

# 巨大数の因数分解に関する調査 A Survey on Factoring Large Numbers

## Abstract

It is a very difficult problem to factor a large number in short time, though it is very easy to compute the product of two large numbers.

In this survey paper, we introduce four factoring methods, Trial Division, Difference of Squares, Quadratic Sieve (QS), and General Number Field Sieve (GNFS). We also show some implementations for these methods.

GNFS is a very fast method and is now still developing, but it is not good to factor a general large number into small prime numbers with GNFS, because it takes long time to factor a number whether it has only small primes or large primes.

Some cryptographic techniques are based on that difficulty to be hard to solve, e.g. factoring large numbers. A famous cryptograph, RSA is one of them. The key numbers to decipher RSA cryptograph have 1024 bit length. How long does it take to factor that size numbers?

## 1 はじめに

一般に「因数分解」といわれる物には 2 種類ある。一方は中学や高校で習う  $x$  や  $y$  などについての高次多項式をより次数の低い多項式の積へと分解することで、もう一方は  $15 = 3 \times 5$  のようにある自然数をそれより小さい数の積へと分解することであり、その中でも特に素数だけの積の形にすることを素因数分解という。特に素因数分解については、一つの自然数は順序を除くと一通りにしか分解されないという特徴がある。本稿では後者の意味での因数分解についての調査を記している。

教育課程上は小学校で習うことから分かるように、因数分解を行うことは概念としては非常に簡単なことである。しかし、因数分解の対象となる数が大きな数となると話は別になる。具体的な数として  $8051^{*1}$  を因数分解することで体感してもらいたい。

人間の頭では先の例のように 4 桁程度の数を因数分解するにもかなり時間がかかる。これはコンピュータで行うとしても同様である。そのため、因数分解を高速に行うために様々なアルゴリズムが開発されてきているが、これまでに実際に行われた因数分解の中で最も速いものでも、10 進 200 桁の数を分解するのに実際の経過時間として 1 年もの期間を必要としている。

もし 300 桁程度の数の因数分解が現実的な時間、例えば 1 ヶ月程度で行えることになれば、RSA のような公開鍵暗号の一部を同程度の時間で解読できることになる。

高速に因数分解を行うアルゴリズムの一つとして、量子コンピュータで動作する Shor のアルゴリズム [5] があるが、現在はまだきちんと動作する量子コンピュータが存在しないため、実際上使用することはできない。

本稿では、このような現状にある大きな数の因数分解に対し、汎用コンピュータ向けのアルゴリズムのいくつかを紹介する。

## 2 因数分解の手法

因数分解問題の解法は、分解の対象となる合成数やその分解結果となる素因数が大きくなるにつれて様々な手法が編み出されるようになってきた。[8]

もちろん、手法によって計算量に違いはあるが、因数分解アルゴリズムに関しては計算量が依存する値にも違いがあり、それにより大きく 2 つに分割されている。

まず、因数分解すべき数の大きさに依存するグループである。これに含まれるアルゴリズムとしては後で述べる 2 次篩法やその発展型である複数多項式 2 次篩法、自己初期化 2 次篩法、そして現在最も速いといわれている数体篩法がある。

次に、因数分解した後に出てくる素因数の大きさに依存するグループである。これには単純な試し割り法から Pollard の  $p-1$  法、 $p+1$  法、楕円曲線法、連分数法などが含まれる。

今回はこれらのうち 4 つについて、そのアルゴリズムと高速化方法を紹介する。

### 2.1 試し割り法 (Trial Division)

因数分解の手段として、まずは素直に与えられた数  $n$  を  $n$  より小さな数で割ってみる方法が挙げられる。また、 $n$  が  $p$  で割り切れるとき、 $n = pq$  ( $p \leq q$ ) とすると

$$p^2 \leq pq = n \tag{2.1}$$

から、 $p \leq \sqrt{n}$  の範囲で探索すれば十分であることが分かる。アルゴリズムを具体的に示すと Fig. 2.1 のようになる。

```

for  $i = \{2, 3, 4, \dots, \lfloor \sqrt{n} \rfloor\}$ 
  while  $n \bmod i = 0$ 
    print  $i$ 
     $n \leftarrow n/i$ 
    
```

Fig. 2.1 Algorithm of “Trial Division”

\*1  $8051 = 83 \times 97$

この手法を原理通りに実装すると、合成数での除算試行が無駄であることに気付く。素数だけを用いて試行することが理想だが、一般に素数を全て列挙する式は知られていないため、代替手法として一部の合成数を取り除く手法が提案されている。まずは簡単に、除算候補から 2 以外の偶数を除く、3 以外の 3 の倍数も除く、5 以外の 5 の倍数も除く、という手法がある。この程度の数の除外ならば、 $30k + 1, 7, 11, 13, 17, 19, 23, 29$  ( $k = 1, 2, \dots$ ) だけの候補しか試す必要が無いため、プログラムソースが煩雑になることもないまま除算試行の候補数を  $8/30$  に減らすことができる。

また、ある程度の大きさまでの素数表をあらかじめ保持することで素数だけの試行を行うことができる。この手法では試行の効率は最も良くなるが、その表を保持するためにメモリやディスクの容量を使ってしまうことがネックとなってくる。

この原理に基づく場合の現実的な計算量の限界を考えてみよう。 $\pi(x)$  は  $x$  以下の素数の個数を示す関数で、具体的な値のいくつかを Table 2.1[8] に示した。

Table 2.1 Values of  $\pi(n)$

$n$	$\pi(n)$
$10^{11}$	4,118,054,813
$10^{12}$	37,607,912,018
$10^{13}$	346,065,536,839
$10^{14}$	3,204,941,750,802
$10^{15}$	29,844,570,422,669
$10^{16}$	279,238,341,033,925
$10^{17}$	2,623,557,157,654,233
$10^{18}$	24,739,954,287,740,860

Table 2.1 から考えると、仮に 1 回の除算試行が 50 クロックでできるとしても、3GHz のマシン 1 台しか使わない場合、式 (2.2) から 5.8 日かけての  $10^{30}$  程度の数の分解が 1 つの限界といえる。勿論、マシンの台数や時間をもっと費やせばさらに大きな数の分解ができるかもしれないが、分解される数が 10 桁も伸びることは考えられない。

$$\pi(10^{15}) \times 50[\text{clock}] \div 3\text{G}[\text{Hz}] \simeq 500\text{K}[\text{sec}] \simeq 5.8[\text{day}] \quad (2.2)$$

## 2.2 平方差分法 (Difference of Squares)

平方差分法では文字式の因数分解

$$x^2 - y^2 = (x + y)(x - y) \quad (2.3)$$

を利用して数の因数分解を行う。因数分解したい数  $n$  が 2 つの奇数  $p, q$  の積<sup>\*2</sup>  $n = pq$  であるとする、 $x = \frac{p+q}{2}, y = \frac{p-q}{2}$  とすれば  $p = x + y, q = x - y$  より

$$n = p \cdot q$$

\*2 もし  $n$  が偶数ならば、あらかじめ必要なだけ 2 で割ることで素因数として 2 を持つことがなくなる。

$$\begin{aligned} &= (x + y)(x - y) \\ &= x^2 - y^2 \end{aligned} \quad (2.4)$$

となる。従って、 $x^2 - y^2 = n$  を満たす  $x, y$  を見付けることができれば、 $p = x + y, q = x - y$  へと分解できることが分かる。

ここで  $n$  が 3 つ以上の素数からなる合成数だった場合、 $p$  と  $q$  のうち、少なくとも一方は合成数となるが、その時は合成数と判断されたものをさらに因数分解すればよい。

この計算アルゴリズムの実装法はいくつかあるが、その 1 つを Fig.2.2 に示した。

```

x = [sqrt(n)], y = 0
while x^2 - y^2 != n
  if x^2 - y^2 < n then x ← x + 1
  else y ← y + 1
print x + y, x - y

```

Fig. 2.2 Algorithm of “Difference of Squares”

この方法は  $p/q \simeq 1$  ならば特に効果を発揮することは明らかである。しかし実際の実行時間は  $O(p - q) \approx O(y)$  のオーダーになるため、 $n$  がある程度大きくなると  $p/q \simeq 1$  であって遅くなってしまふのが欠点である。

### 2.2.1 発展手法

ここで、式 (2.4) を次のように変形することを考えてみよう。

$$\begin{aligned} x^2 - y^2 &\equiv 0 \pmod{n} \\ x^2 &\equiv y^2 \pmod{n} \end{aligned} \quad (2.5)$$

合同法を使うと、さまざまな式を乗法的に組み合わせることで式 (2.5) を満たす式を作ることができ、必要な計算量を減らすことが期待できる。具体例として以下の式を考える。

$$\begin{cases} 14 \cdot 67 \equiv 3 \pmod{187} \\ 31 \cdot 67 \equiv 20 \pmod{187} \\ 14 \cdot 31 \equiv 60 \pmod{187} \end{cases} \quad (2.6)$$

$$(14 \cdot 31 \cdot 67)^2 \equiv 60^2 \pmod{187} \quad (2.7)$$

式 (2.6) の 3 式の両辺をそれぞれ掛け合わせた結果、式 (2.7) となるので、この結果から

$$\text{GCD}(14 \cdot 31 \cdot 67 + 60, 187) = 17 \quad (2.8)$$

$$\text{GCD}(14 \cdot 31 \cdot 67 - 60, 187) = 11 \quad (2.9)$$

と計算することで  $187 = 11 \cdot 17$  と分解できる。また、結果として出てくる数は Table 2.2 に示すように必ずしも自明でない因数<sup>\*3</sup>とは限らない。Table 2.2 の 9 つのパターンのいずれになるかはほぼ等確率なので、 $6/9$  の確率で自明でない因数が出てくるため、複数の組合せを試すことでほぼ確実に分解できる。[3]

\*3 自明な因数とは 1 と  $n$ 。なのでそれ以外の数を自明でない因数という。

Table 2.2 Factorable  $n$  when  $x^2 - y^2 \equiv 0$

$p x+y$	$p x-y$	$q x+y$	$q x-y$	A	B	Factor?
Yes	Yes	Yes	Yes	$n$	$n$	No
Yes	Yes	Yes	No	$n$	$p$	Yes
Yes	Yes	No	Yes	$p$	$n$	Yes
Yes	No	Yes	Yes	$n$	$q$	Yes
Yes	No	Yes	No	$n$	1	No
Yes	No	No	Yes	$p$	$q$	Yes
No	Yes	Yes	Yes	$q$	$n$	Yes
No	Yes	Yes	No	$q$	$p$	Yes
No	Yes	No	Yes	1	$n$	No

\* $A = \text{GCD}(x+y, n)$ ,  $B = \text{GCD}(x-y, n)$

あとは式 (2.6) のような組み合わせをどのようにして効率的に求めるかが問題となってくるが、この方式を用いる因数分解アルゴリズムでは、式 (2.6) のようになりそうな多数の候補を挙げ、その中から有用なものを選抜するという手段をとるため、「篩」系の手法として総称されている。

### 2.3 2次篩法 (QS)

前節で述べた組み合わせを効率的に求める手法の一つがここで述べる 2 次篩法 (Quadratic Sieve; QS) である。

#### 2.3.1 用語解説

ここで、以後の説明を行うために必要な、いくつかの言葉について簡単に説明する。

Factor Base (FB) 簡単な因数分解を行う際に除算試行する数の集合のことである。整数の範囲で言えばある程度までの大きさの素数に  $-1$  を加えた集合とするのが簡単である。特に QS においては

$$t^2 \equiv n \pmod{p} \quad (2.10)$$

となる  $t$  を探索するため、 $n$  が平方剰余とならない  $p$  は除いて良い。平方剰余となるかどうかは

$$n^{\frac{p-1}{2}} \equiv \begin{cases} 1 & (n \text{ は平方剰余}) \\ -1 & (n \text{ は非平方剰余}) \\ 0 & (n \text{ は } p \text{ の倍数}) \end{cases} \pmod{p} \quad (2.11)$$

で分かる。

smooth 与えられた数が定めた FB の要素だけで因数分解できるとき、その数は (その FB 上で) smooth であるという。

特に QS では平方数になるかどうか問題になるので、smooth でない数についても FB の要素で割った商が等しいもの同士を掛け合わせることで smooth な数と同等の扱いをすることができる。

#### 2.3.2 計算手順

1. FB を決める。実際にはある程度の大きさまでの素数の集まりに  $-1$  を加えた集合とすることが多い。
2.  $t = \lfloor \sqrt{n} \rfloor \pm \{1, 2, \dots\}$  について、 $f(t) = t^2 - n$  を FB の要素だけで素因数分解する。smooth でなかった分のデータは、上手く組み合わせることで smooth になる

ものは組み合わせ、それでも残る部分のデータは使わない。

3. 残ったデータのうち、掛け合わせることで FB の全要素についての指数が全て偶数 (0 も含む) になる組合せ  $A$  を求める。このとき、上の手順で残ったデータの数が FB の数より多ければ、FB の指数だけを行列のように表現し、奇数を掃き出すことでその行列の rank を超える行は全て偶数になる。
4. 求めた組合せ  $A$  において  $x = \prod_{t \in A} t$ ,  $y = \sqrt{\prod_{t \in A} f(t)}$  として  $x \pm y$  と  $n$  との最大公約数を求める。ここで  $y$  の値を求める際には、 $\prod_{t \in A} f(t)$  は FB の要素全てについて指数が偶数になることが分かっているので、それらを  $1/2$  にして掛け合わせれば良い。

具体的に  $n = 3937$  を QS を用いて分解してみよう。 $\sqrt{n} \simeq 62.7$  なので、その近辺の区間からリストを作った。

$$\begin{aligned} 55^2 - n &= -912 = -1 \cdot 2^4 \cdot 3 \cdot 19 \\ 56^2 - n &= -801 = -1 \cdot 3^2 \cdot 89 \\ 57^2 - n &= -688 = -1 \cdot 2^4 \cdot 43 \\ 58^2 - n &= -573 = -1 \cdot 3 \cdot 191 \\ 59^2 - n &= -456 = -1 \cdot 2^3 \cdot 3 \cdot 19 \\ 60^2 - n &= -337 = -1 \cdot 337 \\ 61^2 - n &= -216 = -1 \cdot 2^3 \cdot 3^3 \\ 62^2 - n &= -93 = -1 \cdot 3 \cdot 31 \\ 63^2 - n &= 32 = 2^5 \\ 64^2 - n &= 159 = 3 \cdot 53 \\ 65^2 - n &= 288 = 2^5 \cdot 3^2 \end{aligned}$$

この例では数が小さかったため FB を定めず全ての数を素因数分解したが、仮に FB を定義していたならば、FB の要素で割り切ることができない数が残った場合には仮にそれが合成数であろうともそのまま素因数ということにしておく。

ここで、

$$t^2 \equiv t'^2 - n \pmod{n} \quad (2.12)$$

に注意すると、63 と 65 の式を組み合わせることで

$$(63 \cdot 65)^2 \equiv (2^5 \cdot 3)^2 \pmod{n} \quad (2.13)$$

を得られることが分かる。この組み合わせから  $\text{GCD}(63 \cdot 65 - 2^5 \cdot 3, n) = 31$  が導かれる。また、55 と 59 と 63 の式を組み合わせることで

$$(55 \cdot 59 \cdot 63)^2 \equiv (-1 \cdot 2^6 \cdot 3 \cdot 19)^2 \pmod{n} \quad (2.14)$$

も得ることができるが、こちらからは同様に計算すると

$$\text{GCD}(55 \cdot 59 \cdot 63 - (-1) \cdot 2^6 \cdot 3 \cdot 19, n) = 1 \quad (2.15)$$

$$\text{GCD}(55 \cdot 59 \cdot 63 + (-1) \cdot 2^6 \cdot 3 \cdot 19, n) = n \quad (2.16)$$

となるため、失敗となる。

#### 2.3.3 高速化

アルゴリズムに従って  $f(t)$  を順番に素因数分解することを考えると、毎回除算を行う必要があり、とても効率が悪い。そこで、一つ一つの値を順に素因数分解するのではなく、FB の要素ごとにどの値が割り切れるかを確認する方針を採ってみよう。

ある素数  $p$  について

$$f(t) \equiv 0 \pmod{p} \quad (2.17)$$

が成り立っているとき、 $t$  に  $i$  を加えた  $t+i$  では

$$\begin{aligned} f(t+i) &= (t+i)^2 - n \\ &= i^2 + 2it + f(t) \\ &\equiv i(i+2t) \pmod{p} \end{aligned} \quad (2.18)$$

ということが分かる。ここで  $p$  は素数であるから  $f(t+i) \equiv 0 \pmod{p}$  となるためには

$$i \equiv 0 \text{ または } i+2t \equiv 0 \pmod{p} \quad (2.19)$$

でなければならない。

従って、初期値として条件を満たす  $t$  の 1 つを  $t_0$ 、そして  $t_1$  を  $t_1 = -2t_0 \pmod{p}$  とすれば

$$t = t_0 + kp, t_0 + t_1 + kp \quad (k = \pm 1, \pm 2, \pm 3, \dots) \quad (2.20)$$

の場合のみ  $f(t) \equiv 0$  となることが分かる。

次に  $p^i$  について考えてみると、 $t \equiv 0 \pmod{p}$  ならば (2.17) 式より  $n \equiv 0 \pmod{p}$  となっているはずなので  $t \not\equiv 0 \pmod{p}$  とすることができる。このことから、 $p$  が奇素数の場合は  $i \equiv 0 \pmod{p}$  と  $i+2t \equiv 0 \pmod{p}$  が同時に成り立つことは無いので

$$t = t_{j,0} + kp^j, t_{j,0} + t_{j,1} + kp^j \quad (2.21)$$

の場合のみ  $f(t) \equiv 0 \pmod{p^j}$  となる。

このことを利用すると、各  $j$  についてそれぞれ初期値  $t_{j,0}$  を求め、 $p^j$  ごとの値に関する  $p$  の指数に 1 を加えていけば  $p$  に関する指数が求まる。この間、時間のかかる除算が殆ど行われないうえ、数十倍の高速化が期待できる。

### 2.3.4 篩の効率

ここで FB を固定した場合にどの程度の割合で smooth になるかを確認しよう。FB と数の範囲を変えると、smooth になる数の割合がどのように変化するかを Table 2.3 に示した。

Table 2.3 Percentage of smooth numbers  $< n$  [%]

Factor Base \ $n$	100	1000	10000	100000
{2}	7	1.0	0.14	0.017
{2,3}	20	4.0	0.67	0.101
{2,3,5}	34	8.6	1.75	0.313
{2,3,5,7}	46	14.1	3.38	0.694
{2,3,5,7,11}	55	19.2	5.22	1.197

Table 2.3 から、因数分解を行う数自体は小さい方が、FB は大きい方が割合として smooth になりやすい数が多いことが分かる。

### 2.3.5 発展手法

QS では篩にかける数を生成する式が  $f(t) = t^2 - n$  と定まっているため、2.3.4 節から smooth な数を増やすためには FB を大きくするか  $t$  の探索範囲を広げるかしなければならない。しかしこれはどちらとも効率の面から考えると、smooth な数を得られる量に対して全体の計算量を増やすことになるためありがたくない。

そこで、生成式を  $g_{a,b}(t) = (at+b)^2 - n$  のようにして様々な  $(a,b)$  から作られる多項式で得られる値を色々組み合わせることで、小さな  $|g(t)|$  を大量に生成するために  $t$  の探索範囲を広げる必要が無いようにできるという考えから、複数多項式 2 次篩法 (Multiple Polynomial Quadratic Sieve ; MPQS) や自己初期化 2 次篩法 (Self-Initializing Quadratic Sieve ; SIQS) [4] が考え出された。MPQS や SIQS のアルゴリズムなどについての解説は参考文献を参照してもらいたい。

### 2.3.6 計算量

QS の計算量のオーダーは、最初の原理のままでは

$$O(\exp((9/8)^{1/2}(\log n)^{1/2}(\log \log n)^{1/2})) \quad (2.22)$$

だが、複数多項式 2 次篩法に見られるような様々な工夫を行い理想的にできたと仮定すると

$$O(\exp((\log n)^{1/2}(\log \log n)^{1/2})) \quad (2.23)$$

[8] になる。

### 2.4 一般数体篩法

数体篩法 (Number Field Sieve ; NFS) は 1988 年に Pollard が第 7 番目のフェルマー数  $F_7 = 2^{2^7} + 1$  を因数分解するために生み出した因数分解アルゴリズムを、A.K.Lenstra と H.W.Lenstra, Jr. が発展させ、 $r^e + s$  の形をした数に適用できるようにしたものである。[1] ちなみに  $F_7$  は以下のように分解された。

$$F_7 = 59649589127497217 \times 5704689200685129054721 \quad (2.24)$$

その当初は因数分解の基となる数の条件から、ごく一部の数しか分解できなかった。しかし後に、その条件に当てはまらない一般的な数であっても NFS を適用できるように手法が改良された。

この改良によって、一般的な数に適応できることの代償として計算量が大きく増加した。そのため、元の形の数体篩法も特殊な数向けに研究され続けていることとなり、元の数体篩法を特殊数体篩法 (Special Number Field Sieve ; SNFS)、改良された手法を一般数体篩法 (General Number Field Sieve ; GNFS) と区別するようになった。本稿では一般的な数を対象にした因数分解を取り扱うため、GNFS について述べる。

#### 2.4.1 扱う数体について

数体一般についての解説は長くなるので、ここでは GNFS を行う上で必要となる説明だけにとどめる。

まず  $\mathbb{Z}$  は整数体である。つまり単なる「整数の集合」である。次に  $\mathbb{Z}/n\mathbb{Z}$  は有限の整数体で、0 以上  $n$  未満の整数の集合である。数式で表わすと

$$\mathbb{Z}/n\mathbb{Z} = \{a \mid a \in \mathbb{Z} \cap 0 \leq a < n\} \quad (2.25)$$

となる。また、ある代数的数  $\theta$  を用いて表わす  $\mathbb{Z}[\theta]$  は簡単に言うと  $\theta$  の整係数多項式の集合である。ただし GNFS で使われる  $\theta$  は多項式  $f(x)$  の根なので  $f(x)$  の次数を  $d$  とすれば  $\mathbb{Z}[\theta]$  の  $d$  次以上の項は  $d-1$  次以下の項に落とすことができるので

$$\mathbb{Z}[\theta] = \left\{ \sum_{i=0}^{d-1} a_i \theta^i \mid a_i \in \mathbb{Z} \right\} \quad (2.26)$$

と表わすことができる。このとき、 $f(m) \equiv 0 \pmod{n}$  という関係が成り立っていれば、 $\mathbb{Z}/n\mathbb{Z}$  と  $\mathbb{Z}[\theta]$  には

$$\phi: \mathbb{Z}[\theta] \rightarrow \mathbb{Z}/n\mathbb{Z} \quad \theta \mapsto m \quad (2.27)$$

という写像関係が成り立つ。

ここで、 $\mathbb{Z}[\theta]$  上では素元分解 ( $\mathbb{Z}$  でいう素因数分解) が一意に定まらないことを注意しておく。

具体例として  $\mathbb{Z}[\sqrt{-5}]$  で 6 を分解すると

$$6 = 2 \cdot 3 = (1 - \sqrt{-5})(1 + \sqrt{-5}) \quad (2.28)$$

と 2 通りに分解できるが、ここに出てくる  $2, 3, 1 \pm \sqrt{-5}$  はいずれも  $\mathbb{Z}[\theta]$  の中ではこれ以上分解することはできない。

そこで、イデアル [14] を導入することになる。 $\mathbb{Z}[\theta]$  には含まれないイデアル (理想数; ideal number) というものがあり、先ほどの例はそれを用いることで

$$6 = ABCD \quad (2.29)$$

という形に一意分解できると考える。<sup>\*4</sup> また、

$$\begin{cases} 2 &= A \cdot B \\ 3 &= C \cdot D \\ 1 + \sqrt{-5} &= A \cdot C \\ 1 - \sqrt{-5} &= B \cdot D \end{cases} \quad (2.30)$$

とすることで上記の 2 通りの分解ができることが分かる。

ただ、イデアルを用いることによるデメリットもある。 $\mathbb{Z}[\theta]$  上の元  $a$  が 2 つのイデアル  $\alpha, \beta$  の積で表わされるとき

$$a = \alpha \cdot \beta \quad (2.31)$$

$\alpha | a, \beta | a$  を確認することは可能だが、 $\alpha \cdot \beta \rightarrow a$  の方向で求めることはできない。正確には、複数のイデアルを与えられたときに、それらの積が  $\mathbb{Z}[\theta]$  に収まるとは限らないため、 $\theta$  の式で表わすことができるとは限らない。

## 2.4.2 手順

1.  $f(m) \pmod{n} = 0$  を満たす整係数多項式  $f(x)$  と自然数  $m$  を決める。
2.  $f(x) = 0$  の根の 1 つを  $\theta$  とするとき、 $\mathbb{Z}$  と  $\mathbb{Z}[\theta]$  のそれぞれで FB を定める。この時  $\theta$  は説明のために用いるが、実際の計算では具体的な値を知る必要は無い。
3.  $\{a, b\} = [-M_a, M_a] \times [0, M_b]$  の範囲の整数ペア  $(a, b)$  のそれぞれについて  $\mathbb{Z}$  上の FB で  $a + bm$  を、 $\mathbb{Z}[\theta]$  上の FB で  $a + b\theta$  を分解する。
4. 掛け合わせることで FB の指数が全て偶数になる組み合わせを求める。QS と同様に各 FB の指数だけを行列のように表現することで、奇数の数値を掃きだすことができる。
5. 上で求めた組み合わせにおいて、 $\mathbb{Z}[\theta]$  での積  $g(\theta)$  の平方根  $h(\theta)$ <sup>\*5</sup> を求める。ここで  $\mathbb{Z}[\theta]$  上の FB はイデアルとしているため、単純に指数を半分にして掛け合わせることができないことに注意してもらいたい。
6.  $x = \phi(h(\theta)) = h(m)$ ,  $y = \sqrt{\mathbb{Z}}$  での積とする。ここで  $y$  の計算は QS の場合と同様に FB の指数を半分にしたものを掛け合わせることでできる。
7.  $GCD(x \pm y, n)$  が自明な因数でなければ因数分解できる。

## 2.4.3 計算量

GNFS の計算量については、

$$O(\exp((64/9)^{1/3}(\log n)^{1/3}(\log \log n)^{2/3})) \quad (2.32)$$

程度になるといわれているが、これは計算量の大部分を占める、一部の計算量のオーダーである。

GNFS の手順を大まかに 4 つに分けてみると、

1. 多項式決定部分 (手順 1)
2. 篩部分 (手順 3)
3. 依存解析部分 (手順 4)
4. 平方根計算部分 (手順 6)

になる。上記の分類に含まれない手順は、分類された手順と比較して計算量が少ないため、計算量の推計には殆ど考慮されない。

それぞれの手順が占める計算量の割合を考えると、最も多いのが篩部分である。過去の計算結果を見ると具体的な割合は異なるものの、計算時間・計算機の台数ともに篩部分に最も多く使われており、GNFS の計算時間の予想とは概してこの篩部分のみの計算時間を指す。

次に多い部分は指数の依存解析部分である。これは現在 2 番目に大きい 176 桁の分解記録までは篩部分に対して無視できる程度の計算量しかなかったが、200 桁の分解記録においては篩部分の  $1/2 \sim 1/3$  にも達し [2]、決して無視できる計算量ではなくなってきた。

以上のことから、計算速度を速めるには篩部分や依存解析

<sup>\*4</sup> 実際には  $A = B$  ということが分かるので  $6 = A^2CD$

<sup>\*5</sup>  $h(\theta)^2 \equiv g(\theta) \pmod{f(\theta)}$  を満たす多項式  $h(\theta)$

部分での高速化が有効に聞いてくる事が分かる。

他の手順に関しての必要計算量は、全体の計算量に比べて無視できるほどに小さいため考慮されていない。しかし、多項式決定部分に関してはアルゴリズムとしてランダム探索と同等の手段しか発見されていないことから、時間をかければ良い多項式を導き出せる可能性が高まるということで実質的には多くの計算時間が割かれている。

### 3 アルゴリズムの選択

ここまで4つのアルゴリズムを紹介し、GNFSが最も高速であるという説明を行った。しかし、任意に選ばれた大きな数を因数分解しようとする場合、必ずしもGNFSを用いることが最善策であるとは限らない。

#### 3.1 最初に

まずは与えられた数について、例えばRSA暗号の鍵のように2つの大きな素数の積である、というような予備情報が無い場合にはまず試し割り法を適用することで小さな素因数を除いておくが良い。

試し割り法以外のアルゴリズムの多くは、1つの数を2つの因数に分解することを主目的としているため、小さな素因数が沢山ある場合には結果として出てくる因数がそのような小さな素因数(やその積)となることがあるからである。

例えば、5桁程度の素数まで試すとすると

$$\pi(10^5) = 9,592 \quad (3.1)$$

なので、仮にこれらの指数が大きくても短時間で終了することができる。

#### 3.2 中程度の素因数を無くす

次に、 $10^5 \sim 10^{20}$ 程度の素因数を求める。目的は最初の試し割り法と同様で、中程度の素因数を除くことで後の計算にまわす数が大きな素因数しか持たないようにするためである。

これには、今回紹介しなかった手法の $p-1$ 法、 $p+1$ 法、楕円曲線法を使用することが好ましい。これらの手法の計算量は因数分解する対象の数ではなく出てくる素因数の大きさに依存するため、ある程度の時間かけても因数分解できない場合は大きな素因数しかないと考えても構わない。

#### 3.3 素因数分解する

ここまで行った結果、残った合成数は $10^{20}$ を超える大きな素因数の積となっているはずなので、合成数の大きさに依存するQSやGNFSといったアルゴリズムを用いる。

ここで、QSとMPQS、SIQSの3つは本質的に違いが無いので同じアルゴリズムとして扱おうと、 $10^{90} \sim 10^{120}$ 程度を境にしてそれより小さい場合はQS系を、大きい場合はGNFSを使うのが良いとされている。

## 4 関連研究

### 4.1 RSA暗号

RSA暗号は現在広く用いられている公開鍵暗号の一つで、これを解読するための一般的な手段としては因数分解より効率的な手法は知られていない。

的な手法は知られていない。

RSA暗号鍵の作り方を簡単に説明することで、なぜ因数分解することで暗号解読できるのかを説明したい。

まず鍵を作る元として、十分に大きな2つの素数 $p, q$ を準備する。そしてこれらの積を $n = pq$ とする。また、 $\Phi(n) = (p-1)(q-1)$ について

$$ed \equiv 1 \pmod{\Phi(n)} \quad (4.1)$$

を満たす $e$ と $d$ を求めると、 $(d, n)$ のペアが公開鍵、 $(e, n)$ のペアが秘密鍵となる。

ここで $(d, n)$ は第三者に対して公開されるデータなので、この $n$ を $p$ と $q$ に因数分解できたとすれば、鍵を作り出す過程と同じ手順を踏み、秘密鍵 $e$ を求めることで解読可能となる。

現在、RSA暗号の鍵として使われる $n$ のサイズは標準で1024bitとなっている。この大きさの数をGNFSを用いて因数分解するためには、篩部分の計算量として $5.8 \times 10^5$  [年・CPU]から $1.4 \times 10^6$  [年・CPU]を見積もる報告[15]があるが、その予想が正しいかどうかは読者に任せるとされている。

RSA暗号の安全性を見積もるもう1つの指標として、RSA Security社が行っている“The New RSA Factoring Challenge”[9]というコンテストがある。問題として提示されている576, 640, 704, 768, 896, 1024, 1536, 2048bitの数のそれぞれについて、最も早く因数分解した者に賞金が与えられるというものである。2005年5月6日現在では576bitの数しか分解されていない。

### 4.2 TWIRL

現在、ハードウェアで実装することでGNFSの篩部分を高速に行おうという計画がある。その一つであるTWIRL(The Weizmann Institute Relation Locator)[10]は、2003年にA. Shamirらが提唱したもので、試算では数百万ドルをかけることで製作でき、RSA暗号のキーとして現在標準で用いられている1024ビットサイズの数の分解(のうちの篩部分)を1年で行うことができるとされている。

しかし、TWIRLを含め全ての計画はまだ理論段階で、実際のデバイスを製作しているものが無く、机上の空論の域を出ていない。しかも、これらの研究に対しては現在の技術では非現実的であるという考察[12]もなされている。

### 4.3 因数分解プログラム

#### 4.3.1 GGNFS

GGNFS[7]はGNFSを実装した数少ないプログラムの一つで、テキサス工科大学のChris Monico助教授により作成・配布されているオープンソースのプログラムであったが、2005年6月からはSourceForge[11]に入り、オープンソースプロジェクトとして多くの人間が開発にかかわるようになった。プログラムとしてはSourceForge参入前から頻りにプログラムの更新がなされ、パーツによってはCPUに依存した高速化も図られている。

GGNFSを動作させるための条件として、GNUの多倍長

計算パッケージである GMP[6] を使えることが必須となっている。

#### 4.3.2 msieve

msieve は MPQS を実装したプログラムの一つで、他の QS 系を実装したプログラムと比較しても高速に動作する。因数分解を行う際に、最初から MPQS を適用させるのではなく、5桁程度までの素数で試し割りして小さな素因数をなくしてから計算を行っている。

因数分解データを集めている個人の Web サイト “STUDIO KAMADA”[13] では、95桁より大きければ GGNFS を、小さければ msieve を使うことを薦めている。

### 5 計算記録

計算の最大桁数の記録の伸びをグラフにした (Fig. 5.1)。グラフのマーカーは使用されているアルゴリズムによって変化させている。このグラフを見ても分かるように、ここ数年の記録は全て GNFS によって打ち立てられている。

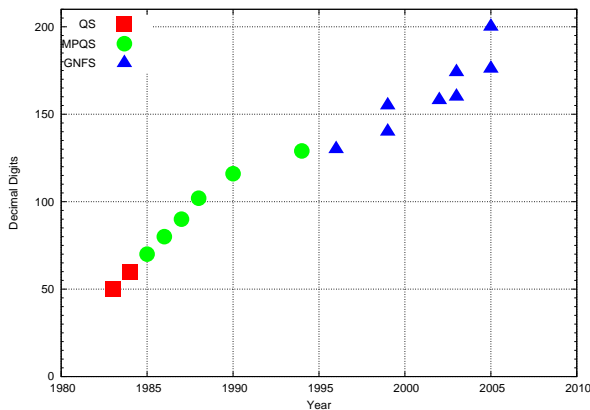


Fig. 5.1 Factoring records

#### 5.1 200桁の数 (RSA200) の分解

現在の最高記録である 200 桁の因数分解 [2] は、2005 年 5 月にボン大学などの研究者のグループによって達成された。分解された数は The New RSA Factoring Challenge の前に行われていた RSA Factoring Challenge の問題の 1 つとして “RSA200” の名称で出題されていた数であり、分解の結果、200 桁の RSA200 は 100 桁の素数 2 つに分解された。この際、因数分解のアルゴリズムには GNFS が用いられた。

この計算に於いて、最も計算量が多くなる篩部分では様々な計算機を不定期に使い、その詳細な時間割り当ては発表されていないが、篩部分の総計算量としては 2.2GHz の Opteron マシン 1 台に換算すると 55 年分になるという形で発表されており、実際には 2003 年のクリスマス前から 2004 年 10 月まで足掛け 11 ヶ月かけている。

また、その次に計算量が多くなる依存解析部分では、80 台の Opteron(2.2GHz) で構成されたクラスタを用いて 2004 年 12 月から約 3 ヶ月かかったと発表されているが、これ以外の部分に関しての計算量は公表されていない。

#### 5.2 176桁の数 ( $11^{281} + 1$ の因数) の分解

現在 2 位の記録となっている 176 桁の分解 [16] は 2005 年 4 月に達成された。RSA200 の分解が達成されるまでは 2 週間ではあるが因数分解桁数の最高記録であった。この記録は立教大学の木田祐司教授と NTT の研究グループによって達成された。分解の対象となった数は  $11^{281} + 1$  の因数の 1 つで、合成数であることが分かっていた 176 桁の数で、87 桁と 89 桁の素数に分解された。 $11^{281} + 1$  の既知の分解は次のようになっていた。C176 とある数がこの記録で分解された数である。

$$\begin{aligned}
 11^{281} + 1 = & 2^2 \times 3 \times 2165143447416427 \\
 & \times 73717873044643423196357 \\
 & \times 47677949460728118782533826487424424473 \\
 & \times 46402408546690245400968842648859676155371 \\
 & \times C176
 \end{aligned}$$

この計算でも GNFS を使用したため、計算量が必要な過程は同じになる。

実際の計算量としてはまず篩部分で RSA200 と同じく様々なマシン (Table 5.1) をバラバラの期間使用し、詳細を明らかにしていないが、篩部分の計算総量として Pentium4(3.2GHz) 換算で 9.7 年かかると発表されている。実際の時間としては 2005 年 3 月 16 日から 27 日かかっている。

Table 5.1 Used machines in factoring C176

台数	CPU	メモリ
6	Pentium III[1.0GHz]	512MB
70	Pentium 4[2.4GHz-3.4GHz]	512M-2GB
1	Athlon 64[2.0GHz]	1GB
148	Pentium 4[2.6GHz]	512MB
32	Pentium 4[3.2GHz]	2GB
28	Pentium III + 4 Opteron[1.0GHz-3.8GHz]	512MB-4GB
100	Pentium III[dual 1.0GHz]	1GB
8	Pentium III[dual 1.4GHz]	1.5GB
7	Pentium 4[3.8GHz]	2GB

行列部分では 36 台の Pentium4(2.8GHz-3.2GHz) で構成されたクラスタで、4 月 16 日から 21 日まで 5.3 日かかった。

また、この記録では多項式の選択にも多くの時間を割いている。2 月 5 日から 3 月 18 日までの約 6 週間のうち、最初の 2 週間は 1 台の PC で、次の 3 週間は 52 台の PC で、最後の 1 週間は 32 台の PC を用い、この多項式探索部分には Pentium4[3.2GHz]1 台に換算すると 3.5 年分の計算量を投じたことになる。後の過程では、真ん中の 3 週間の時に発見した多項式を使用した。

### 6 おわりに

本稿では、試し割り法、平方差分法、QS、GNFS の 4 つの因数分解アルゴリズムについてその背景となる知識や高速

化のための発展手法を説明し、分解しようとする数の大きさに応じたアルゴリズム選択の必要性を述べた。また、素因数分解実験の具体例として現在・過去の世界記録の伸び方とアルゴリズムの変貌について述べた。因数分解を応用する目的の一つとして暗号解読、具体的には RSA 暗号とその解読の難しさについて述べた。

## 参考文献

- [1] A.K.Lenstra and Jr. H.W.Lenstra. *The development of the number field sieve*. Springer, 1991.
- [2] F. Bahr, M. Boehm, J. Franke, and T. Kleinjung. rsa200. <http://www.crypto-world.com/announcements/rsa200.txt>.
- [3] Matthew E. Briggs. An introduction to the general number field sieve. Master theorem, Virginia Polytechnic Institute and State University, 1998.
- [4] S. Contini. Factoring integers with the self-initializing quadratic sieve, 1997.
- [5] Richard Feynman. Simulating physics with computers. *International Journal of Theoretical Physics*, 21(6&7):467–488, 1982.
- [6] GNU. The GNU MP Bignum Library. <http://www.swox.com/gmp/>.
- [7] Chris Monico. GGNFS. <http://www.math.ttu.edu/~cmonico/software/ggnfs/>.
- [8] Hans Riesel. *Prime Numbers and Computer Methods for Factorization*. Birkhäuser, 1994.
- [9] RSA Security Inc. The New RSA Factoring Challenge. <http://www.rsasecurity.com/rsalabs/node.asp?id=2092>.
- [10] A. Shamir and E. Tromer. Factoring large numbers with the TWIRL device, 2003.
- [11] SourceForge. SourceForge.net. <http://sourceforge.net/>.
- [12] (株) 日立製作所. 素因数分解専用集積回路等の実現性についての評価. [http://www.ipa.go.jp/security/enc/CRYPTREC/fy15/documents/rep\\_ID0209.pdf](http://www.ipa.go.jp/security/enc/CRYPTREC/fy15/documents/rep_ID0209.pdf), 2004.
- [13] 鎌田 誠. STUDIO KAMADA. <http://homepage2.nifty.com/m.kamada/math/factorizations.htm>.
- [14] 木田 祐司. 初等整数論. 朝倉書店, 2001.
- [15] 木田 祐司. 素因数分解計算機実験 研究調査報告書. [http://www.ipa.go.jp/security/enc/CRYPTREC/fy15/documents/rep\\_ID0201.pdf](http://www.ipa.go.jp/security/enc/CRYPTREC/fy15/documents/rep_ID0201.pdf), 2003.
- [16] 青木 和麻呂, 植田 広樹, 木田 祐司, and 下山 武司. gnfs176. <http://www.rkmath.rikkyo.ac.jp/~kida/gnfs176.html>.