BIT 20 (1980), 176–184

AN IMPROVED MONTE CARLO FACTORIZATION ALGORITHM

RICHARD P. BRENT

Abstract.

Pollard's Monte Carlo factorization algorithm usually finds a factor of a composite integer N in $O(N^{1/4})$ arithmetic operations. The algorithm is based on a cycle-finding algorithm of Floyd. We describe a cycle-finding algorithm which is about 36 percent faster than Floyd's (on the average), and apply it to give a Monte Carlo factorization algorithm which is similar to Pollard's but about 24 percent faster.

1. Introduction.

Let $S = \{0, 1, 2, \ldots, N-1\}$, where N is a (large) integer. Pseudorandom numbers are often generated by an iteration of the form

$$x_{i+1} = f(x_i)\,, \tag{1.1}$$

where $f\colon S \to S$ is some easily-computable function, and $x_0 \in S$ is given [3]. Since S is finite, there exist $m \geqq 0$ and $n \geqq 1$ such that

$$x_{m+n} = x_m\,, \tag{1.2}$$

and from (1.1) it follows that

$$x_{i+n} = x_i \quad \text{for all } i \geqq m\,. \tag{1.3}$$

The minimal such n and m are called the *period* and the *length of the nonperiodic segment* of the sequence (x_i).

Knuth [3] gives a simple and elegant algorithm, attributed to Floyd, for finding a multiple of the period n, using only a small constant amount of storage. The idea of Floyd's algorithm is to find $j \leqq m+n$ such that

$$x_{2j} = x_j\,. \tag{1.4}$$

The algorithm is:

```
$x := x_0$; $y := x_0$; $j := 0$;
  **repeat** $j := j+1$; $x := f(x)$; $y := f(f(y))$
    **until** $x = y$ .
```

Received December 14, 1979.

On termination $x=x_j$, $y=x_{2j}$, and (1.4) holds, so j is a multiple of n. (The period n may now be found by generating $x_{j+1}, x_{j+2}, \ldots$ until $x_{j+n}=x_j$, or by more sophisticated algorithms which use the prime factorization of j).

Knuth [3] and Pollard [7] analysed the average behaviour of Floyd's algorithm under the following assumptions:

A1: Each of the N^N functions $f: S \to S$ occurs with equal probability N^{-N}.
A2: N is large enough that a continuous approximation is valid.
A3: The work is measured by the number of evaluations (and, sometimes, the number of comparisons) – overheads are ignored.

In Section 2 we describe a family of cycle-finding algorithms which may be faster than Floyd's. The worst case is analysed in Section 3, and the average case (under assumptions A1–A3) in Section 4. The optimal algorithm in our family is about 36 percent faster than Floyd's (on the average).

Pollard [7] gave an ingenious Monte Carlo factorization algorithm based on Floyd's cycle-finding algorithm. In Section 5 we describe how Pollard's algorithm may be modified to use our cycle-finding algorithms instead of Floyd's, and the algorithms are compared in Section 6. With an improvement described in Section 7, the best new factorization algorithm is about 24 percent faster than Pollard's (on the average). The results of some empirical comparisons of the two algorithms are given in Section 8.

Recently Sedgewick and Szymanski [11] have given another family of cycle-finding algorithms. Their algorithms, like Gosper's [1], improve on Floyd's and ours at the expense of using more memory. Unfortunately, it does not seem to be possible to use the Sedgewick–Szymanski or Gosper algorithms to speed up Pollard-like factorization algorithms.

There has recently been much interest in factorization algorithms because of their application to cryptography [2, 10]. Pollard-like algorithms require time $O(p^{\frac{1}{2}})$ on average, where p is the smallest prime factor of the composite number N. The best general approach to factorization of large integers N may be to apply a Pollard-like algorithm to find all prime factors p_i of moderate size (say $p_i < 10^{12}$) with high probability, and then apply more sophisticated algorithms [3, 5, 6, 8, 12] to the quotient $N/\prod p_i$ if it can not be shown to be prime by well-known methods [4, 6, 9, 13].

2. A family of cycle-finding algorithms.

Let $\varrho > 1$ be a free parameter. The cycle-finding algorithm B_ϱ is:

Choose u from a uniform distribution on $[0, 1)$;

```
$y := x_0$; $r := \varrho^u$; $k := 0$; $done :=$ **false**;
  **repeat** $x := y$; $j := k$; $r := \varrho \times r$;
    **repeat** $k := k+1$; $y := f(y)$; $done := (x=y)$
    **until** $done$ **or** $(k \geq r)$
  **until** $done$; $n := k-j$ .
```

It is easy to show that B_ϱ terminates with n set to the period of (x_i). The choice of a (pseudo-) random $u \in [0, 1)$ is not essential: it merely makes the average-case analysis of Section 4 tractable. In practice we usually take $u=0$ and $\varrho=2$: see Section 4.

The algorithm B_2 was originally developed to find the period of the pseudo-random number generator supplied (in ROM) with a popular programmable calculator. This generator was claimed in [15] to be linear congruential with period 199017. Table 2.1 gives the values of n (found by Algorithm B_2) and m (found by the obvious algorithm once n is known) for various starting values x_0. It is clear from the table that the actual generator is quite different from the one described in [15].

Table 2.1. *Period* (n) *and length of non-periodic segment* (m) *for a pseudo-random number generator.*

x_0	n	m
2	11160	1095
13	1897	908
18	467	626
45	406	6683
156	1204	5137
608	717	774
1728	490	12

3. Worst-case analysis.

In this section we consider the worst-case for Floyd's algorithm and Algorithm $B_\varrho, \varrho \geqq 2$. We make assumption A3, and measure work in units of f evaluations.

Floyd's algorithm terminates with

$$(3.1) \qquad j = \begin{cases} m & \text{if} \quad m \equiv 0 \pmod{n} \quad \text{and} \quad m > 0 \\ m+n-(m \bmod n) & \text{otherwise}, \end{cases}$$

where $m \bmod n = m - n\lfloor m/n \rfloor$, and the number of f evaluations is $W_F = 3j$. Thus,

$$(3.2) \qquad 3\max(m, n) \leqq W_F \leqq 3(m+n).$$

Suppose the outer **repeat** loop of algorithm B_ϱ is executed $s \geqq 1$ times. Then on termination $j \geqq m$,

$$(3.3) \qquad j = \begin{cases} 0 & \text{if } s=1, \\ \lceil \varrho^{u+s-1} \rceil & \text{if } s>1, \end{cases}$$

and

$$(3.4) \qquad k = j+n \leqq \lceil \varrho^{u+s} \rceil.$$

Let $\bar{s}$ be the smallest integer such that $\bar{s} \geqq 1$ and

$$\varrho^{u+\bar{s}-1} \geqq \max\left(m, \frac{n+1}{\varrho-1}\right). \tag{3.5}$$

Since

$$\lceil \varrho^{u+\bar{s}} \rceil - \lceil \varrho^{u+\bar{s}-1} \rceil \geqq \varrho^{u+\bar{s}} - \varrho^{u+\bar{s}-1} - 1 \geqq n , \tag{3.6}$$

we see that $s \leqq \bar{s}$. Thus,

$$j \leqq \varrho \max\left(m, \frac{n+1}{\varrho-1}\right), \tag{3.7}$$

and the number of function evaluations is

$$W_\varrho = k \leqq \varrho \max\left(m, \frac{n+1}{\varrho-1}\right) + n . \tag{3.8}$$

If $2 \leqq \varrho \leqq 3$, (3.8) gives

$$W_\varrho \leqq 3(m+n)+2 , \tag{3.9}$$

which is almost the same as the bound (3.2) for Floyd's algorithm.

If $\varrho = 2$ and $u=0$, we have, by (3.2) and a slight modification of the argument leading to (3.8),

$$W_\varrho \leqq 2\max(m,n)+n \leqq W_F . \tag{3.10}$$

Thus, in this case B_ϱ is never slower than Floyd's algorithm.

4. Expected behaviour of algorithm B_ϱ.

In this section we analyse the expected behaviour of the cycle-finding algorithms described above, making assumptions A1–A3. We write $\mu = m/N^{\frac{1}{2}}$, $\nu = n/N^{\frac{1}{2}}$, $\tau = \mu+\nu$, $\mu' = \mu/\tau$, and $\nu' = \nu/\tau$. The expected values of μ and ν are

$$E(\mu) = E(\nu) = (\pi/8)^{\frac{1}{2}} , \tag{4.1}$$

and the joint probability density function of μ and ν is

$$\varphi(\mu, \nu) = e^{-(\mu+\nu)^2/2} \qquad (\mu \geqq 0,\ \nu \geqq 0) . \tag{4.2}$$

These results follow from the discussion in Knuth [3].

Let $w_F = W_F/N^{\frac{1}{2}}$ and $w_\varrho = W_\varrho/N^{\frac{1}{2}}$, where W_F and W_ϱ are respectively the number of f evaluations required by Floyd's algorithm and by algorithm B_ϱ. From (3.1) and (4.1) we have, as in Knuth [3],

$$E(w_F) = (\pi/2)^{5/2} \cong 3.0924 . \tag{4.3}$$

This may be compared with the "optimal" value of $E(\tau) = (\pi/2)^{\frac{1}{2}}$, which is approached (at the expense of memory requirements) by the algorithms of Sedgewick and Szymanski [11].

For algorithm B_ϱ we have

$$w_\varrho = \varrho^{u'} \max\left(\mu, \frac{\nu}{\varrho - 1}\right) + \nu , \tag{4.4}$$

where u' is uniformly distributed in $[0, 1)$. (To prove this, follow the derivation of (3.7) above.) It is essential to note that u', μ and ν are independently distributed. Thus, from (4.4), we have

$$E(w_\varrho) = E(\varrho^{u'}) E(\tau) E\left(\max\left(\mu', \frac{\nu'}{\varrho - 1}\right)\right) + E(\nu) . \tag{4.5}$$

From (4.2), μ' and ν' are uniformly distributed on $[0, 1)$, so (4.5) gives

$$E(w_\varrho) = (\pi/8)^{\frac{1}{2}} \left(\frac{\varrho^2 - \varrho + 1}{\varrho \ln \varrho} + 1\right) . \tag{4.6}$$

From (4.6) with $\varrho = 2$, we have

$$E(w_2) = (\pi/8)^{\frac{1}{2}} (3/\ln 4 + 1) \cong 1.9828 . \tag{4.7}$$

This is within 3 percent of the minimum value

$$\min_{\varrho > 1} E(w_\varrho) \cong 1.9260 \tag{4.8}$$

which is attained when $\varrho \cong 2.4771$ satisfies

$$(\varrho^2 - 1) \ln \varrho = \varrho^2 - \varrho + 1 . \tag{4.9}$$

Thus, we simplify implementation of the algorithm and lose little in efficiency by choosing $\varrho = 2$. This choice is also suggested by the worst-case analysis of Section 3.

From (4.3) and (4.7),

$$E(w_2)/E(w_F) = \frac{(3 + 2 \ln 2)}{\pi^2 \ln 2} \cong 0.6412 , \tag{4.10}$$

so B_2 is significantly faster, on average, than Floyd's algorithm. The variance $V(w_2)$ is smaller too:

$$V(w_2) = 13/\ln 16 + 2/3 - \pi (3/\ln 4 + 1)^2/8 \cong 1.4241 \tag{4.11}$$

and

$$V(w_F) \simeq 2\pi^2 - \pi^5/32 - 6\zeta(3) \cong 2.9638 . \tag{4.12}$$

5. Pollard-like factorization algorithms.

Pollard [7] suggested applying Floyd's algorithm with $f(x)$ a suitable polynomial mod N (e.g. $f(x) = x^2 - 1 \pmod N$), and replacing the termination condition "**until** $x = y$" by "**until** $GCD(|x - y|, N) > 1$". (Here $GCD(M, N)$ denotes the greatest common divisor of M and N.) Let $G = GCD(|x-y|, N)$ on termination of this algorithm. If $G = N$ (i.e. $x = y$) no useful result is obtained, and we have to try different x_0 and/or f. Usually, though, the algorithm terminates with $1 < G < N$, and then G is a nontrivial divisor of N. The algorithm can be applied to N/G and (with different x_0 and/or f) to G, if further factors are required.

Let p be the smallest prime factor of N, and $\hat{x}_i = x_i \bmod p$. Because f is a polynomial, the sequence $(\hat{x}_i)$ satisfies

$$\hat{x}_{i+1} = f(\hat{x}_i) \pmod p , \tag{5.1}$$

and is eventually periodic with $m+n \leqq p$. When $j > 0$ satisfies $\hat{x}_{2j} = \hat{x}_j$ then $GCD(|x_{2j} - x_j|, N) \geqq p$, so Pollard's algorithm terminates after at most $j \leqq p$ iterations.

It is plausible to assume that $f(\bmod p)$ behaves like a "random" function and that, from (4.3), the expected value of j is about $(\pi/2)^{5/2} p^{\frac{1}{2}}$. Empirical results suggest that this is true, except for certain "special" f (e.g. $f(x) = x^2$ or $f(x) = x^2 - 2$: see Pollard [7]); in what follows we shall make this assumption. Since $p \leqq N^{\frac{1}{2}}$, the expected number of f evaluations required by Pollard's algorithm is $O(N^{\frac{1}{4}})$.

Instead of modifying Floyd's algorithm, we could equally well modify algorithm B_ϱ by changing the statement "done := $(x = y)$" to "done := $(GCD(|x-y|, N) > 1)$". From the results of Section 4, we might expect this algorithm (P_ϱ) with $\varrho = 2$ to be faster than Pollard's original algorithm (P_F).

The best-known algorithm for finding GCDs is the Euclidean algorithm [3], which takes $O(\log N)$ times as long as one multiplication mod N. Pollard [7] showed that most of the GCD computations in algorithm P_F could be dispensed with, and a similar trick is applicable to algorithm P_ϱ. The idea is simple: if P_F or P_ϱ computes $GCD(z_1, N), GCD(z_2, N), \ldots$, then we compute

$$q_i = \prod_{j=1}^{i} z_j \pmod N , \tag{5.2}$$

and only compute $GCD(q_i, N)$ when i is a multiple of m, where $\log N \ll m \ll N^{1/4}$. Since $q_{i+1} = q_i \times z_{i+1} \pmod N$, the work required for each GCD computation in algorithm P_F (or P_ϱ) is effectively reduced to that for a multiplication mod N in the modified algorithm P'_F (or P'_ϱ). The probability of the algorithm failing because $q_i = 0$ increases, so it is best not to choose m too large. (This problem can be minimised by backtracking to the state after the previous GCD computation and setting $m = 1$: see algorithm P''_2 in Section 7.)

6. Comparison of algorithms P'_F and P'_2.

Let p be the smallest prime factor of N, and for the sake of simplicity suppose that N/p and its nontrivial factors (if any) are much larger than p. Assume that $f(x)$ has the form $(x^2 + c) \bmod N$, and that the "random f" analysis of Section 4 is valid. (We cannot justify this assumption theoretically for any c, but the results of

Section 8 justify it empirically for $c=3$.) As a unit of work we use one multiplication mod N, and ignore the work required for other operations.

The results of Section 4 are applicable with one important modification: in Section 4 the cost of statements such as "done := $(x=y)$" was ignored, but now they must be counted as one multiplication (the same as one f evaluation). With this change we get

$$E(M'_F) = 4(\pi/2)^{5/2}p^{\frac{1}{4}}/3 \cong 4.1232p^{\frac{1}{4}} \tag{6.1}$$

and

$$E(M'_2) = (\pi/2)^{\frac{1}{2}}(3/\ln 4+1)p^{\frac{1}{4}} \cong 3.9655p^{\frac{1}{4}}, \tag{6.2}$$

where $E(M'_F)$ and $E(M'_2)$ are the expected number of multiplications mod N for algorithms P'_F and P'_2, respectively. (Compare (4.3) and (4.7).) The result is disappointing: P'_2 is only about 4 percent faster than P'_F. We note that Pollard-like methods based on the Sedgewick-Szymanski cycle-finding algorithms [11] are much slower than P'_F, at least with a straight-forward implementation.

7. An improved factorization algorithm.

Algorithm P'_2 can be speeded up (on the average) by omitting terms $(x_r - x_k)$ in the product (5.2) if $k<3r/2$. A nontrivial factor of N contained in these terms must also be contained in the terms with $3r/2 \leqq k < 2r$, so the work required to include the terms with $k<3r/2$ is not worthwhile. An analysis similar to that of Section 4 gives

$$E(M''_2) = (\pi/32)^{\frac{1}{2}}(4+(2\ln\pi-2\gamma+3)/\ln 2)p^{\frac{1}{4}} \cong 3.1225p^{\frac{1}{4}}, \tag{7.1}$$

where $E(M''_2)$ is the expected number of multiplications mod N for this algorithm (P''_2). (By changing the constants 2 and 3/2 slightly, this can be reduced to $3.1207p^{\frac{1}{4}}$.) Comparing (6.1) and (7.1), we see that a speedup of about 24 percent has been achieved.

After incorporating the back-tracking idea mentioned in Section 5, omitting the random choice of u, and making some minor modifications which do not affect the asymptotic analysis, our final factorization algorithm P''_2 is as follows:

```
$y := x_0$; $r := 1$; $q := 1$;
  **repeat** $x := y$;
    **for** $i := 1$ **to** $r$ **do** $y := f(y)$; $k := 0$;
    **repeat** $ys := y$;
      **for** $i := 1$ **to** $min$ $(m, r-k)$ **do**
        **begin** $y := f(y)$; $q := q \times |x-y|$ $mod\, N$
        **end**;
      $G := GCD(q, N)$; $k := k+m$
    **until** $(k \geqq r)$ **or** $(G > 1)$; $r := 2 \times r$
```

```
until $G > 1$;
if $G = N$ then
   repeat $ys := f(ys)$; $G := GCD(|x - ys|, N)$
   until $G > 1$;
if $G = N$ then {failure} else {success}.
```

8. Empirical results.

We ran algorithms P'_F and P''_2 for numbers N having least prime factor p, for all odd primes $p < 10^8$, and counted the number M_p of multiplications (mod N) required to find p. Table 8.1 gives the (predicted and actual) mean and (actual) maximum of $M_p/p^{\frac{1}{2}}$. The maxima were attained at $p = 99,398,833$ (for P'_F), and $p = 48,569,393$ (for P''_2). In all cases we took $x_0 = 0$, $f(x) = x^2 + 3 \bmod N$, and $m = 1$.

Table 8.1. *Mean and maximum of $M_p/p^{\frac{1}{2}}$ for odd primes $p < 10^8$.*

Algorithm	Predicted mean	Actual mean	Maximum
P'_F	4.123238	4.122795	20.3613
P''_2	3.122502	3.122533	18.9972

The agreement between the predictions of Sections 6 and 7 and the actual means of $M_p/p^{\frac{1}{2}}$ is satisfactory, and provides empirical justification for the "random f" assumption. The empirical results for $\max(M_p/p^{\frac{1}{2}})$ show that, if algorithm P''_2 has not terminated after $189,972 + 4m$ multiplications mod N, then N has no prime factors less than 10^8. This compares favourably with the method of trial division by the 5,761,455 primes less than 10^8, even assuming they are available [14]. Guy has conjectured that, for some constant K and all primes p,

$$M_p \leqq K(p \ln p)^{\frac{1}{2}} \tag{8.1}$$

but this appears difficult to prove: see [8].

Acknowledgement.

The author thanks D. E. Knuth and J. M. Pollard for their stimulating correspondence on the subject of this paper, B. P. Molinari and J. M. Robson for their comments on a first draft, and the Australian National University for the provision of computer time.

REFERENCES

1. M. Beeler, R. W. Gosper and R. Schroeppel, *Hakmem*, M.I.T. Artificial Intelligence Lab. Memo No. 239, Feb. 1972, item 132, pg. 64.
2. W. Diffie and M. Hellman, *New directions in cryptography*, IEEE Trans. Information Theory IT-22 (1976), 644–654.
3. D. E. Knuth, *The Art of Computer Programming*, vol. 2, Addison–Wesley, Reading, Mass., 1969.

4. G. L. Miller, *Riemann's hypothesis and a test for primality*, Proc. Seventh Annual ACM Symposium on Theory of Computing, ACM, New York, 1975, 234–239.
5. M. A. Morrison and J. Brillhart, *A method of factoring and the factorization of F_7*, Math. Comp. 29 (1975), 183–208.
6. J. M. Pollard, *Theorems on factorization and primality testing*, Proc. Camb. Phil. Soc. 76 (1974), 521–528.
7. J. M. Pollard, *A Monte Carlo method for factorization*, BIT 15 (1975), 331–334. MR50#6992.
8. J. M. Pollard, *Monte Carlo methods for index computation* (*mod p*), Math. Comp. 32 (1978), 918–924. MR52#13611.
9. M. Rabin, *Probabilistic algorithms*, in *Algorithms and Complexity* (J. F. Traub, ed.), Academic Press, New York, 1976, 31–40.
10. R. L. Rivest, A. Shamir and L. Adleman, *A method for obtaining digital signatures and public-key cryptosystems*, Comm. ACM 21 (1978), 120–126.
11. R. Sedgewick and T. G. Szymanski, *The complexity of finding periods*, Proc. Eleventh Annual ACM Symposium on Theory of Computing, ACM, New York, 1979, 74–80.
12. D. Shanks, *Class number, a theory of factorization, and genera*, Proc. Sympos. Pure Math., vol. 20, Amer. Math. Soc., Providence, Rhode Island, 1970, 415–440. MR47#4932.
13. R. Solovay and V. Strassen, *A fast Monte-Carlo test for primality*, SIAM J. Computing 6 (1977), 84–85.
14. M. C. Wunderlich and J. L. Selfridge, *A design for a number theory package with an optimized trial division routine*, Comm. ACM 17 (1974), 272–276.
15. Anonymous, *ML-15, Random number generator*, TI Programmable 58/59 Master Library, Texas Instruments Inc., 1977, 52–54.

DEPARTMENT OF COMPUTER SCIENCE
AUSTRALIAN NATIONAL UNIVERSITY
CANBERRA, A.C.T.2600
AUSTRALIA