# MATCHING MODULO DIVISORS, AND A SIMPLE $N^{1/4}$ FACTORING ALGORITHM

Eric Bach[1]

Computer Science Division
University of California
Berkeley, CA 94720

ABSTRACT

We present an algorithm for the following problem: given $x_1, \ldots, x_m$, distinct modulo $N$, find two numbers in the list that are congruent modulo a proper divisor of $N$. The number of steps required is less than $m$ times a polynomial in $\log N$. This gives a simple proof that $N$ can be factored in expected time $O(N^{1/4+\epsilon})$, for any $\epsilon > 0$.

## 1. INTRODUCTION

Consider the following simple algorithm for factoring an integer $N$: generate a list of numbers modulo $N$, and see if two members of this list are congruent modulo a prime divisor of $N$. Since we don't know the factorization of $N$, we look for two entries, say $x_i$ and $x_j$, that satisfy $\gcd(x_i - x_j, N) > 1$. Then the result of the gcd computation is a divisor of $N$, and if we are lucky, it will split $N$ into two smaller factors. We then repeat until $N$ is completely factored.

A closer look at this method leads to two questions:

1) How big should the list be?

2) How do we extract a divisor from the list?

The first question can be asked precisely as follows: Let $p$ and $q$ be two relatively prime divisors of $N$. If $x_1, \ldots, x_m$ are selected at random, how big must $m$ be before there are good odds that two members of the list are congruent modulo $p$ but not modulo $q$? The answer is proportional to the square root of $p$; the constant of proportionality depends on what "good odds" means, and on how big $q$ is.

We can also restate the second question: given distinct $x_1, \ldots, x_m$ modulo $N$, how fast can we find a match modulo a divisor of $N$; that is, $d$, $i$, and $j$ such that $d \mid N$, $1 < d < N$, and $x_i \equiv x_j \pmod{d}$? The answer is surprising: finding a match can be done in time that is almost *linear* in the size of the list.

We now have a rough idea of how our factoring algorithm should behave: to have a good chance of removing the smallest prime factor $p$ of $N$, we should generate $O(\sqrt{p})$ random numbers modulo $N$. The match-finding will also take time about $O(\sqrt{p})$, and since some $p \mid N$ must be less than $\sqrt{N}$, we see that $N$ can be split in expected time that is about $O(N)^{1/4}$.

The rest of this paper makes the above argument precise.

## 2. HOW MANY NUMBERS DO WE NEED?

As explained in the introduction, to split $N$ we choose $x_1, \ldots, x_m$ at random, and hope that two entries, say $x_i$ and $x_j$, satisfy $1 < \gcd(x_i - x_j, N) < N$. To make this precise, consider the following process:

[1]This research was sponsored by NSF Grant MCS 82-04506.

Let $N$ have two relatively prime divisors, say $p$ and $q$, with $p<q$. We pick $x_1, x_2, \cdots$ at random satisfying $0 \leq x_i \leq N-1$ until two of them, say $x_i$ and $x_j$, agree modulo $p$. We say we have *succeeded* if $x_i \not\equiv x_j \pmod{q}$, and *failed* otherwise.

We now show that if the process runs long enough, there is a good chance that it will be a success. Precisely, if $\epsilon > 1/p$ and $m \geq [2\ln(\epsilon - 1/p)^{-1}]^{1/2} \cdot \sqrt{p} + 1$, then with probability $\geq 1-\epsilon$, the process terminates successfully in $m$ steps or less.

First, the probability $P_c$ that the process requires more than $m$ steps is

$$P_c = \prod_{k=1}^{m-1} (1 - \frac{k}{p})$$

(see [2], p. 49). Using $\ln(1-x) \leq -x$ for $0<x<1$,

$$\ln P_c \leq -\sum_{k=1}^{m-1} \frac{k}{p} \leq -\frac{(m-1)^2}{2p}.$$

By the hypothesis on $m$, $-(m-1)^2 \leq 2p \ln(\epsilon - 1/p)$, so

$$P_c \leq \epsilon - 1/p.$$

Next, we estimate the probability $P_f$ that the process fails in $m$ steps or less. For $1 \leq i < j$, let $S_{ij}$ be the event that we stop with $x_i \equiv x_j \pmod{p}$ . Then

$$P_f = \sum_{1 \leq i < j \leq m} Pr[S_{ij} \ \& \ x_i \equiv x_j \pmod{q}],$$

since the $S_{ij}$'s are disjoint. Since $p$ and $q$ are relatively prime,

$$Pr[S_{ij} \ \& \ x_i \equiv x_j \pmod{q}] = Pr[S_{ij}] \cdot Pr[x_i \equiv x_j \pmod{q}].$$

Therefore,

$$P_f = \sum_{1 \leq i < j \leq m} Pr[S_{ij}] \cdot \frac{1}{q} \leq \frac{1}{p}.$$

Now the probability $P_s$ of success in $m$ steps or less satisfies

$$P_s \geq 1 - (\epsilon - 1/p) - 1/p = 1 - \epsilon,$$

as desired.

We now make the following observation: the chance that there exist *any* numbers that agree modulo $p$ but not modulo $N$ is at least as good as the chance that two agree modulo $p$ and the earliest such ones are distinct modulo $q$. This proves

THEOREM 1

Let $N$ have at least two distinct prime factors, and let $p$ be the smallest one. Then for any $\epsilon > 1/p$, if we select at least

$$[2\ln(\epsilon - 1/p)^{-1}]^{1/2} \cdot \sqrt{p} + 1$$

random numbers between 0 and $N-1$, two will agree modulo $p$ but not modulo $N$ with probability at least $1-\epsilon$.

## 3. HOW DO WE FIND A MATCH MODULO A DIVISOR?

Assume now that we have enough numbers to expect the difference of two of them to split $N$. The problem is now to *find* them in time roughly proportional to the size of the list.

To fix ideas, let $x_1, \ldots, x_m$ be distinct integers between 0 and $N-1$; we will first decide if something in the first half of the list is congruent modulo a divisor of $N$ to something in the second half of the list. To do this, we form

$$f(X)=\prod_{1\leq i\leq m/2}(X-x_i)(\bmod N);$$

this is the monic polynomial having the the numbers in the first half of the list for roots. Then, generalizing a trick in [5], we evaluate $f$ at each number in the remainder of the list. We have the desired match if and only if there is a $j$, $m/2<j\leq m$, with $\gcd(f(x_j),N)>1$, for this true exactly when $f(x_j)\equiv 0(\bmod p)$ for some prime $p \mid N$. If this is the case, then there must be some $i$, $1\leq i\leq m/2$, with $x_i\equiv x_j(\bmod p)$, and since the $x_i$'s are distinct, some $i$ with $1<\gcd(x_i-x_j,N)<N$.

The most expensive part of the above algorithm is the construction and evaluation of $f$, and so we digress a bit and mention some results in polynomial algebra. Specifically, we are interested in $\mathbb{Z}_N[X]$, the ring of polynomials with integer coefficients modulo $N$; to be definite, let the members of the coefficient ring $\mathbb{Z}_N$ be $0,\ldots,N-1$. We imagine that we are using multiprecision arithmetic, and so we count operations on integers of some fixed "word size".

First, multiplication in $\mathbb{Z}_N[X]$ can be done quickly: if $f$ and $g$ are polynomials in $\mathbb{Z}_N[X]$ of degree at most $k<N$, then we can compute $fg$ in $O(k(logN)^2\log\log N)$ steps.

To do this as explained in [5], we set $D=(k+1)N^2$ and recover the coefficients of $fg$ from the integer product $f(D)\cdot g(D)$. Using a fast integer multiplication algorithm ( [6] ) , this takes $O(n\log n\log\log n)$ steps, where each factor has $n$ bits. Using the hypothesis that $k<N$, we get the bound advertised.

Next, we need the corresponding result for polynomial division. Let $a,b\in\mathbb{Z}_N[X]$, with degrees $k$ and $l$ respectively, and assume that $l\leq k<N$, with the leading coefficient of $b$ a unit modulo $N$. Then we can find polynomials $q$ and $r$ with $a=bq+r$, $\deg(r)<\deg(b)$ in $O(k(logN)^2\log\log N)$ steps.

This can be proved using the argument in [1], p. 95, provided that $B(X)=X^l b(1/X)$ is a unit in the power series ring $\mathbb{Z}_N[[X]]$. For this to be true we need the constant term of $B$ to be invertible mod $N$, and this is exactly the hypothesis on $b$.

Now that we have fast polynomial multiplication and division, the methods in [1] will suffice to prove the following results:

a) Let $x_1,\ldots,x_k\in\mathbb{Z}_N$, and $k<N$. We can compute

$$f(X)=\prod_{i=1}^{k}(X-x_i)$$

in $O(k(\log N)^3\log\log N)$ steps.

b) Let $f(X)\in\mathbb{Z}_N[X]$, with $\deg(f)\leq k<N$. Then we can evaluate $f$ at $x_1,\ldots,x_k\in\mathbb{Z}_N$ in $O(k(\log N)^3\log\log N)$ steps.

We can now solve the problem of finding a match modulo a divisor of $N$ in an arbitrary list $x_1,\ldots,x_m$. Since duplicates are of no use, we remove them by first sorting the list at a cost of $O(m\log m\log N)$. Then we look for a match between the two halves of the list, using the above algorithm. This constructs one polynomial of degree at most $m/2$, evaluates it at about $m/2$ points, and then does $m$ gcd calculations. Since the Euclidean algorithm takes $O(\log N)^3$ steps on numbers less than $N$ ( [3], p. 320 ), this all requires time that is $O(m(\log N)^3\log\log N)$. If we don't find a match between the first and the second halves of the list, we can run the algorithm separately on each half, then on each quarter, and so on; each recursive stage requires the same amount of time, and there are at most $\log m$ stages. Adding everything up, we have proved

THEOREM 2

Let $m<N$ and let $0\leq x_1,\ldots,x_m<N$. Then we can find $x_i,x_j$, and $d \mid N$ (if any) with $x_i\equiv x_j(\bmod d)$ and $x_i\not\equiv x_j(\bmod N)$ in $O(m(\log N)^4\log\log N)$ steps.

## 4. SPLITTING $N$ IN ABOUT $N^{1/4}$ STEPS

Combining the last two sections, here is an incarnation of our factoring algorithm; it will split a composite $N$ with probability at least $1-(1/2)^k$:

a) Verify that $N$ is not a perfect power, and that it has no factors less than 5. (Then $N$ has a prime factor $p$ with $p \leq N^{1/4}$, and $1/p \leq 1/4$.)

b) Let $m \geq 2\sqrt{\ln 2}N^{1/4}+1$, and generate $k \cdot m$ random numbers from $0, \ldots, N-1$. (Using theorem 1 with $\epsilon=1/2$, each block of $m$ numbers contains a pair that agree modulo $p$ but not modulo $N$ with probability $\geq 1/2$, and so the whole list contains such a pair with probability $\geq 1-(1/2)^k$. Since we select each random number with about $\log N$ flips of a fair coin, this takes $O(kN^{1/4}\log N)$ steps.)

c) Run the algorithm in the last section to find two numbers $x$ and $y$ in the above list with $1<\gcd(x-y,N)<N$. (This takes $O(km(\log N)^4\log\log N)$ steps.)

Since $k=\log(|1/2|^k)$ and $m=O(N^{1/4})$, we have proved

THEOREM 3

Let $\epsilon>0$. Then the above algorithm will split a composite number $N$ with probability at least $1-\epsilon$ in $O(|\log\epsilon|N^{1/4}(\log N)^4\log\log N)$ steps.

Now a caveat: this is an asymptotic result, not a practical method. We require space equal to the running time, and fast polynomial algebra of high precision.

Seeking something more tractable, one is tempted to apply the above analysis to Pollard's $\rho$-method [4], which generates $x_1, \ldots, x_m$ by a rule such as $x_1=2$, $x_{i+1}=x_i^2-1(\bmod N)$, and extracts a factor of $N$ using a small amount of space. This apparently takes time about $N^{1/4}$, but $x_1, \ldots, x_m$ is in no sense random, and so our argument does not apply here.

We are left with two unanswered questions, replacing the two in the introduction:

1) The algorithm presented here is an analyzable version of the $\rho$-method, but needs too much space to be practical. Can this requirement be gotten rid of?

2) Let $f(X)$ be a polynomial with integer coefficients. How is the sequence $x_1, f(x_1), f(f(x_1)), \cdots$ distributed modulo $N$? Can we prove that the original $\rho$-method works?

## 5. ACKNOWLEDGEMENTS

Thanks are due to Manuel Blum, Martin Hellman, Shafi Goldwasser, Michael Luby, and Jeff Shallit for discussions and suggestions concerning these ideas.

## 6. REFERENCES

[1] A. Borodin and I. Munro, *The Computational Complexity of Algebraic and Numeric Problems,* New York : American Elsevier (1975).

[2] W. Feller, *An Introduction to Probability Theory and its Applications (volume 1),* New York : Wiley (1968).

[3] D. Knuth, *The Art of Computer Programming (volume 2),* Reading : Addison-Wesley (1969).

[4] J. Pollard, A Monte Carlo Method for Factorization, *BIT 15,* pp. 331-334 (1975).

[5] J. Pollard, Theorems on Factorization and Primality Testing, *Proceedings of the Cambridge Philosophical Society 76,* pp. 521-528 (1974).

[6] A. Schonhage and V. Strassen, Schnelle Multiplikation Grosser Zahlen, *Computing 7,* pp. 281-292 (1971).