NFS with Four Large Primes: An Explosive Experiment

Bruce Dodson[1], Arjen K. Lenstra[2]

[1] Department of Mathematics, Lehigh University, Bethlehem, PA 18015-3174, U. S. A
E-mail: bad0@Lehigh.EDU
[2] MRE-2Q330, Bellcore, 445 South Street, Morristown, NJ 07960, U. S. A
E-mail: lenstra@bellcore.com

Abstract. The purpose of this paper is to report the unexpected results that we obtained while experimenting with the multi-large prime variation of the general number field sieve integer factoring algorithm (NFS, cf. [8]). For traditional factoring algorithms that make use of at most two large primes, the completion time can quite accurately be predicted by extrapolating an almost quartic and entirely 'smooth' function that counts the number of useful combinations among the large primes [1]. For NFS such extrapolations seem to be impossible—the number of useful combinations suddenly 'explodes' in an as yet unpredictable way, that we have not yet been able to understand completely. The consequence of this explosion is that NFS is substantially faster than expected, which implies that factoring is somewhat easier than we thought.

1 Introduction

For the last ten years all 'general purpose factoring records', i.e., those that are relevant for cryptographical applications of factoring, have been obtained by the quadratic sieve factoring algorithm (QS, cf. [13, 15]). The most recent of these records was the factorization of the 129-digit RSA Challenge number, which was published in 1977 and factored in 1994 using the double large prime multiple polynomial variation of QS [1]. The authors of [1], however, suspected that their factorization would be the last factoring record obtained by QS, and that future records will be set by another, faster method, the general number field sieve (NFS, [8]).

It has been known since 1989 that NFS is asymptotically superior to any of the variants of QS: for $n \to \infty$ it can be expected, on loose heuristic grounds, that it takes time

$$\exp((1.923 + o(1))(\log n)^{1/3}(\log\log n)^{2/3})$$

to factor a composite number n using NFS, as opposed to a (slower) heuristic expected run time

$$\exp((1 + o(1))(\log n)^{1/2}(\log\log n)^{1/2}),$$

D. Coppersmith (Ed.): Advances in Cryptology - CRYPTO '95, LNCS 963, pp. 372-385, 1995.

also for $n \to \infty$, for QS. These heuristic run time estimates do not imply that NFS is also faster than QS in practice. Indeed, it has for some time been suspected that NFS would *never* be practical at all, and that, even if we would be able to get it to work, the crossover point with QS would be far beyond our current range of interest.

In this paper we present some evidence that NFS is actually more practical than expected, and that the crossover point with QS is easily within reach of our current computational resources. Our results indicate that NFS is already substantially faster than QS for numbers in the 115 digit range. Since the 'gap' between the factorization times for these methods only widens for larger numbers, our results imply that the 129-digit number factored in [1] could be factored by NFS in about a quarter of the time spent by QS. The consequences for the strength of 512-bit composites, as sometimes used in cryptographic applications, will be commented on in future work.

One of the major reasons that NFS is performing so much better than expected, is that NFS has a certain advantage over QS that has almost no relation to the theoretical advantage of NFS over QS. Roughly speaking, QS only allows efficient usage of two 'large primes', whereas in NFS it should be possible to use four large primes. Practical experiments that exploit this large prime advantage have so far been limited to three large primes [3]. These experiments did not indicate a distinct advantage of three over two large primes, possibly because the numbers that were factored were rather small. The NFS implementation from [6] allowed us to carry out some large scale experiments with four large primes, which, for the first time, unequivocally proved the advantage of more than two large primes.

In Section 2 we describe why it is easier for NFS to take advantage of large primes than it is for QS. Our experiments and results are presented in Section 3, followed by some of the new methods that were used to obtain our results: an alternative 'cycle' counting method in Section 4, and a discussion of the matrix step in Section 5.

2 Large primes in QS and NFS

Let n be an odd number that is not a prime power.

Large primes in QS. To factor n with QS, one begins by selecting a 'factor base' P, consisting of -1 and certain primes up to some bound B. One then employs a 'sieving' process to efficiently collect a set V of more than $\#P$ 'relations', which are identities modulo n of the form

$$v^2 \equiv \prod_{p \in P} p^{e(v,p)} \bmod n,$$

with v, $e(v,p) \in \mathbf{Z}$. Since $\#V > \#P$, the vectors $e(v) = e(v,p)_{p \in P}$ are linearly dependent. This implies that $\#V - \#P$ subsets W of V can be found (using linear algebra) for which there are linearly independent dependencies of the

form $\sum_{v \in W} e(v) = 2(w(p))_{p \in P}$ with $w(p) \in \mathbf{Z}$. Each W therefore leads to an identity

$$\left(\prod_{v \in W} v\right)^2 \equiv \left(\prod_{p \in P} p^{w(p)}\right)^2 \bmod n$$

of the form $x^2 \equiv y^2 \bmod n$ with $x, y \in Z$. For each such identity there is a chance of at least $1/2$ that $\gcd(x - y, n)$ will be a nontrivial factor of n.

In [9] is was shown that it is advantageous to collect identities of a slightly more general form, namely

$$v^2 \equiv q_1(v) q_2(v) \prod_{p \in P} p^{e(v,p)} \bmod n$$

with v, $e(v,p) \in \mathbf{Z}$ and $q_1(v)$, $q_2(v)$ either equal to 1 or to some prime not in P satisfying $q_1(v) \leq q_2(v) \leq B_2 < B^2$, for some bound B_2 with $B_2^2 < B^3$. If $q_1(v) = q_2(v) = 1$ these are the same as the earlier relations, which will be called 'full' relations from now on; otherwise a relation is called 'partial'. The q_i's are referred to as the 'large primes'. Partial relations are potentially useful because it might be possible that they can be combined into 'cycles': collections of partial relations where all occurring large primes can be combined into squares, thus making the combination 'look like' a full relation. As an example, if v and w are two different partial relations for which $q_1(v) = q_1(w) = 1$ and $q_2(v) = q_2(w)$, then

$$(vw/q_2(v))^2 \equiv \prod_{p \in P} p^{e(p,v)+e(p,w)} \bmod n$$

is just as useful as a full relation, unless $\gcd(q_2(v), n) \neq 1$. This implies that the condition that the number of full relations is larger than $\#P$ can be replaced by the condition that the number of full relations plus the number of independent cycles is larger than $\#P$. The large primes that occur in the cycles can be thought of as cheap factor base extenders—cheap because they are found almost for free, without having to sieve or to trial divide with them. The cycles are simply linear combinations of exponent vectors where the coordinates corresponding to the factor base extenders, the large primes, are even.

We explain how partial relations can efficiently be collected during the search for full relations. During the sieving, candidate v's are identified in an efficient manner. For each candidate v the least absolute residue $v^2 \bmod n$ is trial divided, to see if it factors using the elements of P. If so, a full relation has been found. If not, and the remaining cofactor t after trial division with the primes $< B$ is $< B^2$, a partial relation with $q_1(v) = 1$, $q_2(v) = t$ has been found. How many of such partial relations will be found depends on how easily candidates are accepted after the sieving—if many near misses for fulls are accepted, many partials will be found. If we accept even more candidates, we might also find near misses for the partials with $q_1(v) = 1$: if $t > B^2$, $t < B_2^2$, and t is composite we find a partial relation if both of the prime factors of t are $< B_2$ (note that t can have at most two prime factors because $B_2^2 < B^3$). Since compositeness tests are cheap, because composites of this form are fairly easy to factor, and

because relatively many t's will have their factors in the right range, it follows that these partials can also be found at relatively small cost.[3]

Of course, it only makes sense to spend this extra effort if the partial relations are useful in practice, i.e., if cycles among the partials indeed turn up. In [9] it is shown that if only partials with $q_1(v) = 1$ are allowed, the total number of independent combinations (of the type as shown in the example above) can be expected to behave as $c \cdot m^2$, where m is the number of partial relations (with $q_1(v) = 1$), and c is some very small constant depending on the bounds (cf. [9], and the 'birthday paradox'). This quadratic behavior can indeed be observed in practice. Using these restricted partials leads to a speed-up of about a factor 2.5 compared to using only full relations.

Using *all* partials, i.e., also those with $q_1(v) \neq 1$ leads to another speed-up of about a factor 2.5, for sufficiently large numbers. A theoretical analysis of the expected number of cycles has not been given yet, but according to the data from [1] the number of cycles seems to grow almost as $c' \cdot m^4$, where c' is another small constant, and m is now the total number of partials. In any case, the number of independent cycles as a function of the number of partials seems to behave as a very smooth curve, at least over the intervals where it has been observed so far. Thus, reliable estimates of the expected completion time of the relation collection stage can easily be derived from this curve.

Large primes in NFS. To factor n with NFS, one begins by selecting two bivariate polynomials $f_1(X, Y)$, $f_2(X, Y) \in \mathbf{Z}[X, Y]$ that satisfy certain properties that are not relevant for this paper (cf. [8]). Given f_1 and f_2 one selects two factor bases P_1 and P_2, consisting of the primes $\leq B_1$ and $\leq B_2$, respectively. Relations are given by coprime integers a, b, with $b > 1$, such that

$$|f_1(a,b)| = \prod_{p \in P_1} p^{e_1(a,b,p)} \quad \text{and} \quad |f_2(a,b)| = \prod_{p \in P_2} p^{e_2(a,b,p)},$$

with $e_1(a, b, p), e_2(a, b, p) \in \mathbf{Z}$. If more than (approximately) $\#P_1 + \#P_2$ relations have been found, a factorization of n can, with high probability, be derived from linear dependencies modulo two among the $(\#P_1 + \#P_2)$-dimensional vectors consisting of the concatenation of $(e_1(a, b, p))_{p \in P_1}$ and $(e_2(a, b, p))_{p \in P_2}$. How this is done is beyond the scope of this paper (but see [8]; 10]), and neither will we discuss the influence of the (small) amount of 'free' relations.

As in QS, we can allow large primes in the factorizations of the $|f_i(a, b)|$, where those that occur in cycles among the partial relations can be regarded as cheap factor base extenders (where it should be noted that a large prime dividing $f_1(a, b)$ cannot be combined with the same prime dividing $f_2(a, b)$, and

[3] If we relax the conditions on the candidates even more, we might be able to allow three large primes in the factorization of $v^2 \bmod n$. So far, however, this does not seem to lead to a speed-up because it leads to a huge amount of numbers to be trial divided, the vast majority of which will lead to cofactors that do not have the right factoring pattern. Also the factorization of the composite cofactors is substantially more expensive.

vice versa—even more restrictive, a large prime q dividing $f_i(a,b)$ can only be combined with the same large prime q dividing $f_i(a',b')$ if $ab' \equiv a'b \bmod q$). In the NFS implementations described in [8] at most one large prime per $|f_i(a,b)|$, for a total of at most two per relation, was used. Candidate relations are identified using a sieve, similar to QS. For each candidate, $|f_1(a,b)|$ is trial divided with the primes $\leq B_1$, and upon success $|f_2(a,b)|$ is trial divided with the primes $\leq B_2$. This implies that, in principle and as explained above, two large primes per $|f_i(a,b)|$ can quite easily be recognized. In [8] it was reported, however, that actually finding these relations with up to $2+2$ large primes was prohibitively expensive. Fortunately, it was shown in [6] that they can efficiently be found with better sieving and trial division methods. These methods do not seem to apply to QS to efficiently generate three or more large primes per relation in QS. The important difference is that in QS we have one composite cofactor that has to be factored into three or more factors in the right range, whereas in NFS we are dealing with two composite cofactors that each have to factor in the right way — the latter both occurs with higher probability and is easier to decide.

The experiments from [8] indicated that the number of cycles in the $1+1$ large prime variation of NFS is consistently lower than the number of cycles found in the (two) large prime variation of QS, i.e., for NFS substantially more relations are needed than for QS to get the same number of cycles. This is due to the fact that in QS we have a single set of large primes, whereas in NFS we have two 'incompatible' sets of large primes, one for f_1 and one for f_2. Thus, for NFS it takes longer for the 'birthday paradox' to take effect. On the other hand, $1+1$ large primes in NFS behaves markedly better than the single large prime variation of QS, i.e., if only partials with $q_1(v) = 1$ are considered in QS.

Based on these observations, the work from [3], and the 'almost quartic' behavior observed in [1], we hoped that $2+2$ large primes in NFS would produce somewhat better than quartic cycle growth. We also expected that the number of independent cycles as a function of the number of partial relations would, as usual, behave as a nice and smooth curve that would allow easy extrapolation to predict the completion time of the relation collection stage. These expectations turned out to be wrong, as we will see in the next section.

In the sequel, partials with i large primes in $|f_1(a,b)|$ and j large primes in $|f_2(a,b)|$ will be called 'i,j-partials'. Partials for which $i+j=k$ will be called 'k-partials', if $i+j \leq k$ they will be called '$\leq k$-partials', and similarly for '$\geq$'.

3 Experiments and results

In this section we describe the details of three NFS factoring experiments in chronological order: a 116, a 119, and a 107-digit number.

Factoring a 116-digit integer. Let n be the following 116-digit composite factor of the 11887th partition number:

$$n = 1\ 50802\ 87457\ 98463\ 07441\ 49612\ 94413\ 35408\ 90110\ 76626\ 79218\ 10826\ 04486$$
$$78500\ 16206\ 10665\ 65455\ 29820\ 06606\ 21307\ 78648\ 81680\ 71410\ 39443.$$

To factor n using NFS we first spent a few workstation days to find 5 reasonable candidate polynomials f_1 and f_2. Next we sieved a while with each candidate pair, using reasonably sized factor bases. This yielded one pair that stood out from the others with a more than 10% better yield than the next best one: $f_1(X,Y) = X - 49999\ 99918\ 54766\ 46567\,Y$ and

$$\begin{aligned} f_2(X,Y) = {} & 48\ 25692\ 37961\ 89830\,X^5 + 35\ 68080\ 39372\ 65531\,X^4Y \\ & - 4\ 65605\ 61818\ 75120\,X^3Y^2 - 59\ 69883\ 14526\ 21728\,X^2Y^3 \\ & - 13\ 44285\ 55250\ 45260\,XY^4 + 29\ 65432\ 72740\ 38354\,Y^5, \end{aligned}$$

with common root $X/Y = 49999\ 99918\ 54766\ 46567$ modulo n. We could not observe any correlation with properties that were thought to be relevant, like coefficient sizes or number of roots modulo small primes. The issue of polynomial selection in NFS needs to be understood better; we have not pursued this yet as our current trial-and-error approach seems to work satisfactorily, for the moment.

Having thus decided on f_1 and f_2, we selected $\#P_1 = 100,001$, $\#P_2 = 400,001$, and $B_2 = 2^{30}$. From our sieving experiments we derived that this choice could lead to approximately 50 million partials in about 250 mips years, using [6]. Given our experience with QS and NFS with fewer large primes, and the expected counts of the various types of relations (i.e., with a total of 0, 1, 2, 3, or 4 large primes), we expected that this choice would be enough to produce more than 500,000 fulls plus cycles, even without relying on any $\geq$ 3-partials. Furthermore, since 250 mips years is less than QS would need for this number (about 400 mips years), our choice of $\#P_1$, $\#P_2$ and B_2 seemed not too bad.

Initial results were not surprising. The relations were found at the expected rates and the cycle-yield followed our worst-case scenario. The $\leq$ 2-partials combined more or less at their expected rate, and the other partials hardly contributed to the cycles: at 7,336,602 partials (and 12,607 fulls) there were only five 3-partials involved in the 1,474 cycles, and no 2,2-partials.

The results were mixed after we had completed more than half our anticipated sieving, at 28,243,830 partials (and 43,555 fulls). Even though the curve of the cycle-yield as a function of the number of partials resembled a quartic function, extrapolation suggested that we would not even be close to what we needed by the time we would be finished sieving. On the other hand, of the 2,2-partials there were 10,495,464 of which 5,645 (0.054%) that occurred in cycles (of which there were 41,366), which was a marked improvement.

It all looked different by the next time we attempted to count the cycles; attempted, because the counting program failed to work properly, a first indication that something had changed. After replacing the failing counting approach by a new one (cf. Section 4), we found 317,862 cycles among 33,264,762 partials (and 49,680 fulls), with the 2,2-partials at 12,460,866 with 672,773 (5.4%) participating in the cycles. Although this was still not enough, we did not fail to notice that our nice smooth curve had effectively been cut short by an almost vertical line, implying that the sieving was almost complete.

A side effect associated with this explosive cycle behavior was that the average length of the cycles (i.e., the number of partials that together form a cycle)

Table 1

total fulls	total partials	useful partials	%	cycles
12607	7336602	3368	0.04	1474
19456	11521334	8913	0.07	3842
25782	15773678	18627	0.11	7491
29744	18368031	28982	0.15	10934
33215	20736047	42299	0.20	14895
37724	23902815	73039	0.30	22329
40466	25972266	111358	0.42	29415
43555	28243830	224217	0.79	41365
49680	33264762	3486661	10.48	317862
58325	42202890	9369843	22.20	1746609
61849	45876382	13970578	30.45	2605463

seemed to grow rapidly (we failed to keep the number, but see Table 6). Because longer cycles could lead to problems in the matrix step, and because we were curious to see if and how the explosion would continue, we decided to keep sieving for a while—hoping that if we had *many* cycles, we would also be able to find enough short ones, and restrict ourselves to those short ones in the matrix.

This led to two additional counts. At 42,202,890 partials, there were 1,746,609 cycles of average length 93. Of the 16,079,778 2,2-partials 15.4% were useful, i.e., occurred in cycles. At 45,876,382 partials, there were 2,605,463 cycles (and 61,849 fulls). The average length of these cycles had dropped to 74, and the proportion of useful 2,2-partials rose to 23.4% with 4,111,077 useful 2,2-partials out of a total of 17,572,446. Further, we note that 87.3% of the cycles used a 2,2-partial. More details can be found in Tables 1, 2 and 3.

So the explosion indeed continued. Further, the cycle length went down sufficiently that there were 459,922 cycles of length ≤ 23, which was acceptable for our matrix processing method back then. Note that $459,922+61,849 > 500,002$. At this point we stopped sieving, after about 220 mips years. We could have sieved more, and attempted to find the 50 million partials that we hoped to find in 250 mips years, to see if indeed the ≤ 2-partials would have sufficed, as expected. This is unlikely: where we stopped those partials led to only 93,328 cycles (with only 56,328 cycles among the i,j-partials with $i,j \leq 1$ as used in [8]. The ≤ 3-partials would have had a better chance, with 330,485 cycles where we stopped.

We have not yet been able to understand why this very sudden growth in the number of cycles occurs. Counting arguments failed to prove anything, and probabilistic arguments are complicated by the inhomogeneous nature of the data. Connections with well-known 'crystalization' behavior of random graphs have been suggested, but have so far not given any insights that could be used to prove or predict cycle explosions. Obviously this would be useful for a better selection of factor base sizes and minimization of the total sieving time: if no explosion had occurred we would have needed larger factor bases, but if we had

Table 2

cycles	total 1-partials	useful 1-partials	%	total 2-partials	useful 2-partials	%
10934	515439	21906	4.24	3157511	6224	0.19
14895	579057	29991	5.27	3556296	10529	0.29
22329	660552	45965	6.95	4076913	21808	0.53
29415	709775	62463	8.80	4395522	37346	0.84
41365	768018	97711	12.72	4772352	85653	1.79
317862	884118	371434	42.01	5551149	1063330	19.15
1746609	1065782	600653	56.35	6856874	2319722	33.83
2605463	1135723	720965	63.48	7385166	3143071	42.55
cycles	total 3-partials	useful 3-partials	%	total 2,2-partials	useful 2,2-partials	%
10934	7946983	808	0.01	6748098	44	0.00
14895	8970893	1666	0.01	7629801	113	0.00
22329	10338014	4823	0.04	8827336	443	0.00
29415	11194689	10389	0.09	9591557	1160	0.01
41365	12207997	35208	0.28	10495464	5645	0.05
317862	14368629	1379124	9.59	12460866	672773	5.39
1746609	18200456	3968763	21.80	16079778	2480705	15.42
2605463	19783047	5995465	30.30	17572446	4111077	23.39

Table 3

	fulls	%	0,1-partials	%	0,2-partials	%
total	61849		668995		1670521	
useful			417840	62.45	753575	45.11
wrt total useful				2.99		5.39
	1,0-partials		1,1-partials		1,2-partials	
total	466728		5038434		12521329	
useful	303125	64.94	2043376	40.55	3670595	29.31
wrt total useful		2.16		14.62		26.27
	2,0-partials		2,1-partials		2,2-partials	
total	676211		7261718		17572446	
useful	346120	51.18	2324870	32.01	4111077	23.39
wrt total useful		2.47		16.64		29.42

known about it we could have settled for smaller ones, so that the explosion would occur exactly when the sieving is done. This would lead to either denser or much larger matrices than we were used to, and thus would require better matrix techniques than used at the time we sieved this 116-digit number.

For the present number the matrix did not pose a big problem, because of all the extra sieving we had done. After inclusion of 256 columns for quadratic signatures, we had a $521,771 \times 500,258$ bit-matrix, with on average 277 one-bits per row. This matrix was reduced to an almost 9 gigabyte dense $274,696 \times 274,496$ bit-matrix using 'structured Gaussian elimination' [7; 12; 14]. It took 6

CPU-days on a 16K MasPar MP-1 to find the dependencies in the dense matrix using plain Gaussian elimination.

The first dependency was processed by Peter Montgomery at the CWI in Amsterdam, using the method from [10], in less than 2.5 CPU-days on a MIPS R4400 processor. This resulted in the following factorization:

$$n = 3\ 73787\ 18590\ 84719\ 25152\ 67256\ 20648\ 89920\ 58833\ 29019 \times 40344\ 58115$$
$$87486\ 91262\ 33674\ 44522\ 38913\ 92270\ 04005\ 21248\ 10720\ 24860\ 30673\ 43497.$$

More data points can be obtained by pruning the data that we have for smaller values of $B_2 = 2^{30}$, and checking if and where the resulting sets of relations lead to cycle-explosions. We have not done this for the present number, but we did for the 119-digit number discussed below. From these and other experiments of this sort that we carried out in [5] (for a QS factorization) it follows that using a higher value of B_2 indeed saves sieving time, at the cost of substantial amounts of disk space.

Factoring a 119-digit integer. To convince ourselves that the cycle-explosion described above was not an isolated incident, we tried factoring the following 119-digit composite factor of the 13171th partition number:

$$n = 1472\ 39730\ 37795\ 02230\ 11857\ 21506\ 65046\ 38946\ 42104\ 16013\ 39117\ 46791\ 27360$$
$$74474\ 37214\ 92509\ 46318\ 17633\ 03651\ 67483\ 02069\ 42164\ 60898\ 11241.$$

We again found one pair of polynomials that stood out from the rest, for no reasons that we could find: $f_1(X,Y) = X - 1\ 44999\ 99959\ 86876\ 68083\,Y$ and

$$\begin{aligned} f_2(X,Y) = {} & 229\ 71270\ 09947\ 70930\,X^5 - 75\ 24490\ 95044\ 76954\,X^4Y \\ & - 349\ 19223\ 42428\ 31010\,X^3Y^2 + 213\ 34303\ 57653\ 48142\,X^2Y^3 \\ & - 133\ 73262\ 31271\ 45009\,XY^4 - 83\ 88784\ 35301\ 30136\,Y^5, \end{aligned}$$

with common root $X/Y = 1\ 44999\ 99959\ 86876\ 68083$ modulo n. Anticipating the explosion this time, we chose relatively small factor bases ($\#P_1 = 100,001$ and $\#P_2 = 320,001$). This turned out to be too small: after 195 mips years sieving was complete with only about 24,000 cycles among 30 million partials (and 30,000 fulls). Extension of $\#P_2$ to 360,000 and about 45 additional mips years of sieving led to 470,000 cycles among 35.8 million partials (and 39,000 fulls), occupying almost 2 gigabytes of storage. Details can be found in Tables 4 and 5.

As expected this led to an unusually large matrix problem. Structured Gaussian elimination would have required in excess of 15 gigabytes of storage and more than two CPU-weeks on a 16K MasPar MP-1, which is hardly feasible. Instead we used an experimental MasPar implementation of the blocked Lanczos method from [11] (cf. Section 5, and [4]). This took 2.5-CPU-days. The dependencies were again processed by Peter Montgomery, this time in one CPU-day per dependency—the factorization was found on the third one:

Table 4

(with $\#P_2 = 320,001$)

total fulls	total partials	useful partials	%	cycles	total $\geq$ 3-partials	useful $\geq$ 3-partials	%
5607	3683540	588	0.01	280	2898623	2	0.00
10426	7262853	2525	0.03	1157	5771533	6	0.00
13157	9387071	4451	0.04	1997	7492921	24	0.00
16363	12152617	7840	0.06	3371	9755753	69	0.00
18297	14017542	10881	0.07	4549	11301609	138	0.00
21100	16950688	16743	0.09	6621	13765642	365	0.00
25233	21930116	31702	0.14	11118	18018822	1339	0.00
28194	26465817	56158	0.21	16716	21950471	4146	0.01
30289	30107586	110348	0.36	23886	25129694	15258	0.06
		(after recounting with $\#P_2 = 360,001$)					
35953	30101922	219051	0.72	34728	24735646	44746	0.18
38199	34567876	3980288	11.51	337351	28636652	2548939	8.90
38741	35763524	4725203	13.21	472426	29680006	3116297	10.49

Table 5

	fulls	%	0,1-partials	%	0,2-partials	%
total	38741		459082		1254636	
useful			207979	45.30	314563	25.07
wrt total useful				4.40		6.66
	1,0-partials		1,1-partials		1,2-partials	
total	303127		3603958		9746884	
useful	148271	48.91	794788	22.05	1197510	12.29
wrt total useful		3.14		16.82		25.34
	2,0-partials		2,1-partials		2,2-partials	
total	462715		5476700		14456422	
useful	143305	30.97	770361	14.07	1148426	7.94
wrt total useful		3.04		16.30		24.30

n = 54 67135 70838 33359 03092 89109 24162 82702 67063 91975 73477 × 26 93178 62646 37991 75226 13452 71731 62372 36050 27370 88674 71432 35361 24533.

With less than 250 mips years spent on sieving, this factorization was completed about 2.5 times faster than it would with QS. Had we known how well our blocked Lanczos implementation would work, then we would have used larger factor bases, but sieved shorter per 'special-q' (cf. [6]), which would have reduced the sieving time considerably. Combining this observation with many other possible improvements, the 250 mips year figure should not be taken too seriously.

As indicated above, we obtained more data points by pruning the final data sets for smaller values of B_2. Some results are given in Table 6, including results for the data sets as they were at 88%, 90%, and 95% of the sieving. Note the sharp increase of the cycle length (of the cycles as found by our method, cf. Section 4) 'early' in the explosion, and the subsequent decrease of the cycle length as the explosion continues. Apparently, a first indication of an upcoming

Table 6

B_2	total partials	% useful	cycles	longest cycle	average length	total ≥ 3-partials	% useful
using the data sets after 88% of the sieving:							
2^{26}	3716735	2.12	22920	66	4	2435866	0.28
2^{27}	7141411	1.49	26652	106	5	5172320	0.27
2^{28}	12566475	1.14	29867	564	6	9708135	0.22
2^{29}	20536267	5.37	32464	1072	8	16560014	3.17
2^{30}	31471940	5.92	61402	$\geq 10^6$	≥ 388360	26118418	3.95
using the data sets after 90% of the sieving:							
2^{26}	3801311	2.28	23477	65	4	2491247	0.34
2^{27}	7303838	1.69	27794	134	5	5289915	0.33
2^{28}	12852293	1.42	31665	997	8	9928854	0.35
2^{29}	21003354	8.26	59148	$\geq 10^6$	≥ 214885	16936513	5.56
2^{30}	32187175	7.35	98433	$\geq 10^6$	≥ 57252	26712006	5.18
using the data sets after 95% of the sieving:							
2^{26}	4012410	3.93	33026	230	6	2629594	0.95
2^{27}	7709526	12.30	55897	127536	742	5583741	7.52
2^{28}	13566166	15.49	137720	50136	948	10480347	11.30
2^{29}	22169973	13.75	237435	34279	835	17877242	10.44
2^{30}	33975344	11.02	319882	32783	788	28195708	8.46
using the final data sets:							
2^{26}	4223679	4.86	37133	619	8	2768053	1.41
2^{27}	8115376	17.25	86015	≥ 29761	≥ 664	5877684	11.87
2^{28}	14280326	19.21	216173	21728	663	11032061	14.77
2^{29}	23337060	16.62	357792	17235	612	18818348	13.13
2^{30}	35763524	13.21	472426	16524	590	29680006	10.49

cycle-explosion is the sudden growth of the number of useful relations and of the average cycle length. Note also that for larger B_2 there are relatively more partials with more large primes.

Factoring a 107-digit integer. As a third experiment we factored a 107-digit number, with assistance from Magnus Alvestad and Paul Leyland (using our program described in [6]), Peter Montgomery (using his implementation at the CWI in Amsterdam), and Jörg Zayer (using his implementation from [3]). Zayer's and our program both use 'special q's' (cf. [6]), so that non-overlapping sieving tasks could easily be distributed among Zayer and the users of our program. To avoid overlap with Zayer's and our results, Montgomery used his more traditional siever with 'large prime special q's', i.e., special q's that would be considered as large primes by the other programs. As a consequence, Montgomery's program also produced i,j-partials with $i,j > 2$ (but with $i \leq 4$ and $j \leq 6$).

Although smaller numbers lead to smaller factor bases, and therefore to smaller large primes with a higher cycle-yield, the cycle-yield for the 107-digit

number was initially even smaller than the cycle-yield for the 116 and 119-digit numbers reported above: at 12,921 fulls and 24,660,318 partials, there were only 7,552 cycles among 26,070 useful partials, compared to 22,239 cycles among 73,039 usefuls in 23,902,815 partials (for the 116-digit number) and 11,118 cycles among 31,702 usefuls in 21,930,116 partials (for the 119-digit number). This lower than expected cycle-yield cannot be explained by the $\geq$ 5-partials, because at this point we had hardly received any partials from Montgomery yet.

At 29,061,638 partials (and 13,918 fulls) on the other hand, the cycle-yield was better than for the 119-digit number: 202,387 cycles among 2,856,606 usefuls, compared to 34,728 cycles among 219,051 usefuls in 30,101,922 partials for the 119-digit number (good comparison data with the 116-digit number were not available). When we stopped sieving, we had 34,135,923 partials, 5,531,053 of which were useful (we did not count the cycles).

We conclude that also for this smaller number we got a cycle-explosion, even a bit more dramatic than before. The behavior of the cycle-yield is however sufficiently erratic that we have not been able to derive a reasonable 'rule-of-thumb' that could be useful to predict the explosion for future NFS factorizations.

4 Counting

In [9] some elementary methods were given to count and build the useful combinations among relations with at most two large primes per relation. A generalization of these methods to the case of at most three large primes per relation was presented in [2]. This generalization does not extend to our case where relations can have four (or more) large primes. We give an outline of our methods.

From the data presented in the previous section it should be clear that we are dealing with large amounts of data: it takes at least 24 bytes to store one a, b pair with four large primes, which already implies several hundred megabytes for the 2,2-partials (which actually take more space than that). The first concern while counting cycles therefore is to quickly weed out partials that are useless, i.e., that do not occur in any cycle (of course they will be kept, because they might be useful in later counts). Note that a partial is useless if it contains a large prime that does not occur in any other partial.

We hashed each large prime, without collision resolution, to a 2-bit location where the hits were counted (not counting further than 2). We kept only those relations for which all large primes had hashed to a location with count 2. This process was repeated on the resulting collection, until the resulting relations could be handled by another version of the same program that did use collision resolution. The latter version was repeated until no relations were deleted. The number of cycles among the partials in the resulting collection can then easily be estimated by subtracting the total number of distinct primes from the number of partials. The latter can also be done on 'earlier' collections of partials as long as there is enough disk space for the sorting and uniqueing.

To do an exact count of the cycles, or to build them, we always first computed the collection of useful partials, as sketched above, and next applied the following

'greedy elimination'. First we process the single large prime partials, storing each large prime when it is encountered for the first time, and counting (and building) a cycle each time it is encountered after that. Next, we remove all those stored primes from all other partials (counting, and building, cycles for partials where all large primes get removed), separating the resulting partials that still have at least one large prime into those with one, and those with at least two large primes. The entire process is repeated until no new partials with one remaining large prime are kept. Usually, at this point, no other partials remain; if there are, the cycles among them can easily be found using a similar strategy. Note that for the 'building' this greedy elimination requires some additional administration to account for the 'history' of the large prime deletions; so far this has not taken serious amounts of disk space.

5 The matrix step

In all previous QS and NFS factorizations, including the 116-digit one from Section 3, we used the following strategy for the matrix step: first build all cycles to get a, roughly, $\#P \times \#P$ (or $(\#P_1 + \#P_2) \times (\#P_1 + \#P_2)$) bit-matrix, next apply structured Gaussian elimination [7; 12; 14], and finally apply ordinary Gaussian elimination. The extent to which the first step should be carried out before applying the last two steps is debatable. That is, we may remove all of the large primes before starting elimination on the matrix; some of the large primes may be removed in the first step, with the others included in the matrix; or none of the large primes need to be removed—without a clear advantage among these possible strategies. For the 119-digit number from Section 3, only the large primes that occurred at most three times in the useful partials were removed by constructing cycles. This resulted in a sparse $1,475,898 \times 1,472,607$ bit-matrix, which could have been 'reduced' to a dense $362,597 \times 362,397$ bit-matrix by structured Gauss. In our experience about the same would have happened if we had removed all large primes and processed the (unusually dense) 'sparse' $461,001 \times 460,001$ matrix with structured Gauss.

We did, however, not actually build this dense $362,597 \times 362,397$ bit-matrix. Instead we used the blocked Lanczos method from [11] to process the sparse $1,475,898 \times 1,472,607$ bit-matrix for the 119-digit number. The reason that we removed only the large primes that occur ≤ 3 times in the useful partials before using blocked Lanczos, is the following. The expected run time of blocked Lanczos applied to a matrix of (approximately) m rows and m columns with on average w non-zero entries per row, is proportional to m times the total weight (i.e., mw) of the matrix. Initially, we may assume that all rows have about equal weight. Therefore, removal of a large prime that occurs in k different rows results in $m-1$ rows, $m-k$ of the same average weight w as before, and $k-1$ of average weight at most $2w-2$, and thus expected run time for blocked Lanczos proportional to $m-1$ times $(m-k)w+(k-1)(2w-2)$. In realistic circumstances, the latter is less than m^2w as long as $k \leq 3$.

Evidently, removal of large primes destroys the even distribution of the nonzeros over the rows, so the same argument cannot be used to analyse the effect of removing more large primes. Nevertheless, similar arguments imply that the run time can reasonably be expected to decrease if large primes that occur in at most 3 rows are removed, but that an increase can be expected if large primes that occur more often are removed. This explains the choice that we made for the 119-digit number. For a detailed description of the blocked Lanczos algorithm we refer to [11], and to [4] for a description of the implementation that we used.

Acknowledgments. Acknowledgments are due to M. Alvestad, S. Contini, P. Leyland, P. Montgomery, and J. Zayer for their assistance.

References

1. D. Atkins, M. Graff, A.K. Lenstra, and P.C. Leyland, *THE MAGIC WORDS ARE SQUEAMISH OSSIFRAGE*, Asiacrypt'94, to appear.
2. J. Buchmann, J. Loho, and J. Zayer, *Triple-large-prime variation*, manuscript, 1993.
3. J. Buchmann, J. Loho, and J. Zayer, An implementation of the general number field sieve, Advances in Cryptology, Crypto'93, Lecture Notes in Comput. Sci. **773** (1994), 159–165.
4. S. Contini and A. K. Lenstra, *Implementations of blocked Lanczos and Wiedemann algorithms*, in preparation.
5. T. Denny, B. Dodson, A. K. Lenstra, and M. S. Manasse, *On the factorization of RSA-120*, Advances in Cryptology, Crypto'93, Lecture Notes in Comput. Sci. **773** (1994), 166–174.
6. R. Golliver, A. K. Lenstra, and K. McCurley, *Lattice sieving and trial division*, ANTS'94, Lecture Notes in Comput. Sci. **877** (1994), 18–27.
7. B. A. LaMacchia and A. M. Odlyzko, *Solving Large Sparse Linear Systems over Finite Fields*, Advances in Cryptology, Crypto'90, Lecture Notes in Comput. Sci. **537** (1991), 109–133.
8. A. K. Lenstra and H. W. Lenstra, Jr. (eds), *The development of the number field sieve*, Lecture Notes in Math. **1554**, Springer-Verlag, Berlin, 1993.
9. A. K. Lenstra and M. S. Manasse, *Factoring with two large primes*, Math. Comp **63** (1994), 785–798.
10. P. L. Montgomery, *Square roots of products of algebraic numbers*, Proceedings of Symposia in Applied Mathematics, Mathematics of Computation 1943-1993, Vancouver, 1993, Walter Gautschi, ed.
11. P. L. Montgomery, *A block Lanczos algorithm for finding dependencies over GF(2)*, Advances in Cryptology, Eurocrypt'95, Lecture Notes in Comput. Sci. **921** (1995), 106–120.
12. A. M. Odlyzko, *Discrete Logarithms in Finite Fields and their Cryptographic Significance*, Advances in Cryptology, Eurocrypt'84, Lecture Notes in Comput. Sci. **209**, 224–314.
13. C. Pomerance, *The quadratic sieve factoring algorithm*, Advances in Cryptology, Eurocrypt'84, Springer, Lecture Notes in Comput. Sci. **209**, 169–182.
14. C. Pomerance and J. W. Smith, *Reduction of huge, sparse matrices over finite fields via created catastrophes*, Experiment. Math. **1** (1992) 89–94.
15. B. Silverman, *The multiple polynomial quadratic sieve*, Math. Comp. **48** (1987), 329–339.