MPQS with Three Large Primes

Paul Leyland[1], Arjen Lenstra[2], Bruce Dodson[3],
Alec Muffett[4], and Sam Wagstaff[5]

pleyland@microsoft.com
[2] Citibank, 1 North Gate Road, Mendham, NJ 07945-3104, USA,
arjen.lenstra@citicorp.com
[3] Lehigh University, Bethlehem, PA 18015-3174, USA,
bad0@Lehigh.edu
[4] Sun Microsystems Professional Services,
Riverside Way, Watchmoor Park, Camberley, GU15 3YL, UK,
alec.muffett@uk.sun.com
[5] CERIAS and Department of Computer Sciences, Purdue University,
West Lafayette, IN 47907-1315, USA,
ssw@cs.purdue.edu

Abstract. We report the factorization of a 135-digit integer by the triple-large-prime variation of the multiple polynomial quadratic sieve. Previous workers [6][10] had suggested that using more than two large primes would be counterproductive, because of the greatly increased number of false reports from the sievers. We provide evidence that, for this number and our implementation, using three large primes is approximately 1.7 times as fast as using only two. The gain in efficiency comes from a sudden growth in the number of cycles arising from relations which contain three large primes. This effect, which more than compensates for the false reports, was not anticipated by the authors of [6][10] but has become quite familiar from factorizations obtained using the number field sieve. We characterize the various types of cycles present, and give a semi-quantitative description of their rather mysterious behaviour.

1 Introduction

The use of large primes in the Multiple Polynomial Quadratic Sieve (MPQS) has been suggested many times. Earlier studies [10][13] suggested that using one large prime is always better than using none, and that the double large prime variation (often called PPMPQS) is more efficient than using fewer large primes when factoring integers with more than about 80 decimal digits. For N near 10^{100} these studies showed that the double-prime version of MPQS is 2 to 2.5 times faster than the single-prime variant. A spectacular example of a PPMPQS factorization was the record-breaking factorization of the 129-digit Scientific American RSA challenge RSA-129 in 1993-4, reported in [1].

In this paper we introduce TMPQS, i.e., MPQS with three large primes. In [6][10] it was suggested that it would be less efficient than PPMPQS because of

C. Fieker and D.R. Kohel (Eds.): ANTS 2002, LNCS 2369, pp. 446–460, 2002.

the very large number of false reports from the sievers, that is quadratic residues which are flagged as possibly being smooth but which subsequently prove not to be. The analysis underlying the suggestion from [10], however, did not take into account an effect that has since then become familiar from number field sieve (NFS) factorizations. In NFS, relations with more than two large primes are found at very little extra cost. Experience with such NFS-relations has shown that, after a slow start, the number of cycles suddenly grows very rapidly [7]. We expected that similar behaviour may also occur with TMPQS, and that it may compensate for the extra false reports. A few preliminary and small-scale computations by the first author suggested that TMPQS was certainly no worse than PPMPQS for N in the 100-digit range and may be slightly better.

To put this hypothesis to the test, we factored the 135-digit cofactor of $2^{803} - 2^{402} + 1$, also known as 2,1606L.c135 in the Cunningham newsletters. It was chosen because it is: somewhat larger than the previous record-breaking MPQS factorization but close enough in size to RSA-129 that it should give an indication of a possible speed-up; comparable with recent large factorizations using the General NFS; not easily factored with the Special NFS. It should be noted, however, that GNFS would have been much faster than the computation we decided to embark upon. We factored 2,1606L.c135 solely to satisfy our curiosity with respect to the relative speed of TMPQS and PPMPQS and to get more insight in the cycle behaviour. Setting the current record for a factorization with the Quadratic Sieve algorithm was entirely incidental...

In Section 2 we describe our implementation of TMPQS. Section 3 provides a detailed description of the growth in the number of cycles and gives a semi-quantitative analysis of the results. Evidence that TMPQS may be expected to outperform PPMPQS for sufficiently large numbers is presented in Section 4.

2 The Multiple Polynomial Quadratic Sieve

We assume that the reader is familiar with MPQS and PPMPQS (see [6] [10]). Compared to PPMPQS, TMPQS requires a different way to process the reports from the siever and different cycle counting and cyle finding software. The implementation of the siever that the first four authors used was almost identical to the 'factoring by email' version described in [9]. Even though other implementations of PPMPQS are readily available, some of them perhaps more efficient than ours, using this old implementation had the advantage that we were able to compare our performance figures directly to those obtained in previous large-scale factorizations, such as that of the 129-digit RSA challenge number [1] which was also based on the software from [9]. The fifth author used an adapted version of his Self-Initializing Quadratic Sieve siever [2], making sure the reported relations were compatible with the ones found by the others. Because of the nature of this version of MPQS, these relations contained markedly more small primes but the large-prime statistics within the relations having large primes, and the relative proportions of the three types of relations, was closely similar.

The remainder of this section contains a brief description of the five stages of TMPQS: parameter selection, sieving, counting and finding cycles, linear algebra, and combination of relations to find the factorization.

2.1 Parameter Selection

In MPQS a **multiplier** $k \in \mathbf{Z}_{>0}$ is chosen such that kN is a quadratic residue modulo relatively many small primes, where N is the number to be factored. For 2,1606L.c135 the value $k = 1$ is optimal.

The **factor base size** is the number of primes with which one sieves. It was chosen as 550 000, so $B_1 = 17\,157\,953$ was the largest prime in the factor base.

The **large prime bound** was chosen as $B_2 = 2^{30}$. Thus, relations consist of integers v such that $v^2 \bmod (2{,}160\mathrm{L.c}135)$ is the product of primes $\leq B_1$ and at most three primes $< B_2$.

The size of the factor base was chosen in a rather ad hoc manner. Sieving experiments were performed for various choices between 400 000 and 750 000. For each experiment a number of sieve-report bounds were tried, attempting to maximize the yield. The value eventually chosen may not be optimal, but it appears not to be too bad. Compared to PPMPQS our choice may be considered to be on the small side, but it is a logical consequence of allowing three as opposed to just two large primes.

The **sieving range**, the number of values sieved per polynomial, was determined experimentally. The siever used by four of us was tested with values lying between 17 million and 100 million, again attempting to maximize the yield. The yield was only slightly dependent on the sieving range but was somewhat higher for the lower values we tried. The bulk of the computation used a value of 17 158 000. The siever employed by the fifth author used a sieving range of only 2 000 000 because that method initializes polynomials so efficiently.

2.2 Sieving

The only difference between the siever used by most of us and that used to factor RSA-129 is that we attempted to factor quadratic residues less than $B_2^3 = 2^{90}$ after primes $\leq B_1$ had been divided out. We used a fast pseudoprimality test to reject candidates greater than B_2 that were not recognized as composites, we used ECM to factor those which were, and rejected those for which ECM failed or found a prime factor larger than B_2. Per report, the pseudoprime test may have to be applied to two different numbers (one $< B_2^3$ and one $< B_2^2$) and ECM may have to be used to factor both numbers (if composite). Thus, in some cases, testing candidate quadratic residues may be more expensive in TMPQS than PPMPQS.

The output of the siever was a series of relations, each consisting of an integer v and the prime factorization of $v^2 \bmod (2{,}1606\mathrm{L.c}135)$. As in [1], we denote a relation as a *ful*, *par*, or *ppr* according to whether it contains zero, one, or two large primes. In addition, we have *tpr* relations which contain three large primes.

As in [1], the sieving process was distributed over many client machines and the relations produced sent to a central machine which checked the correctness of newly arrived data and discarded duplicates. Relations were then added to one of four files, according to whether they were *ful*, *par*, *ppr*, or *tpr* relations.

We began sieving on 10th January 2001 and finished 231 days later on 29th August. By the time we finished sieving, we had accumulated a total of 13 441 627 relations, made up of 62 626 *ful*s, 790 129 *par*s, 4 080 732 *ppr*s and 8 508 140 *tpr*s.

2.3 Counting and Finding Cycles

In principle, once enough relations have been output by the sieving phase, linear dependencies in the matrix of exponent vectors modulo 2 could be found by standard techniques of linear algebra, such as Gaussian elimination or block Lanczos. There are two problems, however. In the first place, one needs to be able to recognize that enough relations have been found. Secondly, the resulting matrix would be far too large to be easily dealt with by these methods.

In [10] 'cycles' were introduced as sets of relations where each large prime occurs an even number of times. It follows that each cycle gives rise to integers w and s such that $(w^2 \bmod (2,1606L.c135))/s$ is a product of primes $\leq B_1$, where w is the product of the v's and s is the product of all large primes in the cycle and thus a square. For our purposes, cycles are therefore equivalent to (less sparse) *ful* relations. Identifying a relation with a vector with bits set for its large primes, cycles are linear dependencies modulo 2 among those vectors.

It follows that in order to recognize if there are enough relations, it suffices to count the number C of independent cycles among the non-*ful* relations, and to check if C plus the number of *ful* relations is larger than 550 000, the size of the factor base. If so, there are enough relations. Because we were interested in the growth of the number of independent cycles, we computed a lower bound for C on a daily basis. This can be done by means of an easy two-step process:

1. From the set of *par*, *ppr*, and *tpr* relations, remove the 'singletons', i.e., relations that have a large prime that does not occur in any other relation. Removing a singleton *ppr* or *tpr* relation may generate further singletons from the other prime(s) present in the relation. So, singleton removal consists of a number of 'pruning passes' that must be carried out iteratively until, during the last pruning pass, no more singletons are removed. Singleton removal can easily be implemented in a variety of ways.
2. Once all singletons have been removed, all remaining *par*, *ppr*, and *tpr* relations belong to a cycle. A close lower bound estimate for C is given by the difference δ of the total number of remaining large primes (not counting multiplicities) and the number of remaining relations, i.e., the oversquareness of the matrix of vectors. (If no *tpr*s are used and all cycles contain a *par* relation, then $\delta = C$; with *tpr*s this is the case if fairly uncommon types of cycles do not occur.)

The last step can conveniently be done using the Union-Find algorithm as in [10]. Union-Find can also directly be applied to the full set of relations, but that leads to a less precise approximation of C. Another reason that we preferred to remove singletons first is that the change in the number of pruning passes provided interesting information about the progress of our computation. This is shown in Section 3, along with a detailed account of the behaviour of the growth of C as relations were added.

Our final collection of relations produced 494 077 independent cycles, only 67 543 of which (14%) did not contain at least one *tpr* relation; there were 62 626 *ful* relations. Linear algebra techniques could have been applied to the collection of *ful* and singleton-less non-*ful* relations but, as mentioned above, this leads to an unattractively large matrix. In NFS this problem is usually dealt with by a partial merging of the complete singleton-less set of relations, removing most of the primes by combining the relations containing them, but keeping some in order to make the linear algebra step as fast as possible [3]. In MPQS, on the other hand, it has been traditional (and certainly not optimal!) to remove all large primes by building a complete set of independent cycles among the non-*ful* relations. This can be achieved using a relatively straightforward adaptation of the method from [10]; we do not elaborate. Of the 494 077 independent cycles, the 487 424 least dense ones were actually used; the longest cycle of these contained 215 relations (whereas the longest present had well over a thousand). With the addition of the 62 626 *ful* relations a matrix with 550 000 rows and 550 050 columns was produced, requiring 616 megabytes of storage (in ASCII format). The average density of the matrix was 411 bits set per row. This is much denser than usual for an MPQS factorization. More sieving would have reduced the density of the matrix, but there was no need to do so as the matrix was tractable with our resources.

2.4 Linear Algebra

Finding dependencies was done with the block Lanczos method of [11] and [12]. We performed this computation twice, once on a conventional uniprocessor machine and once on a cluster of workstations, for two reasons: to be able to make a comparison of the resources used by each implementation; and because having two simultaneous and independent computations increased our chances to meet a tight deadline. In both cases, the Lanczos algorithm used 128–bit vectors and took 4324 iterations to find 57 dependencies.

The uniprocessor machine was fitted with a 1.33GHz AMD Athlon processor and had 768 megabytes of memory. The software was compiled with the GCC compiler; RedHat Linux 7.0 was the operating system. The complete computation took 146 446 seconds, or a little under 40.7 hours. The amount of active virtual memory reached a maximum of 537 megabytes, but fell to 480 megabytes for the main part of the computation. These figures are well below the size of real memory available and so paging was not a problem.

The parallel implementation ran on a cluster of sixteen machines, each of which contained two 300MHz Pentium-II processors and 384 megabytes of memory. (Note that the memory available on a single cluster node would not have been sufficient to hold the entire data set.) We used the Microsoft Visual C++

compiler, together with the MPIPro multi-processor harness communicating via 100Mbps ethernet; the nodes ran the Windows 2000 operating system. We used 12 nodes and only one processor per node as the remainder of the cluster was required by others. Detailed records of the resources used are available only for the master node, which is the node responsible for all the I/O required for the computation. It took 33 808 seconds, or 9.4 hours, for its share of the computation and used 63 megabytes of active memory. The other nodes would have taken about the same cpu time, or very slightly less. One of the slave nodes was observed to be using 53 megabytes of memory. The 10 megabyte difference between these two figures is accounted for by the data structures needed by the master node to co-ordinate the computation as a whole. If we assume the eleven slave nodes each used 53 megabytes, the total memory usage came to 646 megabytes, substantially more than the 480 megabytes used by the uniprocessor.

If we assume that all the nodes took the same 9.4 hours of cpu time, the total computation comes to $12 * 9.4 = 112.8$ hours. Earlier experiments with heavily instrumented versions of the parallel code, admittedly working on much less dense matrices, showed that this is a reasonable assumption. A naive computation of the total number of cpu cycles used by each implementation yields 1.9×10^{14} for the uniprocessor, and 1.2×10^{14} for the cluster. Although it appears at first sight that the parallel implementation is *more* efficient, it must be stressed that this is an over-simplified analysis: the code was compiled with substantially different compilers and run under very different operating systems; and, even when the compilers and operating systems are identical, the runtime can be heavily dependent on memory bandwidth, cache efficiency, and other non-computational effects. In both cases, the linear algebra phase took less than 0.1% as much computation as did the sieving (cf. Section 4).

We warn against extrapolation of our parallel Lanczos result. Nevertheless, based on the apparent feasibility of parallelized Lanczos and the economical feasibility of clusters of small machines, we caution against assuming that fairly small RSA moduli are safe because the matrix problem is said to be too hard.

2.5 Combination of Relations and Production of the Factors

The combination of linearly-dependent relations produced by the linear algebra used exactly the same code as in [9]. Processing each dependency took 18 minutes on a 400MHz machine. The third dependency yielded the factors $p =$ 337 779 774 700 456 816 455 577 092 228 603 627 733 197 301 999 086 530 154 776 370 553 and $q =$ 346 129 173 115 857 975 809 709 331 088 291 920 685 569 205 287 238 835 924 196 565 083 957 of 2,1606L.c135 = 116 915 434 112 329 921 568 236 283 928 181 979 297 762 987 646 390 347 857 868 153 872 054 154 807 376 462 439 621 333 455 331 738 807 075 404 918 922 573 575 454 310 187 518 221. At 66 digits, p is the largest penultimate factor ever found by the MPQS algorithm.

3 Cycle Behaviour

3.1 Introduction

As the number R of relations increases, the number C of independent cycles increases at an ever increasing rate. With one large prime per relation, the growth rate in C is very nearly proportional to R^2. This can be explained using the birthday paradox [9] and was fully analyzed in [10]. For RSA-129, factored with PPMPQS, the quadratic behaviour broke down about half way through the computation [1], and a better approximation for the later stages was that C is proportional to R^4. Obviously, a large fraction of the cycles arose from the *ppr*s.

Our results for the TMPQS factorization of 2,1606L.c135 are summarized in Fig. 1, where $\ln(C)$ is plotted against $\ln(R)$ during the course of the computation. The line is the least squares fit to the data: its slope of 2.7665 shows that the growth is faster than quadratic, but it is clear that a power law is not a good approximation to the behaviour.

3.2 Cycle Types

The observation that the cycle-growth behaviour depends on the type of relations appearing in cycles was made in [1]. As part of this investigation we characterize the cycles more fully to get a better (though far from complete) understanding of the situation. Cycles consisting only of *par* relations we termed S-cycles (S for single large prime). Of the remainder, cycles not containing *tpr* relations we

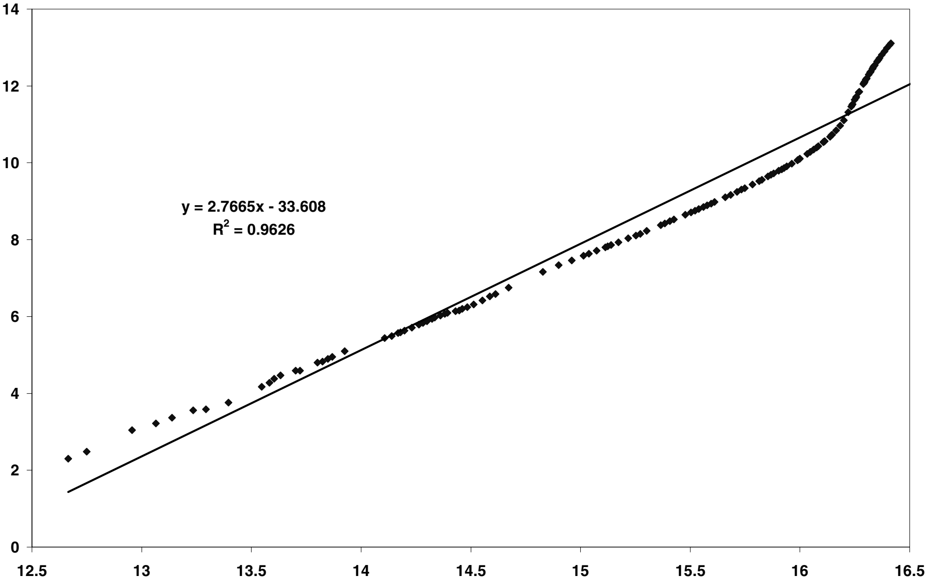

Fig. 1. Behaviour of ln(# cycles) with ln(# relations). Data with $C < 10$ are omitted.

call D-cycles (D for double, since D-cycles contain at least one *ppr* relation and possibly *par* relations), and the remaining cycles are the T-cycles (T for triple). Independent S-cycles may be assumed to consist of two relations. The numbers of independent S, D, and T-cycles are referred to by S, D, and T, respectively. Thus, $C = S + D + T$.

During the course of the computation we calculated C, S, D, and T approximately daily. This allowed us to examine in detail the variation of these quantities as a function of the total number of relations R. The behaviour of each of these quantities as a function of R is described below.

S-cycles. In Fig. 2 we plot $\ln(S)$ against $\ln(R)$. Based on the analysis from [9] and [10] we expect a linear plot with slope close to 2, which is indeed what we find: the least-squares fitted slope is 2.078 with a correlation coefficient squared of 0.9997. At the end of the factorization we had $S = 25\,603$, or 5.18% of the total.

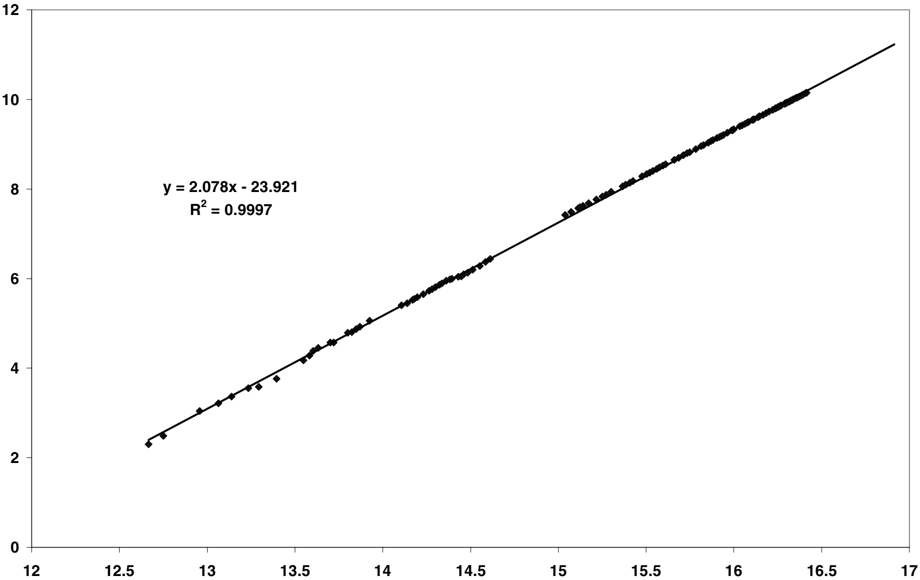

Fig. 2. Behaviour of ln(# S-cycles) with ln(# relations). Data with $S < 10$ are omitted.

D-cycles. In [1] it was observed that $S+D$ grew faster than proportional to R^2, with a suggestion that it may have been proportional to R^4 towards the end of the computation. We are not aware of any other analysis of this quantity. In Fig. 3 we plot $\ln(D)$ against $\ln(R)$. The least square fitted line, with a slope of 3.4969, can be seen to be a remarkably good fit for the data points, though there is a hint that the relationship breaks down at very small values of D. It is tempting to suggest that the slope should be exactly 7/2, but we have *no* theoretical justification for this claim; this would be an interesting subject for further study. By the end of the factorization we had $D = 41\,940$, or 8.49% of the total.

The large gap between $14.7 < \ln(R) < 15.0$ corresponds to a period when counts broken down into S, D, and T were not taken. The edges of the missing-data gap correspond to $D = 100$ and $D = 389$. Prior to this period, T was zero or one — understandably the first T-cycle was eagerly awaited — and D could be calculated from C and the number of large primes occurring in the cycles.

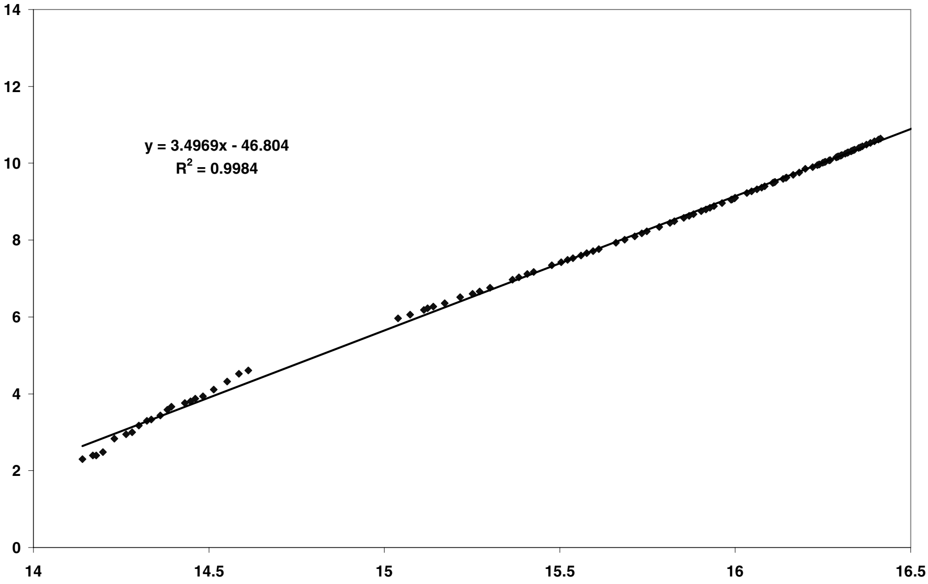

Fig. 3. Behaviour of ln(# D-cycles) with ln(# relations). Data with $D < 10$ are omitted.

T-cycles. When Figs. 2 and 3 are compared with Fig. 1, it is clear that any major non-linearity in the log-log plots must arise from the T-cycles. This is indeed what we find in Fig. 4. The fitted straight line, with a slope of 7.3207, is not a good fit to the data and the data between $16.1 < \ln(R) < 16.3$ shows pronounced curvature. This region corresponds approximately to $1 < R/10^6 < 1.2$. At the end of the factorization we had $T = 426\,534$, or 86.33% of the total.

As can be seen from Fig. 4, the initial growth of T is quite smooth. For $\ln(R) < 15.9$ (i.e., $R < 8\,100\,000$) the data points lie close to a straight line with slope 5.4785 corresponding to a power law with this exponent. Beyond this point, the plot shows an initially rising gradient, reaching a maximum of around 15, and then tailing off again to about 6 near $\ln(R) = 16.35$. In [7] a somewhat similar effect observed during a GNFS factorization was described as explosive growth.

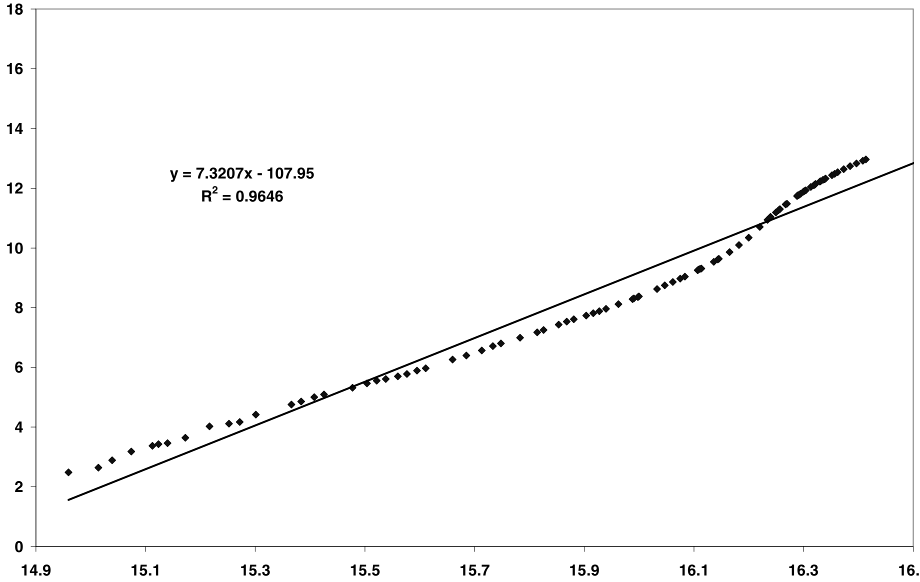

Fig. 4. Behaviour of ln(# T-cycles) with ln(# relations). Data with $T < 10$ are omitted.

In our computation, T showed strongly superpolynomial growth for a short while. The term "explosion" seems too strong for our experience with TMPQS and we prefer "phase transition" as being more appropriate. The two cases, GNFS in [7] and TMPQS, would be expected to show different cycle behaviour. In the former work, relations have four large primes — two algebraic and two rational. The large primes may only be combined with others of the same type, which limits the possible matches. In TMPQS, although there are fewer primes per relation, any of the three primes appearing in a *tpr* relation may combine with any of the (same) primes in other relations. Therefore, it is perhaps not too surprising that we see a phase transition towards the end of the sieving phase.

3.3 Phase Transition in T-Cycle Growth

We present a simple physical model for addition of relations that may justify the phrase "phase transition" to describe the superpolynomial growth in T-cycles. We picture relations as having some of the properties of atoms in chemistry: *par*s are represented as univalent atoms, *ppr*s as divalent, and *tpr*s as trivalent atoms. The silicon - oxygen system shows some of the phenomena we are attempting to model, though care must be taken not to push the analogy too far.

At the beginning of the computation there are no relations, and the model consists of a vacuum without atoms. As relations are added most will not share any primes with relations already present. In our model, these form a monatomic gas. A few will share a prime with another relation; the two atoms will form a chemical bond with one of their valencies. Initially at least, only pairs of *par* relations will have their primes matched (two *par*s with the same large prime). This corresponds in our model to a diatomic molecule formed of two univalent atoms. Other matched relations (diatomic molecules) contain unmatched primes (unsatisfied valencies) and remain available for the addition of further relations (are very reactive in the presence of other atoms and molecules).

As relations are added, the number with matched primes increases. Equivalently, as atoms are added to the gas the density rises and more molecules are formed. Eventually, the density rises high enough that large molecules are produced, some forming rings and chains. Each divalent atom added to a molecule is likely to bind only to one free valency, thereby growing a chain. Initially, at least, each trivalent atom will add an extra free valency to the growing molecule.

The connection between the chemical model and singleton removal is clear: each pruning pass removes those primes which occur only once. In the chemical model, this corresponds to breaking the bonds to those atoms which contain a free valency. When singleton removal has run its course, only cycles remain; when all bonds to reactive sites have been broken, only stable molecules remain.

As the density continues to increase, not only will rings of atoms within a molecule build up, adjacent molecules will connect to each other via a di- or trivalent atom. In real chemical systems, such as the addition of silicon and oxygen atoms to a container, at first the atoms are largely isolated. Then stable O_2 and reactive SiO and Si_2 molecules are formed, and then stable SiO_2 molecules together with a whole raft of highly reactive silicon-oxygen molecules with free valencies. All these atoms and molecules are still in the gas phase. Eventually the density becomes so high that the molecules cross-link in profusion and the system as a whole becomes a liquid, glass, or solid (depending on temperature and pressure) in equilibrium with a vapour. A phase transition has taken place and the properties of the condensed phase are very different from the properties of the earlier gaseous phase.[1] In the TMPQS case, the condensed phase does not appear to be crystalline (we do not find one massive and almost fully interconnected component), but rather more akin to a glass. Many of the relations are connected, but in a number of components and with a large number of lengthy chains which terminate in a free valency and, in all, roughly three-quarters of the relations remain in the gas phase — to mix metaphors badly.

When phase changes occur in physical systems, it is usual for many physical properties to change dramatically over a small variation in a quantity such as temperature, density, or magnetic field strength. Some properties, such as the density increase on condensation or viscosity on polymerization, show themselves as near-step functions with an increased gradient at the phase transition in the phase diagram for the system. Such behaviour is similar to that seen in Fig. 4. Other quantities, such as the specific heat capacity, show relatively flat behaviour well away from the phase transition and rise to a sharp peak at the transition itself. It is normal for the heat capacity of the two phases to be different. To

[1] In a real physical system, bonds are not unbreakable and the condensed phase is constantly exchanging material with the vapour. Matched relations do not split and match with other relations. We did warn against pushing the model too far.

take a physical system almost at random, Doye, Sear and Frenkel [8] describe a phase transition in the molecular configuration of four moderately-sized polymers. Fig. 6a of their paper looks somewhat similar to our Fig. 4 though, it must be admitted, the slopes are very different. When we realise that there is a systematic slope in the $\ln(T)$ versus $\ln(R)$ graph for small values of R and compensate for this feature by subtracting the least squares fitted line to this data ($\ln(T) = 5.4785 \ln(R) - 79.469$) we produce Fig. 5. The resemblance between this plot and Fig. 6a of [8] is remarkable. An even more startling similarity can be seen between Fig. 6b of [8] (the temperature - specific heat capacity plot) and our Fig. 6 which shows a plot of the number of pruning passes during singleton removal against R. The two are almost identical in appearance! In our experiment, the “pruning capacity” rises slowly from about 10 to around 25 in the gaseous phase, grows through a peak of almost 140 at the phase change itself and settles down at about 40 in the condensed phase.

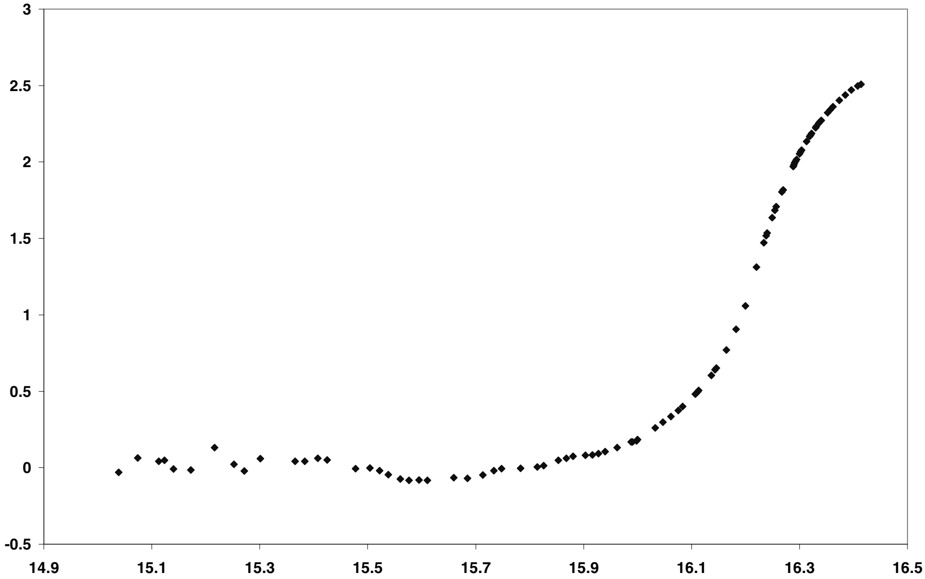

Fig. 5. $\ln(T) - 5.4785 \ln(R) + 79.469$ plotted against $\ln(R)$.

We conclude this section with a repeat of our warning: cycles among relations in a TMPQS factorization are *not* isomorphic to molecules in a chemical mixture, and the analogy should not be pushed too far. Nonetheless, the two systems show remarkably similar behaviour and it would appear that this may be a fruitful field for further study, not least because a similar phenomenon also seems to occur in the GNFS. If the phenomenon were better understood, we may have a hope of selecting sieving parameters which bring forward the onset of the phase transition without unduly slowing down the rate at which relations are produced.

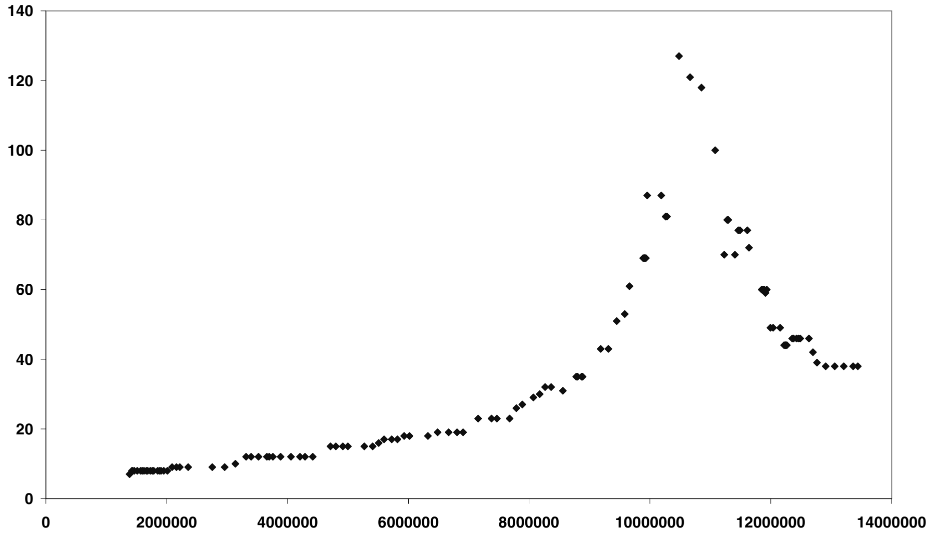

Fig. 6. Behaviour of (# pruning passes) with (# relations).

4 Performance Comparison of TMPQS and PPMPQS

As mentioned in the abstract, one of our objectives was to compare TMPQS and PPMPQS and to test the assertion made in [10] and [6] that TMPQS would be the slower algorithm. Because of the magnitude of the computations required, it is unreasonable to repeat the factorization of a 135-digit integer with PPMPQS purely to make the performance comparison. Fortunately, we can base our comparison on different arguments.

During past PPMPQS factorizations it has often been observed that sieving is approximately half-completed when the number of independent cycles is equal to the number of *ful* relations. As noted in Section 2.3, when we finished we had 62 626 *ful*s and 67 543 cycles not involving *tpr* relations, i.e., the type of cycles generated by PPMPQS. Following this rule of thumb, PPMPQS-sieving would have been about half-completed by the time we were finished. However, this comparison is inaccurate because TMPQS and PPMPQS find *par*s and *ppr*s at different rates: the lower sieve threshold used for TMPQS leads to more false reports (and to *tpr*s) and must therefore affect the *par* and *ppr* yield. For that reason we ran PPMPQS for about a week on the same number, with the same factor base size and large prime bound as used for TMPQS and optimal sieving range. We found that PPMPQS finds non-*tpr*s in 88% of the time of TMPQS (counting only the non-*tpr*s found by TMPQS, but of course TMPQS find *tpr*s too). This would imply that TMPQS does not run about twice as fast than PPMPQS, but only about $0.88 * 2 \approx 1.75$ times faster. An additional small correction may be applied to account for the suboptimality of the PPMPQS factor base size; in our experience this affects the runtime in a very minor way (say, at most 5%).

Based on this analysis, we may conclude that for 2,1606L.c135 and our implementation, TMPQS is more than 1.5 times faster than PPMPQS. This is corroborated by an independent, but admittedly less precise, comparison based on the factorization of RSA-129. During that computation it was found that the complete sieving step for RSA-129 would have taken almost 400 years on a DECStation 5000/25, a fairly common machine in the early 1990s. The first author still has access to such a machine, and ran TMPQS on it for 2,1606L.c135 for 7 117 214 seconds, finding 4901 relations. Given that 13 441 627 relations were needed, this machine would have spent $\frac{13\,441\,627}{4901} \approx 2743$ times 7 million seconds, i.e., about 620 years, to complete the TMPQS sieving. Thus we conclude that this machine would have spent almost 1.6 times as much computation to factor 2,1606L.c135 by TMPQS as it would have done to factor RSA-129 by PPMPQS.

The asymptotic runtime of all variants of the quadratic sieve algorithm to factor N is known to be $L[N] = \exp((1+o(1))\sqrt{\log N \log\log N})$, for $N \to \infty$. We find that, omitting the $o(1)$'s as usual,

$$\frac{L(2{,}1606\mathrm{L.c135})}{L(5 * \text{RSA-129})} \approx 2.7,$$

(where we use 5 ∗ RSA-129 because there the multiplier was 5) so that we may expect that factoring 2,1606L.c135 by PPMPQS is about 2.7 times harder than factoring RSA-129 by PPMPQS. In 13 years of experience with PPMPQS on many numbers in the 100 to 120 digit range, we have never experienced large deviations from the runtime as ‘predicted’ by the above method. Thus, in our experience, this estimate is reasonably reliable. Since the actual TMPQS over PPMPQS ratio is 1.6, we conclude that TMPQS leads to a speed-up of $2.7/1.6 \approx 1.7$ over PPMPQS. This estimate is consistent with the one given earlier.

Although TMPQS appears to be an improvement on PPMPQS, it is still not competitive with GNFS when factoring integers of around 135 digits. Results for the GNFS factorizations of RSA-130 and RSA-140 have been published in [5] and in [4] respectively. On a scale where all costs are normalized to RSA-129 = 5000 MIPS-years we used 8000 MIPS-years to factor 2,1606L.c135 whereas RSA-130 required 750 MIPS-years and RSA-140 took 2000 MIPS-years.

5 Conclusions

We have shown that for our implementation and a particular 135 digit number TMPQS outperforms PPMPQS, despite the gloomy prognostications of [6] and [10]. However, for numbers of this size, GNFS is still about six times faster.

The reason that TMPQS performs so much better than expected is due to a sudden growth in the number of cycles if more than two large primes are used. This phenomenon is familiar from the NFS. A full understanding of what happens is still lacking. We have given an interpretation in terms of a phase transition, borrowing terminology from chemical systems. Further work would be valuable, since it may enable us to improve the efficiency of the NFS.

Acknowledgment

We thank Scott Contini and another (anonymous) ANTS reviewer for their very thorough and insightful criticism and comments.

References

1. D. Atkins, M. Graff, A.K. Lenstra, P.C. Leyland, *THE MAGIC WORDS ARE SQUEAMISH OSSIFRAGE*, Proceedings Asiacrypt'94, LNCS 917, Springer-Verlag (1995), 265-277.
2. B. Carrier, S.S. Wagstaff Jr., *Implementing the hypercube quadratic sieve with two large primes*, Technical Report 2001-45, Purdue University CERIAS (2001) URL `http://www.cerias.purdue.edu/papers/archive/2001-45.{ps,pdf}`.
3. S. Cavallar, *Strategies in filtering in the number field sieve*, ANTS-IV, LNCS 1838, Springer-Verlag (2000) 209–231.
4. S. Cavallar, B. Dodson, A.K. Lenstra, P.C. Leyland, W. Lioen, P.L. Montgomery, B. Murphy, H. te Riele, P. Zimmermann, *Factorization of RSA–140 using the number field sieve*, Proceedings Asiacrypt'99, LNCS 1716, Springer-Verlag (1999), 195–207.
5. J. Cowie, B. Dodson, R.-M. Elkenbracht-Huizing, A.K. Lenstra, P.L. Montgomery, J. Zayer, *A world wide number field sieve factoring record: on to 512 bits*, Proceedings Asiacrypt'96, LNCS 1163, Springer-Verlag (1996), 382–394.
6. R. Crandall, C. Pomerance, *Prime Numbers, a computational perspective*, Springer (2001) 237.
7. B. Dodson, A.K. Lenstra, *NFS with four large primes: an explosive experiment*, Proceedings Crypto'95, LNCS 963, Springer-Verlag (1995) 372–385.
8. J.P.K. Doye, R.P. Sear, D. Frenkel, *The effect of chain stiffness on the phase behaviour of isolated homopolymers*, J. Chemical Physics 108, (1998) 2134–2142 URL `http://brian.ch.cam.ac.uk/~jon/papers/homop/homop.html`.
9. A.K. Lenstra, M.S. Manasse, *Factoring by electronic mail*, Proceedings Eurocrypt'89, LNCS 434, Springer-Verlag (1990) 355–371.
10. A.K. Lenstra, M.S. Manasse, *Factoring with two large primes*, Math. Comp. 63 (1994) 785-798.
11. P.L. Montgomery, *A block Lanczos algorithm for finding dependencies over GF(2)*, Proceedings Eurocrypt'95, LNCS 921, Springer-Verlag (1995), 106–120.
12. P.L. Montgomery, *Distributed Linear Algebra*, 4th Workshop on Elliptic Curve Cryptography, Essen (2000), URL `http://www.cacr.math.uwaterloo.ca/conferences/2000/ecc2000/montgomery.{ppt,ps}`.
13. C. Pomerance, *Analysis and comparison of some integer factoring algorithms* in H.W. Lenstra Jr, R. Tijdeman, editors, *Computational methods in number theory, Part I*, Math. Centre Tracts, Math. Centrum (1982) 89–139.