Factor: An Integer Factorization Program for the IBM PC

Richard P. Brent
Computer Sciences Laboratory
Australian National University
Canberra, ACT 0200

Report TR-CS-89-23
October 1989
Revised March 1994

Abstract

Factor is a program which accesses a large database of factors of integers of the form $a^n \pm 1$. As of March 1994 the database contains more than 175,000 factors of size at least 10^4. The program *factor* implements a simple version of the Elliptic Curve algorithm if it is unable to complete a factorization using trial division and the factor database.

Factor is written in Turbo Pascal and runs on IBM PC or compatible computers. This report describes *factor* and various related programs. The programs and the factor database are available from the author.

1. Introduction

For many years there has been an interest in the prime factors of numbers of the form $a^n \pm 1$, where a is a moderately small integer (the *base*) and n is a positive exponent. Such numbers often arise. For example, if a is prime then there is a finite field F with a^n elements, and the multiplicative group of F has $a^n - 1$ elements. Also, for prime a the sum of divisors of a^n is $\sigma(a^n) = (a^{n+1} - 1)/(a - 1)$. Numbers of the form $a^n + 1$ arise as factors of $a^{2n} - 1$ and in other ways.

An extensive table of factors of $a^n \pm 1$ for $a \le 12$ has been published by Brillhart *et al* [6]. For historical reasons, the computation of [6] is referred to as the *Cunningham Project* after the pioneering computations of Cunningham and Woodall [10]. For a history, see the Introduction in [6].

In the course of proving [4, 5] that there is no odd perfect number less than 10^{300}, we found the tables of [6] very useful, but needed to extend them to higher bases. For example, we needed many factorizations of $a^n - 1$ for $a = 13, 19, 31, 127$. Some of the required factorizations for $a \le 30, n \le 60$ were found in an unpublished table of Silverman [21], but the majority were computed using Lenstra's *Elliptic Curve Method* (ECM) and in some difficult cases the *Multiple Polynomial Quadratic Sieve* (MPQS). The factors were kept in a machine-readable file on a VAX 11/750 computer. Although parts of this file have been published, printed tables of factors are less useful than machine-readable files, and much harder to update as new factors

E-mail address: rpb@cslab.anu.edu.au
 rpb117 typeset using TEX

are found. Hence, we thought it would be useful to make our file of factors generally available. The program *factor* was written with this in mind. It should be considered primarily as a means of accessing a file of known factors, rather than as a general-purpose factorization program. For surveys of factorization algorithms and programs, we refer the reader to [3, 7, 8, 11, 14, 16, 17, 18, 19, 20].

2. The File of Known Factors

A file *hints.dat* contains a list of known factors f of numbers $a^n \pm 1$. Each entry may be thought of as a triple (h, n, f), where $h = a \bmod 10000$, and f is a prime divisor of $a^n \pm 1$. Thus f is a "hint" which may be useful when attempting to factor $a^n \pm 1$. For $a < 10000$ we have $h = a$, so the file is indexed by a and n. For the large a occurring in [4, 5], h may be thought of as a "hash function" [13] derived from a. The file is sorted by increasing h, then by increasing n.

Several devices are used to reduce the size of the file *hints.dat* –

1. Factors f less than a moderate bound B are omitted. These factors may easily be found by trial division. Currently $B = 10,000$.
2. The largest prime factor of a number of the form $a^n \pm 1$ is omitted. These factors may be found by division by the other prime factors.
3. Similarly, the largest prime factors of any Aurifeuillian factors (see Section 3) of $a^n \pm 1$ are omitted.
4. A factor f of $a^m \pm 1$ is not listed as a factor of $a^{km} \pm 1$ for any $k > 1$. When attempting to factorize $a^n \pm 1$, we check each f occurring in an entry $(a \bmod 10000, m, f)$ with $m|n$.

Even with these devices, the file *hints.dat* is rather large (too large to fit on one 360K floppy disk). Thus, we use a *compressed* format which saves space (compared to the *expanded* format) at the expense of some loss of comprehensibility of the file. A program *convert* is provided to convert between the compressed and expanded formats. The compressed format uses the following additional devices to save space –

5. Entries (h, n, f_j) with common h and n may be given on a single line in the format $(h, n, f_1, f_2, \ldots)$.
6. An entry (h, n, F) with *even* F may be used to abbreviate an entry $(h, n, nF+1)$. This takes advantage of the fact that most factors f (of $a^n \pm 1$) in the table satisfy $f = 1 \bmod 2n$.
7. Two decimal digits are packed into one 8-bit byte, i.e. we use a "base one hundred" rather than "base ten" representation.
8. Since the base one hundred representation requires only 7 bits for two decimal digits, the eighth bit is available as a delimiter, and there is no need for blanks or commas to separate $h, n, f_1, f_2, \ldots$

With the space-saving devices just described, the file *hints.dat* occupies 195,745 bytes in compressed format (555,500 bytes in expanded format) for a total of 30,227 entries (h, n, f).† The factors f have an average length of about 10 decimal digits.

† These numbers have increased since this Report was written in October 1989. As of 7 March 1994 there are 175,820 entries and the size has increased in proportion to 1,158,447 bytes in compressed format.

Use of the compressed format increases the amount of CPU time required to scan the file, but may actually decrease the overall time required because of the reduced amount of I/O.

3. The Program *Factor*

Factor accepts positive integers N of the form $(a^n \pm c)/d$ and attempts to factorize them. Of course, any integer N can be expressed in this form (with $a = N$, $n = 1$, $c = 0$, $d = 1$). However, the file *hints.dat* is only useful if $c = \pm 1$ and a is not a prime power. For example, the Fermat number F_6 should be written as $2^{64} + 1$ so that the relevant entry (2, 64, 274177) will be found in *hints.dat.* This would not be the case if F_6 were written as $4^{32} + 1$ or as 18446744073709551617.

Factor first performs trial division by all primes less than the bound B. This finds factors which are too small to be included in the file *hints.dat.* If B were increased then the time required for trial division would increase but the size of the file *hints.dat* would be reduced.

Let N be the number to be factorized after division by those factors less than B. If $N > 1$, the file *hints.dat* is scanned for *relevant* entries, that is entries (h, m, f) with $h = a \bmod 10000$ and $m|n$. For each relevant entry, *factor* checks if $f|N$ and, if so, f (to the appropriate power) is added to a the list of factors found, and N is reduced by division.

Factor next performs greatest common divisor computations with possible Aurifeuillian factors. For example, numbers of the form $2^{4k-2} + 1$ have Aurifeuillian factors $L = 2^{2k-1} - 2^k + 1$ and $M = 2^{2k-1} + 2^k + 1$. Such factorizations were generalized to bases $a > 2$ by Lucas (but the generalizations are usually called Aurifeuillian factors rather than Lucasian factors). We refer the reader to [19] for a description of these factorizations, and a table of the coefficients required†.

Factor implements Aurifeuillian factorizations provided that the square-free part of a is less than 51 (a limitation imposed by the size of our table of coefficients).

Let L, M be possible Aurifeuillian factors, i.e. Aurifeuillian factors of $a^m \pm 1$ for some $m|n$. *Factor* computes GCD (L, N) and GCD (M, N); a nontrivial GCD gives a (possibly composite) factor of N. Composite factors are pushed on a stack for later processing, and N is reduced by division by any factors found.

If the quotient N is nontrivial, *factor* computes GCD (A, N) for possible algebraic factors A of N, that is $A = a^m \pm 1$ for $m|n$. This process may give prime and/or composite factors.

If the quotient N is still nontrivial, it is tested for primality using a probabilistic algorithm [12]. Primality testing is often the most time-consuming part of the whole computation, which is the reason why it is deferred as long as possible. Because the algorithm used is probabilistic, there is a small but positive probability P that a

† See also R. P. Brent, "Computing Aurifeuillian factors", *Proceedings of a Conference on Computational Algebra and Number Theory held at University of Sydney,* Nov. 1992 (edited by W. Bosma and A. van der Poorten), to appear; and R. P. Brent, "On computing factors of cyclotomic polynomials", *Mathematics of Computation* 61 (1993), 131-149.

factor will be treated as prime when it is in fact composite. This probability may be reduced as much as desired by increasing the the number T of trials performed – we have the conservative bound $P \leq 4^{-T}$.

If the factorization is not yet complete, *factor* is left with a (possibly empty) set of prime factors and one or more composite factors. A simple implementation of Lenstra's *Elliptic Curve Method* (ECM) [15] is used in an attempt to complete the factorization. Only the first phase of ECM is used. Implementation of the second phase [2] requires more memory than is usually available on a PC. At each iteration (i.e. each choice of elliptic curve), one of the composites is selected at random – in this way all small factors should be found in a reasonable time.

To simplify implementation of group operations in the groups defined by elliptic curves, *factor* uses the Cauchy symmetric form

$$x^3 + y^3 + z^3 = Dxyz$$

rather than the usual Weierstrass normal form

$$y^2 = x^3 + \alpha x + \beta$$

or the "Montgomery-Chudnovsky" form [9, 16]

$$\beta y^2 = x^3 + \alpha x^2 + x.$$

Use of the Cauchy form leads to symmetric formulas and avoids the need to compute reciprocals (mod N). The cost is a slight reduction in speed, but *factor* is intended to be a simple rather than a particularly efficient implementation of ECM.

4. Implementation Details

Factor and related programs are written in Turbo Pascal, a variant of Pascal which is implemented on IBM PC and compatible machines [1]. We have used Version 4.0 of Turbo Pascal, which allows 32-bit integers (though 16-bit integers are used where possible, since operations on them are faster).

Multiple-precision numbers are represented as decimal strings. In Turbo Pascal, strings have a maximum length of 255 characters (since the length is stored in an 8-bit field). Thus, *factor* can only handle numbers less than 10^{255}. This is not a very serious limitation, since operations such as primality testing on numbers of 200 or more decimal digits are very slow on a PC. We considered that the advantages of using decimal strings outweighed their disadvantages. The main advantages are –

1. Input and output are very straightforward.
2. Debugging is simplified.
3. Functions can return strings, so programming and storage allocation are simplified.

In order to speed up the most time-consuming operations – primality testing and implementation of ECM – *factor* optionally uses a base β larger than 10 in certain critical sections. In fact, *factor* uses $\beta = 180$ because of the constraint $\beta^2 < 2^{15}$.

Factor may be interrupted at any time by pressing a key. Most keys cause *factor* to give a summary of factors found so far, and then to continue. The *escape* key stops execution of *factor* after it gives a summary.

When the ECM phase of *factor* finds a factor, it is written on a file *hints.new* (unless this is impossible because the disk is full or write-protected). The new factor(s) can be added to the file *hints.dat* using a program *update* (see Section 5).

5. Related Programs

Several programs are distributed with *factor* –

Table is a variant of *factor* in which the ECM phase is omitted, the output is less verbose, and a sequence of (possibly incomplete) factorizations is written to a file *table.out*. Thus, *table* is intended for the production of tables of factors, in a format similar to that of the "short" tables at the beginning of [6]. *Table* optionally writes a separate file *tablec.out* of incomplete factorizations – this is useful if one wants to use more powerful programs to find factors which are not currently in the file *hints.dat*. Since *table* repeatedly scans the file *hints.dat*, it is desirable for this file to be on a virtual disk (or on a hard disk with a disk caching program such as *Smartdrive*).

Convert is a program which converts a compressed file of hints to an expanded file. This may be useful if one wants to edit the file or use it as input to another program. *Convert* also converts from expanded to compressed format.

Update is a program which takes a (large) file of hints in compressed format, a (small) file of additional hints in expanded format, and merges them to form a new file in compressed format. Typically *update* is applied to the files *hints.dat* and *hints.new*. The merge requires temporary disk space at least as great as that occupied by the new file (but this temporary space may be on a different disk).

Check is a program which checks a file of hints in expanded format. Each entry (h, e, f) is checked to see if f is prime, if f is a factor of $h^e \pm 1$, and if $f^2 <= h^e$. It is recommended that *check* should be run to check any new factors before adding them to *hints.dat* using *update*. Note that *check* gives spurious warnings about entries with $a \geq 10000$, since it checks if f is factor of $h^e \pm 1$ rather than of $a^e \pm 1$.

Although *factor* and *table* are limited to numbers of less than 256 decimal digits, the file *hints.dat* contains some factors of larger numbers, e.g. factors of $2^n \pm 1$ for $n > 847$, and these may be processed by *convert, update* and *check*.

6. The Distribution Disk(s)

Factor, *hints.dat* and the programs mentioned in Section 5 are available on 3.5" (1440K) and 5.25" (1200K) floppy disks. Both Pascal source files and executable files are available. The programs have been tested on IBM PC AT and compatible machines with at least 640K of memory.

When requesting a copy of *factor*, please specify whether you want 3.5" or 5.25" media, and the densities that you can read. (For 360K 5.25" floppy disks we can only supply an abbreviated version of *hints.dat*.) A small charge may be made to cover media and postage costs, etc. Enquiries may be made by e-mail to `rpb@cslab.anu.edu.au` .

Alternatively, if you have access to the Internet, the files are available by anonymous ftp from `dcssoft.anu.edu.au` in the directory `pub/Brent`, file `rpb117.tar.Z` (see also the `README` and `contents` files).

Acknowledgements

We gratefully acknowledge the assistance of –

Henk Boender, Harvey Dubner, Wilfrid Keller, Peter Montgomery and Mitsuo Morimoto for contributing many factors found by various algorithms (including variants of MPQS, $p-1$, and ECM).

Graeme Cohen, without whom the odd perfect number project [4, 5] would not have been started.

Herman te Riele, who cracked many "difficult" composites with his MPQS program [18]; and the Dutch Working Group on the Use of Supercomputers, for the provision of computer time to run his MPQS program on a Cyber 205 and NEC SX/2.

Hans Riesel for his assistance with the generation of Aurifeuillian factors;

Robert Silverman, for his unpublished tables [21] and various factors found by the special NFS algorithm.

Samuel Wagstaff, for much useful information on the Cunningham project, and the provision of updates to [6].

The staff of the ANU Microcomputer Unit, for their assistance with optical character readers.

The ANU Supercomputer Facility, for the provision of computer time to run our ECM program MVFAC [3] on a Fujitsu VP 100 and VP 2200/10 computers.

Borland International, whose Turbo Pascal environment [1] is a pleasure to use.

References

1. *Turbo Pascal Owner's Handbook Version 4.0*, Borland International, 4585 Scotts Valley Drive, Scotts Valley, California 95066 (IBM version, 1987).
2. R. P. Brent, "Some integer factorization algorithms using elliptic curves", *Australian Computer Science Communications* 8 (1986), 149-163.
3. R. P. Brent, "Parallel algorithms for integer factorisation", *Number Theory and Cryptography* (edited by J. H. Loxton), London Mathematical Society Lecture Note Series 154, Cambridge University Press, 1990, 26-37.
4. R. P. Brent and G. L. Cohen, "A new lower bound for odd perfect numbers", *Mathematics of Computation* 53, 187 (July 1989), 431-437. Supplement, *ibid*, S7-S24.
5. R. P. Brent, G. L. Cohen and H. J. J. te Riele, *Improved techniques for lower bounds for odd perfect numbers*, Report CMA-R50-89, Centre for Mathematical Analysis, Australian National University, October 1989. To appear in *Mathematics of Computation.*
6. J. Brillhart, D. H. Lehmer, J. L. Selfridge, B. Tuckerman and S. S. Wagstaff, Jr., *Factorizations of* $b^n \pm 1, b = 2, 3, 5, 6, 7, 10, 11, 12$ *up to high powers*, American

Mathematical Society, Providence, Rhode Island, second edition, 1985. (Also updates up to update 2.7 and # 3624.)

7. D. A. Buell, "Factoring: algorithms, computations, and computers", *J. Supercomputing* 1 (1987), 191-216.
8. T. R. Caron and R. D. Silverman, "Parallel implementation of the quadratic sieve", *J. Supercomputing* 1 (1988), 273-290.
9. D. V. Chudnovsky and G. V. Chudnovsky, *Sequences of numbers generated by addition in formal groups and new primality and factorization tests*, Dept. of Mathematics, Columbia University, July 1985.
10. A. J. C. Cunningham and H. J. Woodall, *Factorisation of* $y^n \mp 1, y = 2, 3, 5, 6, 7, 10, 11, 12$ *Up to High Powers (n)*, Hodgson, London, 1925.
11. R. K. Guy, "How to factor a number", *Congressus Numerantum XVI*, Proc. Fifth Manitoba Conference on Numerical Mathematics, Winnipeg, 1976, 49-89.
12. D. E. Knuth, *The Art of Computer Programming*, Vol. 2, Addison Wesley, 2nd edition, 1982.
13. D. E. Knuth, *The Art of Computer Programming*, Vol. 3, Addison Wesley, 1973.
14. A. K. Lenstra and M. S. Manasse, *Factoring by electronic mail*, preprint, to appear in *Proceedings Eurocrypt '89.*
15. H. W. Lenstra, Jr., "Factoring integers with elliptic curves", *Ann. of Math.* (2) 126 (1987), 649-673.
16. P. L. Montgomery, "Speeding the Pollard and elliptic curve methods of factorization", *Mathematics of Computation* 48 (1987), 243-264.
17. C. Pomerance, "Analysis and comparison of some integer factoring algorithms", in *Computational Methods in Number Theory* (edited by H. W. Lenstra, Jr. and R. Tijdeman), Math. Centrum Tract 154, Amsterdam, 1982, 89-139.
18. H. J. J. te Riele, W. Lioen and D. Winter, Factoring with the quadratic sieve on large vector computers, *Belgian J. Comp. Appl. Math.* 27 (1989), 267-278.
19. H. Riesel, *Prime Numbers and Computer Methods for Factorization*, Birkhäuser, Boston, 1985.
20. R. D. Silverman, "The multiple polynomial quadratic sieve", *Mathematics of Computation* 48 (1987), 329-339.
21. R. D. Silverman, *Private communication*, June 1987.