An Approximate Probabilistic Model for Structured Gaussian Elimination

Edward A. Bender
Department of Mathematics, University of California, San Diego

November 2, 1998

[summary by François Morain]

1. Introduction

Modern algorithms [5, 7] for factoring integers or for solving discrete logarithms problems work in two phases. In the first one, one collects a huge amount of data that help create a large matrix that is triangularized in the second phase. For instance, the largest non-trivial number ever factored as of today is $(10^{211}-1)/9$, which involved finding dependencies in a $4,820,249 \times 4,895,741$ boolean matrix (see [2]). The easiest way to solve the problem is to find a computer with enough memory so that the matrix fits in core and Gaussian elimination can be used. If such a behemoth is not available, alternative methods have to be used. A method that is widely used relies on the fact that the matrix we are interested in is sparse. For instance the matrix referred to above has only 48.1 non-zero coefficients per row on average. Moreover the structure of the matrix is very peculiar: the leftmost columns are very dense, while the rightmost ones are very sparse. This has led several authors to work out what is called the *Structured Gaussian Elimination* (SGE) method. Apart from this approach, one can use two probabilistic iterative methods to solve the problem, namely Wiedemann's method [9] or Lanczos's [6]. In practice, these approaches are commonly combined.

We will concentrate here on linear algebra for integer factorization. For the discrete logarithm problem, the same ideas can be used with some (sometimes not so trivial) modifications. The aim of the talk is to describe one variant of SGE and try to analyse it.

2. Integer Factorization and Linear Algebra

Suppose we want to factor a large integer N. The basic idea is to look for two integers X and Y such that $X^2 \equiv Y^2 \bmod N$, but $X \not\equiv \pm Y \bmod N$. If this is the case, then $\gcd(X-Y, N)$ is a non-trivial factor of N. How do we proceed to find X and Y? This is usually done using a combination of congruences of the type:

$$x_i^2 \equiv \prod_{j=1}^{k} p_j^{\alpha(i,j)} \bmod N \tag{1}$$

where the p_j's are prime numbers forming the factor basis $\mathcal{B} = \{p_1, p_2, \ldots, p_k\}$. Now we look for a subset I of these such that $\prod_i \prod_j p_j^{\alpha(i,j)}$ is the square of an integer, say Y^2, leading to $\left(\prod_{i\in I} x_i\right)^2 \equiv Y^2 \bmod N$ and we have solved our problem. The method of generation of the auxiliary congruences (1) vary from an algorithm to the other, see the references given above for more details.

Finding a subset I boils down to a linear algebra problem. Indeed, $\prod_i \prod_j p_j^{\alpha(i,j)}$ is the square of an integer if and only if $\prod_j p_j^{\sum_i \alpha(i,j)}$ is, or equivalently, $\sum_i \alpha(i,j)$ is an even integer for all j, which

in turn is equivalent to the fact that we have found a relation between some rows of the matrix $M = (m_{i,j})$ where $m_{i,j} = \alpha(i,j) \bmod 2$, that is a vector x such that $xM = 0$.

What is the shape of the matrix? It is rather easy to guess that a prime number p_j occurs in a factorization with probability $O(1/p_j)$. If we number our primes w.r.t. their magnitude, the left columns of M are seen to be much more dense than the right ones.

3. Structured Gaussian Elimination

The idea of the method [4, 8] is to try to perform Gaussian elimination on the sparse part of the matrix, as long as the fill-in is not too important. We will present here the version given in [1][1]. The *weight* of a row (resp. column) is the number of its non-zero coefficients.

Step 0. [Initial deactivation] deactivate some (small) fraction of the columns and call all remaining columns *light*.

Step 1. [Initial clean up] repeat the following steps (a) and (b) until all columns have weight greater than 1:

(a) Eliminate columns of weight 0 (it is of no use).

(b) If a column has weight 1, eliminate it and the row intersecting it (since it cannot be part of a dependency).

Step 2. [Deactivation] repeat steps (a) and (b) until all columns have been either deactivated or eliminated:

(a) If any row has weight 1, eliminate it and the light column intersecting it. Repeat as often as possible.

(b) Deactivate a column of high weight and repeat (a).

Step 3. [Final step] Find the dependencies of the small matrix and build back the solutions of the initial system.

This algorithm is an iterative process that decreases the size of the matrix. It can happen that more and more rows are eliminated in Step 2a, leading to what is called a "catastrophe" in [8]. Among the questions we ask are: what is the size of the smallest reduced matrix that we can obtain before the catastrophe begins?

4. Branching Processes and the Critical Product

Let us review the basic properties of a Galton-Watson branching process, in the view of applying it to SGE. Theoretical properties on this topic can be found in [3, pp. 3–7].

In a Galton-Watson branching process, time is divided into generations. We start with one object alive in the 0th generation and at generation i, the number of objects depends on the number of objects in generation $i-1$. Let $f(x)$ denote the probability generating function of this number. The probability generating function for the number of objects in the nth generation is $f_{(n)}(x) = f(f(\cdots f(f(x))\cdots))$. Therefore, the expected number of objects in the nth generation is

$$(f_n(x))'\big|_{x=1} = (f'(1))^n = \varphi^n.$$

This quantity φ acts as a threshold. If $\varphi \leq 1$, the process terminates with probability 1, whereas if $\varphi > 1$, the process has a nonzero probability of surviving forever.

How do we apply this theory to our problem? In Step 2a of the algorithm, an object is a row of weight 1 that disappears and may create new objects of weight 1 for the next generation. The associated value φ (called the *critical product*) of this branching process is given by:

[1] There is no *canonical* algorithm for SGE: from the same idea, details can differ in the implementation and the choice of some parameters.

Theorem 1. *Assuming that the rows and columns are independent, we have*

$$\varphi \approx \left(\frac{\sum_k k(k-1)c_k}{\sum_k kc_k}\right)\left(\frac{2r_2}{\sum_k kr_k}\right),$$

where c_k (resp. r_k) is the probability that a randomly chosen column (resp. row) contains exactly k nonzero entries and the approximation is due to the fact that the matrix is finite.

For our purpose, we see that if $\varphi > 1$, then our algorithm loops forever and the catastrophe occurs.

5. Critical Parameters and their Analysis

5.1. **Probability Model.** Let $M = (m_{i,j})$ denote our $m \times n$ matrix. We make the assumption that $\text{Prob}(m_{i,j} \neq 0) = D/j$ for some fixed parameter D. This sounds realistic since the column j of M is related to the prime $p_j \approx j \log j$ dividing some number, thus with probability $1/p_j \approx D/j$ since log is a slowly increasing function. On the other hand, a reasonable model for the weight of rows is that of a Poisson model. We will let $C = Dm/n$.

5.2. **Effect of the Initial Clean Up.** This step causes $n\alpha_1$ columns (and corresponding rows) of weight 1 and $n\alpha_0$ columns (and no rows) of weight 0 to be eliminated. At the end of this step, one gets a new matrix with $m - n\alpha_1$ rows and $n(1 - \alpha_0 - \alpha_1)$ columns.

Theorem 2. *One has:*

$$\alpha_1 \approx \overline{\alpha}_1 = \frac{CE_1(C)}{1 - D(E_0(C) - E_1(C))}, \qquad \alpha_0 \approx CE_2(C) + DE_1(C) \times \overline{\alpha}_1,$$

where $E_r(C) = \int_C^\infty e^{-t}t^{-r}\,dt$ is the exponential integral.

Numerically, for $m/n = 1.0$ and $D = 3.0$, this yields $\alpha_0 = 0.012$ and $\alpha_1 = 0.044$. More generally, the values of α_0 and α_1 are rather small. Moreover, the new matrix will have the same shape as the original one, meaning that the Poisson model will still apply.

5.3. **Matrix After Reduction.** What happens in our case is that the value of φ increases from one iteration of the algorithm to the other, reaching a value > 1 and thus causing a catastrophe. We are interested in the parameters associated with the matrix when this occurs. We suppose we enter Step 2 of the algorithm with an $m \times n$ matrix $M = (m_{i,j})$. At Step 2b, we suppose that some fraction τ of the columns have been deactivated and eliminated. For simplicity, assume these are the leftmost τn columns of M. We want to relate τ and φ.

We suppose that our model is still valid for M, and in particular the model used for the row weight is again a (truncated, since there are no rows of weight 0 or 1) Poisson model of parameter λ. We first have:

Theorem 3. *With the notations above:*

$$-D\log\tau \approx \lambda\left(\frac{e^\lambda - 1}{e^\lambda - \lambda - 1}\right).$$

From this, one gets:

Theorem 4. *Letting $C = Dm/n$, one has*

$$\varphi \approx \frac{C(1-\tau)}{-\tau\log\tau}\frac{\lambda}{e^\lambda - 1}.$$

This formula enables one to compute the value τ_0 corresponding to $\varphi = 1$, i.e., when a catastrophe occurs. For instance, when $m/n = 1.0$ and $D = 3.0$, $\tau_0 = 0.318$. In any case, τ_0 is always far from 0, which indicates that the SGE algorithm ends up with a matrix of size proportional to that of the original matrix.

5.4. **Final Matrix.** By final, we understand the matrix which is rebuilt after SGE and to which we need apply another linear algebra algorithm. Among the τn columns discarded in Step2 of the algorithm, a fraction δ of them were deactivated (a remaining $\tau - \delta$ having been eliminated), therefore being eligible to the final matrix.

Theorem 5. *With the notations above:*

$$\delta(\tau) \approx \int_0^\tau \left(1 + \frac{C\lambda}{\tau(1-\varphi)(e^\lambda - 1)}\right)^{-1} d\tau.$$

For the same numerical values, $m/n = 1.0$ and $D = 3.0$, one finds $\delta_0 = 0.186$. This again shows that the final matrix is of size proportional to that of the original matrix.

6. Conclusions

The authors have attempted an analysis of the structured Gaussian elimination. With rather crude assumptions, they were able to derive an approximate model that is close, at least qualitatively, to that observed by their own simulations as well as by people who really factor numbers. Going further, that is have a model close to reality in quantity would require more work.

Bibliography

[1] Bender (Edward A.) and Canfield (E. Rodney). – An approximate probabilistic model for structured Gaussian elimination. *Journal of Algorithms*, vol. 31, n° 2, 1999, pp. 271–290.

[2] CABAL. – 211-digit SNFS factorization. – `ftp://ftp.cwi.nl/pub/herman/NFSrecords/SNFS-211`, April 1999.

[3] Harris (T. E.). – *The theory of branching processes.* – Dover Publications, 1989.

[4] LaMacchia (B. A.) and Odlyzko (A. M.). – Solving large sparse linear systems over finite fields. In Menezes (A. J.) and Vanstone (S. A.) (editors), *Advances in Cryptology. Lecture Notes in Computer Science*, vol. 537, pp. 109–133. – Springer-Verlag, 1990. Proceedings Crypto '90, Santa Barbara, August 11–15, 1988.

[5] Lenstra (A. K.) and Lenstra, Jr. (H. W.) (editors). – *The development of the number field sieve.* – Springer, *Lecture Notes in Mathematics*, vol. 1554, 1993.

[6] Montgomery (P. L.). – A block Lanczos algorithm for finding dependencies over GF(2). In Guillou (L. C.) and Quisquater (J.-J.) (editors), *Advances in Cryptology – EUROCRYPT '95, Lecture Notes in Computer Science*, vol. 921, pp. 106–120. – 1995. International Conference on the Theory and Application of Cryptographic Techniques, Saint-Malo, France, May 1995, Proceedings.

[7] Pomerance (Carl) (editor). – *Cryptology and computational number theory.* – American Mathematical Society, Providence, RI, 1990, xii+171p. Lecture notes prepared for the American Mathematical Society Short Course held in Boulder, Colorado, August 6–7, 1989, AMS Short Course Lecture Notes.

[8] Pomerance (Carl) and Smith (J. W.). – Reduction of huge, sparse matrices over finite fields via created catastrophes. *Experimental Mathematics*, vol. 1, n° 2, 1992, pp. 89–94.

[9] Wiedemann (Douglas H.). – Solving sparse linear equations over finite fields. *IEEE Transactions on Information Theory*, vol. 32, n° 1, 1986, pp. 54–62.