Some comments on Quadratic sieve

Johan Håstad

November 12, 2014

Abstract

We give a short recap of the quadratic sieve algorithm.

1 The setup

We want to factor N. Let a be $\sqrt{N}$ rounded to the closest integer. Let B be a bound on primes. The idea of the collect many equations of the form

$$(i_j + a)^2 - N = \prod_{p \le B} p^{e_p^j} \tag{1}$$

where we note that most exponents are 0. We think of the factor -1 as a special "prime". If follows from (1) that

$$(i_j + a)^2 \equiv \prod_{p \le B} p^{e_p^j} \bmod N.$$

We want to find a subset S of the equations such that

$$\sum_{j \in S} e_p^j$$

is even for each p, say it equals $2b_p^S$. Then

$$\prod_{j \in S} (i_j + a)^2 \equiv \prod_{p \le B} p^{\sum_{j \in S} e_p^j} \equiv \prod_{p \le B} p^{2b_p^S} \bmod N.$$

gives a solution to $x^2 \equiv y^2 \bmod N$ with $x = \prod_{j \in S}(i_j + a)$ and $y = \prod_{p \le B} p^{b_p^S}$. Heuristically it turns out that for at least half of all equations produced this way, we have $x \not\equiv \pm y \bmod N$ and in this case $\gcd(x + y, N)$ gives a factor in N.

2 Finding the equations

One could, in principle, generate the integers $(i + a)^2 - N$ for all small (in absolute value) i and do trial division. This is too slow and sieving is needed. This is done as follows.

1. Initialize an array that in position i has $\log(|(i + a)^2 - N|)$ as a floating point number.
2. Do the below for all $p \leq B$.
3. Find the solutions i_0 and i_1 to $(i+a)^2 = N \bmod p$ and subtract $\log p$ from all numbers of the forms $i_0 + kp$ and $i_1 + kp$ in the array for integers k. For each $t > 1$ find also the solutions mod p^t and subtract an additional $\log p$ from the corresponding entries.
4. For the elements in the array that are reduced to a number close to 0, reconstruct the original number and factor it by trial division to see if we get a complete factorization.

Note that you need only address primes p such that $x^2 = N \bmod p$ is solvable and this is equivalent to $N^{(p-1)/2} \equiv 1 \bmod p$. Other primes need not be included in the factor base (do remember -1).

The solutions i_0 and i_1 can be found efficiently by an algorithm by Shanks and Tonelli (or other algorithms, for $p \equiv 3 \bmod 4$ the solution to $x^2 \equiv c \bmod p$ can be found as $c^{(p+1)/4} \bmod p$).

For all primes except 2, once you have the solution modulo p it is easy to find the solution modulo higher powers of p as follows. If i_0 is a solution modulo p^{t-1} then you know that solutions modulo p^t will be of the form $i_0 + kp^{t-1}$. Substituting this into the equation we get

$$(i_0 + a + kp^{t-1})^2 \equiv N \bmod p^t.$$

Expanding the square and dropping the term $k^2p^{2(t-1)}$ since it is 0 mod p^t we get

$$(i_0 + a)^2 + 2k(i_0 + a)p^{t-1} \equiv N \bmod p^t$$

which is equivalent to

$$2k(i_0 + a)p^{t-1} \equiv N - (i_0 + a)^2 \bmod p^t. \tag{2}$$

Now as i_0 is a solution modulo p^{t-1} we have that $N - (i_0 + a)^2 = p^{t-1}j_0$ for some integer j_0. This implies that (2) is solved by

$$k \equiv \frac{j_0}{2(i_0 + a)} \bmod p. \tag{3}$$

Note that $(i_0 + a) \neq 0 \bmod p$ as N is not divisible by p and thus the $i_0 + a$ in the denominator in (3) is not a problem. When trying to find solutions modulo 2^t then the 2 in the denominator of (3) does cause a small problem and you need to proceed as follows.

The question whether 2^t divides a number corresponding to i depends only on i modulo 2^{t-1}. You also have to look at factors 2 and 4 and 8 specially but for t larger than 3 you can do lifting as above (keeping in mind that divisibility by 2^t is determined by i modulo 2^{t-1}).

It might be good to know that the above procedure for lifting solutions modulo p^{t-1} to solutions modulo p^t is called "Hensel lifting".

I recommend, however, finding solutions by trial and error in the first implementation and if this is a bottle-neck implement a faster algorithm.

3 Finding the set S

This is a linear algebra problem modulo 2 and is solved by Gaussian elimination. As you only need to keep track of coefficients mod 2, you can pack 32 or 64 coefficients in a computer word. Most languages have primitive operations taking bitwise xor of two computer words. This makes for a very efficient implementation of Gaussian elimination as you can operate on up to 64 coefficients in one operation.

If you have b primes in your factor base (including -1) and c relations then you have rows in your matrix of length $b + c$ bits . The first b positions corresponds to the primes and the other c are used to find S.

A row corresponding to relation j has a one in a position corresponding to a prime p iff the exponent e_p^j is odd. The last c positions are all zero except the $b + j$'th position which is one.

Do Gaussian elimination to get a vector with 0 in the first b positions. The identity of the set S is now read in the last c positions of such a vector with initial zeroes.

To be able to get such a vector you need $c > b$ and you should be able to get $c - b$ different S's. As each S works with probability $\frac{1}{2}$, using $c = b + 20$ should be sufficient. In other words, the number of relations to find should be slightly larger than the number of primes in your factor base.

4 Optimizing B and sieving interval

The main parameter to optimize is B. If it is too small, you rarely find any relations. If it is too large sieving takes a long time and you get a very large matrix in the end making the linear algebra inefficient.

Start with moderately large N and, find a good value B and increase N and B to get the numbers you want.

Another parameter to optimize is the size of the sieving array. This is not so crucial. If a given interval does not give the needed number of relations one can simply start over again with some more numbers i. The reason for a large array is that the overhead of finding the solutions mod p becomes less important. Note, however, that to have efficiency in the sieving process it is good if the length of the sieving interval is much larger than B. If this is not the case no real sieving takes place as each p larger than the length of the sieving interval only divides at most 2 numbers.