MATHEMATICS OF COMPUTATION
Volume 79, Number 269, January 2010, Pages 583–602
S 0025-5718(09)02198-X
Article electronically published on July 27, 2009

THE NUMBER FIELD SIEVE FOR INTEGERS OF LOW WEIGHT

OLIVER SCHIROKAUER

Abstract. We define the weight of an integer N to be the smallest w such that N can be represented as $\sum_{i=1}^{w} \epsilon_i 2^{c_i}$, with $\epsilon_1, \ldots, \epsilon_w \in \{1, -1\}$. Since arithmetic modulo a prime of low weight is particularly efficient, it is tempting to use such primes in cryptographic protocols. In this paper we consider the difficulty of the discrete logarithm problem modulo a prime N of low weight, as well as the difficulty of factoring an integer N of low weight. We describe a version of the number field sieve which handles both problems. In the case that $w = 2$, the method is the same as the special number field sieve, which runs conjecturally in time $\exp(((32/9)^{1/3} + o(1))(\log N)^{1/3}(\log\log N)^{2/3})$ for $N \to \infty$. For fixed $w > 2$, we conjecture that there is a constant ξ less than $(32/9)^{1/3}((2w-3)/(w-1))^{1/3}$ such that the running time of the algorithm is at most $\exp((\xi + o(1))(\log N)^{1/3}(\log\log N)^{2/3})$ for $N \to \infty$. We further conjecture that no ξ less than $(32/9)^{1/3}((\sqrt{2}w - 2\sqrt{2} + 1)/(w - \ 1))^{2/3}$ has this property. Our analysis reveals that on average the method performs significantly better than it does in the worst case. We consider all the examples given in a recent paper of Koblitz and Menezes and demonstrate that in every case but one, our algorithm runs faster than the standard versions of the number field sieve.

1. Introduction

We define the weight of an integer N to be the smallest w with the property that there exists a representation of N as a sum

$$\sum_{i=1}^{w} \epsilon_i 2^{c_i}, \tag{1.1}$$

with $\epsilon_1, \ldots, \epsilon_w \in \{1, -1\}$. In 1999, Solinas observed that arithmetic modulo a prime can be made more efficient if the prime is chosen to be of small weight ([14]). More recently, Koblitz and Menezes, in their investigation of the impact of high security levels on cryptosystems which are based on the Weil and Tate pairings on elliptic curves over finite fields have asked whether there is a downside to using a field in this context whose characteristic has small weight ([6]). In particular, they raise the concern that discrete logarithms modulo a prime N might be easier to compute with the number field sieve (NFS) when N is of low weight than they are in general. The increase in vulnerability may then offset any efficiency advantages gained by implementing the protocol with a low weight prime.

Received by the editor July 31, 2006 and, in revised form, June 15, 2008.
2000 *Mathematics Subject Classification.* Primary 11Y16; Secondary 11T71, 11Y05, 11Y40.
Key words and phrases. Discrete logarithm, integer factorization, number field sieve.

There is no doubt that for weight two primes, the concern about vulnerability is justified. In this case the special number field sieve (SNFS) can be used to solve the discrete logarithm problem. This algorithm, which is designed to compute logarithms modulo primes N of the more general form $kr^c + s$, with k, r, and s small, has a conjectural running time of

$$L_N[1/3; (32/9)^{1/3} + o(1)], \tag{1.2}$$

where

$$L_N[\rho; \xi] = \exp(\xi(\log N)^\rho(\log\log N)^{1-\rho})$$

and the $o(1)$ is for $N \to \infty$ with k, r, and s bounded in absolute value. By comparison, the general NFS has a conjectural running time of

$$L_N[1/3; (64/9)^{1/3} + o(1)],$$

for $N \to \infty$. The suggestion, which is supported by the details of the running time analyses, is that the time it takes to compute a discrete logarithm modulo a large general prime N is about the time required for a special prime of size N^2.

The purpose of this paper is to extend the SNFS to primes of arbitrary weight. In all versions of the NFS, an extensive search takes place for smooth elements in two different number rings. The goal at the outset of the algorithm is to choose the best possible rings. In the SNFS, one ring is taken to be $\mathbb{Z}$ and the other is generated over $\mathbb{Z}$ by the root of a polynomial with integral coefficients having a root mod N. The reason the SNFS is faster than the NFS is that the special form of the modulus N allows for the construction of a suitable polynomial with extremely small coefficients. The algorithm we describe in the next section takes advantage of the low weight of N in much the same way. In particular, we convert (1.1) into a representation of a multiple of N as a sum of the form

$$\sum_{i=1}^{w} \epsilon_i 2^{a_i + b_i e}$$

and make use of the associated polynomial with coefficients $\epsilon_i 2^{a_i}$ and root 2^e mod N. Because our results are applicable to the problem of factoring as well as to the discrete logarithm problem, and in order to avoid a discussion about the existence of primes of fixed weight, we formulate our algorithm as a general method which takes an arbitrary odd integer N as input and which can be incorporated either into a method to factor N when N is composite or a method to compute discrete logarithms mod N when N is prime.

In §3, we analyze the algorithm of the previous section. When $w = 2$, the method is the same as the SNFS and so runs conjecturally in time (1.2) for $N \to \infty$. For fixed $w > 2$, we conjecture that the running time of the algorithm is at most $L_N[1/3; \xi + o(1)]$, where ξ is a constant which is less than

$$\left(\frac{32}{9}\right)^{1/3}\left(\frac{2w-3}{w-1}\right)^{1/3}$$

and the $o(1)$ is again for $N \to \infty$. Thus, for fixed w, our algorithm is asymptotically faster than the general NFS. We also exhibit, for each w, an infinite set S of integers of weight w having the property that for $N \in S$ tending to ∞, the conjectural running time of the algorithm is bounded below by

$$L_N[1/3; (32/9)^{1/3}\tau^{2/3} + o(1)],$$

where

$$\tau = \frac{\sqrt{2}w - 2\sqrt{2} + 1}{w - 1}.$$

The details of the running time analysis for these sets of inputs are set aside as an appendix. The reader who consults Figure 3.11 will see that the difference between $((2w-3)/(w-1))^{1/3}$ and $\tau^{2/3}$ is quite small. Though the asymptotic results we obtain are satisfying in that they indicate how to parametrize the gap between the SNFS and general NFS running times in terms of w, their value is diminished by the fact that the weight of an input only loosely determines the time required by the method. As we will see, the spacing of the exponents in (1.1) has a significant impact on the algorithm, producing a wide range of performances for a fixed weight. To partially remedy the situation, we also investigate in §3 what happens on average for inputs of a fixed weight.

In the fourth and final section of the paper, we turn to practical considerations. To get a better sense of the speedup afforded by small weight inputs, we compare the size of the numbers that are tested for smoothness in our algorithm with those arising in other versions of the NFS. The discrete logarithm problems we use for this investigation are the seven examples appearing in [6]. Of these, four involve a finite field of degree two over its prime field. In all but one of the seven examples, the method we describe in §2 is superior to the other versions of the NFS, and in many of these cases, dramatically so.

One practical issue we do not discuss is how to determine whether a given input N has low weight. We do, however, note that fast algorithms exist to determine the weight of an integer and to produce all representations of the form displayed in (1.1). Thus, for a value of N not given in this form, we can quickly determine whether to look for a representation for use in the method of this paper. For more on representing integers in signed-binary form, see [10].

2. The algorithm

Whether it is being used to factor an integer N or to compute discrete logarithms in a prime field of size N, the number field sieve (NFS) begins with the selection of two number rings R_1 and R_2 for which there exist maps $\phi_1 : R_1 \to \mathbb{Z}/N\mathbb{Z}$ and $\phi_2 : R_2 \to \mathbb{Z}/N\mathbb{Z}$. An element in a number ring is said to be smooth with respect to a bound B if each prime factor of its norm in $\mathbb{Z}$ is at most B. Once R_1 and R_2 are obtained, sieving techniques are used to collect many pairs $(\beta_1, \beta_2) \in R_1 \times R_2$ such that β_1 and β_2 are smooth with respect to an optimally chosen bound and such that $\phi_1(\beta_1) = \phi_2(\beta_2)$. Finally, a massive linear algebra computation leads to the desired factors or logarithms. The challenge when choosing R_1 and R_2 is to minimize the size of the norms being tested for smoothness. Our purpose in this section is to describe a method for choosing rings which is particularly well-suited to the case that N is of low weight.

The algorithm we present is an extension of the special number field sieve (SNFS), which is designed for inputs N of the form $kr^c + s$. We give a brief overview of the SNFS in the event that $k = 1$. In this case, one of the number rings used is $\mathbb{Z}$, and the other is obtained by adjoining to $\mathbb{Z}$ a root $\alpha \in \mathbb{C}$ of the polynomial

$$f = X^d + sr^{c'-c},$$

where d is the precomputed optimal degree of f and c' is the smallest multiple of d greater than or equal to c. Observe that $r^{c'/d}$ is a root of f mod N. Therefore, the canonical ring homomorphism from $\mathbb{Z} \to \mathbb{Z}/N\mathbb{Z}$ can be extended to $\mathbb{Z}[\alpha]$ by sending α to $r^{c'/d} + N\mathbb{Z}$. The pairs checked for smoothness in the algorithm are of the form $(a - br^{c'/d}, a - b\alpha)$, with $a, b \in \mathbb{Z}$. Since the norm of $a - b\alpha$ is equal to $b^d f(a/b)$, the coefficients of f have a significant impact on the size of the smoothness candidates. In the special case that $|r|$ and $|s|$ are small, these candidates are much smaller than those arising in the general NFS, and as a result, the SNFS is the faster method.

Since our focus is on the polynomial construction that occurs at the beginning of the NFS and since there are many good accounts of the NFS available, we do not provide a complete description here of how to factor N or determine logarithms mod N for a low weight input N. Instead, we carry the story line only through the sieving stage. For a description of the remaining steps, see [1] and [15] in the case of factoring and [3] and [13] in the case of discrete logarithms.

Algorithm 2.1 (Extended SNFS). *This algorithm takes as input a positive, odd integer N of weight w, represented as the sum*

$$\sum_{i=1}^{w} \epsilon_i 2^{c_i},$$

with $c_w > c_{w-1} > \ldots > c_1 = 0$ and $\epsilon_1, \ldots, \epsilon_w \in \{1, -1\}$, as well as integer parameters $e, B, M, J > 0$, with $e \leq c_w$. Its purpose is to produce a matrix A which can be used for factoring N or computing discrete logarithms mod N. In what follows, B serves as a smoothness bound, M determines the size of the region sieved for smooth elements, and J is used to specify how many smooth pairs are needed.

Step 1. This step is devoted to producing a polynomial with integral coefficients and a root mod N. The goal is to have the coefficients and root be small. For $i = 1, \ldots, w$, let $\bar{c}_i$ be the least non-negative residue of c_i mod e. In addition, let $\bar{c}_{w+1} = e$. Assume that σ is a permutation on the set $\{1, \ldots, w, w+1\}$ which fixes $w+1$ and has the property that the sequence

$$\bar{c}_{\sigma(1)}, \bar{c}_{\sigma(2)}, \ldots, \bar{c}_{\sigma(w)}, \bar{c}_{\sigma(w+1)} \tag{2.2}$$

is non-decreasing, and let I be the largest number such that $\bar{c}_{\sigma(I+1)} - \bar{c}_{\sigma(I)}$ is greater than or equal to the difference between any other pair of consecutive terms in (2.2). Finally, write μ for the quantity $e - \bar{c}_{\sigma(I+1)}$, and let

$$f = \sum_{i=1}^{w} \epsilon_i 2^{a_i} x^{b_i},$$

where

$$a_i = \bar{c}_i + \mu \quad \text{and} \quad b_i = \lfloor c_i/e \rfloor \qquad \text{if} \quad \sigma^{-1}(i) \leq I,$$

and

$$a_i = \bar{c}_i - \bar{c}_{\sigma(I+1)} \quad \text{and} \quad b_i = \lfloor c_i/e \rfloor + 1 \qquad \text{if} \quad \sigma^{-1}(i) > I.$$

It is a straightforward matter to verify that

$$\max |a_i| = \bar{c}_{\sigma(I)} + \mu = e - \left(\bar{c}_{\sigma(I+1)} - \bar{c}_{\sigma(I)}\right) \tag{2.3}$$

and that

$$f(2^e) = 2^\mu N.$$

Notice that the weight of N does not appear in (2.3) but instead influences the bound on the size of the coefficients of f indirectly by imposing a lower bound on the size of the maximal gap in sequence (2.2).

In the case that N is prime, it follows from a result in [4] that f is irreducible. In the case that N is composite and f factors into a product $g_1 \dots g_m$ of non-constant irreducible polynomials, we either obtain a splitting of N by plugging 2^e into the factors of f or we find that $g_i(2^e)$ is a multiple of N for some i. In the former case, our problem is solved, and in the latter, replacing f by g_i only improves the algorithm that follows. For this reason, we assume from this point onwards that f is irreducible.

Let $\alpha \in \mathbb{C}$ be a root of f and let $\mathcal{O}$ be the ring of integers of $\mathbb{Q}(\alpha)$. If f is monic, then the description of the next two steps is easily reconciled with the outline of the NFS given at the beginning of this section. As in our sketch of the SNFS, the two rings being used are $\mathbb{Z}$ and $\mathbb{Z}[\alpha]$ and the pairs being tested for smoothness are of the form $(a - b2^e, a - b\alpha)$, where a and b are integers. If f is not monic, then almost no adjustments are required in the computations in the algorithm but the algebraic structure underlying the method is somewhat different. We refer the reader to §12 of [1] for a discussion of this case.

Step 2. Let d be the degree of f, let Δ be the discriminant of f, and let T be the set of prime ideals of degree one in $\mathcal{O}$ which lie over a rational prime which is prime to Δ and at most B. The objective of this step is to find all pairs (a, b) of relatively prime integers satisfying $|a| \leq M$ and $0 < b \leq M$ such that

$$(a - b2^e)b^d f(a/b) \tag{2.4}$$

is B-smooth and prime to Δ. This can be accomplished by means of a sieve ([1],[9]). Call the set of pairs passing the smoothness test U. If $|U| < \pi(B) + |T| + J$, then the algorithm is not successful and terminates.

Step 3. In this step, we construct the matrix A. For each rational prime q, let ν_q denote the valuation of $\mathbb{Q}$ associated to q, and similarly, for each prime ideal $\mathfrak{q} \in \mathcal{O}$, let $\nu_\mathfrak{q}$ be the valuation of $\mathbb{Q}(\alpha)$ corresponding to $\mathfrak{q}$. Let U' be a subset of U of size $\pi(B) + |T| + J$. For each $(a, b) \in U'$, compute $\nu_q(a - b2^e)$ for $q \leq B$ and $\nu_\mathfrak{q}(a - b\alpha)$ for $\mathfrak{q} \in T$, and let $v(a, b)$ be the vector whose coordinates are these values. Finally, let A be the matrix whose columns are the vectors $v(a, b)$. Note that the norm of $a - b\alpha$ is

$$\frac{b^d f(a/b)}{k},$$

where k is the leading coefficient of f, and that for $\mathfrak{q} \in T$ lying over a rational prime not dividing k, the coordinate $\nu_\mathfrak{q}(a - b\alpha)$ is equal to the exponent to which q divides $b^d f(a/b)$ in the case that $k(a - b\alpha)$ is in $\mathfrak{q}$ and is 0 otherwise. The case that $\mathfrak{q}$ lies over a prime divisor of k is addressed in §12 of [1]. This completes the description of Algorithm 2.1.

The sources cited prior to Algorithm 2.1 describe how to factor N or compute logarithms mod N by solving one or more congruences of the form $A'x \equiv v \bmod l$, where l is prime, v is a specified vector, x is unknown, and A' is a modification of the matrix A. The purpose of J is to ensure that A' has enough columns so that such a congruence is likely to have a solution. In general, J is small and in the analysis that follows, we take it to be $O(\log N)$.

3. Running time

In this section we provide a worst-case running time analysis of Algorithm 2.1 for inputs of a fixed weight. As revealed by equation (2.3), the maximum size of the coefficients of the polynomial f appearing in Step 1, and in turn the running time of the method, depend on the size of the largest gap in sequence (2.2). Since this size can vary widely for different inputs of the same weight, we also consider the performance of the method on average. In particular, we consider the set of all non-decreasing sequences of a fixed length whose terms are non-negative integers at most a given bound and determine the average size of the largest gap in such a sequence. This result gives a better indication than the worst-case analysis of the speedup that follows, both asymptotically and in practice, from using a low weight input.

The special number field sieve (SNFS), described briefly in the previous section and in detail in [9], serves as an initial point of reference. Its conjectural running time for input $N = kr^c + s$ is

$$L_N[1/3; (32/9)^{1/3} + o(1)] \tag{3.1}$$

for $N \to \infty$ with k, r and s bounded in absolute value. This is a significant improvement over the running time of the number field sieve (NFS) for general numbers, which runs conjecturally in time

$$L_N[1/3; (64/9)^{1/3} + o(1)] \tag{3.2}$$

for $N \to \infty$. In the case that $w = 2$, the method for finding f in Algorithm 2.1 is the same technique as is used in the SNFS, and as w grows, we expect the running time of Algorithm 2.1 to span the distance between (3.1) and (3.2). Our first task is to determine precisely how this occurs.

Conjecture 3.3. *Fix w and let $\theta = (2w-3)/(w-1)$. For each positive, odd integer N of weight w, let $J(N)$ be a positive integer and assume that $J(N)$ is $O(\log N)$. Then for each N as just described, there exists $e, B,$ and M such that Algorithm 2.1, upon input of N and parameters $e, B, M,$ and $J = J(N)$, produces a matrix A in time at most*

$$L_N[1/3; (32\theta/9)^{1/3} + o(1)] \tag{3.4}$$

for $N \to \infty$.

Let $N > 3$ be an odd integer of weight w, represented as in Algorithm 2.1. That is,

$$N = \sum_{i=1}^{w} \epsilon_i 2^{c_i},$$

with $c_w > c_{w-1} > \ldots > c_1 = 0$ and $\epsilon_i \in \{1, -1\}$. To obtain (3.4) as a conjectural running time, we let $e = \lfloor c_w/\delta \rfloor$, where

$$\delta = \Big\lfloor \Big(\frac{3\theta \log N}{2 \log\log N} \Big)^{1/3} \Big\rfloor .$$

As in §2, we denote by d the degree of the polynomial f produced in Step 1 of the algorithm. We leave it to the reader to verify that for N sufficiently large, $d = \delta$. In particular, we see that

$$d = \Big(\frac{(3\theta + o(1)) \log N}{2 \log\log N} \Big)^{1/3} \tag{3.5}$$

for $N \to \infty$.

We proceed to determine the size of the numbers which are being tested for smoothness in Step 2 of the algorithm, that is, the numbers given in (2.4). In what follows, assume that N is sufficiently large that $d = \delta > 1$. Since the least non-negative residue of c_w mod e is less than d, the largest gap in sequence (2.2), call it γ, is at least

$$e\Big(\frac{1}{w-1} - \lambda\Big),$$

where

$$\lambda = \frac{d-1}{e(w-1)}.$$

Recall that the coefficients of f are bounded in absolute value by $2^{e-\gamma}$. Thus the quantity in (2.4) is at most

$$2dM^{d+1}(2^e)^{2-\frac{1}{w-1}+\lambda}.$$

Letting $\theta = (2w-3)/(w-1)$ and using the fact that $e = \lfloor c_w/d \rfloor$, we see that this bound is less than or equal to $2dM^{d+1}(2^{c_w})^{(\theta+\lambda)/d}$, which is at most

$$2dM^{d+1}(2N)^{\frac{\theta+\lambda}{d}}. \tag{3.6}$$

The remainder of our argument in support of Conjecture 3.3 is the same as the one given in §11 of [1]. We sketch only the first part here.

Consider the case that Algorithm 2.1 is run with e as given and with

$$B = M = L_N[1/3; (4\theta/9)^{1/3}].$$

Let z equal the quantity in (3.6). For any positive real numbers x and y, let $\psi(x,y)$ denote the number of integers at most x which are y-smooth. It is shown in [2] that for any $\epsilon > 0$,

$$\frac{\psi(x, x^{1/w})}{x} = w^{-w(1+o(1))} \tag{3.7}$$

for $w \to \infty$, uniformly for $x \geq w^{w(1+\epsilon)}$. It follows that

$$\frac{M^2\psi(z,B)}{z} = B^{1+o(1)}$$

for $N \to \infty$. Note that we have relied here on the fact that d satisfies (3.5) and that $\lambda = o(1)$ for $N \to \infty$. We now adopt the assumption that for N sufficiently large, the numbers tested for smoothness in the algorithm behave with respect to the property of being B-smooth like random numbers at most z. The reason that our result is conjectural is that this assumption is unproved. Since the number of pairs (a,b) considered in Step 2 is approximately $12M^2/\pi^2$, we conclude that this step produces a set U of size $B^{1+o(1)}$ for $N \to \infty$. Under the assumption that J is $O(\log N)$, the threshold $\pi(B) + |T| + J$ given in Step 2 is also equal to $B^{1+o(1)}$. Thus, we might expect that U is sufficiently large that the algorithm succeeds. In fact, the values of B and M need only be adjusted slightly. To see that they can be chosen so that

$$B = M = L_N[1/3; (4\theta/9)^{1/3} + o(1)] \tag{3.8}$$

for $N \to \infty$ and so that Step 2 succeeds, see the analysis supporting Conjecture 11.4 in [1]. Finally note that Step 2 runs in time $M^{2+o(1)}$ and that when d satisfies (3.5) and J is $O(\log N)$, Step 3 runs in time at most $B^{2+o(1)}$, even in the case that no information about the factorization of (2.4) is retained from the sieving process and trial division is used to compute the needed valuations. Both quantities equal (3.4) when (3.8) holds. This concludes our argument in support of Conjecture 3.3.

The running time given in Conjecture 3.3 is best possible in the following sense. Let z denote the quantity in (3.6), and assume that the input parameters B and M satisfy

$$\frac{M^2\psi(z,B)}{z} \geq B^{1+o(1)}$$

for $N \to \infty$, as we believe they must for Algorithm 2.1 to succeed. Then it is possible to show rigorously that M is at least $L_N[1/3; (4\theta/9)^{1/3} + o(1)]$. See §10 and §11 of [1] for details.

Conjecture 3.3, however, does not give the right answer. It turns out that for $w > 2$, it is not possible to find an infinite set of integers for which the bound in (3.6) is attained. Indeed, we might expect that for fixed w, Algorithm 2.1 would run in time (3.4) for N of the form

$$2^c + \sum_{i=0}^{w-2} 2^{\left\lfloor \frac{\eta}{w-1} \right\rfloor i},$$

where

$$\eta = \frac{c}{\left(\frac{3\theta \log 2^c}{2 \log\log 2^c}\right)^{1/3}}.$$

Certainly, for these numbers, if e is set equal to $\lfloor \eta \rfloor$, then Step 1 of Algorithm 2.1 produces a polynomial of degree d satisfying (3.5), the bound (3.6) is the right one to use, and the algorithm runs in time (3.4). However, choosing e to be a little larger improves the asymptotic running time by forcing the largest gap in sequence (2.2) to be greater than $e/(w-1)$.

More generally, we have the following strategy. Assume we are given as input an integer N of weight w, represented as $\sum_{i=1}^{w} \epsilon_i 2^{c_i}$, with $\epsilon_i \in \{1, -1\}$ and $c_w > c_i$ for $i = 1, \dots, w-1$. Assume also that $c_w \geq w$ and let δ be a multiple of w which is at most c_w. We proceed by producing two polynomials according to the method described in Step 1 of Algorithm 2.1, one with e equal to $e_1 = \lfloor c_w/\delta \rfloor$ and the other with e set equal to $e_2 = \lfloor e_1 w/(w-1) \rfloor$. We then continue the algorithm with the polynomial which produces the smaller smoothness candidates. It turns out that when we optimize this method, we obtain for all $w > 2$, a running time which is slightly better than that of Conjecture 3.3. To see why this is so, first note that because w divides δ, the least non-negative residue of c_w mod e_2 is less than 2δ and hence asymptotically negligible. Next let γ_1 be the largest gap in sequence (2.2) in the case that e_1 is used, define γ_2 analogously, and observe that if γ_1 is close to $e_1/(w-1)$, then the least non-negative residue mod e_1 of each c_i is necessarily close to a multiple of $e_1/(w-1)$. It follows that, in this case, γ_2 is close to $me_1/(w-1)$ with $m \geq 2$. Thus, when e_1 is not a good choice, γ_2 is significantly better than the worst-case value of $e_2/(w-1)$. We leave the remaining details to the interested reader and offer instead the following result, which shows that the room for improvement over Conjecture 3.3 is quite small.

Conjecture 3.9. *Fix w and let*

$$\tau = \frac{\sqrt{2}w - 2\sqrt{2} + 1}{w-1}.$$

Then there exists an infinite set of integers of weight w with the property that for N in this set, the time required by Algorithm 2.1 to produce a matrix A is at least

$$L_N[1/3; (32\tau^2/9)^{1/3} + o(1)] \tag{3.10}$$

for $N \to \infty$.

The secondary constant in (3.10) is, of course, bounded by the secondary constant in the running time proposed in Conjecture 3.3. Figure 3.11 gives the values of these constants for small values of w. All entries are rounded to the nearest thousandth. We note that in the case that $w = 2$, our upper and lower bounds equal $(32/9)^{1/3}$ and that both constants approach $(64/9)^{1/3}$ as $w \to \infty$.

	λ	$(32\tau^2/9)^{1/3}$	$(32\theta/9)^{1/3}$
$w = 2$	1.526	1.526	1.526
$w = 3$	1.729	1.730	1.747
$w = 4$	1.781	1.796	1.810
$w = 5$	1.805	1.828	1.839
$w = 6$	1.819	1.847	1.857
$w = 7$	1.828	1.860	1.868
$w = 8$	1.835	1.869	1.876
$w = 9$	1.840	1.876	1.882
$w = 10$	1.843	1.881	1.887

FIGURE 3.11

We delay the argument in support of Conjecture 3.9, in which we explicitly demonstrate a set with the stated property, to the appendix at the end of the paper. For now we observe that, although Conjectures 3.3 and 3.9 give important information about Algorithm 2.1 and are easily juxtaposed with analogous conjectures for the SNFS and the general NFS, they are deceptive. On the one hand, the asymptotic improvement over the general NFS that they report suggests, overly optimistically, that for a given integer N, Algorithm 2.1 should be the method of choice. However, for a given weight w, this improvement is not necessarily realized unless N is sufficiently large. Indeed, in §4 we observe that in the worst case, Algorithm 2.1 runs faster than standard NFS methods only if w is at most of the order of d, that is, of the order of $(\log N/\log\log N)^{1/3}$. On the other hand, Conjectures 3.3 and 3.9 are overly pessimistic. As noted earlier, for a given input N, the running time of Algorithm 2.1 is not so much determined by the weight of N but by the largest gap in sequence (2.2). As the next result shows, this gap is significantly greater on average than the value $e/(w-1)$ used to obtain Conjecture 3.3, especially when w is small.

Proposition 3.12. *Let w and e be integers greater than or equal to 2. Let $S = S(e, w)$ be the set of sequences of length $w-2$ of non-negative integers less than e. For $s \in S$, let $g(s)$ be the largest difference between consecutive terms of the sequence obtained by ordering the elements of s, together with the numbers 0 and e, from smallest to largest. Then the average value of $g(s)$ as s ranges over S is*

$$\frac{1}{e^{w-2}} \sum_{k=1}^{w-2} \frac{k^{w-2-k}(e!)\,G(e,k+2)}{(e-k)!}, \tag{3.13}$$

where

$$(3.14)\qquad G(e,k) = \frac{\sum\limits_{g=\lceil \frac{e}{(k-1)} \rceil}^{e} g \sum\limits_{j=1}^{\lfloor \frac{e}{g} \rfloor} (-1)^{j+1} \binom{k-1}{j} \sum\limits_{l=1}^{\lfloor \frac{e}{jg} \rfloor} (-1)^{l+1} \binom{k-2}{l-1} \binom{e-1-(j+l-1)g}{k-(j+2)}}{\binom{e}{k-2}}.$$

Proof. For a given w, let $S_0 = S_0(e,w)$ be the set of increasing sequences of length $w-2$ of non-negative integers less than e. We begin by calculating the average value of $g(s)$ as s varies over S_0. We accomplish this by counting, for a given value g, the number of s such that $g(s) = g$. The first step is to determine for a given g, how many ways there are to partition $e-g$ into $w-2$ parts, subject to the condition that each summand is at least 1 and at most g, and then to multiply by $w-1$ to account for the fact that the maximum gap g can occur at any one of the $w-1$ parts of e. Standard, elementary, combinatorial techniques yield the answer

$$(w-1) \sum_{j=1}^{\lfloor e/g \rfloor} (-1)^{j+1} \binom{w-2}{j-1} \binom{e-1-jg}{w-3}.$$

Of course, this tally is too large since those partitions of e containing more than one g have been counted more than once. We thus need to subtract off the number of ways to partition $e-2g$ into $w-3$ parts, multiplied by $\binom{w-1}{2}$, add back in the number of ways to partition $e-3g$ into $w-4$ parts, multiplied by $\binom{w-1}{3}$, and so on. Multiplying the resulting alternating sum by g, summing over all the possible values of g, and dividing by $|S_0|$ yields the value $G(e,w)$ given in (3.14).

Since $G(e,w)$ is also the average value of $g(s)$ as s ranges over the set of sequences of length $w-2$ of distinct non-negative integers less than e, it only remains to handle those sequences in $S(e,w)$ with repeated terms. We classify these according to how many distinct terms each such sequence contains. In particular, we observe that for $k = 1, \ldots, w-2$, the number of sequences in $S(e,w)$ containing k distinct terms is $k^{w-2-k}|S_0(e,k)|$. The average value of $g(s)$ for these sequences is $G(e,k+2)$. We find, therefore, that the average value of $g(s)$ over $S(e,w)$ is

$$\frac{\sum\limits_{k=1}^{w-2} k^{w-2-k} |S_0(e,k)| G(e,k+2)}{|S(e,w)|}.$$

This expression is the same as (3.13), and the proof of the proposition is complete. □

Proposition 3.12 gives us an idea of what to expect when running Algorithm 2.1 on a number N of weight w. Let c be the largest exponent appearing in the weight w representation of N. Then the quantity in (3.13) represents, roughly speaking, the size of the gap we can expect in sequence (2.2) when Algorithm 2.1 is run with e chosen so that the least non-negative residue of c mod e is negligible. It is easy enough to compare this value with the worst-case value of $e/(w-1)$. We can make a comparison, however, that does not depend on e by noting that (3.13) is asymptotically linear in e. That is, as $e \to \infty$ with w fixed, the ratio of (3.13) to e approaches a constant which we call κ_w. In Figure 3.15, we give for small w, the value of (3.13) divided by e when $e = 200$ and when $e = 1000$, the value of κ_w, and for comparison, the value of $1/(w-1)$. All entries are rounded to the nearest thousandth.

	$e = 200$	$e = 1000$	κ_w	$1/(w-1)$
$w = 3$	.753	.751	.750	.500
$w = 4$	.604	.610	.611	.333
$w = 5$	.508	.518	.521	.250
$w = 6$	.439	.453	.457	.200
$w = 7$	.386	.404	.408	.167
$w = 8$	.340	.364	.370	.143
$w = 9$	.301	.332	.340	.125
$w = 10$	.268	.305	.314	.111

FIGURE 3.15

The chart reveals clearly that one can expect in practice to fare much better than one would in the worst case. Going one step further, we can substitute $2 - \kappa_w$ for θ in (3.6) and modify the statement of Conjecture 3.3 accordingly. In particular, quantity (3.4) with θ replaced by $2 - \kappa_w$ represents a kind of average running time for Algorithm 2.1 to succeed upon input of an integer of weight w. We leave a precise formulation of this statement to the reader.

We conclude this section by remarking that the algorithms which use the matrix A produced by Algorithm 2.1 to factor N or compute discrete logarithms mod N have running times dominated by a linear algebra computation which runs in time $B^{2+o(1)}$ for $N \to \infty$. Since (3.4) is equal to $B^{2+o(1)}$ for the value of B given in (3.8) and since (3.10) is a lower bound, we see that Conjectures 3.3 and 3.9 apply to the full factoring and discrete logarithm algorithms that incorporate Algorithm 2.1.

4. EXAMPLES

In practice, when confronted with a particular N which one wants to factor or modulo which one wants to compute logarithms, asymptotic results are of limited interest. One simply wants to choose the best method for the number at hand. In this section, we compare the performance of Algorithm 2.1 with that of other versions of the number field sieve (NFS).

The chief factor in determing how fast the sieving stage of these methods runs for a given input is the size of the integers being tested for smoothness. The algorithm to choose is the one with the smallest smoothness candidates. In the standard NFS for factoring, these numbers are at most

$$2(d+1)M^{d+1}N^{2/(d+1)}, \tag{4.1}$$

where M as usual is the bound on the size of the coefficients of the elements considered for smoothness and d is the degree of the proper extension of $\mathbb{Q}$ employed in the algorithm. In the case of discrete logarithms modulo a prime, the best approach is due to Joux and Lercier and makes use of two extensions of $\mathbb{Q}$ generated by the roots of two polynomials which have a shared root mod N and whose degrees are consecutive integers ([5]). The smoothness candidates in this method are bounded by

$$\Big(\frac{d^2}{4} + \frac{3d}{2} + 2\Big)M^{d+1}N^{2/(d+2)}, \tag{4.2}$$

where $d+1$ is the sum of the degrees of the two extensions. Note that this sum is necessarily odd. The method of Joux and Lercier can be modified so that two extensions with degrees over $\mathbb{Q}$ of the same parity are used. In this case, however, the bound on the smoothness candidates is at least as big as (4.1).

Comparison of (4.1) and (4.2) with the corresponding bound appearing in the analysis supporting Conjecture 3.3 provides some information about how Algorithm 2.1 measures up to other versions of the NFS. For instance, if we ignore the small factors in (3.6) and (4.1) and compare $M^{d+1}N^{\theta/d}$, where $\theta = (2w-3)/(w-1)$, with $M^{d+1}N^{2/(d+1)}$, we see that in the worst case Algorithm 2.1 only outperforms other NFS methods if w is bounded by a value close to $(d+3)/2$. Since the optimal value of d grows very slowly with respect to N, this condition represents a severe restriction on w.

The usefulness of such a comparison, however, is compromised by the fact that (3.6) is, in general, a poor estimate for the size of the largest smoothness candidate tested in Algorithm 2.1, whereas the bounds for the other methods are generally attained. A comparison of methods, therefore, should be made on a case by case basis, using (4.1), (4.2) for d even, and a tighter bound for Algorithm 2.1. This is precisely what we do with the examples from [6]. We begin with the three examples that involve the computation of logarithms in prime fields. The remaining four examples concern logarithms in fields of degree two and are discussed subsequently.

Let N be a prime input to Algorithm 2.1, given in signed-binary form. Let c be the largest exponent appearing in this representation. The upper bound we use for the size of the smoothness candidates arising in the algorithm is the quantity

$$2dM^{d+1}2^{2e-\gamma},$$

where for a given e, we let γ be the largest gap in sequence (2.2) and for a given d, we choose e between $c/(d+1)$ and $c/(d-1)$ so as to minimize the exponent $2e-\gamma$. For fixed d, we can easily compare this value to (4.1) and (4.2), since the M^{d+1} term can be ignored. When we compare bounds for different values of d, however, the dependency on M creates a problem. We overcome this difficulty by minimizing for a given d, the maximum of M and the smoothness bound B, subject to the constraint that sufficiently many smoothness pairs are found during the sieving stage of the algorithm. We use the resulting minima for our rankings. Our choice of $\max\{M, B\}$ as target function reflects our desire to take into account the linear algebra step that dominates the NFS once the sieving is complete. Indeed, at least asymptotically, this is the right function to minimize since the sieve in the NFS runs in time $M^{2+o(1)}$ and the linear algebra runs in time $B^{2+o(1)}$.

Proceeding with the optimization, we approximate the number of smooth pairs collected in the NFS, as we did in §3, by

$$\frac{(12/\pi^2)M^2\psi(z,B)}{z},$$

where z is a bound on the size of the smoothness candidates. The number of pairs needed is roughly equal to the number of degree one prime ideals of norm at most B in the two rings being used. For each ring, this number is approximately the number of rational primes at most B ([16]), so we adopt the value $2B/\log B$ as our estimate for the number of smooth pairs required. Finally, we use (3.7), with the $o(1)$ dropped, to approximate $\psi(z,B)/z$. We arrive then at the problem of minimizing, for a given d and a given expression for the bound z, the value of $\max\{M, B\}$ subject to the condition that

$$\frac{M^2 u^{-u} \log B}{B} \geq \frac{\pi^2}{6},$$

where $u = (\log z)/\log B$. By comparing the values produced, as we vary d and the NFS version used, we can determine which algorithm is fastest for N and obtain an estimate for its running time in this case. We caution, however, that the estimates we find this way are very rough and are generally higher than those produced by means of extrapolation techniques such as are used in [7].

For each example that follows, we report for a range of values of d, the bit length of z/M^{d+1} evaluated for each of the three expressions for z under consideration and the outcome of the optimization problem for the case that z/M^{d+1} is minimal. Though they are not of practical interest, we include the values obtained when z is given by (4.2) and d is odd. In no case do these yield an estimated running time faster than the one obtained by considering the other bounds given.

For those N for which Algorithm 2.1 is the best choice, we also compute an estimate of the size of a general prime for which the NFS variation of Joux and Lercier runs in time approximately equal to that required by Algorithm 2.1 to compute logarithms mod N. More specifically, we determine the bit length of the smallest general prime having the property that, as we let d vary over the positive even integers, the smallest solution to the optimization problem just described using bound (4.2) is equal to the optimal value of $\max\{M, B\}$ associated to N. This bit length then can be thought of as a measure of the NFS-security of N. The reader can check that these security estimates do not change if the solutions to the minimization problem using (4.1) for d odd are taken into account.

For each example that follows, we only include data for those d which are close to optimal. In each case, the interval from $\lfloor ((3/2)\log N/\log\log N)^{1/3} \rfloor$ to $\lceil (3\log N/\log\log N)^{1/3} \rceil$ is contained in the range of values of d presented. All entries in the charts are bit lengths. Only powers of 2 were considered when determining the optimal values of $\max\{M, B\}$. In those cases that we give the NFS-security of a prime, we round to the nearest 10.

Example 4.3. We consider the weight 7 prime

$$2^{3202} - 2^{3121} + 2^{3038} + 2^{2947} - 2^{2865} + 2^{2690} + 1.$$

In this case, we find that the method of Joux and Lercier beats Algorithm 2.1. The values given below in the $\max\{M, B\}$ column are obtained using the bound from the former method. We note that there are values of d for which the polynomial obtained using Algorithm 2.1 is superior to the pair given by the method of Joux and Lercier. Indeed, when the bit length of e is close to 90 or 180, the polynomial produced by Algorithm 2.1 has very small coefficients. The degree in these cases, however, is far too large to be practical.

	$2d2^{2e-\gamma}$	$2(d+1)N^{2/(d+1)}$	$(\frac{d^2}{4}+\frac{3d}{2}+2)N^{2/(d+2)}$	$\max\{M, B\}$
$d = 5$	975	1071	919	75
$d = 6$	819	919	805	73
$d = 7$	751	805	717	72
$d = 8$	661	716	646	72
$d = 9$	588	645	588	73
$d = 10$	547	587	540	74

Example 4.4. Our second example, also of weight 7, is the prime

$$2^{8376} - 2^{8333} + 2^{8288} - 2^{7991} + 2^{7947} + 2^{7604} + 1.$$

In this case, we find that Algorithm 2.1 is the winner. Moreover, the value of 102 bits in the $\max\{M, B\}$ column corresponds to the optimal value of $\max\{M, B\}$ obtained when using the method of Joux and Lercier on a prime of 7310 bits. Thus the NFS-security of this prime is only about 7310 bits.

	$2d2^{2e-\gamma}$	$2(d+1)N^{2/(d+1)}$	$(\frac{d^2}{4} + \frac{3d}{2} + 2)N^{2/(d+2)}$	$\max\{M, B\}$
$d = 9$	1428	1680	1529	105
$d = 10$	1252	1528	1402	102
$d = 11$	1157	1401	1295	103
$d = 12$	1090	1294	1203	104
$d = 13$	1026	1202	1123	105

Example 4.5. Our final example for discrete logarithms in a prime field is the field of size

$$2^{15474} - 2^{14954} + 2^{14432} + 1.$$

Once again, we find that Algorithm 2.1 is the best method. Based on the optimal $\max\{M, B\}$ for $d = 14$, we estimate the NFS-security of this prime to be approximately 13030 bits.

	$2d2^{2e-\gamma}$	$2(d+1)N^{2/(d+1)}$	$(\frac{d^2}{4} + \frac{3d}{2} + 2)N^{2/(d+2)}$	$\max\{M, B\}$
$d = 12$	1977	2386	2217	132
$d = 13$	1785	2216	2070	130
$d = 14$	1621	2069	1941	129
$d - 15$	1522	1940	1827	130
$d = 16$	1461	1826	1726	132

The remaining examples involve fields of degree two over their prime field. Let F be such a field and denote by N its characteristic. Logarithms can be computed in F using the NFS in much the same way as in the prime case. The major difference stems from the fact that F cannot be represented as a quotient of $\mathbb{Z}$ but instead is represented as the quotient of an order of a quadratic extension K of $\mathbb{Q}$. As a result, the number fields employed in the NFS are extensions of K. In one approach, K itself is used in conjunction with a field produced by adjoining to K the root of a polynomial with integral coefficients, preferably small, which has a root mod N, also preferably small ([13]). Such a polynomial can be obtained by the usual methods employed in factoring versions of the NFS or, in the case that N has small weight, by the technique given in the first step of Algorithm 2.1. A second approach is to follow the method of Joux and Lercier and produce two polynomials with a shared root mod N and, in turn, two extensions of K. Regardless of strategy, if we define M suitably and make various simplifying assumptions regarding the complications introduced by the replacement of $\mathbb{Q}$ by K, we arrive at the same optimization problem as in the prime case, except that the factors other than M^{d+1} appearing in the formula adopted for z must be squared. For the examples that follow, therefore, we provide the same data as given in the charts above except that the bit lengths in the first three columns are computed using the squares of the expressions used previously. For each finite field, we again include an estimate of the field's NFS-security. Because the weights of the primes in these examples are so low, Algorithm 2.1 registers significant gains.

Example 4.6. Our first example of a field of degree two is the one of characteristic

$$2^{520} + 2^{363} - 2^{360} - 1.$$

The optimal $\max\{M, B\}$ value of 42 for $d = 6$ and $d = 7$ corresponds to what we expect for a degree-two field of approximately 850 bits. Note that the best bounds for $d = 2$ and $d = 5$ are obtained using the expression associated to the method of Joux and Lercier. Of course, in the case that $d = 5$, this bound is not realized because $d + 1$ is even.

	$4d^2 2^{2(2e-\gamma)}$	$4(d+1)^2(N^2)^{2/(d+1)}$	$(\frac{d^2}{4}+\frac{3d}{2}+2)^2(N^2)^{2/(d+2)}$	$\max\{M, B\}$
$d = 2$	685	699	526	50
$d = 3$	386	527	423	45
$d = 4$	327	423	354	44
$d = 5$	317	354	306	45
$d = 6$	216	305	269	42
$d = 7$	180	269	241	42
$d = 8$	199	240	218	47

Example 4.7. Our next example is the degree-two field of characteristic

$$2^{1582} + 2^{1551} - 2^{1326} - 1.$$

Based on the case $d = 7$, we obtain an NFS-security of approximately 2240 bits, which is significantly lower than the 3162 bit size of the field.

	$4d^2 2^{2(2e-\gamma)}$	$4(d+1)^2(N^2)^{2/(d+1)}$	$(\frac{d^2}{4}+\frac{3d}{2}+2)^2(N^2)^{2/(d+2)}$	$\max\{M, B\}$
$d = 5$	799	1062	912	70
$d = 6$	594	912	800	64
$d = 7$	520	800	713	63
$d = 8$	521	712	643	66
$d = 9$	521	642	586	70
$d = 10$	491	585	539	72

Example 4.8. In this example, we consider the prime

$$2^{4231} - 2^{3907} + 2^{3847} - 1.$$

We estimate that the NFS-security of the field of degree two in this case is 5780 bits, again far below the size of the field.

	$4d^2 2^{2(2e-\gamma)}$	$4(d+1)^2(N^2)^{2/(d+1)}$	$(\frac{d^2}{4}+\frac{3d}{2}+2)^2(N^2)^{2/(d+2)}$	$\max\{M,B\}$
$d=8$	1477	1889	1703	103
$d=9$	1243	1702	1549	99
$d=10$	1055	1548	1422	96
$d=11$	901	1420	1314	93
$d=12$	830	1312	1221	94
$d=13$	778	1219	1141	96

Example 4.9. Our final example is the weight 3 prime

$$2^{7746} - 2^{6704} - 1.$$

Our estimate for the NFS-security for the field in this case is merely 9520 bits.

	$4d^2 2^{2(2e-\gamma)}$	$4(d+1)^2(N^2)^{2/(d+1)}$	$(\frac{d^2}{4}+\frac{3d}{2}+2)^2(N^2)^{2/(d+2)}$	$\max\{M,B\}$
$d=9$	2093	3107	2827	124
$d=10$	2093	2826	2593	128
$d=11$	2093	2592	2395	131
$d=12$	1802	2393	2225	127
$d=13$	1502	2223	2078	121
$d=14$	1250	2076	1949	117
$d=15$	1062	1947	1836	114
$d=16$	1127	1833	1735	121

Appendix

This additional section is devoted to providing support for Conjecture 3.9, which we relabel and restate here. We continue with the notation introduced in §2 and in §3.

Conjecture A.1. *Fix w, and let*

$$\tau = \frac{\sqrt{2}w - 2\sqrt{2} + 1}{w-1}.$$

Then there exists an infinite set S of integers of weight w with the property that for $N \in S$, the time required by Algorithm 2.1 to produce a matrix A is at least

$$L_N[1/3; (32\tau^2/9)^{1/3} + o(1)] \tag{A.2}$$

for $N \to \infty$.

We proceed by exhibiting a set S with the stated property. Fix w and for any positive real number ρ and any integer $c \geq 2$, let

$$h = h_\rho(c) = \rho\Big(\frac{\log 2^c}{\log\log 2^c}\Big)^{1/3}.$$

In the case that $c \geq (w-1)(h+1)$, let

$$N_\rho(c) = 2^c + \sum_{i=0}^{w-2} 2^{\left\lfloor \frac{c}{(w-1)(h+1)} \right\rfloor i}.$$

Finally, let

$$\zeta = \frac{\sqrt{2}+1}{2(\frac{2}{3}\tau)^{1/3}}.$$

Our set S then is the set of numbers $N_\varsigma(c)$.

To determine what happens when the numbers in S are input into Algorithm 2.1, we consider more generally how the algorithm performs on inputs of the form $N = N_\rho(c)$ for some arbitrary but fixed ρ. We distinguish between three cases, based on the size of the degree d of the polynomial f produced in Step 1. Critical to our approach is the observation that, as a consequence of the way in which f is formed, the parameter e and the degree d satisfy

$$\frac{c}{d+1} \le e \le \frac{c}{d-1}.$$

From this point forward, we write c_{w-1} in place of $\lfloor c/((w-1)(h+1))\rfloor(w-2)$.

Case 1. Assume that $d \le h$. Then $e \ge \frac{c}{d+1} \ge \frac{c}{h+1}$, and consequently,

$$\begin{aligned} e - c_{w-1} &\ge \frac{c}{h+1} - (w-2)\left\lfloor \frac{c}{(w-1)(h+1)} \right\rfloor \\ &\ge (w-1)\left\lfloor \frac{c}{(w-1)(h+1)} \right\rfloor - (w-2)\left\lfloor \frac{c}{(w-1)(h+1)} \right\rfloor \\ &\ge \left\lfloor \frac{c}{(w-1)(h+1)} \right\rfloor. \end{aligned}$$

Because of the special form of N, the maximal gap in sequence (2.2) is equal either to $\lfloor c/((w-1)(h+1))\rfloor$ or to $e - c_{w-1}$, depending on where the least non-negative residue of c mod e appears in the sequence. In both cases, this maximal gap is at most $e - c_{w-1}$, and we conclude that the maximum absolute value of the smoothness candidates in Step 2 is at least

$$M^{d+1}(2^e-1)2^{e-(e-c_{w-1})} = M^{d+1}(2^e-1)2^{(w-2)\left\lfloor \frac{c}{(w-1)(h+1)} \right\rfloor}.$$

Note that this quantity is bounded below by

$$l_1 M^{d+1}(2^c)^{\frac{1}{d+1} + \frac{w-2}{(w-1)(h+1)}}$$

for some constant l_1 which does not depend on c.

Case 2. Assume that

$$h < d \le (w-1)h.$$

Then $e \le \frac{c}{d-1} < \frac{c}{h-1}$. If e is in addition less than c_{w-1}, then the largest gap in sequence (2.2) is at most $\lfloor c/((w-1)(h+1))\rfloor$. If e is not less than c_{w-1}, then it may be that the largest gap in (2.2) is equal to $e - c_{w-1}$ and that this difference is greater than $\lfloor c/((w-1)(h+1))\rfloor$. However,

$$e - c_{w-1} \le \frac{c}{h-1} - (w-2)\left\lfloor \frac{c}{(w-1)(h+1)} \right\rfloor \le \frac{c}{(w-1)(h+1)} + \frac{2c}{h^2-1} + w - 2.$$

We conclude that the maximum absolute value of the smoothness candidates in Algorithm 2.1 is at least

$$M^{d+1}(2^e-1)2^{e - \frac{c}{(w-1)(h+1)} - \frac{2c}{h^2-1} - (w-2)},$$

which is at least

$$l_2 M^{d+1}(2^c)^{\frac{2}{d+1} - \frac{1}{(w-1)(h+1)} - \frac{2}{h^2-1}} \tag{A.3}$$

for some constant l_2. Observe that the upper bound on d ensures that the exponent on 2^c in (A.3) is greater than or equal to

$$\frac{1}{d+1} - \frac{2}{h^2-1},$$

which itself is bounded below by

$$\frac{1}{d+1} - \frac{2(w-1)}{(h-1)(h(w-1)+1)} \geq \frac{1+o(1)}{d+1},$$

where the $o(1)$ is for $c \to \infty$, uniformly in d.

Case 3. Assume that $d > (w-1)h$. In this case, we draw no conclusions about the coefficients of the polynomial f and simply point out that the maximum absolute value of the numbers tested for smoothness in Algorithm 2.1 is at least $M^{d+1}(2^e-1)$. This lower bound, in turn, is at least

$$l_3 M^{d+1}(2^c)^{\frac{1}{d+1}}$$

for some constant l_3.

Considering Cases 1-3 together, we see that when Algorithm 2.1 is run with input $N = N_\rho(c)$, the maximum absolute value of the smoothness candidates is at least

$$z = z(c,d) = lM^{d+1}2^{c\beta},$$

where l is the minimum of l_1, l_2, and l_3 and

$$\beta = \beta(c,d) = \begin{cases} \frac{1}{d+1} + \frac{w-2}{(w-1)(h+1)} & \text{if } d \leq h, \\ \frac{2}{d+1} - \frac{1}{(w-1)(h+1)} - \frac{2}{h^2-1} & \text{if } h < d \leq (w-1)h, \\ \frac{1}{d+1} & \text{if } (w-1)h < d. \end{cases}$$

Recall here that h depends on ρ and is a function of c. Since we are interested in a lower bound on the running time of Algorithm 2.1, we can assume that all the smoothness candidates are, in fact, bounded by z in absolute value. If we then assume that these numbers behave with respect to B-smoothness as random numbers bounded by z, we find that any value of M which is sufficiently large that the method succeeds must satisfy

$$\frac{(12/\pi^2)M^2\psi(z,B)}{z} \geq \pi(B) + |T| + J,$$

where T and J are as in the description of the algorithm.

Next, we rely on results from §10 of [1]. In particular, Lemma 10.1 tells us that because $\pi(B) + T + J \geq B^{1+o(1)}$ for $B \to \infty$, we are guaranteed that

$$M^2 \geq L_z[1/2; \sqrt{2} + o(1)],$$

where the $o(1)$ is for $z \to \infty$. Lemma 10.12 then supplies the necessary manipulations to conclude that $2\log M$ is at least

$$\text{(A.4)} \qquad (1+o(1))\Big(d\log\, d + \sqrt{(d\log\, d)^2 + 2\log(2^{c\beta})\log\log(2^{c\beta})}\,\Big)$$

for $c \to \infty$, uniformly in d, as long as we restrict ourselves to pairs (c,d) such that

(i) $\log(2^{c\beta}) \geq 1$ for c sufficiently large and
(ii) $d + \log(2^{c\beta}) \to \infty$ as $c \to \infty$, uniformly in d.

This constraint, however, is of no consequence. The fact that

$$\beta \geq \frac{1+o(1)}{d+1} \tag{A.5}$$

for $c \to \infty$, uniformly in d, implies that condition (ii) is satisfied. With regard to (i), observe that in Step 2 of Algorithm 2.1, the numbers tested for smoothness are greater than 2^d in absolute value whenever the coefficients a and b are greater than 1. As stated in the discussion of Conjecture 3.3, the running time given in that conjecture cannot be improved upon if we maintain the assumption that the smoothness candidates in Step 2 behave like random positive integers at most the quantity given in (3.6). The reader can verify that with all parameters optimized, (3.6) is equal to $L_N[2/3;\zeta]$ for some constant ζ. We conclude that the only (c,d) of interest are those for which $2^d \leq L_N[2/3;\zeta]$, in which case $d+1 < c/2$. For such (c,d), condition (i) is now met as a consequence of (A.5).

Our final task is to minimize (A.4). Following §11 of [1], we note that for each of the three cases used to formulate the function β, this is accomplished by having $(d \log d)^2$ be of the same order of magnitude as the term accompanying it under the square root sign. Using (A.5) and the fact that β is bounded above by $2/(d+1)$, we determine that in each case, d must be of the same order of magnitude as $(\log 2^c/\log\log 2^c)^{1/3}$. We proceed to express (A.4) as a function of c by using the definitions of β and h, and by substituting $k(\log 2^c/\log\log 2^c)^{1/3}$ for d. As a result, we find that the minimum of (A.4) is

$$(m+o(1))(\log 2^c)^{1/3}(\log\log 2^c)^{2/3}, \tag{A.6}$$

where the $o(1)$ is for $c \to \infty$ and m is the minimum value of

$$f_\rho(k) = \begin{cases} \frac{k}{3} + \left(\frac{k^2}{9} + \frac{4}{3k} + \frac{4(w-2)}{3(w-1)\rho}\right)^{\frac{1}{2}} & \text{if } k \leq \rho, \\ \frac{k}{3} + \left(\frac{k^2}{9} + \frac{8}{3k} - \frac{4}{3(w-1)\rho}\right)^{\frac{1}{2}} & \text{if } \rho \leq k \leq (w-1)\rho, \\ \frac{k}{3} + \left(\frac{k^2}{9} + \frac{4}{3k}\right)^{\frac{1}{2}} & \text{if } k \geq (w-1)\rho. \end{cases}$$

Since $2^c < N_\rho(c) < 2^{c+1}$, we can replace the quantity 2^c in (A.6) with $N_\rho(c)$. All that remains then is to let ρ equal the constant ζ defined at the beginning of the appendix. We leave it to the reader to verify that the minimum of $f_\zeta(k)$ occurs at $k = (3\tau^2/2)^{1/3}$ and $k = (3\tau^2/2)^{1/3}/\sqrt{2}$ and is equal to $(32\tau^2/9)^{1/3}$. Since (A.6) is a lower bound for $2\log M$ and the time required by the sieve in Step 2 of Algorithm 2.1 is equal to $M^{2+o(1)}$, we see that the running time of the method is at least (A.2) on the set of numbers $N_\zeta(c)$. This concludes our argument in support of Conjecture A.1.

Acknowledgement

The author wishes to thank Neil Koblitz and Alfred Menezes for inquiring about a low weight version of the number field sieve and for their subsequent comments on this paper.

References

[1] J.P. Buhler, H.W. Lenstra, Jr., C. Pomerance, *Factoring integers with the number field sieve*, in [8], pp. 50–94. MR1321221

[2] E.R. Canfield, P. Erdös, C. Pomerance, *On a problem of Oppenheim concerning "factorisatio numerorum"*, J. Number Theory **17** (1983), 1–28. MR712964 (85j:11012)

[3] A. Commeine, I. Semaev, *An algorithm to solve the discrete logarithm problem with the number field sieve*, Public Key Cryptography, Lecture Notes in Comput. Sci., vol. 3958, Springer-Verlag, Berlin, 2006, pp. 174–190. MR2423189

[4] M. Filaseta, *A further generalization of an irreducibility theorem of A. Cohn*, Canad. J. Math. **34** (1982), 1390–1395. MR678678 (85g:11014)

[5] A. Joux, R. Lercier, *Improvements on the general number field sieve for discrete logarithms in prime fields*, Math. Comp. **72** (2003), 953–967. MR1954978 (2003k:11192)

[6] N. Koblitz, A. Menezes, *Pairing-based cryptography at high security levels*, Cryptography and Coding: 10th IMA International Conference, Lecture Notes in Comput. Sci, vol. 3796, Springer-Verlag, Berlin, 2005, pp. 13–36. MR2235246 (2007b:94235)

[7] A.K. Lenstra, *Unbelievable security: Matching AES security using public key systems*, Advances in Cryptology - ASIACRYPT 2001, Lecture Notes in Comput. Sci., vol. 2248, Springer-Verlag, Berlin, 2001, pp. 67–86. MR1934516 (2003h:94042)

[8] A.K. Lenstra, H.W. Lenstra, Jr., (eds.), *The development of the number field sieve*, Lecture Notes in Mathematics, 1554, Springer-Verlag, Berlin, 1993. MR1321216 (96m:11116)

[9] A.K. Lenstra, H.W. Lenstra, Jr., M.S. Manasse, J.M. Pollard, *The number field sieve*, in [8], pp. 11–42. MR1321219

[10] G. Manku, J. Sawada, *A loopless Gray code for minimal signed-binary representations*, 13th Annual European Symposium on Algorithms, Lecture Notes in Comput. Sci., vol. 3669, Springer-Verlag, Berlin, 2005, pp. 438-447. MR2257959

[11] K. McCurley, *The discrete logarithm problem*, Cryptology and Computational Number Theory, Proc. Sympos. Appl. Math., vol. 42, Amer. Math. Soc., Providence, RI, 1990, pp. 49–74. MR1095551 (92d:11133)

[12] O. Schirokauer, *Discrete logarithms and local units*, Theory and Applications of Numbers without Large Prime Factors, Philos. Trans. Roy. Soc. London, Ser. A, vol. 345, Royal Society, London, 1993, pp. 409–424. MR1253502 (95c:11156)

[13] O. Schirokauer, *The impact of the number field sieve on the discrete logarithm problem*, Algorithmic Number Theory: Lattices, Number Fields, Curves, and Cryptography, Mathematical Sciences Research Institute Publications, Cambridge University Press, Cambridge, 2008.

[14] J. Solinas, *Generalized Mersenne numbers*, Technical Report CORR 99-39 (1999).

[15] P. Stevenhagen, *The number field sieve*, Algorithmic Number Theory: Lattices, Number Fields, Curves, and Cryptography, Mathematical Sciences Research Institute Publications, Cambridge University Press, Cambridge, 2008.

[16] P. Stevenhagen, H.W. Lenstra, Jr., *Chebotarëv and his density theorem*, The Mathematical Intelligencer **18** (1996), 26–37. MR1395088 (97e:11144)

Department of Mathematics, Oberlin College, Oberlin, Ohio 44074
E-mail address: `oliver.schirokauer@oberlin.edu`