J. Math. Crypt. **1** (2007), 105–124

DOI 10.1515 / JMC.2007.007

Optimal Parameterization of SNFS

Robert D. Silverman

Communicated by Rainer Steinwandt

Abstract. The Special Number Field Sieve factoring algorithm has a large number of parametric choices, each of which can affect its run time. We give guidelines for these choices along with a discussion of useful coding optimizations. We also give a theoretical argument which proves that the choice of sieving region that has been used so far in successful factorizations is not optimal and show how to obtain an improved sieve region. The improvement has yielded a 15% speed increase in practice.

Keywords. Factoring, GNFS, SNFS, number field sieve, parameter selection.

AMS classification. 11Y05, 11Y16, 65K10, 68W15, 68W40.

1 Introduction and notation

In this paper all logarithms are natural unless otherwise noted. The Number Field Sieve is the fastest known algorithm for factoring large integers and has two versions: the General NFS and the Special NFS. The former factors arbitrary integers N in time $L(N,c) = \exp((c+o(1))((\log N)^{1/3}(\log\log N)^{2/3}))$ and space $L(N,c)^{1/2}$, with $c = (64/9)^{1/3}$. The latter factors integers of special form such as $N = h_1a^n + h_2b^n$ for small integers h_1, h_2 in time $L(N,c)$ and space $L(N,c)^{1/2}$, with $c = (32/9)^{1/3}$. See [8] for a discussion of integers susceptible to the Special version.

Both methods have a fairly large set of parameters associated with them. GNFS has all of the choices of SNFS, along with the additional choice of finding good polynomials for the method. To eliminate that choice, since it is still an open research topic, this paper discusses optimal parameter selection for SNFS, although methods presented here carry over to GNFS. We assume familiarity with the algorithm, although a brief description is given below to motivate parameter selection.

SNFS works as follows: Given (say) an integer of the form $N = a^n + 1$, start by selecting d, a degree for a polynomial. We will represent N as an instance of a degree d polynomial $f(x)$ with small coefficients evaluated at a point. Typically, for numbers of a size that can be handled by current computers, $d = 4, 5, 6$. Find the nearest multiple wd to n of d, and re-write N as $c_1(a^w)^d + c_2$, perhaps by multiplying N by a small power of a. E.g. for $N = 41^{83}+1$, one may write $N = 41^3(41^{16})^5+1 = 41^3M^5+1$ or perhaps $41^2N = (41^{17})^5 + 41^2 = M^5 + 41^2$. Then a^w is a root of the polynomial mod N. Throughout this paper, we shall denote this root mod N as M. A second, linear polynomial with the same root is then $g(x) = x - M$. When N has a form such as $N = h_1a^n + h_2b^n$, the root M equals a^c/b^c for an appropriate value of c. In this case the linear polynomial becomes $g(x) = b^cx - a^c$ because $(b, N) = 1$. There is a ring

homomorphism ϕ that takes a root α of $f(x)$ to M mod N. Thus, $\phi(\alpha) = M \bmod N$, whence for rational integers (b, a) we have

$$a + bM \equiv \phi(a + b\alpha) \bmod N. \tag{1.1}$$

We shall refer to the left hand (integer) side and the right hand (algebraic) side of this congruence throughout this paper. The algorithm tries to factor both sides of this congruence simultaneously, the left side over rational primes and the right side over prime ideals and units of $Q(\alpha)$. One does this using ordinary integer arithmetic by factoring the norms of $a+bM$ and $a+b\alpha$. Then, given a large number of such factored congruences, one constructs a congruence of the form $A^2 = B^2 \bmod N, A \neq \pm B \bmod N$, from which $GCD(A + B, N)$ and $GCD(A - B, N)$ split N. The details of how $A^2 = B^2 \bmod N$ is constructed are given in numerous other papers [7, 14, 16] and will not be repeated here, except to note that the construction depends upon finding the null space of a very large matrix over $\mathbb{Z}/2\mathbb{Z}$. Thus far, attempts to parallelize this linear algebra have met with some, but not strong success, and solving the matrix remains essentially a serial problem for a very fast single computer with very large memory. Parallel implementations [9] show poor per-processor utilization and have not scaled well. The Number Field Sieve is therefore strongly constrained by space requirements as well as time requirements. Asymptotically, the algorithm takes as long to solve the matrix as it does to find the factored smooth relations. In practice, solving the matrix for numbers that can be factored today takes only a small fraction of the total CPU time, but since it has not yet been possible to scale a parallel solver, solving the matrix can take a substantial fraction of the total elapsed time. For the factorization of RSA-512 [10], sieving was accomplished in parallel in about 8 weeks, while solving the matrix took 10 days on a single Cray. This ratio between elapsed times gets worse as the numbers get larger as long as the parallel matrix reduction cannot be scaled. This paper will concentrate upon minimizing the sieve time.

The algorithm proceeds as follows: Select the sizes k_1 and k_2 of two separate factor bases F and $\mathcal{F}$. F is comprised of rational primes $F = \{2, 3, 5, \ldots, p_{k_1}\}$, and $\mathcal{F} = \{I_1, I_2, I_3, \ldots, I_{k_2}\}$ where the I_j are prime ideals of $Q(\alpha)$ ordered by norm. We put $P_j = \text{norm}(I_j)$ and also require that $f(x) = 0 \bmod P_j$ has a solution for each j. There can be $\deg(f)$ solutions if P_j splits completely in $Q(\alpha)$. Thus, $\mathcal{F}$ is the set of smallest k_2 prime ideals of $Q(\alpha)$ corresponding to linear roots of $f(x)$. We also include $p_0 = P_0 = -1$ to hold the sign.

Solve $g(x) = 0 \bmod p_i$ and $f(x) = 0 \bmod P_j$ for all primes in the two factor bases. This may be readily accomplished by e.g., the Cantor-Zassenhaus algorithm [11]. One then identifies fully factored relations by sieving $\text{norm}(a + bM)$ and $\text{norm}(a + b\alpha)$. In practice, one allows relations fully factored over the factor bases plus perhaps up to three somewhat larger primes per polynomial, which will be referred to as the "large primes". Thus, in the two large prime case, we have for some (b, a)

$$\text{norm}(a + bM) = \prod_{p_i \in F} p_i^{r_i} \cdot B1 \cdot B2 \tag{1.2}$$

where $B1, B2$ are the large primes and

$$\text{norm}(a + b\alpha) = \prod_{I_j \in \mathcal{F}} P_j^{s_j} \cdot B3 \cdot B4 \tag{1.3}$$

where $B3, B4$ are also large primes. Any of $B1, B2, B3, B4$ might turn out to equal 1. As of January 2005, there have been two published instances where three large primes were used in (1.2). This was during the factorizations of $2^{773}+1$ and $2^{811}-1$. However, it is not known whether the parameters used were optimal, so they are not reported herein. We place an upper bound LP on these large primes as a parameter of the algorithm. We sieve all pairs (b, a) over some region of the plane. The sieving is performed by adding suitably scaled approximations of $\log p_i$ and $\log P_i$ at locations (b, a) where the primes are known to divide $a + bM$ and $a + b\alpha$ respectively. Factored relations are then identified by having the accumulated logs exceed a threshold value, which depends on (b, a). The thresholds are different for f and g. The algorithm sieves a value of b for all a in the range $[-a_{\max}, a_{\max}]$, then replaces b with $b + 1$. Updating the starting points for the sieve requires only a single modular addition per prime as b changes and takes almost no time at all. The sieve region is therefore a rectangle in the right half plane. This is what has been used historically, but Section 2 shall prove that this is not optimal. A variation of this procedure, known as the lattice sieve [18] sieves over a region which is an affine transform of this rectangle. The transform is defined in such a way that one of the norms is *a priori* divisible by some chosen moderately large prime. This makes the norm more likely to be smooth, but it also has the effect of increasing the corresponding norm on the other side of the congruence. One can then vary the choice of this 'lattice prime'. Details of the ordinary and lattice sieving procedures may be found in [7, 14]. Once sieving has identified a factored (smooth) relation, the actual factorization can be constructed by a second sieve which stores the primes, rather than accumulating their scaled logs. Typically, one factors the relations by trial division up to some small bound T, then identifies larger primes dividing the norms from the sieve. This is a very worthwhile implementation practice. It can more than double the speed of the algorithm, especially for large factor bases, as opposed to using only trial division.

If N has several known factors it might be worthwhile to factor the cofactor with GNFS, rather than the full value of N with SNFS. Such a choice depends on the size of the cofactor and the relative speed of the GNFS implementation. Trial sieving experiments might be necessary to make this determination. Once the degree has been selected, the parametric choices for the algorithm are:

1. The size of the factor bases, k_1 and k_2 respectively. The norms of the largest primes in the factor bases are denoted as $p_{\max_1}$ and $p_{\max_2}$ respectively. We shall frequently assume that $k = k_1 = k_2$ and $p_{\max} = p_{\max_1} = p_{\max_2}$. We also have $p_{\max} \approx k \log k$ by the Prime Number Theorem.
2. The size and shape of the sieving region.
3. The number of and bound LP on the large primes.
4. The trial division bound T.

5. The threshold values for the sieve.

6. The choice of polynomial, although for SNFS this is very limited.

7. Partitioning of the sieve region.

8. The base for the logarithms. Since the sieving uses single byte approximations for these, the base needs to be large enough so that 8 bits can accomodate the log of the largest norm that will occur in the sieve region.

9. As with the Quadratic Sieve, the 'small prime' variation is effective at speeding sieving. See [19] for a discussion and suggested parameters.

2 The sieving region

Let $G(b, a)$ denote the probability that both norms are simultaneously smooth for the point (b, a). Technically, $G(b, a)$ is either 0 or 1. What is really meant is that $G(b, a)$ is the probability that randomly chosen integers near the two norms are smooth. As with the asymptotic analysis in [14] we assume that the norms behave as random integers with respect to their divisibility properties. We will assume for the moment that the large prime variation is *not* in use. The reason for this is that without the large prime variation we know the minimum number of smooth relations that are needed for the algorithm to succeed. This is $k_1 + k_2 + 1$. However, a particular successful factorization given by (1.2) or (1.3) may or not be useful. In order for one to be useful, we must find other such results with matching large primes. Large primes B_i which only appear once can never be part of a square. Work by Lenstra [15], Lambert [13], and Kovalenko [12] shows that the number of double large prime factorizations needed is Poisson distributed and therefore has variance equal to its mean. Practical experience has shown that the number of two-large-prime factorizations needed is unpredictable up to perhaps a factor of 1.5, but most of this variation comes from changing the degree d. Degree 4 seems to require fewer total successes for an as yet unexplained reason. There is very little variation in practice when both N and d are fixed. See Section 4 for further discussion. For full factorizations, $G(b, a)$ may be derived from a theorem of Canfield, Erdös, and Pomerance [5].

Theorem 2.1. *The probability that an integer y has all of its prime factors less than x is $u^{-u+o(1)}$ where $u = \log y / \log x$. This holds uniformly as $u \to \infty$ and $u < (1 - \epsilon) \log x / \log \log x$.*

A result of Bach and Peralta [1] shows that the $o(1)$ term is quite small, even for numbers of the size that appear as norms in SNFS. Without the large prime variation we may therefore take $G(b, a)$ as $u^{-u}v^{-v}$ where $u = \frac{\log |\text{norm}(a+bM)|}{\log(k_1 \log k_1)}$ and $v = \frac{\log |\text{norm}(a+b\alpha)|}{\log(k_2 \log k_2)}$. Graph #1 shows $G(b, a)$ when $f(x)$ has two real roots. The "ridges" are regions where $-a/b$ is close to a real root of $g(x)$. When $-a/b \simeq \alpha$, the norm is smaller, thus the probability of smoothness is larger. The ridge lies along a vector whose slope in the (b, a) plane equals the real root.

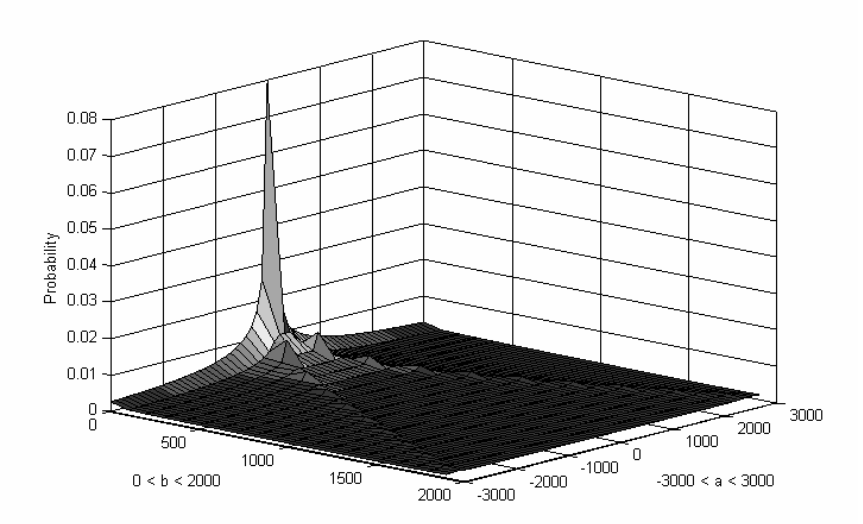

Figure 1. Graph #1: An example of $G(b, a)$

Note. This shape of this graph is in close agreement with that of Figure 3.1 in [7], which gives actual yields for $G(b, a)$ *with* the two large prime variation. Asymptotically, $k_1 = k_2$ is optimal, although one might want to make one larger in practice, depending on circumstances. See the discussion in Section 3 for details. In the analysis that follows and for Theorem 2.2 we assume that the sieve region is symmetric about the b axis. The reasons for this assumption, as well as a suggested technique for relaxing this assumption and creating an asymmetric sieve region, are given following equation (2.9).

For a fixed value of b we sieve from $-a_{\max}$ to $a_{\max}$ with each of the two factor bases. This takes work proportional to

$$W = 2a_{\max}\left(\sum_{p_i \in F} 1/p_i + \sum_{I_j \in \mathcal{F}} 1/\text{norm}(I_j)\right) \tag{2.1}$$

which is approximately

$$W = 4a_{\max} \log\log p_{\max} \tag{2.2}$$

when $k_1 = k_2$. Equation (2.1) represents the total number of additions made to lattice point locations during the sieving phase. This is not, however, the total number of machine instructions. It ignores array address calculations, loop overhead, trial division, the arithmetic involved in splitting the large primes, changing the sieve start locations, and other overhead such as cache misses.

We now suppose that the boundary of an optimal sieve region can be given as a function of b. This function need not be continuous, but it does need to be Riemann

integrable for the following theorem to work. It applies whether or not the large prime variation is used.

Theorem 2.2. *Suppose the boundary of an optimal sieve region is a Riemann integrable function of b. Then at that boundary, the probability of finding a successful relation must be constant.*

Proof. If we allow the sieve interval for a to vary from $-\Omega(b)$ to $\Omega(b)$ and allow b to range from 1 to B, the total work is proportional to

$$4 \log\log p_{\max} \int_1^B \Omega(b)\, db. \tag{2.3}$$

We require a total of at least $k_1 + k_2 + 1$ successes, which yields

$$\int_1^B \int_{-\Omega(b)}^{\Omega(b)} G(b,a)\, da\, db > k_1 + k_2. \tag{2.4}$$

Note that in practice we might want to increase the RHS of (2.4) by a very small amount to account for fundamental units of the field and the quadratic characters used in computing the final square root. We want to minimize (2.3) subject to (2.4). Put $k = k_1 = k_2$ and form the following Lagrangian:

$$\mathcal{L}(\Omega, \lambda, B, k) = 4 \log\log p_{\max} \int_1^B \Omega(b)\, db + \lambda \left[\int_1^B \int_{-\Omega(b)}^{\Omega(b)} G(b,a)\, da\, db - 2k \right] \tag{2.5}$$

where λ is a suitable Lagrange multiplier to be determined. If we perturb Ω by replacing it with $\Omega + \tau\xi$ for a suitable function ξ, a variational equation for Ω is then

$$\frac{\partial}{\partial \tau} \mathcal{L}(\Omega + \tau\xi, \lambda, B, k) \Big|_{\tau=0} = 0 \tag{2.6}$$

whence

$$\frac{\partial}{\partial \tau} \left(4 \log\log p_{\max} \int_1^B \Omega(b) + \tau\xi(b)\, db + \lambda \left[\int_1^B \int_{-\Omega(b)-\tau\xi(b)}^{\Omega(b)+\tau\xi(b)} G(b,a)\, da db - 2k \right] \right) \Big|_{\tau=0} = 0. \tag{2.7}$$

This simplifies to

$$4 \log\log p_{\max} \int_1^B \xi(b)\, db + \lambda \int_1^B \left(G(b, \Omega(b)) + G(b, -\Omega(b)) \right) \xi(b)\, db = 0. \tag{2.8}$$

The first variational principle [2] says that (2.8) must hold for all functions ξ, which in turn requires

$$4 \log\log p_{\max} + \lambda \left(G(b, \Omega(b)) + G(b, -\Omega(b)) \right) = 0 \tag{2.9}$$

for all $1 \le b \le B$ since G is strictly positive. Let $G_1(b,c) = \int_{-c}^{c} G(b,a)\,da$. We assumed throughout that the sieve region is symmetric. The reason for this is as follows. $G(b,\Omega(b))$ is very nearly equal to $G(b,-\Omega(b))$. The smoothness probability for norm$(a+bM)$ does not depend on a, because bM very strongly dominates. At the boundary of much of the sieve region $a \gg b$ and $|\text{norm}(a+b\alpha)|$ is very close to $|\text{norm}(-a+b\alpha)|$. Thus, with a symmetrical sieve region, (2.9) becomes

$$G(b,\Omega(b)) = \frac{-2\log\log\, p_{\max}}{\lambda}. \tag{2.10}$$

Equation (2.10) shows that at the boundary $\Omega(b)$, the probability of success is constant. □

Furthermore, the boundary clearly can not be linear. The optimal sieve region is neither a rectangle with boundary parallel to the b axis, nor an affine transform of the rectangle. Its actual shape is shown in Graph #2.

The above argument has assumed that the optimal region is symmetric around the b-axis. For an odd degree polynomial, one typically expects (and observes) more smooth relations when a and b have opposite signs. The proof above can be modified so that the sieve interval extends from (say) $-\Omega_1(b)$ to $\Omega_2(b)$. However, as previously stated, for much of the sieve region, a is much larger than b, so that Norm$(a+b\alpha) \sim$ Norm$(a-b\alpha)$. Thus, near the sieve boundary the norms are quite close. Numerical experiments were performed which allowed the upper and lower bound of the sieve interval for a to vary. There was very little observable difference between allowing modestly different upper and lower bounds than with assuming $\Omega_1 = \Omega_2$. If we do assume an asymmetric region, then equation (2.10) becomes

$$(G(b,\Omega_1(b)) + G(b,\Omega_2(b))) = \frac{-4\log\log\, p_{\max}}{\lambda},$$

thus showing that a rectangle is suboptimal even when the symmetry assumption is dropped.

We now derive two more equations. The variation in B yields the following equation.

$$\frac{\partial}{\partial B}\mathcal{L}(\Omega,\lambda,B,k) = 4\log\log\, p_{\max}\Omega(B) + \lambda\left(G_1(B,\Omega(B)) - G_1(B,-\Omega(B))\right) = 0. \tag{2.11}$$

By the Prime Number Theorem, we may take $p_{\max} \simeq k\log k$ and the variation in k yields

$$\frac{\partial}{\partial k}\mathcal{L}(\Omega,\lambda,B,k) = \tag{2.12}$$
$$\frac{1+\log k}{\log(k\log k)k\log k}\int_1^B \Omega(b)\,db + \lambda\left(\int_1^B \int_{-\Omega(b)}^{\Omega(b)} \frac{\partial G(b,a)}{\partial k}\,da\,db - 1\right) = 0.$$

Put $b = B$ in (2.9), replace the inequality in (2.4) with equality, and together (2.4), (2.9), (2.11), and (2.13) are four equations in four unknowns: $B, \Omega(B), k,$ and λ. This

can be reduced to three unknowns, as it is possible to derive an optimal value for k independently as shown in Section 3. We could then solve these numerically, substitute the resulting value for λ in (2.9), and solve this for $\Omega(b)$ for $1 \leq b \leq B$. This exercise would be academic, however, since in practice we allow large prime factorizations and the right hand side of (2.4) is unknown in that case.

What is important about this derivation is that (2.10) shows that a rectangular sieve region is not optimal regardless of whether $G(b, a)$ is based on full factorization or large-prime factorization. When the algebraic norm polynomial has no real root, $G(b, a)$ is uniformly decreasing in both b and a. Regardless of the true value of λ, as b increases monotonically, $\Omega(b)$ must decrease monotonically for (2.10) to hold. When $f(x)$ does have a real root, the value of $\Omega(b)$ that satisfies (2.9) lies sufficiently far from a/b that the real roots make no difference. The asymptotic analysis from which $L(N, c)$ is derived results in a square sieve region with $B = a_{\max} = L(N, c)^{1/2}$. Our region is smaller by the ratio $4 \log\log p_{\max} \int_1^B \Omega(b)\, db / L(N, c)$, but determining a closed form for this ratio seems analytically intractable. The difference between our optimal region and the square region is subsumed by the $o(1)$ term in $L(N, c)$. Getting an analytic estimate for the improvement given by the optimal sieve region would require a second main term in the theorem of Canfield, Erdös and Pomerance, and subsequently a main term and error term in place of the $o(1)$ in $L(N, c)$. Thus, the improved sieve region does not affect the asymptotic run time, but as data in Section 7 shall show, it does improve actual performance. Contini has done some experiments on altering the shape of the sieve region. His empirical results [6] also show that a rectangular region is not optimal.

As an example take $N = 11^{113} - 1$, with $f(x) = x^6 - 11$ and $M = 11^{19}$. We shall use this number as a canonical example throughout this paper.

Suppose $B = 500\,000$ and $\Omega(B) = 10^6$. These may not be optimal; we choose a value merely for illustrative purposes. Solving (2.10) for λ yields

$$\lambda \simeq \frac{-2 \log\log p_{\max}}{G(B, \Omega(B))}. \tag{2.13}$$

With $\log\log p_{\max} \simeq 2.6$, we get $\lambda \simeq -8.5 \cdot 10^7$ and the following graph shows an approximation to $\Omega(b)$. Choosing different values for B and $\Omega(B)$ does very little to alter the shape of this graph. We solved (2.10) for each b by binary search on $\Omega(b)$.

As implemented, the optimal boundary will actually be a step-function of b. Efficient implementations partition the sieve interval for each b into pieces that fit in cache. If the cache size is C, then the sieve length is chosen to be $h * C$ for some integer h. The sieve boundary changes most rapidly near the origin. To compute the sieve boundary, we therefore compute $\Omega(b)$ more frequently near the origin and proceed as follows:

1. Select B, the largest anticipated value of b which will be sieved. This value depends upon the size of the factor base, and the size of the large primes, as well as the size of N. Values for B may be approximated by $\sqrt{\mathit{Work}}/20$ where *Work* may be taken from Table 1. $\Omega(B)$ is approximately $10 \cdot B$ for numbers in the range of Table 1.
2. Compute λ and $G(B, \Omega(B))$ using the methods of [20].

3. For $b = 100*j$, $j = 1, \ldots, 100$ and using binary search on $\Omega(b)$, find $\Omega(b)$ to satisfy (2.10), rounding $\Omega(b)$ to the nearest multiple of C.
4. For $b = 10\,000 * j$, $j = 1, \ldots, B/10\,000$ again find $\Omega(b)$ to satisfy (2.10), rounding appropriately.

The step sizes for how frequently $\Omega(b)$ is computed may be adjusted for convenience.

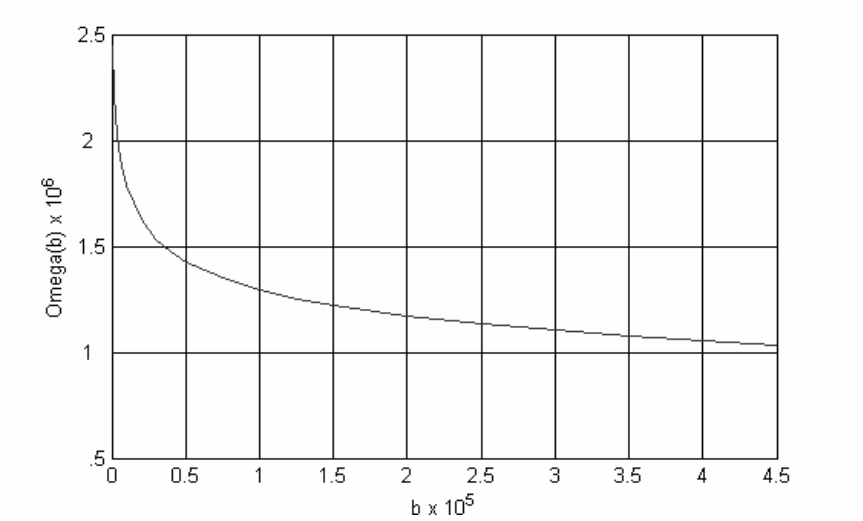

Figure 2. Graph #2: Qualitative behavior of $\Omega(b)$

Geometric Interpretation. Theorem 2.2 results from finding the region of smallest area lying in the (b, a) plane such that the volume over that region and under $G(b, a)$ equals the number of needed relations. It is not surprising that the optimal $\Omega(b)$ follows contours of high probability on the surface. It is clearly more beneficial to sieve in a region of higher probability. The shape of Ω is exactly determined by the intersection of a horizontal plane with $G(b, a)$. The height of that plane is given by the right hand side of (2.10).

In practice, when using two large primes, $\Omega(b)$ can only be determined empirically by trying different values of $\Omega(b)$ for N of fixed size. The optimal value of B is also derived empirically by trying different values for fixed sizes of N. But the graph gives qualitative guidelines for the shape of this region. In Section 7, one of the factorizations used a sieve region whose general shape followed that of Graph #2. It resulted in about a 17% reduction in sieve time over the best rectangular region. Furthermore, (2.9) can be used as a guide for GNFS, as well as SNFS, once the GNFS polynomials have been selected.

To compute points along $\Omega(b)$ one can use the methods of Sorenson [20] for computing G with two large primes. For N of a given size, B and $\Omega(B)$ are derived from

experience. It is easy to solve (2.10) for $\Omega(b)$ by binary search. A high accuracy solution is not needed.

The discussion in Section 5 suggests that for a given b, the sieve line over a should be partitioned into pieces that fit in the L_2 cache of the computer (if it has one). Thus, $\Omega(b)$ is rounded to the nearest multiple of that cache size and becomes a step function. This means that in practice the sieve region consists of a set of rectangles of decreasing width and increasing length. For composites ~ 200 digits near $b \sim 0$, we select $\Omega(b)$ to be quite large; perhaps 100 times the cache size. Near the largest values of b it decreases to about 10 times the cache size. See the data in Section 7.

Note. It takes extensive CPU time to factor numbers of the sizes given in Section 7. Acquiring optimal values for B and $\Omega(B)$ from experience will require CPU resources not available to this author. Equation (2.10) is a recent discovery. Data on possibly optimal values of $\Omega(B)$ was not collected from earlier factorizations.

3 The size of the factor bases and LP bound

We show here that without the large prime variation, the optimal size of the factor base may be derived independently from equations (2.9) and (2.13). The analysis will show that the time complexity function is fairly shallow near its minimum. This means that one can be a little bit off in choosing the size of the factor base without affecting the run time very much. We will also show, however, that if one takes the factor base too small (less than a specific quantity to be shown), then the run time increases faster than exponentially. The penalty for choosing a factor base that is too big is smaller than that for choosing one too small. Also, if the factor base is too large, it is much easier to discard excess relations if we decide to reduce the factor base size before doing the matrix reduction. When using the large prime variation, the optimal size is determined empirically and actual values are suggested in Section 7.

The number of lattice points needed to be examined in the neighborhood of a typical point $P = (b, a)$ on average, for one success, is $1/G(b, a)$. We need at least $2k$ total, so

$$T(k) = 2k/G(b, a) \tag{3.1}$$

represents the total work required. $G(b, a)$ is given by $u^{-u}v^{-v}$ where $u = \log|a + bM|/(\log(k \log k))$ and $v = \log|\text{norm}(a + b\alpha)|/\log(k \log k)$. Over much of the sieve region the norms are approximately equal, so we take $u = v$ for simplification. We also make the approximation $\log(k \log k) \approx \log k$ because $\log \log k$ is quite small. These simplifications make $G(b, a)$ more tractable to deal with. The formal derivation of $L(N, c)$, given in [14] takes $P = (b_{\max}, a_{\max})$ as its typical point to determine the size of the norms that need to be smooth. Choice of a different point, say $P_1 = (b_{\max}/4, a_{\max}/4)$ makes very little difference to the value of u. Indeed, for our canonical example, choosing P gives $u = 59.86/\log k$, whereas choosing P_1 gives $u = 58.48/\log k$. As long as the point is chosen not too close to the origin, the value of $G(b, a)$ does not change too much as shown by Graph #1. The optimal factor

base size is not very sensitive to the selection of P. The graph below shows $T(k)$ for $u = 59.17/\log(k)$ and $5 * 10^5 < k < 2 * 10^8$ corresponding to $b = 500\,000$ from our canonical example.

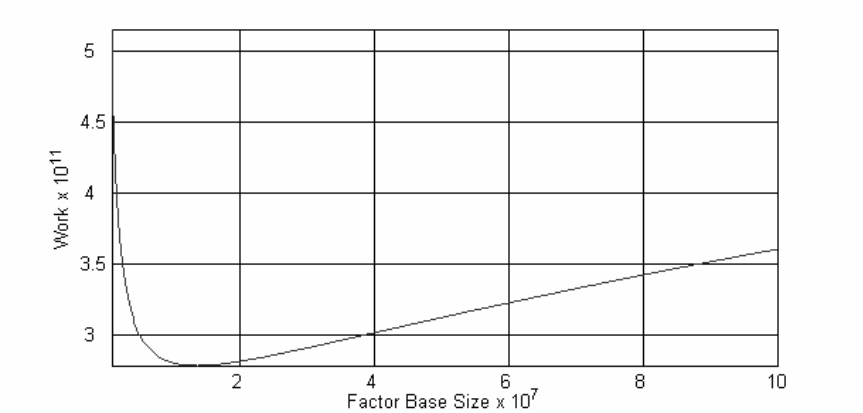

Figure 3. Graph #3: Optimization of Factor Base Size

Graph #4 also shows $T(k)$, but in a region close to the optimum: $k_{\text{opt}} - 10^7 < k < k_{\text{opt}} + 10^7$.

As can be seen, the penalty for choosing a factor base that is too small becomes much larger than the penalty for choosing one too big as soon as one drifts 'reasonably' far from the optimum. We quantify this next.

Let $C = \log(\text{norm}(a_{\max} + b_{\max}M))$. Since we assume that $u \approx v$, we have

$$T(k) = k(C/\log k)^{(2C/\log k)} = kr(k). \tag{3.2}$$

We have $\lim_{k\to 1+} T(k) = \infty$, thus confirming that $T(k)$ must at some point grow faster than exponentially as k decreases below the minimum. Also $\lim_{k\to\infty} r(k) = 1$, so $T(k)$ is asymptotic to k. Thus, the penalty for a factor base that is too big grows linearly with k.

We can determine the exact optimal point and exactly where $T(k)$ starts to grow rapidly with the aid of the Lambert W function, $W(c) = z$, such that $c = ze^z$. Put $y = C/\log k$ and differentiate $T(y)$ logarithmically to get

$$T'(y) = T(y)\left(\frac{-C}{y^2} + 2 + 2\log y\right) = T(y)s(y). \tag{3.3}$$

We set this equal to zero and replace $2\log y$ with $\log C - \log(C/y^2)$. This gives

$$-C/y^2 + 2 - \log(C/y^2) + \log C = 0. \tag{3.4}$$

Put $z = C/y^2$ and exponentiate, giving

$$Ce^2 = ze^z. \tag{3.5}$$

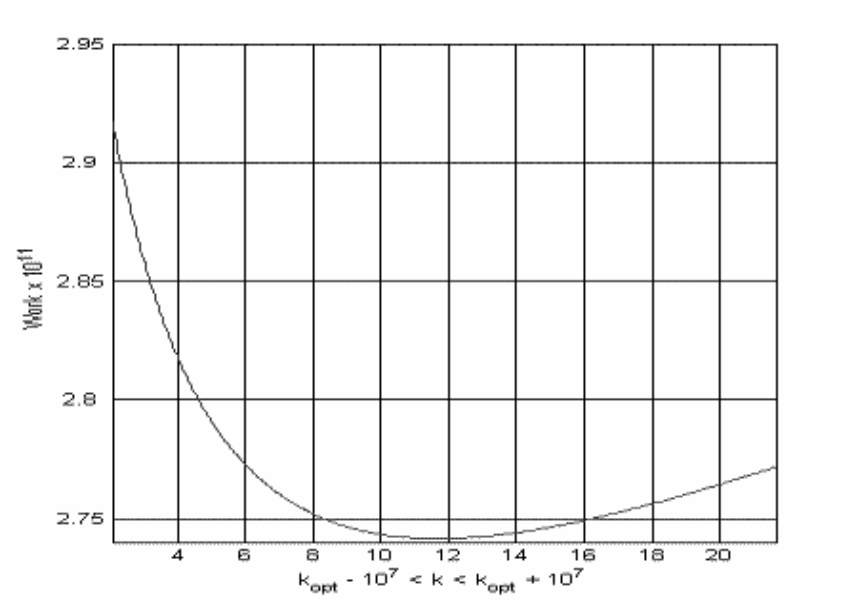

Figure 4. Graph #4: Optimization of Factor Base Size; near the minimum

We have $z = (\log k)^2/C$ and thus the optimal value of k is

$$k_{\text{opt}} = e^{\sqrt{CW(Ce^2)}}. \tag{3.6}$$

For the canonical example we obtain $k_{\text{opt}} \approx 13\,660\,000$ from (3.6). To show the effect of selecting a different point in computing C, we have $k_{\text{opt}} = 10\,480\,000$ with b taken to be 100 000 instead of 500 000. We now consider the rate of growth of $T(k)$ as k moves to the left of its optimum. Note that $s(y)$ is an increasing function of y as y increases. Therefore we have $T'(y) > T(y)$ provided that $s(y) > 1$. Note that y increases as k decreases. We can solve $s(y) > 1$ using the technique from equations (3.4) and (3.6). This means that $T(k)$ grows faster than exponentially as k decreases provided that

$$k < e^{\sqrt{CW(Ce)}}. \tag{3.7}$$

Inequality (3.7) shows how small we can force the factor base without starting to incur a large penalty. It has been suggested that one way to alleviate the space difficulties with NFS is to use much smaller faster bases, but "sieve longer". These results show that the amount of additional work rapidly becomes prohibitive. This is also seen in the comparison of the factorization of $10^{181} + 1$ and $2^{617} + 1$ in Section 7.

Instances exist where $k_1 = k_2$ is not desired. Consider $N = 31^{125} + 1$. This has the algebraic factor $31^{25} + 1$ leaving $N = 31^{100} - 31^{75} + 31^{50} - 31^{25} + 1 = M^4 - M^3 + M^2 - M + 1$. For a typical lattice point (say) $(10^6, 10^6)$, the norm of

$a + bM$ is approximately $10^{43}b$ while the norm of the algebraic polynomial is near 10^{25}. Thus, the linear norm is much larger than the algebraic norm and it makes sense to use a larger rational factor base in instances such as this.

As LP increases, so does the yield rate per lattice point, but so also does the number of needed factorizations. Further, once smooth relations have been identified and the primes in the factor base divided out, we are left with a cofactor. This must first be tested for primality, then if not prime, factored with a method such as SQUFOF or Pollard Rho. The author's original implementation found SQUFOF, followed by Pollard Rho if SQUFOF fails, to be quite effective. However, a more recent revision of the code shows that using a well tuned version of QS without a large prime variation is even faster than SQUFOF. Section Section 7 gives data on the actual improvement obtained with QS. However, as LP increases, so does the amount of time dealing with the cofactors relative to the sieve time. Further, a large fraction of the cofactors will be discarded. Either they will be prime, or they will split into the product of a relatively small prime times a prime bigger than LP. These are referred to as 'false hits'. If LP is too big, then too much time is spent processing and discarding them. Furthermore, the existence of false hits influences the computation of the sieve threshold, discussed in Section 5. For very small numbers if two large primes are used for both sides of the congruence it has been observed that the time splitting the cofactors totally dominates the computation. The table in Section 7 gives guidelines, derived from experience both for the number of large primes and for the value of LP.

The value of LP and number of primes also influence the size and density of the matrix to be solved, although not in any way that has a discernable pattern. The size of the matrix is not very predictable as a function of large prime parameters except to say in a general way that the size increases as LP and the number of large primes increases. The size of the Galois group of f also influences the size and density of the matrix. SNFS tends to run slightly faster when there are more 'small' primes that split completely in $Q(\alpha)$ and the frequency of such primes is determined by the reciprocal of the order of normal closure for the Galois group. Research needs to be done to quantify this effect. We need a function for NFS similar to the Knuth-Schroeppel function for the Quadratic Sieve, but which takes into account the size of the Galois group. Finally, if more than the minimum number of required relations is collected, it aids in forming a smaller matrix. More relations gives more freedom in combining relations with large primes and results in smaller matrices.

Note. It is possible that one might want to vary the number and size of the large primes over the sieve region. Near the origin the norms are smaller. It is possible that excess time is spent processing and discarding false hits in this region. One possibility is to try reducing the number and/or size of the large primes to cut down on false hits when the norms are known to be small. To this author's knowledge no one has investigated this possibility.

4 Choice of polynomial

For SNFS, the choices of polynomial are limited. See [17] for a discussion of selection techniques for GNFS. The table in Section 7 gives guidelines for the suggested degree of the polynomial as a function of the size of N, but there is sometimes more than one choice for a given degree. We discuss this by example.

Suppose $N = 41^{89} + 1$. There are a number of choices for the polynomial:

1. $41x^4 + 1$ with $M = 41^{22}$
2. $x^5 + 41$ with $M = 41^{18}$
3. $x^6 + 41$ with $M = 41^{15}$
4. $41^4x^5 + 1$ with $M = 41^{17}$.

The last two are readily discarded. The decrease in M from 41^{18} to 41^{15} does not compensate for the fact that for most (b, a) pairs, the norm for the sextic is larger than that for the quintic by more than a factor of 41^3. Similarly, the larger coefficient 41^4 more than offsets the decrease in M by the factor 41 than in using polynomial 2. The choice between polynomials 1 and 2 is fairly close and performance will be comparable. Also note that N has the algebraic factor 42. If we remove it we are left with a dense degree 88 polynomial for N. Finding a small degree polynomial becomes impossible; GNFS would have to be used.

Note that the run time of SNFS is not uniform in the size of N. Larger numbers sometimes take less time than smaller ones. This may be seen by comparing (say) $41^{83} + 1$ and $41^{86} + 1$. With degree 5 the former uses $x^5 + 41^2$ and the latter $41x^5 + 1$ and $M = 41^{17}$ in both cases. However, the coefficient 41^2 on the first polynomial is larger than the coefficient 41 on the second. Numbers of the form $x^{5k-2} + 1$ will generally run slightly slower than $x^{5k+1} + 1$ even though the latter is larger.

We show here a technique for numbers of the form $x^{11k} + y^{11k}$ or $x^{13k} + y^{13k}$ that allows us to take advantage of the algebraic factor, yet still use a polynomial of moderate degree. Put $N = x^{11k} + y^{11k}$. Then $y^{-11k}N = (x/y)^{11k} + 1$ yielding the polynomial $z^{11} + 1$ with root $z = x^k/y^k \bmod N$. This is a reciprocal polynomial. Thus, there is a quintic that takes $z + 1/z$ to $\frac{1}{z^5}\frac{(z^{11}+1)}{(z+1)}$. This polynomial is $z^5 - z^4 - 4z^3 + 3z^2 + 3z - 1$. For degree 13, the polynomial becomes $z^6 - z^5 - 5z^4 + 4z^3 + 6z^2 - 3z - 1$. They both have root $(x/y)^k + (y/x)^k \bmod N$. Thus, the norm on the linear polynomial becomes $(xy)^k a + (x^{2k} + y^{2k})b$ for point (b, a).

This last example gives us an instance where two numbers of the same size take different amounts of time. Consider $N_1 = 10^{121}+3^{121}$ and $N_2 = 10^{121}+9^{121}$. Using the technique above yields linear norms of $30^{11}a + (10^{22} + 3^{22})b$ and $90^{11}a + (10^{22} + 9^{22})b$ respectively. For small b the coefficient on a is so much larger in the second case that the linear norm is larger over much of the sieve region, resulting in a longer run time even though both numbers have 122 digits. This happens because N_1 is closer to a perfect power than N_2.

See also [7] for additional instances and examples of when specially tailored polynomials might be used.

5 Coding considerations

Some simple optimizations can be applied to the code which will give noticeable speed improvements.

1. To avoid L_2 cache misses while sieving, it is very worthwhile to partition the interval $[-\Omega(b), \Omega(b)]$ into pieces. The reason for this is that if each piece fits in cache, one can avoid cache misses while sieving. The speed at which the siever operates is primarily determined by the ability to update an address, retrieve the contents of that address, add a value to it, then put the result back. This process is much quicker if the memory being accessed is L_2 cache-resident. The optimal size of the pieces will be implementation and platform dependent. It can also be beneficial to make the sieve loop code small enough to fit in the L_1 cache so that no cache misses are incurred when fetching sieve instructions. This, however, is highly architecture dependent.

2. When a relation is found it is necessary to check that $GCD(b, a) = 1$. If not, the same relation will have been found earlier at $(b/GCD(a, b), a/GCD(a, b))$. Therefore when b is even, we only need to add values to odd a. Thus, one can start sieving at an odd value of a and double the stride length for each prime. If the sieve start point is even, simply take a single stride before doubling the stride. This simple trick can save $20 - 25\%$ of the total sieve time. One could do a similar trick for odd multiples of b divisible by 3, but this author did not find it worthwhile. The theoretical savings is only $1/18$th of the total run-time if implemented with no overhead.

3. Once a smooth relation has been identified, its actual factorization must be constructed. One can do this by trial division, combined perhaps with Pollard Rho, but it is better in practice to do by resieving. Construct a compact bitmap which contains the locations of the smooth relations, then sieve again. Except now, instead of adding $\log p_i$, store p_i itself. The resieving is done with all primes greater than some bound T. Primes up to T are identified by trial division. Primes that are stored by the resieve process are also checked to see if they divide the norm more than once. This author found that the selection of T had very little impact on run time. Anything in the range 1000 to 25 000 worked well. The tradeoffs in selecting T depend upon the cache size and the amount of memory needed to store the primes while resieving. The smaller the value of T, the more memory is needed. Sieving with the very smallest primes is actually slightly slower than trial division with them since there are not very many of them and sieve time for each is proportional to $1/p$. This author's code takes so little time to actually store the p_i that the actual storage mechanism was not important. The p_i were stored in a simple list.

4. Once factor base primes have been pulled from a norm, one should test any cofactor for primality before applying SQUFOF, Pollard Rho, or QS to pull apart the large primes. It is worthwhile to write a custom prime testing procedure optimized for 64-bit integers, as opposed to using a general multi-precision prime tester. The same applies to the SQUFOF, Pollard Rho, or QS routines.

5. One identifies smooth relations after sieving by checking that the accumulated logarithms are close to $\log(\text{norm}(a + bM))$ on the integer side and $\log(\text{norm}(a + b\alpha))$ on the algebraic side. Since large prime factors are allowed, these thresholds must

be adjusted by subtracting off $\log(LP^2)$. The integer side threshold can be computed once per b value since it changes very little as a varies, but the norms of the algebraic side can vary considerably, especially near the origin. It is worthwhile to recompute this threshold fairly frequently since it changes with $(d-1)\log a$. This author found it worthwhile to recompute every 200 to 300 a values and that optimizing a routine for doing this computation was productive. Furthermore, since the logs of the factor base primes are only byte approximations, it is worthwhile to allow a little "fuzziness" in the sieve thresholds. One way to do this is to subtract $\log(LP^{2+\epsilon})$ rather than just $\log(LP^2)$. The value of ϵ will be dependent on the base one chooses for the logs of the factor base primes. It is worthwhile to use as small a logarithm base as possible, to allow as much dynamic range in a single byte as possible.

6. It is worthwhile to partition the factor bases into subsets according to the size of the primes. The size in each subset is determined by how many hits in a partitioned piece of the sieve is made by each prime. The sieve loops should be unrolled accordingly.

6 Space requirements

Composites requiring factor bases sizes that yield primes larger than 32 bits are still well beyond computer range. Even composites of 240 digits only require factor bases where the largest prime is less than 10^8. Thus, the assumption below that variables only require 4 bytes of storage will remain correct for some time. A speed-efficient implementation must store the following variables:

1. The factor bases; 4 bytes per prime ideal.

2. The roots of $f(x)$ and $g(x)$ modulo the factor bases; 4 bytes per root.

3. The start points for the sieve; 4 bytes per prime.

4. The ending points for each prime within a partition if the sieve is partitioned into pieces; 4 bytes per prime.

5. Storage for the primes when factoring by resieving. Storage for this depends on the number of expected successes in each partitioned piece of the sieve interval. This in turn depends on the size of the number being factored since smaller numbers have more successes.

6. Space for the sieve array. This depends on how it is partitioned.

The author's code requires only 53 Megabytes for a factor base of 1.2 million prime ideals for each polynomial; this is adequate for a 200-digit SNFS number. This includes storage for each ideal, along with its corresponding root. This should scale as $\sqrt{L(N,c)}$ as N increases. If $2^{1024}+1$ were done with SNFS, it should take about 3800 times as long and require about 62 times as much space, or about 3.3 Gigabytes per sieve processor. If, owing to inadequate memory, one tries to use an out-of-core siever in which some of the variables are written to a file then retrieved as needed, it slows the computations down considerably. Also note that although 64-bit machines are readily available, 32-bit machines are still prevalent, and some operating systems will not allow users to address more than 2 Gigabytes of data in their programs.

7 Numerical results

The author has factored hundreds of numbers from the Cunningham project [4] and from its extensions [3]. The tables below are derived from these factorizations. Table 1 shows the raw data from actual factorizations. Table 2 gives suggested choices, as a function of the size of the number being factored, for the algorithm parameters. These have been found to work well on Pentium based processors. For other processors, with different cache sizes and instruction sets, these choices may not be optimal. (M = million, K = thousand). Some of the parameterizations were deliberately chosen to be sub-optimal to show the effect of such choices. The Notes section below the table gives an explanation of the columns and a discussion of which factorizations used parameters that were optimal.

NB. The derivation of equation (2.9) is a recent result. It was applied only for $2^{619}+1$ among the results given here. It resulted in about a 17% run-time improvement over $2^{617}+1$ even though the latter is slightly smaller. While $2^{617}+1$ used $4x^5+1$, and $2^{619}+1$ used x^5+2 so that the coefficient on the former is larger, the latter had a value of M that was also exactly a factor of 2 larger. $2^{617}+1$ was sieved with $a_{\max} = 20$M and $B = 2$M. $2^{619}+1$ had $\Omega(b) = 50$M from $b = 1$ to 100K, $\Omega(b) = 30$M from $b = 100$K to 200K, $\Omega(b) = 25$M from $b = 200$K to 300K, $\Omega(b) = 20$M from $b = 300$K to 800K, $\Omega(b) = 19$M from $b = 800$K to 1100K, $\Omega(b) = 18$M from $b = 1100$K to 1300K, $\Omega(b) = 17$M from $b = 1300$K to 1400K, and $\Omega(b) = 16$M from $b = 1400$K to 1500K. k was equal to 1M for both factorizations.

N	Digits	Poly	LP	FB Size	Work	Matrix Size	Matrix Memory	Total Relations
$4^{232}+3^{232}$	140	$16x^5+9$	50M	70K	1.42×10^{13}	427K	64M	4.06M
$97^{73}-1$	146	x^5-9409	60M	70K	1.76×10^{13}	565K	142M	4.7M
$35^{97}+1$	150	$1225x^5+1$	50M	80K	3.53×10^{13}	652K	160M	4.4M
$2^{587}+1$	177	$4x^5+1$	100M	250K	8.7×10^{14}	860K	224M	5.6M
$5^{257}+1$	179	$25x^5+1$	100M	300K	1.45×10^{14}	N/A	N/A	5.2M
$12^{167}+1$ (*)	180	$144x^5+1$	150M	750K 400K	1.2×10^{14}	1.97M	488M	13.0M
$10^{181}+1$	181	$10x^5+1$	100M	300K	1.25×10^{15}	.97M	241M	6.17M
$10^{184}+1$	184	x^5+10	100M	300K	1.29×10^{15}	1.06M	258M	6.33M
$2^{617}+1$	185	$4x^5+1$	300M	1M	2.24×10^{14}	1.9M	457M	24.8M
$2^{619}+1$	186	x^5+2	300M	1M	1.86×10^{14}	2.0M	480M	23.5M
$10^{191}+1$	191	$10x^5+1$	100M	300K	1.52×10^{15}	N/A	N/A	8.9M
$10^{194}+1$	194	x^5+10	100M	300K	2.06×10^{15}	N/A	N/A	9.4M
$2^{653}+1$	197	x^6+2	500M	1.2M	3.3×10^{14}	4.5M	1020M	39.0M
$10^{197}-1$	197	x^6-10	300M	1.2M	1.11×10^{15}	2.4M	562M	22.4M
$6^{256}+1$	200	$6x^5+1$	300M	1.2M	7.66×10^{14}	2.41M	604M	24.7M

Table 1. Actual Results

Notes.

1. LP is the large prime bound.
2. Work is defined as $\log\log p_{\max}$ times the area actually sieved.
3. Matrix size is given by the number of rows.
4. Total is the total number of relations that were found.
5. $2^{617}+1$ took less time by a factor of 5.6 than $10^{181}+1$ even though it is five digits larger. This clearly shows that the parameters were not optimally chosen for $10^{181}+1$; 300 000 is too small for the factor base and 100M is too small for LP.
6. The numbers in the 190-200 digit range could have been done slightly faster with slightly larger factor bases, but the need to run on machines with only 64 Mbytes of memory constrained the largest factor base to be about 1.2 million primes per polynomial. The parameters for $10^{191}+1$ and $10^{194}+1$ were significantly sub-optimal. $10^{197}-1$ required half of the amount of sieving as $10^{194}+1$.
7. The factorization of $12^{167}+1$ used different values for k_1 and k_2. A larger factor base was used for the integer norms. This result is due to Peter Montgomery.
8. Additional results for smaller numbers may be found in 3.10 of [7].

To give an idea of the time required to sieve we give the following data for $2^{653}+1$. The given times are for a 3.2GHz Pentium IV with a 512Kbyte L_2 cache. Total sieving time for this machine would be $\approx$ 100 days. This is an estimate because the actual factorization was effected using 8 machines of varying speeds and cache sizes. The following data shows actual CPU times for just a sub-region of the entire sieve region.

1. Factor base size: 1.2M for each polynomial. $LP = 400$M
2. Total time for $400\,000 < b < 450\,000$: 211 478 seconds = 4.229 seconds/b-value.
3. $\Omega(b) = 13$M for this interval
4. Sieving time: 165 372 seconds
5. Resieving time: 18 533 seconds
6. Time spent splitting large primes via SQUFOF: 10 093 seconds
7. Time to scan sieve array for successes: 5297 seconds
8. Trial Division and testing cofactors for primality: 4423 seconds
9. All other operations: 7760 seconds

The main sieve takes 78.2% of the total time. Factoring the identified successes (including resieving) takes 15.6%. Scanning the sieve array for successes takes 2.5%. Everything else takes 3.7%.

Note. Since the above data was collected, this author has changed from using SQUFOF to split the large primes to using QS customized for composites in the 50 to 60 bit range. The result was a factor of slightly more than 2 decrease in the time to split the large primes. This would have saved about 5000 seconds of the total 211 000 seconds in the above data, a savings of about 2.3%.

Size of N	80-100	100-120	120-140	140-160	160-180	180-200
k	5K-10K	10K-30K	30K-80K	80K-300K	300K-1.2M	1.2M-2M
LP	20M-30M	30M-50M	60M-70M	70M-150M	150M-300M	300M-500M
#Large Primes	(1,1)	(2,1)	(2,1)	(2,2)	(2,2)	(2,2)
Poly deg.	3 or 4	4 or 5	5	5	5 or 6	5 or 6

Table 2. Suggested Parameter Choices

The next table gives guidelines for choosing the factor base, the number and size of large primes, and the polynomial degree as a function of the size of the number being factored. This data was drawn from the author's experience with hundreds of factorizations. The notation (c, d) for the number of large primes refers to the number on the left and right hand side of each relation respectively.

Acknowledgments. I would like to thank Professor Lawrence Evans of Berkeley for very helpful comments on the variational problem. David Cantrell suggested the use of the Lambert W function to solve equation (3.4). I would also like to thank Peter Montgomery for his data filtering and square root code as well as for solving some large matrices. My computers had insufficient memory to do so. Comments of the anonymous referees greatly added to the quality of this paper.

This paper was developed prior to my current employment at Raytheon.

References

[1] E. Bach and R. Peralta, *Asymptotic semismoothness probabilities*, Math. Comp. 65 (1996), pp. 1701–1715.

[2] O. Bolza, *Lectures on the Calculus of Variations*. Dover, New York, 1961.

[3] R. P. Brent, P. Montgomery, and H. te Riele, *Factorizations of Cunningham numbers with bases 13 to 99: Millennium edition*, Australian National University, Report, December 2000. Report PRG TR-14-00, Oxford University Computing Laboratory, Oxford, UK.

[4] J. Brillhart, D. H. Lehmer, J. Selfridge, B. Tuckerman, and S. S. Wagstaff Jr., *Factorizations of $b^n \pm 1$ for $b = 2, 3, 5, 6, 7, 10, 11, 12$ up to High Powers*, Contemporary Mathematics 23. Amer. Math. Soc., 1987.

[5] E. R. Canfield, P. Erdös, and C. Pomerance, *On a problem of Oppenheim concerning "factorisatio numerorum"*, J. Number Theory 17 (1983), pp. 1–28.

[6] S. Contini, *Two implementation ideas for fast line sieving*, http://www.cwi.nl/~herman/Contini.doc.

[7] R.-M. Elkenbracht-Huizing, *Factoring integers with the Number Field Sieve*, Ph.D. thesis, Leiden University, 1997.

[8] R.-M. Elkenbracht-Huizing, Peter L. Montgomery, R. D. Silverman, R. K. Wackerbarth, and S. S. Wagstaff Jr., *The number field sieve on many computers*. Proceedings of the Fifth Conference of the Canadian Number Theory Association (Rajiv Gupta and Kenneth S. Williams, eds.), Centre de Recherches Mathematiques (CRM, Montreal) Proceedings and Lecture Notes Series, pp. 81–85. American Mathematical Society (AMS), 1999.

[9] Personal email from NFSNET, www.nfsnet.org.

[10] S. Cavallar et al., *Factorization of a 512-bit modulus*. Advances in Cryptology-EUROCRYPT '00 (B. Preneel, ed.), Lecture Notes in Computer Science 1807, pp. 1–18. Springer-Verlag, 2000.

[11] D. E. Knuth, *The Art of Computer Programming Vol 2*. Addison-Wesley, 1997, 3rd ed.

[12] I. N. Kovalenko, *Theory of random graphs*, Kibernetika 4 (1970), pp. 575–579.

[13] R. Lambert, *Computational Aspects of Discrete Logarithms*, Ph.D. thesis, University of Waterloo, 1996.

[14] A. K. Lenstra and H. W. Lenstra (eds.), *The Development of the Number Field Sieve*, Lecture Notes in Mathematics 1554. Springer-Verlag, 1993.

[15] A. K. Lenstra and M. S. Manasse, *Factoring with two large primes*. Advances in Cryptology-EUROCRYPT '90 (I. Damgård, ed.), Lecture Notes in Computer Science 473, pp. 72–82. Springer-Verlag, 1990.

[16] P. Montgomery, *Square roots of products of algebraic numbers*. Proceedings of Symposia in Applied Mathematics, Mathematics of Computation 1943-1993 (Walter Gautschi, ed.), pp. 667–671, 1993, Vancouver.

[17] B. Murphy, *Polynomial selection for the number field sieve integer factorization algorithm*, Ph.D. thesis, Australian National University, 1999.

[18] J. M. Pollard, *The Lattice Sieve*, The Development of the Number Field Sieve (A. K. Lenstra and H. W. Lenstra, eds.), Springer-Verlag, 1993, pp. 43–49.

[19] R. D. Silverman, *The Multiple Polynomial Quadratic Sieve*, Math. Comp. 48 (1987), pp. 329–339.

[20] J. Sorenson, *A Fast Algorithm for Approximately Counting Smooth Numbers*. Proceedings of 4th International Symposium ANTS 2000 (W. Bosma, ed.), Lecture Notes in Computer Science 1838, pp. 539–550. Springer-Verlag, 2000.

Received 30 March, 2006; revised 27 September, 2006

Author information

Robert D. Silverman, Raytheon Co., Integrated Defense Systems, 225 Presidential Way, Woburn, MA 01801, USA.
Email: robert_silverman@raytheon.com