiment series A started with 65.7M raw relations, B with 68.5M from 5 different sites. We removed 1.4M and 1.6M duplicates, respectively, with `mergelevel` 0 runs on each contributor's data. The experiments in Table 5 start with the remaining 64.3M respectively 66.9M relations having 54.2M and 54.7M large prime ideals, respectively. After the pruning step (with `filtmin`= 10M) we need an excess of $\frac{239}{120}\pi(10\text{M}) = 1.3\text{M}$ for the small prime ideals. For a summary of `mergelevel` 0 and 1 runs, see Table 4.

In this paragraph we only describe experiment series A. The `mergelevel` 1 run on the whole bunch of data removed another 9.2M duplicates and added 0.1M free relations for large primes. Note, that

*In particular, all quadratic character rows are omitted. The pseudo-dependencies being found for this reduced matrix must be combined to real dependencies afterwards.

†These figures match with the implementation for $K = 64$. For $K = 128$, we could even have dropped the prime ideals up to norm 180. The resulting lighter matrices would have led to shorter timings for that implementation. However, for simplicity, we used the same matrices for both the $K = 64$ and the $K = 128$ versions.

at this point the excess 64.3M − 54.2M − 9.2M + 0.1M = 1.1M[‡] was less than the needed 1.3M. The excess was sufficient only after removing the singletons, when we were left with 26.8M relations having 21.5M large prime ideals. The clique algorithm removed a total of 17.6M relations to approximate the excess of 1.4M = 9.2M − 7.8M.

The factorization was done using matrix A1.1 which took 100h on the Cray. Only 2- and 3-way merges were performed, because the code for higher than 3-way merges was not ready by then. For logistic reasons we had built the matrix before we received all the data.

With the complete data (experiment series B) the excess was enough from the beginning. Furthermore, a matrix constructed from this data by applying the same filter strategy as for A1.1 would have performed better than A1.1 as one can imagine when comparing A1.1.2.1 to B1.2: both did merges up to prime ideal frequency 5 and the latter is smaller in size *and* weight.

We also tried `mergelevel` 8 (B2) with $m_{max} = 7$ which was introduced only just before the factorization of RSA-155. The program stopped with $k$-way merges, $k \geq 3$ at shrinkage pass 10 after having deleted 381K relations. This means that only merges with a maximum weight increase of 6 original relations had been done. Matrix B2 beats the `mergelevel` 5 matrix of the same series (B1.2).

In Table 6 one can see from the percentages that each size reduction should have a favourable effect on Block Lanczos's running time which is confirmed by the time column.

These experiments confirm our idea of the advantage of higher-way merges. They show that collecting more data than necessary is recommendable. It does not become clear, however, how much excess data one should keep after the pruning step.

| experiment | `mergelevel` | `filtmin` | `maxdiscard` | `maxrels` | `maxpass` | sets | discarded | excess | not merged |
|---|---|---|---|---|---|---|---|---|---|
| A | 1 | 10M | `keep` 1.4M | | | 9.2M | 46 040K | 90K | - |
| A1 | 2 | 10M | - | 4.0 | 6 | 6.0M | 54K | 36K | 59 |
| A1.1 | 3 | 10M | unlim. | 10.0 | 10 | 4.7M | 3K | 33K | 0 |
| A1.1.1 | 4(0) | 10M | 20K | 10.0 | 10 | 4.2M | 20K | 13K | 243K |
| A1.1.2 | 4(0) | 10M | 20K | 12.0 | 10 | 4.0M | 14K | 20K | 0 |
| A1.1.3 | 4(0) | 10M | 20K | 11.0 | 10 | 4.0M | 20K | 13K | 48K |
| A1.1.2.1 | 5(0-0) | 8M | 17K | 15.0 | 10 | 3.5M | 17K | 4K | 0 |
| B | 1 | 10M | `keep` 1.7M | | | 9.8M | 46 906K | 384K | - |
| B1 | 4(5) | 10M | 300K | 8.0 | 12 | 4.3M | 170K | 208K | 6K |
| B1.1 | 5(1-3) | 10M | 200K | 11.5 | 10 | 3.6M | 85K | 128K | 1K |
| B1.2 | 5(1-3) | 10M | 200K | 10.5 | 10 | 3.4M | 200K | 14K | 28K |
| B2 | 8 | 10M | 375K | 8.0 | 15 | 3.3M | 383K | 1K | 909K/455K |

Table 5: RSA-140 filter runs

With each timing column, we fitted a surface $t = s_1 n^2 + s_2 nw$ to the points $(n, w, t)$. The fits were done by `gnuplot`'s implementation of the nonlinear least-squares (NLLS) Marquardt-Levenberg algorithm. The quotient $s_1/s_2$ corresponds to the $C$ from (2.2). Table 7 gives some possible values for $C$.

$C = 14$ is much smaller than we had initially expected. According to Table 2, with $C = 14$ and assuming $\frac{w}{n} = 30$ we have that 4-way merges are convenient with pivot relations up to weight 31, which is slightly above average whereas 5-way merges should be done with lighter than average (max. 21 entries) pivot relations. When assuming $\frac{w}{n} = 50$ the maxima are higher but below average also for 4-way merges.

[‡]The apparent arithmetical error is due to rounding all numbers to units of a million.

| exp. | matrix size | % | weight | % | col.w. | $w_{eff}$ | col.w. | Cray | SGI [a] | |
|---|---|---|---|---|---|---|---|---|---|---|
| A1.1 | 4 671K × 4 704K | −11 | 151.1M | +8 | 32.1 | 147.4M | 31.5 | 75h | 59d | 24d |
| A1.1.1 | 4 180K × 4 193K | −4 | 163.1M | +3 | 38.9 | 161.3M | 38.6 | 65h | 56d | 22d |
| A1.1.3 | 3 999K × 4 012K | −1 | 168.7M | +1 | 42.0 | 166.8M | 41.7 | 63h | 54d | 21d |
| A1.1.2 | 3 960K × 3 980K | −12 | 171.1M | +12 | 43.0 | 168.1M | 42.4 | 62h | 53d | 20d |
| A1.1.2.1 | 3 504K × 3 507K | | 191.3M | | 54.5 | 190.8M | 54.4 | 56h | 51d | 18d |
| B1.2 | 3 380K × 3 394K | −3 | 178.8M | +2 | 52.7 | 176.8M | 52.3 | 51h | 46d | 16d |
| B2 | 3 285K × 3 286K | | 182.1M | | 55.4 | 182.0M | 55.4 | 50h | 43d | 15d |

Table 6: RSA-140 matrices

[a] The second column gives timings from the $K = 128$ implementation.

| Block Lanczos implementation | $s_1$ | | $s_2$ | | $C$ | |
|---|---|---|---|---|---|---|
| vectorized Cray code with $K = 64$ | 1.84 | ±0.06 | 0.0499 | ±0.0014 | 37 | ±2 |
| SGI code with $K = 64$ | 0.86 | ±0.14 | 0.060 | ±0.003 | 14 | ±3 |
| improved SGI code with $K = 128$ [a] | 0.69 | ±0.08 | 0.0140 | ±0.0018 | 49 | ±12 |

Table 7: C values for different Block Lanczos implementations

[a] This version 'under development' by Montgomery is being optimized for cache usage rather than vectorization. It is being redesigned to allow parallelization, but we used only one processor.

Why then did the matrices, which were constructed by more or less brutally doing all possible 3-, 4- and 5-way merges[§], perform better than we would expect from looking at the figures in Table 3 and 2? It seems most merges were able to find a pivot relation with much smaller weight than average. Furthermore, we must consider that the inequalities (2.4) and (2.5) do not take account of the weight and size reduction obtained by discarding relation-sets which are made of more than `maxrels` relations. Some benefit also comes from the minimum spanning tree algorithm.

With $C = 49$ and $\frac{w}{n} = 30$, even above average 6-way merges can be beneficial.

*R211*

The following two tables give data concerning filter experiments with the special 211-digit number R211:= $(10^{211} - 1)/9$, which is a so-called "repunit", since all its digits are 1. It was factored on April 8, 1999. Five sites produced a total of 57.6M raw relations. 1.2M duplicates were removed during `mergelevel` 0 runs on the individual data. The experiment series A and B both started with the remaining 56.4M relations having 49.5M prime ideals of norm above 10M. This means that we had 6.9M more relations than prime ideals which seemed to be enough since we needed to reserve $\frac{23}{12}\pi(10\text{M}) = 1.3\text{M}$ more relations accounting for the small prime ideals. Unfortunately, the `mergelevel` 1 run on the complete data set revealed 9.4M duplicates. The remaining 47.0M relations plus 0.8M free relations were *less* than the number of prime ideals. However, we did not need to sieve further since we had an excess after removing the 26.5M singletons. The clique algorithm started hence with 21.3M relations having 18.6M prime ideals of norm larger than 10M, which is an excess of 2.8M. See Table 4.

Experiment series A gives the parameters and results of the `filter` runs that led to the matrix that was used to factor the number; it took 120 hours on the Cray. B shows a different approach, where we kept 1.1M more relations than for A after the pruning step, leaving more choice for merging.

[§] For A1.1.2.1, all possible merges up to prime ideal frequency 5, for prime ideals of norm larger than 8M, had been performed.

| experiment | mergelevel | filtmin | maxdiscard | maxrels | maxpass | sets | discarded | excess | not merged |
|---|---|---|---|---|---|---|---|---|---|
| A | 1 | 10M | keep | 1.7M | | 13.9M | 33 839K | 433K | - |
| A1 | 4(5) | 20M | 300K | 6.0 | 10 | 6.8M | 304K | 124K | 1 637K |
| A1.1 | 5(5-10) | 20M | 15K | 12.0 | 15 | 5.6M | 15K | 109K | 796K |
| A1.1.1[a] | 5(5-10) | 8M | 50K | 15.0 | 15 | 4.9M | n.a. | 63K | n.a. |
| B | 1 | 10M | keep | 2.8M | | 21.3M | 26 488K | 1 484K | - |
| B1 | 4(5) | 20M | 1 300K | 6.0 | 10 | 6.7M | 1 310K | 206K | 1 410K |
| B1.1 | 5(5-10) | 20M | 170K | 12.0 | 15 | 4.8M | 170K | 11K | 97K |
| B1.1.1 | 5(1-3) | 8M | 10K | 18.0 | 10 | 4.6M | 4K | 8K | 2 |
| B2 | 8 | 10M | 1 400K | 9.0 | 15 | 4.7M | 1 421K | 30K | 1 244K/925K |
| B3 | 8 | 10M | 1 400K | 10.0 | 15 | 4.5M | 1 423K | 64K | 918K/777K |

Table 8: R211 filter runs

[a]This run was done with the flag `regroup`, which splits up existing relation-sets and does merges from scratch, which leads to different relation-sets.

Both `mergelevel` 4 runs can actually be considered `mergelevel` 3 runs, since the maximum number of discards, `maxdiscard`, was reached before 4-way merges would have started.

| exp. | matrix size | % | weight | % | col.w. | $w_{eff}$ | col.w. | Cray | SGI |
|---|---|---|---|---|---|---|---|---|---|
| A1.1.1 | 4 820K × 4 896K | −0 | 234.2M | −5 | 47.8 | 221.2M | 45.88 | 118h - 97h 93h | 96d |
| B1.1 | 4 863K × 4 877K | −3 | 223.3M | +4 | 45.8 | 221.3M | 45.92 | 119h - 97h 95h | 97d |
| B2 | 4 723K × 4 754K | −2 | 231.9M | −0 | 48.8 | 228.2M | 49.10 | - 95h 93h 92h | 95d |
| B1.1.1 | 4 661K × 4 670K | −2 | 231.2M | +7 | 49.5 | 229.3M | 49.60 | 115h - 93h 91h | 95d |
| B3 | 4 503K × 4 569K | | 247.5M | | 54.2 | 239.0M | 53.06 | - 90h - - | - |

Table 9: R211 matrices

Experiment series B achieved smaller matrices than A. The reason must be the different `keep` values during the pruning stage. Experiment series A kicked out 7.4M relations with the clique algorithm whereas B kept all the excess relations, performed more merges and discarded more relations during the merge steps. We can conclude that for this data the best thing was to skip the clique algorithm. This is strongly connected to the fact that we barely had enough relations. Sieving any longer would surely have led to smaller matrices.

Matrix A1.1.1 performed better than matrix B1.1, which may seem counter-intuitive since B1.1 produced the smaller and lighter matrix. However, matrix A1.1.1 contained fewer rows (fewer prime ideals) than matrix B1.1 and due to the default truncation taking place in the Block Lanczos algorithm the effective A1.1.1 matrix was smaller in size and weight than the effective B1.1 matrix.

At B2 we also tried `mergelevel` 8 while having $m_{max} = 7$. `maxdiscard` was reached already at shrinkage pass 9 (with 15 possible passes) when the allowed weight increase was 5 original relations. The final matrix was larger than B1.1.1. We had chosen `maxrels` too low. It was 9, compared to 18 in B1.1.1. With `maxrels` 10 we achieved the desired reduction (B3).

*RSA-155*

The 155-digit number RSA-155 (512 bits!) was factored on August 22, 1999. A total of 130.8M relations were collected from 12 different sites. 6.1M relations were removed in individual `mergelevel` 0 runs. Another 39.2M duplicates where removed in a `mergelevel` 0 run on the whole amount of

data. All the experiments below started with the remaining 85.5M relations and its 0.2M free relations. Therefore, in contrast to the previous examples, the figures in the discarded column do not contain any duplicates. See Table 4 for details.

Matrix B2 was used for the factorization. It took 225 hours on the Cray.

| experiment | `mergelevel` | `filtmin` | `maxdiscard` | `maxrels` | `maxpass` | sets | discarded | excess | not merged |
|---|---|---|---|---|---|---|---|---|---|
| A | 1 | 10M | `keep` 1.7M | | | 19.1M | 66 593K | 385K | - |
| A1 | 5(1-3) | 10M | 370K | 11.0 | 12 | 7.1M | 370K | 15K | 67K |
| B | 1 | 10M | `keep` 2.0M | | | 20.2M | 65 531K | 684K | - |
| B1 | 8 | 10M | 600K | 9.0 | 15 | 6.9M | 603K | 81K | 1 611K/764K |
| B2 | 8 | 7M | 670K | 9.0 | 15 | 6.7M | 672K | 13K | 1 576K/716K |
| B3 | 8 | 7M | 670K | 10.0 | 15 | 7.1M | 366K | 317K | 1 432K/744K |
| B4 | 16 | 7M | 670K | 9.0 | 15 | 6.6M | 690K | −5K | 4 130K/694K |
| B5 | 16 | 7M | 670K | 10.0 | 15 | 6.8M | 482K | 193K | 3 797K/562K |
| B6 | 18 | 7M | 670K | 10.0 | 15 | 6.3M | 672K | n.a. | n.a. |
| C | 1 | 10M | `keep` 3.0M | | | 23.6M | 62 092K | 1 682K | - |
| C1 | 8 | 10M | 1 670K | 8.0 | 15 | 6.8M | 1 675K | 7K | 1 710K/698K |
| D | 1 | 10M | `keep` 5.0M | | | 30.3K | 55 402K | 3 677K | - |
| D1 | 8 | 10M | 3 670K | 7.0 | 15 | 7.1M | 3 698K | −20K | 2 118K/780K |

Table 10: RSA-155 filter runs

The experiments indicate that retaining more data (`keep` $\geq$ 3.0M) after the pruning stage did not help to reduce the size of the matrix.

Experiments B4 and D1 discarded too many relation-sets which is recognizable from the negative excess.

In B2 merging was stopped at shrinkage pass 11, while $m = 6$. Since there were still many unmerged ideals in B2, we tried to make the matrix smaller by increasing `maxrels` in B3 which allows also relation-sets with 10 relations, which were deleted in test B2. But even after this run many potential merge candidates remained unmerged, although `maxdiscard` was not reached. This indicates that the weight increase of the merges was considered too high and the merges were subsequently not executed. Next, we tried `mergelevel` 16, which is the maximum prime ideal frequency you can have a merge with for $m_{max} = 7$. Some reduction was achieved (B4 and B5). Finally, we took $m_{max} = 8$ together with `mergelevel` 18 and `maxrels` 10. `maxdiscard` was reached during shrinkage pass 14, when $m = m_{max}$.

| exp. | matrix size | % | weight | % | col.w. | $w_{eff}$ | col.w. | Cray |
|---|---|---|---|---|---|---|---|---|
| B2 | 6 699K × 6 711K | −5 | 417.1M | +7 | 62.2 | 415.5M | 62.0 | 218h |
| B6 | 6 342K × 6 354K | | 445.3M | | 70.1 | 443.4M | 69.9 | 213h |

Table 11: RSA-155 matrices

Matrix B6 is 5% smaller than B2 but also 7% heavier. With $C = 14$ we can expect to save $1 - \frac{14 \cdot 6.342^2 + 6.342 \cdot 445.3}{14 \cdot 6.699^2 + 6.699 \cdot 417.1} \approx 1\%$ running time, which is too small a gain to accept the weight increase, whereas with $C = 37$ or $C = 49$ we may save 3% or 4%, respectively. The effective runs on the Cray ($C = 37$) indicate a saving of 2%.

## 5. Conclusions

We extended our previous `filter` program to allow higher-way merges and proved theoretically and practically that we can reduce Block Lanczos running time by performing higher-way merges. We determined limits for the weight of pivot columns.

During a merge, instead of merging by pivoting we calculate a minimum spanning tree in order to assure minimum weight increase.

A denser matrix allows for more weight increase during a merge than a lighter one: this means we can merge with denser pivot columns. Therefore we do the light merges before the heavier ones.

We determined the ratio between the two terms characterizing the running time of Block Lanczos for different implementations. To which extent we can profit from higher-way merges depends on this ratio. We saw values ranging from 14 to 49. With the help of this constants we can estimate the running time of a matrix, given the running time of another matrix.

Collecting more data than necessary is advisable. The clique algorithm enables us to get rid of excess data quickly and in a sensible way. It is a useful tool when having abundant excess.

# References


1. Hendrik Boender. *Factoring Large Integers with the Quadratic Sieve*. PhD thesis, Rijksuniversiteit Leiden, 1997.
2. Joe P. Buhler, Hendrik W. Lenstra, Jr., and Carl Pomerance. Factoring integers with the number field sieve. In Arjen K. Lenstra and Hendrik W. Lenstra, Jr., editors, *The development of the number field sieve*, number 1554 in Lecture Notes in Mathematics, pages 50–94. Springer-Verlag, 1993.
3. Stefania Cavallar, Bruce Dodson, Arjen K. Lenstra, Paul Leyland, Walter Lioen, Peter L. Montgomery, Herman te Riele, and Paul Zimmermann. 211-digit SNFS factorization. Available from ftp://ftp.cwi.nl/pub/herman/NFSrecords/SNFS-211, April 1999.
4. Stefania Cavallar, Bruce Dodson, Arjen K. Lenstra, Paul Leyland, Walter Lioen, Peter L. Montgomery, Brian Murphy, Herman te Riele, and Paul Zimmermann. Factorization of RSA-140 using the number field sieve. In Kwok Yan Lam, Eiji Okamoto, and Chaoping Xing, editors, *Advances in Cryptology - Asiacrypt '99*, volume 1716 of *Lecture Notes in Computer Science*, pages 195–207. Springer-Verlag, 1999.
5. Stefania Cavallar, Bruce Dodson, Arjen K. Lenstra, Walter Lioen, Peter L. Montgomery, Brian Murphy, Herman te Riele, Karen Aardal, Jeff Gilchrist, Gérard Guillerm, Paul Leyland, Joël Marchand, François Morain, Alec Muffett, Chris Putnam, Craig Putnam, and Paul Zimmermann. Factorization of a 512-bit RSA modulus. Submitted to Eurocrypt 2000.
6. James Cowie, Bruce Dodson, R.-Marije Elkenbracht-Huizing, Arjen K. Lenstra, Peter L. Montgomery, and Jörg Zayer. A world wide number field sieve factoring record: on to 512 bits. In Kwangjo Kim and Tsutomu Matsumoto, editors, *Advances in Cryptology - Asiacrypt '96*, volume 1163 of *Lecture Notes in Computer Science*, pages 382–394. Springer-Verlag, 1996.
7. Thomas F. Denny. Solving large sparse systems of linear equations over finite prime fields. Transparencies of a lecture of the Cryptography Group at CWI, May 1995.
8. Reina-Marije Elkenbracht-Huizing. An implementation of the number field sieve. *Experimental Mathematics*, 5(3):231–253, 1996.
9. Ronald L. Graham and Pavol Hell. On the history of the minimum spanning tree problem. *Annals of the History of Computing*, 7(1):43–57, January 1985.
10. Jürgen Neukirch. *Algebraische Zahlentheorie*. Springer-Verlag, 1992.

11. Donald E. Knuth. *The Stanford GraphBase: A Platform for Combinatorial computing*. Addison-Wesley, 1993.

12. Donald E. Knuth. *Sorting and Searching*, volume 3 of *The Art of Computer Programming*. Addison-Wesley, second edition, 1998.

13. Brian A. LaMacchia and Andrew M. Odlyzko. Solving large sparse linear systems over finite fields. In A. J. Menezes and S. A. Vanstone, editors, *Advances in Cryptology - Crypto '90*, volume 537 of *Lecture Notes in Computer Science*, pages 109–133. Springer-Verlag, 1991.

14. Serge Lang. *Algebraic Number Theory*. Springer, 1986.

15. Peter L. Montgomery. Square roots of products of algebraic numbers. In W. Gautschi, editor, *Mathematics of Computation 1943–1993: a Half-Century of Computational Mathematics*, volume 48 of *Proceedings of Symposia in Applied Mathematics*, pages 567–571. American Mathematical Society, 1994.

16. Peter L. Montgomery. A block Lanczos algorithm for finding dependencies over GF(2). In Louis C. Guillou and Jean-Jacques Quisquater, editors, *Advances in Cryptology - Eurocrypt '95*, volume 921 of *Lecture Notes in Computer Science*, pages 106–120. Springer-Verlag, 1995.

17. Phong Nguyen. A Montgomery-like square root for the number field sieve. In J. P. Buhler, editor, *Algorithmic Number Theory - ANTS-III*, volume 1423 of *Lecture Notes in Computer Science*, pages 151–168. Springer-Verlag, 1998.

18. J. M. Pollard. The lattice sieve. In Arjen K. Lenstra and Hendrik W. Lenstra, Jr., editors, *The development of the number field sieve*, number 1554 in Lecture Notes in Mathematics, pages 43–49. Springer-Verlag, 1993.

19. Carl Pomerance and J. W. Smith. Reduction of huge, sparse matrices over finite fields via created catastrophes. *Experimental Mathematics*, 1(2):89–94, 1992.