Improvement in the filtering step of integer factorization algorithms

Cyril Bouvier

Improvement in the filtering step of integer factorization algorithms

Cyril BOUVIER

CARAMEL project-team, LORIA

Université de Lorraine, CNRS, INRIA, France

Cyril.Bouvier@loria.fr

Abstract

The security of the RSA public-key cryptosystem is based on the difficulty of factoring large integers which have exactly two prime factors of approximately the same size.

Most factoring algorithms aiming at RSA moduli (Quadratic Sieve and Number Field Sieve), can be described in 3 main steps: data collection, filtering and linear algebra. The goal of the filtering step is to generate a small, sparse matrix over $\mathrm{GF}(2)$, for which one will compute the kernel during the linear algebra step. The filtering step is mainly a structured Gaussian elimination (SGE) over $\mathrm{GF}(2)$. For the current factorization records, billions of data are collected in the first step and have to be processed in the filtering step. One part of the filtering step is to remove heavy rows of the matrix. The choice of the weight function to sort heavy rows is critical in order to obtain the smallest matrix possible. In this paper, several weight functions are studied in order to determine which one is more suited in the context of factorization algorithms.

Keywords

factorization, Number Field Sieve, filtering, structured Gaussian elimination, sparse linear system

I. Introduction

The difficulty of the integer factorization problem is the key of the security of the RSA cryptosystem. The problem is to factor large integers with exactly two prime factors of approximately the same size. Most of the integer factorization algorithms aimed at RSA moduli use the *relation collection method*. The filtering step is a common step of all such algorithms, like the Quadratic

Sieve (QS), the Multiple Polynomial Quadratic Sieve (MPQS), the Special and General Number Field Sieve (SNFS and GNFS). The main goal of the filtering step is to reduce the size of a large sparse matrix over $\mathrm{GF}(2)$, constructed from the data collected previously by the algorithm, in order to being able to compute its kernel.

The filtering step of integer factorization algorithms is mostly a structured Gaussian elimination (SGE) [1], [2] over $\mathrm{GF}(2)$. A complete description of the state of the art can be found in Cavallar's thesis [3]. Some integer factorization tools that contain code for filtering are CADO-NFS [4], Msieve [5] and GGNFS [6].

During the filtering step, the "heaviest" rows are removed. The definition of heavy depends on the choice of a weight function. In this paper, we propose new weight functions for the filtering step and compare them on real-case factorizations, exhibiting better weight functions than the ones used in current integer factorization tools.

A. Overview of the NFS factorization method

In order to understand the goal of the filtering step, we give a description of the Number Field Sieve (NFS) algorithm. For others algorithms based on relation collection, there are some differences but the filtering step remains the same. For a more detailed description of NFS, see [7].

The goal of the algorithm is to find an equality between two squares modulo the number n that one wants to factor. The algorithm looks for two integers x and y such that $x^2 \equiv y^2 \pmod{n}$.

The NFS algorithm can be described in 3 main steps. A first step (data collection) computes lots of relations of the form

$$\prod_i p_i^{e_i} = \prod_j q_j^{f_j},$$

where the p_i and the q_j represent prime ideals in two distinct number fields. In the following, we will not make the distinction between prime ideals of the two number fields and call them *ideals*.

The goal is to find a subset of all the relations whose product is a square on each member of the equality. This can be rephrased as: for each ideal p the sum of the exponents of p in all chosen relations must be even.

This can be translated in a linear algebra problem. If one sees the relations as rows of a matrix where each column corresponds to an ideal, the coefficients of the matrix are the exponents of

the ideals in the relations. As we are looking for even exponents, one can reduce the matrix over $\mathrm{GF}(2)$. Finding a linear combination of relations such as every exponent is even is equivalent to computing the (left) kernel of the matrix.

The *excess* is defined as the difference between the number of rows (relations) and the number of columns (ideals). The excess should always be positive in order for a nontrivial kernel to surely exist (in practice one tries to have an excess around 100 before linear algebra to have enough vectors in the kernel).

During the filtering step, the matrix is reduced in order to accelerate the linear algebra step, which consists in computing the (left) kernel of the matrix and constructing the solution from vectors of this kernel.

By construction, the matrices at the beginning of the filtering step are sparse (about 20 non-zero coefficients per row). In the linear algebra step, algorithms dedicated to sparse matrices are used in order to deal with matrices from millions to hundreds of millions of rows and columns. The cost of those algorithms depends on the matrix dimensions and on its density. So at the end of the filtering step one should try to keep the matrix as sparse as possible (around 100-150 non-zero coefficients per rows).

B. Usefulness of the filtering step

The goal of the filtering step is to reduce the size of the matrix that is going to be solved in the linear algebra step. In order to realize in which proportion the reduction is made, this is illustrated with data from record factorizations with GNFS (NFS for general number) and SNFS (NFS for special number of the form $r^s \pm e$, where r and e are small).

The current GNFS record factorization is RSA-768 [8], an integer of 768 bits (232 digits). At the beginning of the filtering step, once duplicate relations were removed, the matrix had about 47 billion rows and 35 billion columns. After the first part of the filtering step (see the following section for more detail), the matrix had about 2.5 billion rows and 1.7 billions columns. At the end of the filtering step, the matrix used in the linear algebra had about 193 million rows and columns. So the filtering step reduced the size of the matrix by more than 99% (it was reduced by almost 95% after the first part).

The current record factorization with SNFS is $2^{1061}-1$ [9], an integer of 320 digits. The initial matrix had about 671 million rows, then 282 million rows after the first part of the filtering step

and the final matrix had 90 million rows at the beginning of the linear algebra step. So the matrix was reduced by 87% during the filtering step.

After describing in details the filtering step in section II, we will propose, in Section III, new weight functions for the filtering step and present experiments on three real-case factorizations in Section IV. The main contribution of this article is to present several weight functions, including ones previously described in the literature, in an unified formalism and to compare them on real-case factorizations. In this article, the computation were done with the development version of the CADO-NFS software.

II. Description of the filtering step

The goal of the filtering step is to construct a matrix over $\mathrm{GF}(2)$ which is the smallest and the sparsest possible. Then, in the linear algebra step, one finds the kernel of that matrix. The following description of the state of the art for the filtering step can be found in [3].

The filtering step is classically divided in 3 parts. The first part removes duplicate relations. It will not be studied in this article, as the set of input relations will always be a set of unique relations. The second part, called *purge*, removes singletons (ideals appearing only once, or equivalently columns with only one non-zero coefficient) and reduces the size of the matrix thanks to the *clique removal* algorithm. The last part, called *merge*, is a beginning of Gaussian elimination that reduces the size of the matrix by adding rows (and thus making the matrix less sparse).

The purge algorithm is similar to a structured Gaussian elimination (SGE) [1], [2]. SGE begins by removing columns without non-zero coefficients, which do not exist in our case by construction of the matrix. Then, SGE removes columns with exactly one non-zero coefficient and heavy rows, which corresponds to the purge algorithm. The difference between SGE and the filtering step of NFS is that purge is followed by the algorithm merge, so the best weight function adapted to these two cases may be different.

Definition 1. The weight of a row or a column of the matrix is the number of non-zero coefficients of the row or the column. By extension, one can talk about the weight of a relation or an ideal. The total weight of a matrix is the number of non-zero coefficients in the matrix.

A. *Purge*

A singleton is a column of the matrix of weight 1 (with only one non-zero coefficient). The corresponding row and column in the matrix can be deleted without loss of any information on the kernel. The first part of purge is to remove all singletons and the correspondings rows and columns. When one removes a singleton, one removes a row and a column, so the excess does not change (except in the very rare case where a relation contains more than one ideal of weight 1).

Example 1. Let us consider the matrix

$$M_0 = \begin{pmatrix} 0 & 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}.$$

One can see that the first column is a singleton, so one can remove the first column and the second row. By doing this, one creates a singleton (the second column of M_0) and so one can remove the second column and the first row. The remaining matrix is

$$M_1 = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

with no more singleton and excess 2.

Removing excess rows is possible while one is sure that the kernel is still of positive dimension, which is always true if there are more rows than columns in the matrix. When one removes

a row, one looses some information on the kernel, but as only a few vectors of the kernel are necessary, one can remove rows until the target excess of 100-150.

In particular, if there is a column of weight 2 and one of the rows corresponding to a non-zero coefficient is removed, then the column becomes a singleton and will be removed during the next singleton phase (as well as the other row corresponding to the other non-zero coefficient). Thus two rows and a column of the matrix will have been deleted, reducing the excess by 1.

Definition 2. Let us consider the graph where the nodes are the rows of the matrix and the edges are the columns of weight 2, connecting the two corresponding rows / nodes. A "clique" is a connected component of this graph.

Remark 1. Even though the component is not a clique in the common sense of graph theory, in the context of filtering it is traditional to call it clique [3], thus we keep that terminology.

Example 2. Let us consider the matrix M_1 at the end of Example 1. One can see that there are 3 cliques: one containing 3 rows (the first, second and third rows), one containing two rows (the fourth and the fifth) and one containing one row. The graph obtained from this matrix is presented in Figure 1. Removing one of these three cliques reduces the excess by 1. Removing the clique with three rows connects the two other cliques *via* the last column.

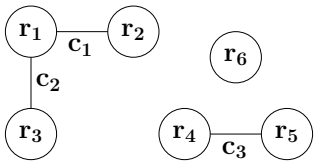

Figure 1. A graph with 3 cliques, constructed from matrix M_1 of Example 1 (r_i corresponds to the ith row and c_j to the jth column).

The stage of purge that removes excess rows is called *clique removal* (or *pruning* as in [3]), as it will, once all the singletons have been removed, compute all the cliques of the matrix and remove some of them. As one can remove any row, one can choose the cliques to remove in order to obtain the smallest, sparsest matrix possible. On the clique removal stage, a weight is given to each clique and the heaviest ones (with respect to that weight function) are deleted. The choice of the weight function associated to a clique is crucial to obtain a small and sparse

matrix. Every algorithm removing rows can be written as a clique removal algorithm with an appropriate weight function, as every row belongs in a clique (possibly a clique with only one row).

B. Merge

The merge stage is a beginning of Gaussian elimination, as one combines some rows and deletes some columns to reduce the size of the matrix. In this stage, the total weight of the matrix usually increases.

In order to understand the idea behind the merge algorithm, let us begin with two examples. Let p be an ideal representing a column of weight 2 and let r_1 and r_2 be the two rows corresponding to those two non-zero coefficients. If one replaces in the matrix the row r_1 by r_1+r_2 and deletes the row r_2, then the column corresponding to the ideal p has no non-zero coefficient anymore and can be deleted. The idea is that if the relation r_1 (resp. r_2) is used in the kernel, as the exponent of the ideal p must be even and the only other relation with an odd exponent of p is r_2 (resp. r_1), it must also be used. This operation is called a 2-merge. A 2-merge removes one row and one column (the one corresponding to p) of the matrix, thus leaving the excess inchanged.

Now, if the column corresponding to the ideal p has weight 3 and r_1, r_2 and r_3 are the rows corresponding to those 3 non-zero coefficients, then, as the previous case, one can delete the column corresponding to p, if one replaces r_1 by r_1+r_3, r_2 by r_2+r_3 and deletes the row r_3. This operation is called a 3-merge. A 3-merge removes one row (in this case r_3) and one column (the one corresponding to p) of the matrix, thus leaving the excess inchanged. An important difference with a 2-merge is that there is a choice to be made because there is more than one way of combining the 3 relations. Indeed, one can also choose to add r_2 to both r_1 and r_3 and to delete r_2, or to add r_1 to both r_2 and r_3 and to delete r_1.

Example 3. For this example, let us consider the matrix M_0 of Example 1, while forgetting that the first column is a singleton. The second column has weight 2, so one can do a 2-merge. If one adds the second row to the first row, then column 2 becomes a singleton and the second column and the second row can be deleted.

An example of 3-merge comes from the third column which has weight 3. One can see that, in order to minimize the total weight of the matrix, one should add row 4 to row 1 and row 3.

The following definition generalizes those examples to a k-merge, for any integer $k \geq 2$.

Definition 3. Let $k \geq 2$ be a positive interger. Let p be an ideal representing a column of weight k and let $r_1, \ldots, r_k$ be the k rows corresponding to these k non-zero coefficients. A k-merge is a way of performing successive rows additions of the form $r_i \leftarrow r_i + r_j$, such that the column corresponding to p becomes a singleton that is deleted (as well as the only remaining row containing a non-zero coefficient in the column corresponding to p).

A k-merge removes one row and one column, so a k-merge does not change the excess of the matrix (except in the very rare case where the excess can increase if doing a k-merge allows one to delete more than one column).

A 2-merge always reduces the total weight of the matrix. If the two relations r_1 and r_2 had respectively w_1 and w_2 non-zero coefficients, then the combined row $r_1 + r_2$ will have at most $w_1 + w_2 - 2$ non-zero coefficients. So every 2-merge reduces the number of rows by 1 and the total weight by at least 2. For a k-merge ($k > 2$), the total weight of the matrix will increase in general. In order to control the increase of the total weight of the matrix, one will choose the way of combining the k rows that minimize this increase. In order to do that, one can see the rows as nodes of a graph where all nodes are connected with weights corresponding to the weight of the row resulting of the addition of the two nodes. Finding the best way to combine the k rows is equivalent to find a spanning tree of minimal weight in this graph (an example is given in Figure 2).

	r_2	r_4	r_5	r_7	r_8
r_1	4	2	4	2	3
r_2		4	2	4	3
r_4			2	2	1
r_5				2	1
r_7					1

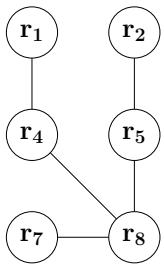

Figure 2. Example of a 6-merge from the sixth column of matrix M_0 of Example 1. The table on the left gives the weight of the sum of r_i and r_j, $i < j$. The figure on the right shows the minimal spanning tree associated to these weights; the edges link two nodes that are going to be added in the 6-merge.

Markovitz algorithm [10] is used to choose the merges that are going to be made. The merges

are in a heap sorted by the weight increase corresponding to that merge.

The merge algorithm always reduces the number of rows but usually increases the total weight of the matrix. So one has to stop while the matrix is still sparse (e.g. 100 non-zero coefficients on average per row).

The time of the linear algebra step is proportional to the product of the number of rows and the total weight of the final matrix (with the Wiedemann [11] or Lanczos [12] algorithms). To sum up, the problem of the filtering step is: given as input a set of unique relations, a target excess and a target average weight for the final matrix, what weight function should be used in the clique removal algorithm in order for the filtering step to produce the smallest matrix possible?

III. Several weight functions for clique removal

What one wants to improve is the size of the matrix at the end of the filtering step (for a given target excess and average weight), it means that the weight function used in the clique removal stage has to produce the smallest matrix *at the end of merge*, and not necessarily *at the end of purge*. So a good weight function should remove lots of heavy relations but also reduce the total weight of the matrix and create many 2-merges that will reduce the total weight of the matrix during merge.

Before defining the weight functions that are going to be studied, some notations are needed. At the beginning of the clique removal stage, it is supposed that there is no more singleton. The matrix has N rows and the total weight is denoted Z (number of non-zero coefficients). Let c be a clique. The number of relations contained in the clique is denoted $n(c)$. By definition of a clique, the $n(c)$ relations of the clique c are linked by $n(c)-1$ columns of weight 2. The weight associated to the clique is denoted $w(c)$. If p is an ideal, then $z(p)$ is the weight of the column that corresponds to the ideal p and $z_c(p)$ is the number of occurrences of the ideal in the clique c.

One wants to delete ideals of small weight in order to obtain more singletons and 2-merges, so the weight associated to a column appearing in the clique should be a decreasing function of $z(p)$. Also, due to the nature of the data collecting step, large ideals have very few occurrences, so the weight associated to a column appearing in the clique could also be a increasing function of the "size" of p.

An experiment on a 124-digit number showed that ideals that disappear during purge have an initial weight less than or equal to about 15. As small ideals have many occurrences, they are not usually considered during the purge stage. Experiments on data used for the factorization of RSA-155 and RSA-704 showed that only very few cliques (15 at most over more than 20 million) contain a large ideal p such that $z_c(p) > 1$ and $z(p) \neq 2$. It means that except the ideals of weight 2, there are almost never other large ideals considered during purge that appear more than once in a clique. So in the following we can consider that for any ideal p such that $z(p) \geq 3$, $z_c(p) = 1$ if the ideal p belongs to a relation of the clique and $z_c(p) = 0$ otherwise. In the case where $z(p) = 2$, we know, by definition of a clique, that $z_c(p) = z(p) = 2$ if p belongs to a relation of the clique, and $z_c(p) = 0$ otherwise.

We propose to consider weight functions of the following form: $w(c) = \Omega(c) + \epsilon(c)$. The function Ω measures the contribution of the ideals of weight greater or equal to 3. This function will favor removing cliques will many ideals, or cliques which contain ideals that can create singletons or 2-merges. The following formulae for the Ω function were tried (as said above, one can consider that $z_c(p) = 0$ or 1 for $z(p) \geq 3$):

$$\Omega_0(c) = \sum_{z(p)\geq 3} z_c(p), \quad \Omega_1(c) = \sum_{z(p)\geq 3} z_c(p)\log(p),$$

$$\Omega_2(c) = \sum_{z(p)\geq 3} \frac{z_c(p)}{2^{z(p)-1}}, \quad \Omega_3(c) = \sum_{z(p)\geq 3} \frac{z_c(p)}{z(p)} \quad \text{and}$$

$$\Omega_4(c) = \sum_{z(p)\geq 3} \frac{z_c(p)}{\log(z(p)+1)}.$$

The case where the Ω function is zero is treated in the additional weight functions described below. The function ϵ measures the contribution of the ideals of weight 2, that is the ideals that link the relations of the clique together. So the function ϵ is in fact a function depending of $n(c)$ and will try to remove cliques with many relations in order to reduce the size of the matrix. The following formulae for the ϵ function were tried: $\epsilon_0(c) = 0$, $\epsilon_1(c) = \frac{n(c)}{2}$, $\epsilon_2(c) = n(c)$ and $\epsilon_3(c) = 2n(c)$.

With these 5 formulae for Ω and 4 formulae for ϵ, one can construct 20 different weight functions $w_{xy}(c) = \Omega_x(c) + \epsilon_y(c)$. For example,

$$w_{21}(c) = \Omega_2(c) + \epsilon_1(c) = \sum_{z(p)\geq 3} \frac{z_c(p)}{2^{z(p)-1}} + \frac{n(c)}{2}.$$

In Cavallar's thesis, a weight function is proposed by the author [3]:

> "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."

In CADO-NFS, within purge, the distinction between relations and free relations is not made, since the free relations only represent a small percentage of the total amount of relations. So the weight function proposed by Cavallar is almost the same as the weight function w_{21} defined above (up to a factor 2).

In addition to the 20 weight functions constructed above, 5 others were tried:

1) The weight of the clique c is the weight of the remaining relation once all the 2-merges possible inside the clique have been made. The relations of the clique are linked with columns of weight 2 that can be used to perform 2-merges. Once all the 2-merges are done, it remains only one relation, its weight is used as the weight of the clique. The weight of the remaining relation is exactly the number of ideals that appear an odd number of times in the clique,
$$w_{50}(c) = \sum_{z(p)\geq 3} \frac{1-(-1)^{z_c(p)}}{2}.$$
This function is similar to w_{00}, if one consider that $z_c(p) = 0$ or 1 when $z(p) \geq 3$.
2) The weight of the clique c is the number of relations in the clique, $w_{51}(c) = \epsilon_2(c) = n(c)$. It corresponds to the case where the Ω function is zero. This is, up to a constant, the weight function used in CADO-NFS 1.1.
3) The weight of the clique c is the ratio of the number of relations in the clique and the total number of relations, plus the ratio of the number of non-zero coefficients (in the ideals of weight stricly greater than 2) of the clique and the total weight of the matrix, $w_{52}(c) = \frac{n(c)}{N} + \frac{\Omega_0(c)}{Z}$. The expression of this weight function is inspired from the derivative of the product $N \times Z$, which is what we are trying to minimize.
4) This weight function is the same as the previous one, except that the ideals of weight 2 are taken into account, $w_{53}(c) = \frac{n(c)}{N} + \frac{\Omega_0(c)+2(n(c)-1)}{Z}$.
5) This is the weight function used in Msieve: $w_{54}(c) = n(c) + \sum_{3\leq z(p)\leq 15} \frac{z_c(p)}{2^{z(p)-2}}$. This weight function is close (up to a factor 2) to w_{21} similar to Cavallar's, except the sum is truncated.

In total, 25 weight functions were tested, the results of their comparison is given in the next section.

IV. Results

The 25 weight functions described in the previous section have been implemented in the development version of CADO-NFS and have been tested on data from three different real-case factorizations: RSA-155, B200 and RSA-704. The original data were used for B200 and RSA-704, and the data for RSA-155 were recomputed with CADO-NFS.

The purge algorithm used in these computations interleaves 50 stages of singleton removal and 50 stages of clique removal. On each singleton removal stage all the singletons are removed from the matrix. On each clique removal stage, the excess is decreased by one fiftieth of the difference beetwen the initial excess and the desired final excess. This strategy is used in the development version of CADO-NFS.

As CADO-NFS 1.1 and Msieve have different methods of combining the singleton and clique removal stages, the implementation in the development version of CADO-NFS of their weight functions (respectively w_{51} and w_{54}) allows us to have a fair comparison where only the weight functions differ (and not the interleaving of the singleton removal and clique removal stages).

In all the following experiments, the target final excess is 160. The target average weight of a row depends on the integer being factored.

The filtering step (purge and merge) was run for the 25 weight functions without any difference in the other parameters. As the excess and the average weight of a row are fixed and the same for all the weight functions, the smaller the size of the matrix is at the end of merge, the better the weight function is. In this section, only results for the three best weight functions, the worst one, the one corresponding to Cavallar's thesis (w_{21}) and the ones used in CADO-NFS 1.1 (w_{51}) and Msieve (w_{54}) are presented. The complete data are shown in Appendix.

A. *RSA*-155

RSA-155 is a 155-digit, 512-bit integer factored in 1999 [13] using the NFS algorithm. It was part of the RSA challenge.

Before beginning the purge and merge algorithms, there were 91.3 million initial unique relations. After the first pass (over 50) of singleton removal, 54.9 million relations remained

with an excess of 7.34 million (13.4%). During merge, only k-merges with k from 2 to 30 were considered. At the end of merge, the average weight of a row (i.e., average number of non-zero coefficients per row) was $Z/N = 100$.

Table I gives a summary of the results, the complete data are shown in Table VI in Appendix.

Table I
RESULTS FOR RSA-155, SEE TABLE VI FOR COMPLETE DATA

	After purge		After merge	
	N	$Z \times N$	N	$Z \times N$
(best) 30	22304109	9.99e+15	**6977331**	4.87e+15
21	**21768640**	9.50e+15	7045617	4.96e+15
54	**21768640**	9.50e+15	7045617	4.96e+15
(Cavallar) 21	**21768640**	9.50e+15	7045617	4.96e+15
(Msieve) 54	**21768640**	9.50e+15	7045617	4.96e+15
(CADO-NFS 1.1) 51	22155372	9.83e+15	7285330	5.31e+15
(worst) 51	22155372	9.83e+15	7285330	5.31e+15

B. B200

B200 is the 200th Bernouilli number. In the factorization of the numerator of B200, a 204-digit (677-bit) composite integer remains unfactored until August 2012 [14].

Before beginning the purge and merge algorithms, there were 702 million initial unique relations. After the first pass (over 50) of singleton removal, 452 million relations remained with an excess of 67.3 million (14.9%). During merge, only k-merges with k from 2 to 30 were considered. At the end of merge, the average weight of a row (i.e., average number of non-zero coefficients per row) was $Z/N = 120$.

Table II gives a summary of the results, the complete data are shown in Table VII in Appendix.

C. RSA-704

RSA-704 is a 212-digit, 704-bit integer factored in July 2012 using CADO-NFS [15]. It was part of the RSA challenge.

Before beginning the purge and merge algorithms, there were 833 million initial unique relations. After the first pass (over 50) of singleton removal, 650 million relations remained

Table II

RESULTS FOR B200, SEE TABLE VII FOR COMPLETE DATA

	After purge		After merge	
	N	$Z \times N$	N	$Z \times N$
(best) 30	156736538	5.41e+17	**44359520**	2.36e+17
40	155951650	5.36e+17	44823273	2.41e+17
41	152683226	5.13e+17	44979411	2.43e+17
(Cavallar) 21	**150424785**	4.98e+17	45029461	2.43e+17
(Msieve) 54	**150424785**	4.98e+17	45029461	2.43e+17
(CADO-NFS 1.1) 51	154025018	5.21e+17	47029434	2.65e+17
(worst) 51	154025018	5.21e+17	47029434	2.65e+17

with an excess of 98.5 million (15.2%). During merge, only k-merges with k from 2 to 50 were considered. At the end of merge, the average weight of a row (i.e., average number of non-zero coefficients per row) wass $Z/N = 173$.

Table III gives a summary of the results, the complete data are shown in Table VIII in Appendix.

Table III

RESULTS FOR RSA-704, SEE TABLE VIII FOR COMPLETE DATA

	After purge		After merge	
	N	$Z \times N$	N	$Z \times N$
(best) 30	316478268	2.24e+18	**83566005**	1.21e+18
40	315962066	2.23e+18	84496548	1.24e+18
20	335993975	2.52e+18	84654737	1.24e+18
(Cavallar) 21	**305215466**	2.08e+18	84776712	1.24e+18
(Msieve) 54	**305215466**	2.08e+18	84776712	1.24e+18
(CADO-NFS 1.1) 51	307415189	2.11e+18	87759201	1.33e+18
(worst) 51	307415189	2.11e+18	87759201	1.33e+18

D. Interpretation of the results

The only difference between the two weight functions w_{21} and w_{54} is the fact that w_{54} does not take into account the ideals of weight greater than 15. On these three examples, they gave

the same results. This illustrates that in the case of w_{21}, the contribution of ideals of large weight is negligible.

The matrices produced with w_{00} and w_{50} are almost the same, confirming the fact that almost always, if a ideal p of weight $z(p) \geq 3$ appears in a clique, then it appear only once.

As one can see the best weight function w_{30} for the whole filtering step (purge and merge) is not necessarily the one that produces the smallest matrix at the end of purge ($w_{21} \simeq w_{54}$).

If one compares the product $Z \times N$ at the end of merge (which gives the time spent in the linear algebra step), one can see that w_{30} decreases that product by 2% compared to w_{21} and w_{54} (Msieve-Cavallar) and 9% compared to w_{51} (CADO-NFS 1.1), for RSA-155. For B200, the weight function w_{30} is again the best weight function and improves the product $Z \times N$ at the end of merge by 3% compared to Msieve-Cavallar and 11% compared to CADO-NFS 1.1. For RSA-704, the weight function w_{30} is still the best weight function and improves the product $Z \times N$ at the end of merge by 3% compared to Msieve-Cavallar and 9% compared to CADO-NFS 1.1.

In all cases, the weight function w_{30} is the best among the 25 that were tried. The weight functions that have no contribution from the ideals of weight 2 seem to be better (e.g. w_{30}, w_{20} and w_{40}). This means that the number of relations in the clique should not be taken into account when computing the weight of the clique.

Table IV shows that the bigger the initial excess is, the larger the difference between w_{30} and Msieve-Cavallar or CADO-NFS 1.1 is, at the end of the filtering step. This shows that the choice of deleted cliques made with w_{30} is better, as the more choice (i.e. excess) there is, the better w_{30} is.

Table IV

FOR RSA-155, GAIN FROM w_{30} OVER MSIEVE-CAVALLAR (w_{21}) AND CADO-NFS 1.1 (w_{51}), IN TERM OF $Z \times N$, DEPENDING ON THE EXCESS AFTER THE FIRST SINGLETON PASS

excess	2.25%	6.19%	9.88%	13.4%	18.0%
w_{30}	7.48e+15	6.21e+15	5.43e+15	4.87e+15	4.28e+15
$\frac{w_{21}}{w_{30}}$	1.004	1.010	1.015	1.020	1.026
$\frac{w_{51}}{w_{30}}$	1.014	1.038	1.059	1.090	1.106

In order to try to explain why weight functions with no contribution from ideals of weight 2 are better, some data on the number of relations in a clique and on the number of 2,3,4,5-merges

for the best and the worst weight functions, w_{30} and w_{51} (CADO-NFS 1.1), were computed and are shown in Table V.

Table V

FOR RSA-155, DATA ON THE SIZE OF THE CLIQUES AND THE NUMBER OF k-MERGES FOR THE BEST AND THE WORST WEIGHT FUNCTIONS

	Number of cliques c with $n(c)$ relations						Number of k-merges		
	At pass 1		At pass 25		At last pass		Beginning of merge		
	$5 \leq n(c) < 10$	$n(c) \geq 10$	$5 \leq n(c) < 10$	$n(c) \geq 10$	$5 \leq n(c) < 10$	$n(c) \geq 10$	$k = 2$	$k = 3$	$k = 4$
w_{30}	837758	73194	232112	92	136836	100	7439747	3789512	2241202
w_{51}	837758	73194	25402	6	23636	14	5902092	4635965	2562140

Remember that the weight function w_{51} removes cliques with the largest $n(c)$ whereas w_{30} has no contribution from the number of relations in the clique. Data in Table V show that even if the weight function does not take into account the number of relations in the cliques, the cliques with many relations are deleted. The major difference between these two weight functions is the number of possible 2-merges at the beginning of the merge algorithm. The gain in term of numbers of relations at the end of purge by w_{51} is not worth the loss in term of the number of 2-merges available at the beginning of merge.

A clique with a large number of relations will anyway be reduced in one single row during merge while decreasing the total weight of the matrix. So it is more important, during the clique removal, to create many 2-merges, or at least k-merges with k small. This explains why the best weight functions are the ones with no contribution from the number of relations in the cliques.

V. CONCLUSION

In this article, we presented several weight functions, including ones previously described in the literature, in an unified formalism. Then, we compared them on three real-case factorization with the same parameters and where only the weight function differs.

These experiments showed that the best weight functions are the ones with no contribution from the ideals of weight 2 (or equivalently from the number of relations in the clique).

We also showed that the best strategy for the purge algorithm only is not necessarily the best one for the whole filtering step, this is what makes the difference between the filtering step for factorization algorithms and SGE.

The weight function called w_{30} in this article is better than all the others weight functions tested on the three real-case factorizations. This new weight function can improve the matrix output at the end of the filtering step by 3% with respect to the state of the art (w_{21} described by Cavallar and w_{54} used in Msieve) and 11% with respect to the weight function used in CADO-NFS 1.1.

Relation collection algorithms can also be adapted to compute discrete logarithms. But in this case, the matrix is not over $\mathrm{GF}(2)$ anymore but over a finite field of large characteristic. One can wonder if the weight functions described here can be adapted for the discrete logarithm case and if the results still stand.

REFERENCES

[1] B. LaMacchia and A. Odlyzko, "Solving large sparse linear systems over finite fields," in *Advances in Cryptology-CRYPT0' 90*, ser. Lecture Notes in Computer Science, A. Menezes and S. Vanstone, Eds. Springer Berlin / Heidelberg, 1991, vol. 537, pp. 109–133.

[2] C. Pomerance and J. W. Smith, "Reduction of huge, sparse matrices over finite fields via created catastrophes," in *Experimental Mathematics*, 1992, vol. 1, no. 2, pp. 89–94.

[3] S. Cavallar, "On the number field sieve integer factorization algorithm," Ph.D. dissertation, Universiteit Leiden, 2002.

[4] S. Bai, P. Gaudry, A. Kruppa, F. Morain, E. Thomé, and P. Zimmermann, "Crible algébrique: distribution, optimisation (CADO-NFS)," http://cado-nfs.gforge.inria.fr/.

[5] J. Papadopoulos, "Msieve 1.50," http://sourceforge.net/projects/msieve/.

[6] C. Monico, "GGNFS," http://www.math.ttu.edu/~cmonico/software/ggnfs.

[7] A. Lenstra, H. Lenstra, M. Manasse, and J. Pollard, "The number field sieve," in *The development of the number field sieve*, ser. Lecture Notes in Mathematics, A. Lenstra and H. Lenstra, Eds. Springer Berlin / Heidelberg, 1993, vol. 1554.

[8] T. Kleinjung, K. Aoki, J. Franke, A. K. Lenstra, E. Thomé, J. W. Bos, P. Gaudry, A. Kruppa, P. L. Montgomery, D. A. Osvik, H. T. Riele, A. Timofeev, and P. Zimmermann, "Factorization of a 768-bit RSA modulus," in *CRYPTO 2010*, ser. Lecture Notes in Computer Science, T. Rabin, Ed., vol. 6223. Santa Barbara, USA: Springer Verlag, 2010, pp. 333–350.

[9] G. Childers, "Factorization of a 1061-bit number by the Special Number Field Sieve," Cryptology ePrint Archive, Report 2012/444, 2012, http://eprint.iacr.org/.

[10] H. Markowitz, "The elimination form of the inverse and its application to linear programming," *Management Science*, vol. 3, no. 3, pp. 255–269, 1957.

[11] D. Wiedemann, "Solving sparse linear equations over finite fields," *Information Theory, IEEE Transactions on*, vol. 32, no. 1, pp. 54–62, 1986.

[12] D. Coppersmith, "Solving linear equations over gf (2): block lanczos algorithm," *Linear algebra and its applications*, vol. 192, pp. 33–60, 1993.

[13] S. Cavallar, B. Dodson, A. K. Lenstra, W. Lioen, P. L. Montgomery, B. Murphy, H. Te Riele, K. Aardal, J. Gilchrist, and G. Guillerm, "Factorization of a 512-bit RSA modulus," in *Advances in Cryptology—EUROCRYPT 2000*. Springer, 2000, pp. 1–18.

[14] Archives of the Number Theory Mailing List (NMBRTHRY), August 2012, https://listserv.nodak.edu/cgi-bin/wa.exe?A0=NMBRTHRY.

[15] S. Bai, E. Thomé, and P. Zimmermann, "Factorisation of RSA-704 with CADO-NFS," Cryptology ePrint Archive, Report 2012/369, 2012, http://eprint.iacr.org/.

APPENDIX

The appendix contains the complete data for all the weight functions for RSA-155, B200 and RSA-704. It also contains more information about the parameters used in purge and merge that were not necessary for the understanding of the comparison between the weight functions.

For RSA-155, B200 and RSA-704, during merge, the 32 heaviest columns (with most non-zero coefficients) are buried because they are discarded during the linear algebra step.

For B200 and RSA-704, the original relations were used, for RSA-155, they were generated with CADO-NFS, with the following polynomials:

$$g(x) = 3216929091619x - 730598439063554685035469587 00$$

$$f(x) = 5256451200x^5 - 408896411797380x^4 + 1161522740010687 9434x^3$$
$$+ 1895260253840098645553615 0x^2 - 3101178076280120395421431 96507x$$
$$- 1232050446355944886746177 05124790497$$

For RSA-155, during purge, only ideals greater than 16M on rational side and 32M on algebraic side were taken into account. The complete set of data is shown in Table VI.

For B200, during purge, only ideals greater than 250M on rational side and 250M on algebraic side were taken into account. The complete set of data is shown in Table VII.

For RSA-704, during purge, only ideals greater than 250M on rational side and 500M on algebraic side were taken into account. The complete set of data is shown in Table VIII.

Table VI

COMPLETE DATA FOR RSA-155

	After purge			After merge		
	N	Z	$Z \times N$	N	Z	$Z \times N$
00	22308346	447761860	9.99e+15	7147662	714766219	5.11e+15
01	22138422	444257945	9.84e+15	7143706	714370742	5.10e+15
02	22066002	442712752	9.77e+15	7150251	715025216	5.11e+15
03	22002456	441347469	9.71e+15	7157614	715761484	5.12e+15
10	22269888	447050932	9.96e+15	7115118	711511973	5.06e+15
11	22244465	446523945	9.93e+15	7114542	711454238	5.06e+15
12	22219463	446004363	9.91e+15	7115059	711506058	5.06e+15
13	22178976	445159615	9.87e+15	7117969	711797112	5.07e+15
20	23964247	480416327	1.15e+16	7067081	706708249	4.99e+15
21	21768640	436507287	9.50e+15	7045617	704561836	4.96e+15
22	21761845	436316238	9.50e+15	7070828	707083046	5.00e+15
23	**21761781**	436314357	9.49e+15	7070791	707079105	5.00e+15
30	22304109	447747669	9.99e+15	**6977331**	697733144	4.87e+15
31	21849189	438285423	9.58e+15	7058925	705892556	4.98e+15
32	21830037	437817252	9.56e+15	7082259	708226070	5.02e+15
33	21829481	437801991	9.56e+15	7083249	708324969	5.02e+15
40	22214885	445969514	9.91e+15	7046702	704670365	4.97e+15
41	21984540	441156656	9.70e+15	7072749	707275021	5.00e+15
42	21918065	439713161	9.64e+15	7094332	709433500	5.03e+15
43	21894368	439160151	9.62e+15	7110145	711014530	5.06e+15
50	22308348	447761880	9.99e+15	7147564	714756650	5.11e+15
51	22155372	443752266	9.83e+15	7285330	728533258	5.31e+15
52	21984965	440952438	9.69e+15	7162566	716256615	5.13e+15
53	21984965	440952438	9.69e+15	7162566	716256615	5.13e+15
54	21768640	436507287	9.50e+15	7045617	704561836	4.96e+15

Table VII

COMPLETE DATA FOR B200

	After purge			After merge		
	N	Z	$Z \times N$	N	Z	$Z \times N$
00	156537781	3446487128	5.40e+17	45549605	5465952870	2.49e+17
01	154064914	3390799340	5.22e+17	45494110	5459293298	2.48e+17
02	153258565	3372195268	5.17e+17	45535394	5464247294	2.49e+17
03	152309074	3350320106	5.10e+17	45653988	5478478807	2.50e+17
10	156652384	3449109637	5.40e+17	45394730	5447367910	2.47e+17
11	156350477	3442387722	5.38e+17	45381480	5445777829	2.47e+17
12	156043741	3435529061	5.36e+17	45376760	5445211467	2.47e+17
13	155634299	3426284667	5.33e+17	45391855	5447022603	2.47e+17
20	167572079	3686704165	6.18e+17	45145279	5417433653	2.45e+17
21	**150424785**	3308291955	4.98e+17	45029461	5403535441	2.43e+17
22	150537586	3309260024	4.98e+17	45512339	5461480884	2.49e+17
23	150536536	3309239028	4.98e+17	45512337	5461480545	2.49e+17
30	156736538	3452552015	5.41e+17	**44359520**	5323142470	2.36e+17
31	151100876	3324491802	5.02e+17	44989760	5398771398	2.43e+17
32	150880533	3317838531	5.01e+17	45430958	5451715100	2.48e+17
33	150903512	3317900228	5.01e+17	45540161	5464819395	2.49e+17
40	155951650	3434578589	5.36e+17	44823273	5378792980	2.41e+17
41	152683226	3360741556	5.13e+17	44979411	5397529618	2.43e+17
42	151780058	3339537582	5.07e+17	45194795	5423375489	2.45e+17
43	151351955	3328604360	5.04e+17	45496984	5459638320	2.48e+17
50	156537784	3446487174	5.40e+17	45549399	5465927982	2.49e+17
51	154025018	3382830549	5.21e+17	47029434	5643532104	2.65e+17
52	151960357	3341281013	5.08e+17	45885686	5506282610	2.53e+17
53	151960357	3341281013	5.08e+17	45885686	5506282610	2.53e+17
54	**150424785**	3308291955	4.98e+17	45029461	5403535441	2.43e+17

Table VIII

COMPLETE DATA FOR RSA-704

	After purge			After merge		
	N	Z	$Z \times N$	N	Z	$Z \times N$
00	318482490	7116527816	2.27e+18	86068929	14889924962	1.28e+18
01	314374563	7021744305	2.21e+18	85976226	14873887230	1.28e+18
02	312257402	6972431665	2.18e+18	86023351	14882039823	1.28e+18
03	309581688	6909144385	2.14e+18	86289347	14928057114	1.29e+18
10	317600088	7096215399	2.25e+18	85353396	14766137519	1.26e+18
11	316774858	7077359841	2.24e+18	85334630	14762891123	1.26e+18
12	316100182	7061898811	2.23e+18	85335648	14763067383	1.26e+18
13	315222418	7041707284	2.22e+18	85374125	14769723670	1.26e+18
20	335993975	7498462296	2.52e+18	84654737	14645269906	1.24e+18
21	**305215466**	6807845443	2.08e+18	84776712	14666371520	1.24e+18
22	305731404	6816412545	2.08e+18	85956527	14870479407	1.28e+18
23	306052381	6822840501	2.09e+18	86276348	14925808283	1.29e+18
30	316478268	7070667268	2.24e+18	**83566005**	14456919204	1.21e+18
31	306703756	6844378022	2.10e+18	84859979	14680776843	1.25e+18
32	306291074	6830816506	2.09e+18	86024372	14882216573	1.28e+18
33	306755193	6838239642	2.10e+18	86843779	15023973773	1.30e+18
40	315962066	7059762082	2.23e+18	84496548	14617903212	1.24e+18
41	310287263	6928789463	2.15e+18	84800639	14670510832	1.24e+18
42	308089611	6876418573	2.12e+18	85274098	14752419356	1.26e+18
43	306866671	6844339025	2.10e+18	86200932	14912761674	1.29e+18
50	318482486	7116527709	2.27e+18	86068616	14889870935	1.28e+18
51	307415189	6849817515	2.11e+18	87759201	15182342039	1.33e+18
52	307156090	6846108844	2.10e+18	87323537	15106972294	1.32e+18
53	307156090	6846108844	2.10e+18	87323537	15106972294	1.32e+18
54	**305215466**	6807845443	2.08e+18	84776712	14666371520	1.24e+18