IMPLEMENTATION OF THE CONTINUED FRACTION INTEGER FACTORING ALGORITHM

by

Carl Pomerance*
Department of Mathematics
University of Georgia
Athens, Georgia 30602

and

Samuel S. Wagstaff, Jr.*
Department of Statistics and
Computer Science
University of Georgia
Athens, Georgia 30602

1. Introduction.

A person wishing to factor a large number on a computer has many options. Trial division is usually a good starting point and may be all that is needed if the number is not too large, say less than 10^{12}. Any residual factor (perhaps the original number) which has not been cracked by trial division should then be subjected to a pseudoprime test - we do not wish to apply factoring algorithms to a number that is probably prime!

A next step could be the Pollard ρ algorithm [10]. This should uncover a prime factor p of N with $O(\sqrt{p})$ arithmetic operations mod N (and with very little space). Thus if $p < 10^{16}$, it should be findable in reasonable time. Other attacks that can find special prime factors of N include the Pollard $p - 1$ method [9] and the related $p + 1$ method (see Williams [17]).

A "general" factoring algorithm is one whose running time depends mainly on the gross order of magnitude of

*Research supported in part by grants from the National Science Foundation.

CONGRESSUS NUMERANTIUM, VOL. 37 (1983), pp. 99-118

the number being factored. A very speedy general factoring algorithm is the SQUFOF method of Shanks [13]. It is recommended for numbers N less than about 10^{24}; the precise useful range, though, is machine dependent. The probable running time is $O(\sqrt[4]{N})$ and very little space is needed. An attractive feature is that all arithmetic is done with numbers only half the length of N.

For larger numbers, the most popular general factoring algorithm has been the continued fraction method (or CFRAC) of Brillhart and Morrison [8]. If

$$L(N) = \exp\{(\ell n\ N\ \ell n \ell n\ N)^{1/2}\}\ ,$$

then the running time for CFRAC should be $L(N)^{\sqrt{2}+o(1)}$ and the space required should be $L(N)^{1/\sqrt{2}+o(1)}$ (see [11]). (Actually the asymptotic analysis for CFRAC, and every other algorithm mentioned in this paper, except for trial division, is based on certain unproved, but reasonable hypotheses.)

There are other good general factoring algorithms, some of which are asymptotically faster than CFRAC (for example, Pomerance's quadratic sieve [11], [4] and the Schnorr-Lenstra method [12]) and they certainly deserve practical investigations. However, at this time, almost all of the practical experience has been with CFRAC.

It thus seems a useful enterprise to collect together in one place some of the shortcuts, programming tricks and variations that greatly extend the useful range of CFRAC. It is against this modified and refined version of CFRAC that competing algorithms should be judged. We are quick to point out that many of the suggestions presented here are not original with us. When this is the case, we of course so indicate.

We view our contribution here as not only putting some major improvements to CFRAC under one roof, but also

refining some of these ideas and advocating their widespread use. This paper is not intended as the last word on CFRAC. Some of the improvements mentioned below have not yet been tried, others perhaps can be implemented better than we have. In addition, our experience has been on a serial mainframe computer. Those working on parallel or vector processors, microcomputers, or special purpose hardware may well have to rewrite the book on CFRAC.

2. Description of CFRAC.

The continued fraction factoring algorithm as presented here is due to Brillhart and Morrison [8]. It is based on an older pre-computer-age method of Lehmer and Powers [7]. The algorithm employs the simple continued fraction expansion of $\sqrt{N}$. Say A_n/B_n is the n-th convergent to $\sqrt{N}$. Define $Q_n = A_n^2 - NB_n^2$. Then $Q_n \equiv A_n^2 \bmod N$. The values of $A_n \bmod N$ and Q_n are generated by a simple iterative procedure described below.

For each Q_n produced a small effort is made to factor it into primes. Since the Q_n are not too large, each $|Q_n| < 2\sqrt{N}$, some of them will factor easily. When enough of them are factored, a subset of factored Q_n's is determined whose product is a square, say X^2 (we only compute $X \bmod N$). Let $Y \bmod N$ be the product of the corresponding $A_n \bmod N$. Then $X^2 \equiv Y^2 \bmod N$ and there is a good chance that $\gcd(X - Y, N)$ is a non-trivial factor of N. If not, try another subset of the factored Q_n's or factor more Q_n's.

The attempt to factor each Q_n involves trial division of Q_n by a set of small primes $p_1, p_2, \ldots, p_m$ called the "factor base". The primes p_i consist of the prime 2 and all of the odd primes up to some point for which the

Legendre symbol $(N/p_i) = 1$. The determination of the subset of factored Q_n's whose product is a square is accomplished by Gaussian elimination over $\mathbb{Z}/2\mathbb{Z}$ on a matrix with $m + 1$ columns and a row for each factored Q_{n_i}. The $i, 0$ entry is 0 or 1 depending on whether Q_{n_i} is positive or negative. The i,j entry for $1 \leq j \leq m$ is $a_{ij} \bmod 2$, where $p_j^{a_{ij}}$ exactly divides Q_{n_i}. For numbers N of moderate size (20 to at least 55 digits), the Gaussian elimination and subsequent steps of the algorithm take negligible time compared to the trial division and the $A_n \bmod N$, Q_n generation.

The recursive procedure for computing the pairs $A_n \bmod N$, Q_n is as follows. Out of necessity we simultaneously compute three auxiliary sequences G_n, q_n, r_n. Let $g = [\sqrt{N}]$. Then we recursively compute our five sequences by the formulas:

$$Q_n = Q_{n-2} + q_{n-1}(r_{n-1} - r_{n-2})$$

$$G_n = 2g - r_{n-1}$$

$$q_n = [G_n/Q_n]$$

$$r_n = G_n - q_n Q_n$$

$$A_n \equiv q_n A_{n-1} + A_{n-2} \bmod N$$

where we initialize as follows

$$Q_{-1} = N,\ Q_0 = 1,\ q_0 = g,\ r_{-1} = g,\ r_0 = 0,$$

$$A_{-1} = 1,\ A_0 = g\ .$$

To give an idea of the complexity of the arithmetic involved in these calculations, we remark that q_n is usually single precision, g, Q_n, G_n and r_n are about half the length of N, and A_n is about the length of N.

3. The "Large Prime" variation.

When trial division of Q_n has been performed all the way to p_m and the remaining cofactor is still larger than 1 but less than p_m^2, then the number Q_n has been completely factored because the cofactor is a prime larger than p_m. In [8] Brillhart and Morrison point out that even though such Q_n do not factor completely over the factor base, they still can be of use. Indeed, if Q_n and Q_k have the same large prime factor, then Q_nQ_k gives a row in our matrix of exponents mod 2.

Our program always saves Q_n and A_n mod N if Q_n factors over the factor base but for one large prime less than p_m^2. In contrast, the Brillhart-Morrison program has a bound UB as a parameter; it saves Q_n and A_n mod N only when the cofactor is less than UB. Usually they chose UB much smaller than p_m^2. They argued that if Q_n has a large prime factor, then it has little chance of being paired with another Q_k having the same cofactor and is thus nearly valueless.

However, we occasionally did find a repeated large prime nearly as large as p_m^2. We saved every Q_n which factored with a large prime because the cost of handling them was small. The Brillhart-Morrison parameter would be useful if there were a non-trivial cost to store the factored Q_n's. For $m = 959$ and N near 10^{50} our experiments lead us to estimate that the choice of $UB = p_m^2/10$ would halve the number of Q_n's to be saved and cause the loss of about 2% of the pairs of Q_n's with a repeated large prime factor. Thus one need store only half as many Q_n's if one is willing to do a little bit more A_n mod N, Q_n generation and trial division.

We say a few words on the implementation of the Large Prime variation. If Q_n factors with the large prime p, then it can be stored as a partially sparse vector. That is, if v_n denotes the 0 - 1 vector of length $m + 1$ corresponding to the prime factorization (and sign) of Q_n/p, then Q_n is stored as the pair v_n, p. At the Gaussian elimination stage of the algorithm, the pairs v_n, p are first ordered by size of the large prime p. Then one pass down the list can eliminate all of the large primes. Indeed, if p is not repeated, then v_n, p is deleted. If the pairs v_{n_i}, p; $v_{n_{i+1}}$, p;...;$v_{n_{i+k}}$,p all have the same large prime p, then v_{n_i},p is deleted and the other pairs are replaced by the vectors $v_{n_i} + v_{n_{i+1}}$,...,$v_{n_i} + v_{n_{i+k}}$ (reduced mod 2, of course). Thus at little cost of space and time, the large primes can be eliminated. This tactic was first implemented by Wunderlich [20], acting on a suggestion of Brillhart-Morrison [8], Remark 5.14.

4. The "Early Abort Strategy".

The Early Abort Strategy (EAS) consists of giving up on a Q_n before trial division is completed if it does not look likely to factor. Specifically, the EAS with parameters

$$1 \leq m_1 < m_2 < \ldots < m_k \leq m,\ 2\sqrt{N} > B_1 > B_2 > \ldots > B_k > 1$$

involves as many as k tests of a Q_n: if after trial division to p_{m_i}, the unfactored portion exceeds B_i, then Q_n is aborted and Q_{n+1} is obtained. Thus the trial division of a Q_n proceeds normally to p_m if and

only if each of the k tests is passed. This procedure was analyzed in [11] and the optimal asymptotic choices of the abort parameters were computed there. Perhaps surprisingly, it was shown that there indeed is a strategic improvement in the running time. CFRAC with k optimally chosen early aborts has a running time of $L(N)^{\sqrt{1.5+(2k+2)^{-1}}+o(1)}$, the case $k = 0$ being straight CFRAC. In contrast, the Large Prime variation only affects the "$o(1)$" in the exponent.

The EAS was anticipated in the work of Brillhart-Morrison, Schroeppel, and deVogelaere (see Remarks 4.2 and 4.6 of [8], page 384 of [6], and [14]). However, until now there does not seem to have been much experimentation with the concept, nor was such a large speed-up expected.

Let $D_1 = 2\sqrt{N}/B_1$ and for $1 < i \leq k$ let $D_i = 2\sqrt{N}/(B_i D_1 \ldots D_{i-1})$. We call the numbers m_i cuts and the numbers D_i bound divisors. Thus at the cut m_i we abort Q_n if the unfactored portion exceeds $2\sqrt{N}/(D_1 \ldots D_i)$. Asymptotically, the optimal choices for $m, m_1, \ldots, m_k, D_1, \ldots, D_k$ are

$$m = L(N)^{(6+2/(k+1))^{-1/2}}$$

$$m_i = m^{i/(k+1)}$$

$$D_i = N^{i/(3k^2+7k+4)}.$$

However, an asymptotic analysis necessarily glosses over details that can be important in practice with finite numbers. In experiments with the EAS we were only roughly guided by the above choices of parameters.

Most of the numbers we factored during our experiments with EAS were 47 to 51 digit divisors of various numbers of the form $b^n \pm 1$ from the Cunningham Pro-

ject [3]. We began with a factor base of $m = 639$. We soon increased m to 959 and almost all of our experiments are with this choice of m. When we made the increase from 639 to 959 we achieved a modest speed up of about 5%. We thus concluded that we were probably near the optimal choice for our program and for numbers near 10^{50}, so we did not experiment further with m. (We chose $m \equiv -1 \bmod 32$ so that the vectors described in §2 would exactly fill a whole number of 32-bit IBM machine words.)

We factored several 47 digit numbers using one abort; with the other numbers we used two or three aborts. With the one abort numbers we experimented by choosing the cut after $m_1 = 10,20,30,40,50,60$ or 70 primes and the bound divisor D_1 between 50,000 and 16,000,000. Of course, the choice of parameters for two or three aborts involved more degrees of freedom. The computation for each factorization was divided into jobs in which 100,000 to 1,000,000 Q_n's were considered with a fixed choice of parameters.

Performance was measured by the rate at which factored Q_n's were produced. (A value of Q_n is said to "factor" if it either factors completely over the factor base or factors with a large prime, see §3.) Assume that a Q_n has the probability p of factoring (with fixed EAS parameters) and that different Q_n's are independent trials in a random process. If we try t values of Q_n in a job, we expect tp of them to factor and the variance to be $tp(1 - p)$. Since p is nearly zero, the standard deviation is approximately $\sqrt{tp}$, or the square root of the expected number of Q_n's factored. As mentioned above each job had $10^5 \leq t \leq 10^6$. We usually factored between 100 and 1000 values of Q_n per job. Thus the standard devi-

ation was about 3% to 10% of the expected number of Q_n's factored.

The running time measured by the computer timer and printed at the end of a job is accurate to within 2% or 3%. Since this has a smaller percentage variation than the observed number of Q_n's factored, we used this time rather than a more complicated measure in evaluating the performance of each job. Thus we computed the statistic for each job

performance = (number of Q_n's factored)/(running time)

and sought to maximize it.

Those Q_n which factor completely over the factor base are more valuable than those which factor with a large prime because the former always give a row in the matrix of exponents mod 2 while the latter often do not. Hence the former should have greater weight than the latter in the performance measure. For $m = 959$ and N near 10^{50} we found that only about 3% of the Q_n which factored were factored completely over the factor base. Thus very few of the latter Q_n were factored per job and hence there was great variation in their number per job. Because of the difficulty of drawing a statistically significant conclusion we did not use a weighted performance measure. The matter deserves further study.

"Evolutionary operation" [1] was used to optimize the performance. This technique is widely used to improve the yield of industrial processes which have similar problems with statistical fluctuations. Suppose we have run a job using one abort with cut m_1 and bound divisor D_1. Choose nearby cuts m_1', m_1'' and nearby bound divisors D_1', D_1'' with

$$m_1' < m_1 < m_1'' , \quad D_1' < D_1 < D_1'' .$$

We then run jobs with the four possible pairs of m_1', m_1'' and D_1', D_1''. We then replace the pair m_1, D_1 with the one of the four pairs which gives the best performance and repeat the process.

Attempting to graph the performance as a function of the parameters chosen, we found that it was essentially constant over a wide range of parameters. The following choices of cuts and bound divisors seem to work well for $10^{40} < N < 10^{54}$, a factor base of $m = 959$, and our program. We believe these choices are close to optimal. In any case they are a good starting point for evolutionary operation.

number of cuts	m_1	m_2	m_3	D_1	D_2	D_3
1	50	-	-	2M	-	-
2	15	80	-	1K	40K	-
3	10	50	150	500	600	1K

In this table, K stands for 1000 and M stands for 1,000,000.

The EAS makes a big difference in the running time of CFRAC. A factorization of a number near 10^{50} which would take 100 hours with no aborts would take about 30 hours with one abort, 14 hours using two aborts, and 12 hours with three aborts. We did not try four aborts because we felt the additional acceleration for numbers in our range would be negligible.

§5. Choice of the multiplier.

As Brillhart-Morrison [8] and Wunderlich [19], [20] point out, it is often more favorable to use the continued fraction expansion of $\sqrt{kN}$ for some small squarefree k than that of $\sqrt{N}$. (This changes the algorithm for the production of $A_n \bmod N$ and Q_n described in §2 as follows: we have $g = [\sqrt{kN}]$ rather than $[\sqrt{N}]$

and $Q_{-1} = kN$ rather than N.) Knuth [6], p. 383 humorously reports that this appears a strange way to proceed "if not downright stupid." In fact, it can be a smart thing to do since the factor base for kN may contain smaller primes on average than that of N. (The factor base for kN consists of $p = 2$ and those odd primes p with $p|k$ or $(\frac{kN}{p}) = (\frac{k}{p})(\frac{N}{p}) = 1$.) Another reason to use a multiplier is if the continued fraction period for $\sqrt{N}$ is too short.

When the length of the period of the continued fraction is not an issue, we found that the Knuth-Schroeppel function $F(k,N)$ defined in [6], §4.5.4 is a good measure of the worth of the multiplier k. Specifically we wish to choose a square-free k so as to maximize

$$F(k,N) = \sum_{i=1}^{m} f(p_i,kN)\log p_i - \frac{1}{2}\log k$$

where for odd p

$$f(p,kN) = \begin{cases} \frac{1}{p+1}, & \text{if } p|k \\ \frac{2p}{p^2-1}, & \text{if } p\nmid k \end{cases}$$

and

$$f(2,kN) = \begin{cases} \frac{1}{3}, & \text{if } 2|k \text{ or } kN \equiv 3 \bmod 4 \\ \frac{2}{3}, & \text{if } kN \equiv 5 \bmod 8 \\ \frac{4}{3}, & \text{if } kN \equiv 1 \bmod 8 \end{cases}.$$

If k is not square-free, then the definition of $F(k,N)$ is more complicated. However, it follows from this more complicated formula that for any prime p

$$\frac{1}{\log p}(F(p^2k,N) - F(k,N)) = \frac{2}{p+1} + \frac{f(p,kN)}{p} - 1 - f(p,kN) < 0$$

so that $F(k,N)$ is always larger than $F(p^2k,N)$. Thus the optimal choice for k is always square-free.

Not only is the use of the function $F(k,N)$ a good way to choose k, but the function $\frac{1}{2}\log N - F(k,N)$ accurately measures the difficulty of factoring N via CFRAC with multiplier k. That is, if N_1 with optimal multiplier k_1 gives a smaller value of this function than N_2 with optimal multiplier k_2, then N_1 should take less time to factor than N_2.

§6. The $A_n \bmod N$, Q_n generation.

Although asymptotically the time it takes to produce the next $A_n \bmod N$, Q_n is negligible, in practice this can take a non-trivial portion of the total running time. When using the EAS (see §4) necessarily some and perhaps many of the good Q_n's are discarded before it is realized that they will factor. Hence the EAS causes us to compute more values of $A_n \bmod N$, Q_n than would ordinarily be required. Thus it is especially important with the EAS to have an efficient algorithm to produce these numbers.

We used two programming tricks to accelerate the production of $A_n \bmod N$, Q_n pairs. At one point in the algorithm (see §2) one computes the quotient $q_n = [G_n/Q_n]$ and the remainder $r_n = G_n - q_nQ_n$. The numbers q_n are the denominators in the continued fraction expansion of $\sqrt{N}$ (or $\sqrt{kN}$ if the multiplier k is used). For "random" irrational numbers, most denominators in the continued fraction expansion are quite small. Thus one would expect, and this is indeed what is observed, that most q_n are small. Knowing beforehand that q_n is likely to be small can help us compute q_n and r_n faster. (See Remark 5.1 in [8].)

Our program makes a quick estimate of q_n based on the high order digits (in radix 2^{30}) of G_n and Q_n. If it appears that $q_n \leq 5$, the division is performed by repeated subtraction, which also computes r_n. It if appears that q_n will exceed 5 but will be less than 10^6, then the program forms a good approximation to q_n using real arithmetic, multiplies it by Q_n, subtracts from G_n and adjusts q_n by addition or subtraction to the correct value. If $q_n \geq 10^6$, the program performs the multiprecise division via the classical algorithm (see §4.3.1 of [6]).

The other trick concerns the calculation of A_n mod N by the congruence $A_n \equiv q_n A_{n-1} + A_{n-2} \bmod N$. Since the reduction modulo N is time consuming and the q_n are usually small, the reduction is performed only occasionally. The program reduces A_n (and A_{n-1})mod N only when (i) A_n approaches N^2 in size, (ii) q_n exceeds 10^6, or (iii) Q_n has been factored so that it and A_n mod N must be output.

Some ideas that we did not program, but might be worth a try include the following. When A_n is reduced mod N, this necessarily involves some multiplication of N by small numbers. Perhaps a table of multiples of N can be stored so that a reduction mod N consists solely of table look-ups, shifts and subtractions. This idea might even be combined with reducing A_n mod N for each n. For then the quotient

$$g_n \overset{\text{def}}{=} [(q_n(A_{n-1} \bmod N) + (A_{n-2} \bmod N))/N]$$

does not exceed q_n and so is usually small. Thus the large majority of reductions mod N could involve a real arithmetic estimation of g_n, a single table look up for, say, a multiple of N up to 100N, plus a single subtraction. If $g_n > 100$, then A_n mod N could be computed the long way.

Another idea is to define A'_n as follows:

$$A'_n = q_n A'_{n-1} + A'_{n-2} - k_n N$$

where k_n is defined by the following algorithm. Let $k_1 = 1$. If $A'_{n-1}, A'_{n-2} > 0$, then let $k_n = k_{n-1} + 1$. If $A'_{n-1}, A'_{n-2} < 0$, then let $k_n = k_{n-1} - 1$. If $A'_{n-1}A'_{n-2} < 0$, then let $k_n = 0$. This actually requires no multiplication by N, since if $k_n \neq 0$, then $k_n N = k_{n-1} N \pm N$. The only danger is that a huge value for q_n or a quick succession of very large values of q_n could produce a huge value of A'_n. If this occurs, it and the next value could be replaced by their residues mod N.

An alternative way of choosing k_n in the above is to let it be 0 or the largest power of 2 such that $A'_n > 0$. This involves no multiplication by N and no table look-ups. Probably there would never be an exceptional case requiring special treatment.

§7. Other ideas.

In this section we discuss two variations of CFRAC that we have not tried to implement. One of these is to use the Pollard ρ method [10], [2] in place of trial division. It was shown in [11] that CFRAC with Pollard ρ has running time $L(n)^{\sqrt{3/2}+o(1)}$ and that if k optimally chosen early aborts are also used, then the running time is $L(n)^{3/\sqrt{7-(k+1)^{-1}}+o(1)}$. (In [11], actually the Pollard-Strassen method was considered, not Pollard ρ, since the former has a rigorous running time analysis. The Pollard-Strassen method is based on the fast Fourier transform and is probably not practical. If one is prepared to accept that Pollard ρ finds a prime factor p of m in time $O(\sqrt{p}\ \log^3 m)$, then the

analysis in [11] holds for Pollard ρ as well.) For this variation to give an improvement in the running time, probably a substantially larger factor base should be used. Perhaps trial division could be used for the very smallest primes, followed by an early abort, followed by the ρ method. This in turn could be cut at several points and aborted if Q_n were not doing well. The asymptotics suggest that the matrix reduction will be a bottle neck, both in space and time. In practice, the space problem appears to be more important.

Another, but less radical idea that we have not yet had time to try is to use the nearest integer continued fraction (NICF) expansion of $\sqrt{N}$ rather than the ordinary continued fraction (OCF). This idea was suggested recently by Williams [16]. There is an iterative algorithm for the NICF that is very similar to the one described in §2 for the OCF. This algorithm is described in detail in Williams-Buhr [18]. Every pair $A'_n \bmod N$, Q'_n generated by the NICF is also one of the pairs $A_m \bmod N$, Q_m found by the OCF (Williams [15]), but not necessarily vice versa. The advantage to using the NICF is that the largest of the $|Q_m|$, those satisfying $|Q_m| \geq (\sqrt{5} - 1)\sqrt{N}$, do not appear in the NICF expansion (Hurwitz [5]). Since the largest Q's are bypassed, the remaining Q's should prove a richer set for factoring into small primes. Moreover, there appears to be zero overhead in implementing this idea.

We plan to try the NICF soon. We predict that for numbers near 10^{50} there will be a 5% to 10% speed-up using the NICF.

§8. Factorizations.

We split about 70 numbers of the form $b^n \pm 1$ from the Cunningham Project [3]. Actually, the numbers we factored were not themselves of the form $b^n \pm 1$, but large composite factors of these numbers that had been previously obtained in [3]. Our work increased the size of the smallest composite listed in [3] from 47 digits to 51 digits. We give below the numbers which were the first "hole" in their table and also the 53 and 54 digit numbers.

Parent Number	Digits in Unfactored Portion	Factorization
$2^{193}-1$	51	6165444023324834061655 9*P29
$2^{362}+1$	50	178925762979037*3830538323149121* 95016376135553173181
$2^{276}+1$	54	577033894648179874459 3*P32
$3^{133}-1$	52	875794894196183806 7*P33
$3^{175}-1$	53	1430128198787051*P38
$5^{73}+1$	51	63554310563*776127677721902 89* 3570677866897550288203
$5^{76}+1$	48	21363981507860375753*P29
$6^{67}-1$	52	90494902832991041546 7529*P28
$6^{77}-1$	47	48484757451097008256 7*P26
$6^{74}+1$	51	330004340083552 9*P36
$6^{76}+1$	53	355518408146401*5028187486478069273* 13038313680704041577
$6^{79}+1$	48	996873129087496 81*P31
$6^{83}+1$	49	2160853606264485187777 1*P26
$7^{64}+1$	50	1110623386241*P38
$12^{47}-1$	50	10617249990997021*P34

The notation Pxx means a prime number of xx digits. The number $2^{362}+1$ has the "algebraic" factorization $(2^{181}-2^{91}+1)(2^{181}+2^{91}+1)$. We report here the factorization of part of $2^{181}-2^{91}+1$. Recently, Mr. Hiromi Suyama factored $2^{181}+2^{91}+1$ using the Brent-Pollard rho method. See [3] for complete factorizations. Mr. Robert D. Silverman finished the factorizations of $2^{181}-2^{91}+1$ and $6^{76}+1$ after we had removed one of the factors. We are grateful to him for letting us publish his factorizations. We found all factors of $5^{73}+1$.

Running times in hours on an IBM 370/158 are given in the following table for those numbers factored using more than one early abort.

Digits (decimal)	Number of Factorizations	Execution Time (in hours)
47	5	6-10
48	12	7-11
49	19	8-19
50	8	12-24
51	7	17-37
52	3	21-51
53	2	29-37
54	1	37

The rather large spread of times for numbers of a given size can be explained by the discussion in §5. These times were needed for a factor base of $m = 959$ primes. Some of the smaller numbers were done with two aborts, the other numbers with three aborts (and with parameters close to those in the table in §4). The times for the numbers of 52, 53, and 54 digits should not be freely extrapolated to other numbers of comparable size since

the sample is too small and because these 6 large numbers (with one 52 digit exception) had inordinately favorable values of the Knuth-Schroeppel function (§5). To our knowledge, the 54 digit number is the largest number to have been factored by a general factoring algorithm.

These running times are about 10-15 times faster than the previous experience with CFRAC. This dramatic improvement is mainly attributable to the EAS, but is also partially due to our improved $A_n \bmod N$, Q_n generation (see §6).

We gratefully acknowledge the University of Georgia Computer Center for granting us the time used in this project.

References

1. G.E.P. Box and N.R. Draper, Evolutionary Operation, A Method for Increasing Industrial Productivity, Wiley, New York, 1969.
2. R.P. Brent and J. M. Pollard, Factorization of the eighth Fermat number, Math. Comp. 36(1981), 627-630.
3. J. Brillhart, D.H. Lehmer, J.L. Selfridge, B. Tuckerman, and S.S. Wagstaff, Jr., Factorizations of $b^n \pm 1$ up to High Powers, to be published by the Amer. Math. Soc.
4. J.L. Gerver, Factoring large numbers with a quadratic sieve, to appear.
5. A. Hurwitz, Uber eine besondere Art die Kettenbruchentwicklung reeler Grossen, Acta Math. 12(1889), 367-405.
6. D.E. Knuth, The Art of Computer Programing, vol. 2, Seminumerical Algorithms, 2nd ed., Addison Wesley, Reading, Massachusetts, 1981.
7. D.H. Lehmer and R.E. Powers, On factoring large numbers, Bull. Amer. Math. Soc. 37(1931), 770-776.
8. M.A. Morrison and J. Brillhart, A method of factoring and the factorization of F_7, Math. Comp. 29(1975), 183-205.
9. J.M. Pollard, Theorems on factorization and primality testing, Proc. Cambridge Philos. Soc. 76 (1974), 521-528.
10. J.M. Pollard, A Monte Carlo method for factorization, BIT 15(1975), 331-334.
11. C. Pomerance, Analysis and comparison of some integer factoring algorithms, to appear in Computational Methods in Number Theory, H.W. Lenstra, Jr. and R. Tijdeman, eds., Math. Centrum, Amsterdam.

12. C.P. Schnorr and H.W. Lenstra, Jr., Factoring integers depending on good luck, to appear.
13. D. Shanks, Square-form factorization, a simple $O(N^{1/4})$ algorithm, to appear.
14. R. deVogelaere, The first 9000 complete factorizations of the h series, Notices Amer. Math. Soc. 23 1976), A-56, Abstract 731-10-47.
15. H.C. Williams, Some results concerning the nearest integer continued fraction expansion of $\sqrt{D}$, J. reine angew. Math. 315(1980), 1-15.
16. H.C. Williams, Some results concerning nearest integer continued fractions in $Q(\sqrt{D})$ and $Q(\sqrt[3]{D})$, Abstracts Amer. Math. Soc. 3(1982), Abstract 797-12-41, p. 416.
17. H.C. Williams, A $p + 1$ method of factoring, Math. Comp. 39(1982), 225-234.
18. H.C. Williams and P.A. Buhr, Calculation of the regulator of $Q(\sqrt{D})$ by use of the nearest integer continued fraction algorithm, Math. Comp. 33 (1979), 369-381.
19. M.C. Wunderlich, A running time analysis of Brillhart's continued fraction factoring method, in M.B. Nathanson, ed., Number Theory Carbondale 1979, Lecture Notes in Math. 751(1979), 328-342.
20. M.C. Wunderlich, A report on the factorization of 2797 numbers using the continued fraction method, unpublished manuscript.