

# An Implementation of Zhang's Special Quadratic Sieve and Possible Extension

Eric Landquist  
MATH 488: Integer Factorization

December 20, 2002

## 1 Introduction

While the Quadratic Sieve (QS) is effective at factoring composite integers up to about 110 digits, it cannot take advantage of integers in a special form. In contrast, the Number Field Sieve (NFS) [5] was first used to factor numbers of a special form and later generalized to factor general integers. In 1998 Zhang [8] found a clever way to factor integers of the form  $n = m^3 + a_2m^2 + a_1m + a_0$  using a QS-like algorithm. Of course any integer can be expressed in this form by letting  $m = \lfloor n^{1/3} \rfloor$  and  $a_0 = n - m^3$ . However, the smaller the  $a_i$  coefficients are, the faster the algorithm will run. If the  $a_i$  get too large, this method will be slower than other QS variants. It may be necessary to use a multiplier. For example, if  $n = 2^{64} + 1$ , there is no obvious way to express  $n$  in the desired form, but  $4n = (2^{22})^3 + 4$  is in the proper form, so we apply this algorithm to  $4n$  instead of  $n$  itself.

In this paper we describe the method, how and why it works, and analyze the complexity of the method. It will be shown that for small enough coefficients  $a_i$  and for  $n$  large enough that the running time of SQS beats the running time of QS.

## 2 Zhang's Idea

Let  $n$  be an integer of the form

$$n = m^3 + a_2m^2 + a_1m + a_0,$$

where  $m \sim n^{1/3}$ ,  $a_i = O(n^\epsilon)$ , for all  $i$ , and  $\epsilon > 0$  is very small. Later we will calculate exactly how small  $\epsilon$  should be for there to be a theoretical speed-up over QS. One of the main steps of QS and its variants is to find smooth values of a polynomial,  $Q$ , such that  $x^2 \equiv Q \pmod{n}$ . We will do the same. Let

$$x = b_2m^2 + b_1m + b_0, \quad b_i \in \mathbb{Z}.$$

Then

$$x^2 = b_2^2m^4 + 2b_1b_2m^3 + (2b_0b_2 + b_1^2)m^2 + 2b_0b_1m + b_0^2.$$

We will make the right hand side a lot nicer by making a couple substitutions. Notice that

$$m^3 = n - (a_2m^2 + a_1m + a_0) \equiv -(a_2m^2 + a_1m + a_0) \pmod{n},$$

and

$$\begin{aligned} m^4 &\equiv -(a_2m^3 + a_1m^2 + a_0m) \pmod{n} \\ &\equiv -(a_2(-(a_2m^2 + a_1m + a_0)) + a_1m^2 + a_0m) \pmod{n} \\ &\equiv (a_2^2 - a_1)m^2 + (a_1a_2 - a_0)m + a_0a_2 \pmod{n} \end{aligned}$$

so that

$$x^2 \equiv c_2m^2 + c_1m + c_0 \pmod{n},$$

where

$$\begin{aligned} c_2 &= (a_2^2 - a_1)b_2^2 - 2a_2b_1b_2 + b_1^2 + 2b_0b_2 \\ c_1 &= (a_1a_2 - a_0)b_2^2 - 2a_1b_1b_2 + 2b_0b_1 \\ c_0 &= a_0a_2b_2^2 - 2a_0b_1b_2 + b_0^2. \end{aligned}$$

Recall that since  $m \sim n^{1/3}$ , we want  $c_2 = 0$  so that  $x^2 \pmod{n}$  is not too much larger than  $n^{1/3}$ . The way to do this is to parametrize the  $b_i$ . In fact, there are two ways.

## 2.1 Single Variable Parametrization

The first parametrization is:

$$\begin{aligned}b_2 &= 2 \\b_1 &= 2b \\b_0 &= a_1 - a_2^2 + 2a_2b - b^2,\end{aligned}$$

where  $b \in \mathbb{Z}$ . In this case our equation  $x^2 \equiv Q \pmod{n}$  looks like:

$$\begin{aligned}(x(b))^2 &\equiv Q(b) \pmod{n} \\&\equiv 4(a_1a_2 - a_0 - (a_1 + a_2^2)b + 2a_2b^2 - b^3)m \\&\quad + (4a_0a_2 - 8a_0b + (a_1 - a_2^2 + 2a_2b - b^2)^2) \pmod{n}.\end{aligned}$$

The polynomial  $Q(b)$  is quartic in  $b$  and, assuming the  $a_i$  are all much smaller than  $b$ , is dominated by the term  $b^3m$ . If  $b = O(n^\epsilon)$ , then  $Q(b) = O(n^{1/3+3\epsilon})$ . In order for this method to be a speed-up over QS, we want  $\epsilon$  to be small enough so that  $n^{1/3+3\epsilon} \leq n^{1/2}$ , since the smooth integers we sieve for in QS are about  $n^{1/2}$ . Thus  $\epsilon \leq 1/18$ , and the sieving interval with  $Q(b)$  will be roughly  $[-n^{1/18}, n^{1/18}]$ . For small  $n$  this interval is incredibly small, so this will not work for small  $n$ . Of course we want our interval to be subexponential in size, so if  $n$  is sufficiently large we can take a subexponential sized subinterval of this.

In actuality, the different versions of QS will produce smooth integers of size  $n^{1/2+\epsilon}$ , where  $\epsilon$  is largest for the basic QS, smaller for the multiple polynomial QS (MPQS), smaller still for the self-initializing QS (SIQS), and smallest when we apply standard tricks with large primes and such. So our sieving interval can be widened some, but not a whole lot. Our quartic polynomial grows much faster than other sieving polynomials that we're used to.

## 2.2 Two Variable Parametrization

As noted above, we get a speed-up when we use several polynomials for sieving with QS. The same trick can be applied in this case with a different parametrization of the  $b_i$ . Again, we let  $x = b_2m^2 + b_1m + b_0$ , but now

$$\begin{aligned}b_2 &= 2u^2 \\b_1 &= 2(uv + a_2u^2) \\b_0 &= a_1u^2 - v^2,\end{aligned}$$

where  $u, v \in \mathbb{Z}$ , but with the added restrictions that  $(u, v) = 1$  and  $u > 0$ , since otherwise we will produce redundant relations. Our sieving polynomial is again a quartic:

$$Q(u, v) = d_0u^4 + d_1u^3v + d_2u^2v^2 + d_3uv^3 + v^4,$$

where

$$\begin{aligned} d_0 &= -4a_0(m + a_2) + a_1^2 \\ d_1 &= -4(a_1m + 2a_0) \\ d_2 &= -2(2a_2m + a_1) \\ d_3 &= -4m. \end{aligned}$$

Assuming that  $u$  and  $v$  grow to be much larger than the  $a_i$ , and are on the order of  $n^\epsilon$ ,  $Q(u, v)$  will be dominated by any term, except for  $v^4$ , and will be on the order of  $n^{1/3+4\epsilon}$ . We endeavor to keep this below  $n^{1/2}$  in order to have a theoretical speed-up over QS, so we want  $\epsilon \leq 1/24$ . Recalling the restrictions we have made on  $u$  and  $v$ , this will give approximately  $\frac{12n^{1/12}}{\pi^2}$  distinct inputs into our sieving polynomial versus only  $2n^{1/18}$  for the single variable polynomial. Again, we will bound  $u$  and  $v$  subexponentially, but that will wait until our complexity analysis. Now we will discuss specifics of how this method works.

### 3 Details of the Algorithm

The details of the special QS (SQS) will only be given for the multiple polynomial version, since it is faster in general than the single variable version. The details of the single variable SQS will be very similar to what is described below, though simplified in some cases, since we have no need to switch polynomials. We mention the single variable version of SQS since for small  $n$  it may even be faster to run just the single version, or use it in conjunction with the multiple polynomial SQS if either method is particularly slow on its own. Similarly, either SQS method may be combined with SIQS once the residues of  $Q$  surpass  $n^{1/2}$  in SQS. Even taking into account the switch in algorithms and the initialization time required, the hybrids should run faster than either one alone for small  $n$ . One major technicality is that the factor base would have to be the union of the individual factor bases used. This construction is described first.

### 3.1 The Factor Base

One of the first things we need to do is construct a factor base,  $FB$ . The bulk of the time spent in the algorithm will be finding smooth integers, i.e. those integers such that every prime dividing them are in  $FB$ . For QS, constructing the factor base is quite simple, just take all primes  $p$  such that  $\left(\frac{n}{p}\right) = 1$ . We also include -1, since some integers we sieve for will be negative. In general, we take our factor base to be those primes  $p$ , below a certain bound, for which the sieving polynomial has a root mod  $p$ . Again, we will also include -1 as the first “prime.” If  $Q(u, v) = d_0u^4 + d_1u^3v + d_2u^2v^2 + d_3uv^3 + v^4$  is our sieving polynomial, notice that  $Q(u, v) = u^4f(vu^{-1})$ , where

$$f(t) = d_0 + d_1t + d_2t^2 + d_3t^3 + t^4.$$

Suppose  $p|Q(u, v)$ . It is clear that if  $p|u$ , then  $p|v$ , contradicting our restriction. So assuming that  $p \nmid u$ ,  $p|f(vu^{-1})$ , so we find solutions to

$$f(t) \equiv 0 \pmod{p}.$$

This equation can be solved with trial and error for small primes, but for larger primes one should use a Berlekamp approach [6] or a random splitting technique [3]. One speed-up is achieved if one finds the multiplicities of the roots. For example, if  $f(t) \equiv t^4 \pmod{2}$ , then if  $2|f(t)$ ,  $16|f(t)$ . When sieving we add the number of bits of  $p$  (or any logarithm of  $p$ ) to a location in the sieving interval corresponding to some  $(u, v)$  such that  $p|Q(u, v)$ . If the corresponding root of  $f(t)$  has multiplicity  $s$ , we can instead add  $s$  times the number of bits to that location. This will make more sense when we describe sieving.

### 3.2 Sieving

One very nice attribute of SQS is that we only need to find roots of  $f(t)$  once, so it is similar to SIQS in that respect. Suppose the roots of  $f(t) \pmod{p}$  are given by  $t_{ij}$ , where  $i$  ranges over the (positive) primes  $p_i \in FB$ , and  $0 \leq j \leq 4$ . We will fix  $u$ , beginning with  $u = 1$ , and sieve over some interval  $[-M, M]$ , so that the elements in the sieving interval will represent values of  $v$ . For each  $p_i \in FB$ , add  $s_{ij} \log p_i$  to location  $v \in [-M, M]$  if  $v = t_{ij} \pm kp$  for some  $k \in \mathbb{Z}$  and  $t_{ij}$  is a root of  $f \pmod{p_i}$  of multiplicity  $s_{ij}$ . This is

based off of the property of polynomials that if  $p|f(t)$ , then  $p|f(t + kp)$  for all  $k \in \mathbb{Z}$ .

If  $u > 1$ , we make a slight change. Notice that if  $Q(1, t_{ij}) = f(t_{ij}) \equiv 0 \pmod{p_i}$ , then

$$Q(u, ut_{ij}) = u^4 f(ut_{ij}u^{-1}) = u^4 f(t_{ij}) \equiv 0 \pmod{p_i}.$$

So if  $u > 1$ , take  $r_{ij} = ut_{ij} \pmod{p_i}$ , and sieve as done in the  $u = 1$  case, but replace  $t_{ij}$  with  $r_{ij}$ .

I also mention that since we are sieving with a quartic polynomial, we can have up to four roots and therefore up to four places around which we may sieve. Roots mean small residues of  $Q(u, v)$ ! We have set up a sieving interval at  $[-M, M]$  since we know that residues of  $Q(u, v)$  will be small in a neighborhood of 0. So we can expect that there will be one root in the sieving interval, though not necessarily an integer. In general, using other roots to sieve around doesn't look promising. We illustrate this with a typical example.

Most applications of SQS will be to factor integers of the form  $m^3 + a_0$ . In this case

$$Q(u, v) = -4a_0mu^4 - 8a_0u^3v - 4muv^3 + v^4.$$

Notice that

$$Q(u, 0) = -4a_0mu^4,$$

and

$$\begin{aligned} Q(u, -a_0^{1/3}u) &= -4a_0mu^4 + 8a_0^{4/3}u^4 + 4a_0mu^4 + a_0^{4/3}u^4 \\ &= 9a_0^{4/3}u^4 \end{aligned}$$

if  $a_0 > 0$  and

$$\begin{aligned} Q(u, -2a_0^{1/3}u) &= -4a_0mu^4 + 16a_0^{4/3}u^4 + 32a_0mu^4 + 16a_0^{4/3}u^4 \\ &= 8(4a_0^{4/3} + 7a_0m)u^4 \end{aligned}$$

if  $a_0 < 0$ .

These have opposite signs, so there must be a root in  $(-a_0^{1/3}u, 0)$  if  $a_0 > 0$  and in  $(0, -2a_0^{1/3}u)$  if  $a_0 < 0$ . This last case assumes that  $m > \frac{-8a_0^{1/3}}{7}$ ,

which will be true in our case. Certainly these intervals are within our sieving intervals.

A quick calculation with Mathematica tells us that  $\frac{\partial Q}{\partial v}(u, v) = 0$ , i.e.  $v^3 - 3muv^2 - 2a_0u^3 = 0$ , has exactly one real solution, so there is only one local extremum. This means that there is exactly one other root, and is going to be very close to  $v = 4mu$ , since  $Q(u, 4mu) = -36a_0mu^4$ , which is small, and a small increase in  $v$  will switch the sign of  $Q(u, v)$ . This value of  $Q(u, v)$  is certainly the size of those we sieve for in  $[-M, M]$  as long as  $u$  is not too large, but notice that  $(u, 4mu) = u$ , which is not 1 unless  $u = 1$ . Also notice that  $\frac{\partial Q}{\partial v}(u, 4mu) = 64m^3u^3 - 8a_0u^3$ , which is larger than  $n$  itself! So any values of  $Q(u, v)$  around  $v = 4mu$  will be much too large to sieve for, since the quartic grows so fast. If we are desperate we can always take our value  $Q(1, 4m) = -36a_0m$ , which should be smooth since we'll likely know the factorization of  $m$  and  $a_0$ .

For more general applications of this method, it is not clear if it will be of any benefit to sieve around the other roots of  $Q(u, v)$ . It will probably be the case that we find a root via Newton's Method, pick out a residue near the root, and immediately trial divide it if the residue is sufficiently small to test for smoothness. Actually graphing out a couple of the sieving polynomials may give us a good idea if this will even work.

### 3.3 Smooth Residues

Smooth residues of  $Q(u, v)$  will have an accumulated value of about  $\log |Q(u, v)|$  at location  $v \in [-M, M]$ , and

$$\begin{aligned} \log |Q(u, v)| &= \log | -4u(a_0u^3 + a_1u^2v + a_2uv^2 + v^3)m \\ &\quad + (a_1^2 - 4a_0a_2)u^4 - 8a_0u^3v - 2a_1u^2v^2 + v^4 | \\ &\geq \log m + \log |4u(a_0u^3 + a_1u^2v + a_2uv^2 + v^3)| \\ &\geq \log m + \log |4u(a_0u^3 + a_1u^2v + a_2uv^2 + v^3)| - T \log p_{max} \end{aligned}$$

where  $p_{max}$  is the largest prime in  $FB$  and  $1 < T < 2$  is set depending on the size of large primes we may want to look for. The particular logarithm is the same logarithm that we used in the sieving step. This motivates us to set a threshold of

$$E(u) = \log m + \log 4a_0 + 3 \log u - T \log p_{max},$$

since we will likely have  $a_1 = a_2 = 0$ . If necessary, we can always set different thresholds within a sieving interval depending on different ranges of  $v$ . As usual when sieving is involved, experimentation is usually the best solution.

When a potential smooth value is discovered, we check for smoothness by actually factoring it. One obvious approach is to use trial division: divide by every prime in the factor base until we run out or the smooth number has been factored. In [2], Bernstein gives a comprehensive survey of other techniques for smoothness testing and presents a new method of his own, which he claims is faster than previous methods. Standard tricks involve using Pollard's Rho or ECM to pick out the small factors of a number. Wagstaff has also suggested that SQUFOF is even more effective at factoring 18 digit integers with two large prime factors.

To extend our definition of smoothness, we allow for numbers to be smooth with the exception of large primes. If  $p_{max}$  is the largest prime in  $FB$ ,  $Q(u_i, v_j) = L \prod_{p_i \in FB} p_i^{e_i}$ , and  $p_{max} < L < p_{max}^2$ , then we know that  $L$  is prime. We then store  $L$  with the hope that another partially-smooth  $Q(u, v)$  will have  $L$  as a factor. In practice many people will only store such  $L$  provided  $L < 128p_{max}$ , both for storage reasons and since the chance of finding two large primes beyond that bound is quite small. We can also check if  $p_{max}^2 < L < p_{max}^3$ , for then  $L$  is either prime or the product of two large primes. We only deal with the latter case, and factor  $L = P_1 P_2$  using ECM. It is a lot more complicated dealing with this case, but the basic idea is to let each large prime be a vertex in a graph, with 1 as an additional vertex. An edge is created between 1 and a large prime  $P$  for the single large prime case, and an edge is created between  $P_1$  and  $P_2$  in the double large prime case. We can use primes when a cycle in the graph appears.

### 3.4 Linear Algebra

We form the matrix and solve it as we do with QS and its variants. Each row corresponds to a smooth number and the columns correspond to the primes in the factor base, along with the large primes that were used. In each row, we put in the power  $\pmod{2}$  of the prime that divides the corresponding smooth number. Thus our matrix is over  $GF(2)$ . The next goal is to find a kernel,  $K$ , of the matrix, which will give us a solution. If  $\vec{k} \in K$ , then we have a possible solution, for we have set up the congruence:

$$x^2 = \prod x(u_i, v_j)^2 \equiv \prod Q(u_i, v_j) = \prod p_i^{2e_i} = y^2 \pmod{n},$$

where the nonzero entries of  $\vec{k}$  pick out which of the  $(u_i, v_j)$  pairs we are concerned with. This sets up the potentially final calculation

$$\gcd\left(\prod x(u_i, v_j) \pm \prod p_l^{e_l}, n\right).$$

If this is nontrivial, then we have a nontrivial factor of  $n$ , and we're done.

One method to speed up the linear algebra is Gaussian elimination, which is done in two stages. The motivation is this: in each column, we need at least two nonzero entries in order for that column to be of any use. So if there are no entries in a particular column, we can throw that column out. If there is exactly one, we throw out that column and row the one entry appears in, since it is impossible for that row to be used in forming the kernel. That is the first stage. The second stage is similar. If there is a column with exactly two nonzero entries, we combine the two rows with those two entries, and eliminate the column since the column will now be empty. Once the second stage is done, we go back and check if we can repeat the process.

After Gaussian elimination has been performed, we then find the kernel using Wiedemann's Algorithm [7] over  $GF(2)$ .

This concludes the description of the algorithm. We now analyze the running time.

## 4 Complexity Analysis

Recall that an integer,  $N$ , is  $B$ -smooth if every prime dividing  $N$  is less than  $B$ . Let

$$\Psi(x, y) = \#\{N \leq x \mid N \text{ is } y\text{-smooth}\}.$$

It is known that

$$\Psi(x, y) \sim xu^{-u}, u = \frac{\log x}{\log y}$$

and that

$$\Psi(n^\beta, L_n[\alpha]) = n^\beta L_n \left[ -\frac{\beta}{2\alpha} + o(1) \right],$$

where

$$L_n[\alpha] = e^{\alpha(\log n)^{1/2}(\log \log n)^{1/2}}, 0 < \beta \leq 1, \text{ and } 0 < \alpha \leq 1.$$

If  $B = L_n[\alpha]$  and  $\beta = 1/3$ , then

$$\Psi(n^{1/3}, L_n[\alpha]) = n^{1/3} L_n \left[ -\frac{1}{6\alpha} + o(1) \right].$$

At this point we need to make a couple of assumptions, which are unproven. First, we assume that there exists a constant  $0 < c < 1$  such that there are at least  $c\pi(L_n[\alpha])$  primes  $p_i \leq L_n[\alpha]$  for which  $f(t)$  has a root  $(\text{mod } p_i)$ , where  $f(t)$  is as defined in section 3.1. From experimental data, it seems that this constant will be greater than  $1/2$ . Thus we assume that the factor base will have  $L_n[\alpha]$  primes, including  $-1$ .

Our second assumption is that the values of  $Q(u, v)$  will behave like random integers in the interval  $[1, n^{1/3+o(1)}]$ . If this is true then we expect to find an  $L_n[\alpha]$ -smooth integer every  $L_n[\frac{1}{6\alpha} + o(1)]$  locations in our sieve array. Since we need to find  $L_n[\alpha + o(1)]$  of these, we need to sieve  $L_n[\alpha + \frac{1}{6\alpha} + o(1)]$  locations. The smoothness testing can be done with ECM, so each iteration will run in  $L_n[o(1)]$  time. Thus the entire sieving time has complexity:

$$L_n \left[ \alpha + \frac{1}{6\alpha} + o(1) \right].$$

The linear algebra will be done on an  $L_n[\alpha] \times L_n[\alpha]$  sparse matrix with  $o(1)$  entries in each row, so the complexity of the linear algebra is

$$L_n[2\alpha + o(1)].$$

The total complexity is therefore

$$\max \left\{ L_n \left[ \alpha + \frac{1}{6\alpha} + o(1) \right], L_n[2\alpha + o(1)] \right\}.$$

The optimal value for  $\alpha$  is then given by the solution to  $\alpha + \frac{1}{6\alpha} = 2\alpha$ . So  $\alpha = \frac{\sqrt{6}}{6}$ , and the total asymptotic running time of SQS is

$$L_n \left[ \frac{\sqrt{6}}{3} + o(1) \right] \approx L_n[0.8165 + o(1)],$$

which beats the asymptotic running time of QS for  $n$  sufficiently large.

From this analysis we obtain optimized parameters for the range of values of  $u$  and  $v$ . Considering  $Q(u, v)$ , we should want  $1 \leq u, |v| \leq M = L_n[\gamma]$ , with  $(u, v) = 1$ , so the number of such pairs is

$$2 \sum_{u=1}^M \phi(u) = \frac{6}{\pi^2} M^2 + O(M \log M) = L_n[2\gamma + o(1)].$$

So we choose  $\gamma = \frac{1}{2} \frac{\sqrt{6}}{3} = \frac{\sqrt{6}}{6}$ , and

$$M = L_n \left[ \frac{\sqrt{6}}{6} + o(1) \right].$$

Notice that we also choose the same value for the bound  $B$  for the size of primes in  $FB$ .

## 5 Generalizing SQS

By taking  $n$  to be of the form we did, we were able to sieve for smooth integers that were comparable in size to  $n^{1/3}$ . It is natural to ask then if this method can be generalized to work with integers of the form  $n = m^4 + a$ , and if this will allow us to work with integers of size about  $n^{1/4}$ . Even more generally, will we be able to express  $n$  in the form  $n = m^d + a$  and sieve for numbers which are of size approximately  $n^{1/d}$ ? In his paper, Zhang mentioned that it is improbable that we will be able to generalize further. This section answers more specifically how he came to that conclusion and explores possibilities that he may not have considered.

### 5.1 $n = m^4 + a$

We first consider the simplest case:  $n = m^4 + a_3m^3 + a_2m^2 + a_1m + a_0$ . Let  $x = b_2m^2 + b_1m + b_0$  so that

$$x^2 = b_2^2m^4 + 2b_1b_2m^3 + (b_1^2 + 2b_0b_2)m^2 + 2b_0b_1m + b_0^2.$$

With substitutions similar to what we did before we get

$$x^2 \equiv (2b_1b_2 - a_3b_2^2)m^3 + (b_1^2 + 2b_0b_2 - a_2b_2^2)m^2 + (2b_0b_1 - a_1b_2^2)m + (b_0^2 - a_0b_2^2).$$

So we wish to find parametrizations to

$$\begin{cases} 2b_1b_2 - a_3b_2^2 = 0 \\ b_1^2 + 2b_0b_2 - a_2b_2^2 = 0. \end{cases}$$

If we simplify further and let  $a_1 = a_2 = a_3 = 0$ , then we are forced to choose  $b_2 = 0$  or  $b_1 = 0$  to satisfy the first equation, and are again forced

with a trivial solution for the second. Attempts to find solutions to the more general equation will also result in rather trivial solutions.

Thus we change  $x = b_3m^3b_2m^2 + b_1m + b_0$ . If we find  $x^2$  and make the substitutions

$$m^6 \equiv -(a_3m^5 + a_2m^4 + a_1m^3 + a_0m^2 \pmod{n}),$$

$$m^5 \equiv -(a_3m^4 + a_2m^3 + a_1m^2 + a_0m \pmod{n}),$$

$$m^4 \equiv -(a_3m^3 + a_2m^2 + a_1m + a_0 \pmod{n})$$

we get:

$$\begin{aligned} x^2 &\equiv (b_0^2 - a_0b_2^2 - 2a_0b_1b_3 + 2a_0a_3b_2b_3 + a_0a_2b_3^2 - a_0a_3^2b_3^2) \\ &+ (2b_0b_1 - a_1b_2^2 - 2a_1b_1b_3 - 2a_0b_2b_3 + 2a_1a_3b_2b_3 + a_1a_2b_3^2 + a_0a_3b_3^2 - a_1a_3^2b_3^2)m \\ &+ (b_1^2 + 2b_0b_2 - a_2b_2^2 - 2a_2b_1b_3 - 2a_1b_2b_3 + 2a_2a_3b_2b_3 - a_0b_3^2 + a_2^2b_3^2 + a_1a_3b_3^2 - \\ &a_2a_3^2b_3^2)m^2 \\ &+ (2b_1b_2 - a_3b_2^2 + 2b_0b_3 - 2a_3b_1b_3 - 2a_2b_2b_3 + 2a_3^2b_2b_3 - a_1b_3^2 + 2a_2a_3b_3^2 - a_3^3b_3^2)m^3 \\ &\pmod{n}. \end{aligned}$$

We would like to eliminate the  $m^2$  and  $m^3$  terms so that  $x^2 \equiv c_0 + c_1m \pmod{n}$  for some  $c_0$  and  $c_1$ . Thus we wish to solve the system:

$$\begin{aligned} b_1^2 + 2b_0b_2 - a_2b_2^2 - 2a_2b_1b_3 - 2a_1b_2b_3 + 2a_2a_3b_2b_3 - a_0b_3^2 + a_2^2b_3^2 + a_1a_3b_3^2 - a_2a_3^2b_3^2 &= 0 \\ 2b_1b_2 - a_3b_2^2 + 2b_0b_3 - 2a_3b_1b_3 - 2a_2b_2b_3 + 2a_3^2b_2b_3 - a_1b_3^2 + 2a_2a_3b_3^2 - a_3^3b_3^2 &= 0. \end{aligned}$$

If this can be accomplished, then we have a potential speed-up of SQS.

Simplifying further, if we let  $a_1 = a_2 = a_3 = 0$ , then we only need to parameterize the  $b_i$  so that:

$$b_1^2 + 2b_0b_2 - a_0b_3^2 = 0$$

$$b_1b_2 + b_0b_3 = 0.$$

However, this can only be solved if  $b_1^2 + 2b_0b_2 = a_0b_3^2$ , so we'd need equal powers of  $a_0$  on either side. This can only happen if  $a_0$  is square. We also have a problem with the factor of 2 in the first equation, so  $a_0$  will have to be an even power of 2. For example, if  $a_0 = 4$ , we have:

$$b_1 = -\frac{b_0b_3}{b_2} \quad b_0^2b_3^2 + 2b_0b_2^3 - 4b_2^2b_3^2 = 0,$$

so

$$b_0 = \frac{-2b_2^3 \pm \sqrt{4b_2^6 + 16b_3^4b_2^2}}{2b_3^2}.$$

So  $b_2^4 + 4b_3^4$  needs to be a square, but any solutions to this are very large, so this will not be conducive to finding small residues. So finding a parametrization for the more general case seems to be unlikely.

## 5.2 $n = m^5 + a$

Why not give it a shot? It's possible that  $d = 4$  was just a bad choice. So suppose that  $n = m^5 + a$  and  $x = b_3m^3 + b_2m^2 + b_1m + b_0$ . If we try parametrizing the  $b_i$  so that  $x^2 \pmod{n}$  is small, then we only get a trivial parametrization, so next we try

$$x = b_4m^4 + b_3m^3 + b_2m^2 + b_1m + b_0.$$

In this case we get:

$$\begin{aligned} x^2 &\equiv b_0^2 - 2ab_2b_3 - 2ab_1b_4 + (2b_0b_1 - ab_3^2 - 2ab_2b_4)m \\ &+ (b_1^2 + 2b_0b_2 - 2ab_3b_4)m^2 + (2b_1b_2 + 2b_0b_3 - ab_4^2)m^3 \\ &+ (b_2^2 + 2b_1b_3 + 2b_0b_4)m^4 \pmod{n}. \end{aligned}$$

So we want to solve the system:

$$b_1^2 + 2b_0b_2 - 2ab_3b_4 = 0$$

$$2b_1b_2 + 2b_0b_3 - ab_4^2 = 0$$

$$b_2^2 + 2b_1b_3 + 2b_0b_4 = 0.$$

Combining the equations into one we get:

$$2b_1b_2b_3 + 4b_0b_3^2 + b_1^2b_4 - b_2^3 = 0.$$

If we treat this as a quadratic equation in  $b_1$  :

$$b_1 = \frac{-2b_2b_3 \pm \sqrt{4b_2^2b_3^2 - 4b_4(4b_0b_3^2 - b_2^3)}}{2b_4},$$

We need to factor  $b_4$  out of the square root and cancel it out of the denominator. Substituting  $b_0 = vb_4, b_2 = ub_4, b_4 = wb_3$  will do the trick:

$$b_1 = -ub_3 \pm b_3\sqrt{u^2 + u^3w^2 - 4v}.$$

So all we need to do from here is find integer values of  $\sqrt{u^2 + u^3w^2 - 4v}$  and we're done. Some solutions  $(u, v, w)$  to this is are  $(1, 1, 2)$ ,  $(1, -1, 2)$ ,  $(1, -5, 2)$ , and  $(1, -11, 2)$ , so small nontrivial solutions do exist.

In fact, notice a pattern with those solutions. The  $v$ -coordinate is in a quadratic progression. If  $u = 1$ , and  $w > 1$ , then  $v = -y^2 + y + w^2$ ,  $y > 0$  is a solution, since  $u^2 + 4u^3w^2 - 4v = 1 + 4w^2 - 4(-y^2 + y + w^2) = 1 - 4y + 4y^2$  is indeed a square. Similarly, if  $u = 2$ , we can parametrize solutions:

$$(2, -y^2 + 2y + 2w^2, w), y > 0, w > 0.$$

And if  $u = 3$ , we have two cases:

$$\left(3, -y^2 + 2y + \frac{u^2 + u^3w^2}{4} - 1, w\right), w \equiv 1 \pmod{2}$$

$$\left(3, -y^2 + y + \frac{u^2 + u^3w^2 - 1}{4}, w\right), w \equiv 0 \pmod{2}.$$

From these we can see how to get a square out of  $u^2 + u^3w^2 - 4v$ . There are three cases to consider:  $u^2 + u^3w^2 \equiv 0, 1, 2 \pmod{4}$ . If:

$$u \equiv 0 \pmod{4} \text{ or } (u \equiv 3 \pmod{4} \text{ and } w \equiv 1 \pmod{2}), u^2 + u^3w^2 \equiv 0 \pmod{4}$$

$$u \equiv 1, 3 \pmod{4} \text{ and } w \equiv 0 \pmod{2}, u^2 + u^3w^2 \equiv 1 \pmod{4}$$

$$u \equiv 1 \pmod{4} \text{ and } w \equiv 1 \pmod{2}, u^2 + u^3w^2 \equiv 2 \pmod{4}.$$

So the substitutions are:

$$(u, -y^2 + 2y + \frac{u^2 + u^3w^2}{4} - 1, w), u^2 + u^3w^2 \equiv 0 \pmod{4}$$

$$(u, -y^2 + y + \frac{u^2 + u^3w^2 - 1}{4}, w), u^2 + u^3w^2 \equiv 1 \pmod{4}$$

$$(u, -y^2 + y + \frac{u^2 + 4u^3w^2 - 1}{4}, w), u^2 + u^3w^2 \equiv 2 \pmod{4},$$

where  $u$  is any integer, but  $w, y > 0$ . making these substitutions, we have the parametrizations of the  $b_i$ :

$$b_0 = \left(-y^2 + 2y + \frac{u^2 + u^3w^2}{4} - 1\right) wt, u^2 + u^3w^2 \equiv 0 \pmod{4}$$

$$\begin{aligned}
b_0 &= \left( -y^2 + y + \frac{u^2 + u^3w^2 - 1}{4} \right) wt, \quad u^2 + u^3w^2 \equiv 1 \pmod{4} \\
b_0 &= \left( -y^2 + y + \frac{u^2 + 4u^3w^2 - 1}{4} \right) wt, \quad u^2 + u^3w^2 \equiv 2 \pmod{4} \\
b_1 &= (-u \pm 2(y-1))t, \quad u^2 + u^3w^2 \equiv 0 \pmod{4} \\
b_1 &= (-u \pm (2y-1))t, \quad u^2 + u^3w^2 \not\equiv 0 \pmod{4} \\
b_2 &= uwt \\
b_3 &= t \\
b_4 &= wt.
\end{aligned}$$

I still have to double check to make sure the parametrizations work, but I'm pretty sure they do. So if we make the substitutions into the equation  $x^2 \equiv Q \pmod{n}$ , for  $u^2 + u^3w^2 \equiv 0 \pmod{4}$  we get:

$$\begin{aligned}
x(t, u, w, y)^2 &= (b_3m^3 + b_2m^2 + b_1m + b_0)^2 \equiv \\
Q(t, u, w, y) &= b_0^2 - 2ab_2b_3 - 2ab_1b_4 + (2b_0b_1 - ab_3^2 - 2ab_2b_4)m \pmod{n}.
\end{aligned}$$

The other cases modulo 4 are similar.

This looks very promising. So if this is correct, then we have a sieving polynomial with four variables, whose residues are of the order  $O(n^{1/5+o(1)})$ , since we have chosen  $m \sim n^{1/5}$ . We will call this the Extended SQS, or ESQS.

### 5.3 Efficiency of ESQS

The dominant term in the sieving polynomial  $Q(t, u, w, y)$  is going to be  $2b_0b_1m$ , or  $2ab_2b_4m$ , if  $a$  is excessively large. If all the parameters are of order  $n^\epsilon$ , then the dominant term will be  $O(n^{1/5+9\epsilon})$ . So for this method to be faster than QS, the parameters all need to be bounded by  $O(n^{1/30})$ , but with four variables to deal with, we have  $O(n^{1/7.5})$  locations to sieve, which is better than SQS. Of course we will probably have to deal with the variables being coprime or pairwise coprime, which will shrink down the possible values with which we may sieve.

As for running time, in the complexity analysis above we noted that with residues of size  $n^{1/3}$ , we would find a smooth integer about every  $L_n \left[ \frac{1}{6\alpha} + o(1) \right]$  sieve locations. In this case, if residues are of size  $O(n^{1/5+\epsilon})$ ,

then we will find a smooth integer about every  $L_n \left[ \frac{1}{10\alpha} + o(1) \right]$  sieve locations. This gives us  $\alpha = \frac{1}{\sqrt{10}}$ , and a running time of  $L_n \left[ \frac{\sqrt{10}}{5} + o(1) \right] \approx L_n [0.632 + o(1)]$ , which is much better than QS or SQS!

A potential drawback is that  $Q(u, v, w, y)$  is a sextic in  $u$  and  $w$  because of the  $b_0^2$  term, so those two variables should be kept as small as possible, and the bulk of the sieving should be done with  $t$  and  $y$ . There are also slightly different sieving polynomials depending on  $u^3 + u^2w^3 \pmod{4}$ , so it would be wise to choose  $u$  and  $w$  for sieving in which that congruence is 0, since more values of  $u$  and  $w$  satisfy that condition.

## 5.4 Future Work

Since there are so many polynomials to sieve with, we need to look into how easy it is to switch polynomials. It certainly does not appear that every term in  $Q$  will have the same degree, as we saw with SQS, so that will be work of future investigation.

Since we had so much luck with the case  $n = m^5 + a$ , this begs the question if we can further generalize to  $n = m^5 + a_4m^4 + \dots + a_0$ , and if it is possible that the coefficients won't get in the way so much so that we can use this method to factor general integers. It is also possible that with a higher degree of our choice of  $x$  that we could get a nicer set-up. Lastly, it does look promising for us to consider making an even further extended ESQS to consider  $n = m^7 + a$ . For some reason I think that the case  $n = m^4 + a$  didn't work because 4 is composite, so trying with prime degrees may be the thing to look for. Of course the end goal of this investigation is to see if perhaps this can be adapted to factor general integers, such as RSA numbers faster than NFS can. We know that as  $n$  increases, that NFS is the fastest of all general factoring algorithms, but if we can find an algorithm with running time  $L_n[\alpha + o(1)]$ , with  $\alpha$  very small, say 1/2 or even less, then that will suit our purposes just fine. My guess is that we can get  $\alpha$  arbitrarily small, but we may have to experiment more with other forms for a sieving polynomial.

## 6 Implementations Details

An attempt was made to implement SQS. Included with this paper is the source code and an initialization file:

sqs — The executable.

sqs.cc — The source code.

sqs.h — The header files with global variables.

sqs.ini — This file is read in after the program begins. The variables are:

$n = \frac{m^3 + A_2 m^2 + A_1 m + A_0}{k}$ , where  $k$  is a multiplier, and  $n$  is the number to be factored.

TOLERR = A fudge factor for the bound we accept for recognizing smooths.

M = Half the length of the sieve array. FB = Size of the factor base.

SKIP = The number of small primes we skip when sieving.

LPFAC = We accept large primes less than LPFAC times the largest prime in the factor base.

DPFAC = We accept two large primes whose product is less than DPFAC times the largest prime in the factor base.

ER = Extra relations. We need at least 1 in order to guarantee a possible factorization.

KINT = Read out whenever KINT new smooths have been found.

VERBOSE = 0 if you don't want read-outs, 1 if you do.

As mentioned, the residues of  $Q(u, v)$  will become larger than the residues that are dealt with for QS when  $M > n^{1/24}$ . Since  $0 < u \leq M$ ,  $|v| \leq M$ , and  $(u, v) = 1$ , we want to sieve over than  $\frac{12n^{1/12}}{\pi^2}$   $(u, v)$  pairs. Choosing optimal parameters, we will find a smooth element about every  $L_n \left[ \frac{\sqrt{6}}{6} \right]$  locations with about  $L_n \left[ \frac{\sqrt{6}}{6} \right]$  primes in the factor base. So the first question is of course how practical is this method for use as a stand-alone factoring algorithm. Given the numbers above, the cross-over point for which we can sieve the  $\frac{12n^{1/12}}{\pi^2}$  locations and expect to find  $L_n \left[ \frac{\sqrt{6}}{6} \right]$  smooth elements to fill in our matrix is about  $n = 10^{266}$ ! At this point we have a factor base of size  $1.3 \cdot 10^{11}$  and sieve about  $1.8 \cdot 10^{22}$  locations!

In practice though we don't use as large a factor base as suggested by the complexity analysis and the sieving intervals can be a bit larger. Otherwise we would not be able to fit a matrix that large on a computer to solve. For example, in my own toy examples, I attempted to factor  $n = 2^{64} + 1$ . Both  $M$  and the size of the factor base were chosen to be 500. I allowed for large primes and double large primes to be used, and used a multiplier  $k = 4$  so that  $kn = (2^{22})^3 + 4$ . With these parameters I was only able to obtain about 380 smooth relations for the matrix. Beyond the sieving interval, residues became much too large to deal with. So for numbers of this size I am certain

that a hybrid system would have to be used.

Hoping for better results with a larger  $n$ , I tried  $n^{128} + 1$  with a multiplier of 2, but was unable to find good enough parameters to even get 150 relations. Perhaps I kept setting my sieving interval too small or made my factor base too small. It was quite disappointing. In this case, with  $M = 200$ , residues are still quite small, so we are able to peel off a number of relations for any size factor base. In order to be effective, I believe we would still need to use a hybrid system at this size. This is only a guess, but it probably won't be until we need to factor integers of about 100 digits that this method will work.

It is unknown how efficient the ESQS will be in practice.

## References

- [1] Apostol, T. *Introduction to Analytic Number Theory*, Springer-Verlag, New York, (1976).
- [2] Bernstein, D. *How to Find Small Factors of Integers*, Math. Comp. (to appear), Available from <http://cr.yp.to/papers.html>, (2002).
- [3] Cohen, H. *A Course in Computational Algebraic Number Theory*. Springer-Verlag, Berlin, (1993).
- [4] Crandall, R. and Pomerance, C. *Prime Numbers, A Computational Perspective*, Springer, New York, (2001).
- [5] Lenstra, A. and Lenstra, H. (eds.), *The Development of the Number Field Sieve*, Lecture Notes in Mathematics, 1554, Springer-Verlag, Berlin, (1993).
- [6] Shoup, V. *A New Polynomial Factorization Algorithm and its Implementation*, J. Symbolic Comp. **20**, 363-397, (1995).
- [7] Wiedemann, D. "Solving Sparse Linear Equations over Finite Fields," *IEEE Trans. Inform. Theory*, **32**, 54-62 (1986).
- [8] Zhang, M. *Factorization of the Numbers of the form  $m^3 + c_2m^2 + c_1m + c_0$* , Algorithmic Number Theory (Portland, OR, 1998), 131-136, Lecture Notes in Computer Science, 1423, Springer, Berlin, (1998).