How to Compute the Coefficients of the Elliptic Modular Function $j(z)$

Harald Baier and Günter Köhler

CONTENTS

We discuss various methods to compute the Fourier coefficients of the elliptic modular function $j(z)$. We present run times to compute the coefficients in practice. If possible, we discuss the theoretical complexity of the corresponding method, too. We conclude that, in practice, an approach due to Kaneko and Zagier turns out to be most efficient.

1. INTRODUCTION

The Fourier coefficients $c(n)$ of the elliptic modular function

$$j(z) = \sum_{n=-1}^{\infty} c(n)q^n \ , \tag{1–1}$$

where $q = e(z) = e^{2\pi iz}$ for z in the upper half plane, are important for several purposes. For instance, they can be used to compute singular values of $j(z)$ and Hilbert class polynomials, and they are needed to compute modular equations. The standard definition of $j(z)$ is

$$j(z) = \frac{E_4^3(z)}{\Delta(z)} \ , \tag{1–2}$$

where

$$\Delta(z) = q \cdot \prod_{n=1}^{\infty}(1-q^n)^{24} = 12^{-3} \cdot \left(E_4^3(z) - E_6^2(z)\right) \tag{1–3}$$

is the discriminant function and

$$\begin{aligned} E_4(z) &= 1 + 240 \cdot \sum_{n=1}^{\infty} \sigma_3(n)q^n, \\ E_6(z) &= 1 - 504 \cdot \sum_{n=1}^{\infty} \sigma_5(n)q^n \end{aligned} \tag{1–4}$$

with $\sigma_r(n) = \sum_{d|n} d^r$ are the Eisenstein series of weights 4 and 6, respectively.

We discuss various approaches to compute the Fourier coefficients $c(n)$. We show that the respective performance is, in practice, very different. Furthermore, we

2000 AMS Subject Classification: Primary 42A16; Secondary 11F03

Keywords: Algorithmic number theory, elliptic modular function j, Fourier coefficients, Fourier series

1058-6458/2001 $0.50 per page

give evidence that a method proposed by Zagier ([Zagier 96]) and Kaneko ([Kaneko 99]) is the most efficient one. For most of the methods discussed in this paper, we present the performance in both theory and practice. More precisely, by the performance in theory, we mean the number of multiplications of integers. Thus we do not consider the contribution of integer additions. If we speak of the performance in practice, we mean the CPU time of our implementation.

One of our aims is that our results are easily verifiable by an "ordinary" user. Hence, all our run times are measured on a PC using freely available libraries. Our practical tests are performed on an Athlon XP1600+ running Linux 2.4.10 at 1.4 GHz and having 1 GByte main memory. All programs are implemented in C++ using the GNU compiler gcc 3.0.1 and the GNU multiprecision package gmp 3.2.1 . As stated above, all software is freely available. The implementation of the methods in this paper may be downloaded from the Website of the first author (http://www.cdc.informatik.tu-darmstadt.de/~hbaier).

The rest of the paper is organized as follows. In Section 2, we discuss two algorithms using the definition of the modular function j. In Section 3, we present two methods proposed by Rademacher [Rademacher 38] and Mahler [Mahler 76], respectively. In Section 4, we investigate an algorithm which makes use of ideas due to Herrmann [Herrmann 73]. In Section 5, we present two methods due to the second author. This approach uses Hecke series. The last method, due to Zagier and Kaneko, is discussed in Section 6.

2. TWO APPROACHES USING THE DEFINITION OF j

In this section, we discuss the performance of our first two algorithms. Both methods are based on Equations (1–2), (1–3), and (1–4). We give evidence that for our purposes none of them is efficient in practice.

In order to make use of the defining equations, we have to know the coefficients of the denominator in Equation (1–2). It is well known that Δ may be written in terms of a Fourier series, that is, we have

$$\Delta(z) = \sum_{n=1}^{\infty} \tau(n) q^n \; . \tag{2–1}$$

The coefficients $\tau(n)$ in Equation (2–1) are called *Ramanujan numbers*. A first approach would be to compute $\tau(n)$ by means of the infinite product for $\Delta(z)$ or by means of $E_4^3(z)$ and $E_6^2(z)$, respectively. However, in the first author's thesis, it is shown that both ideas turn out to be rather slow ([Baier 02]).

We investigate two different representations of the Ramanujan numbers in what follows. The first one is due to Ramanujan himself ([Ramanujan 27]); the second one was proposed by Niebur ([Niebur 75], [Gouvêa 97]).

In 1916, Ramanujan ([Ramanujan 27, page 152]) proved the recursion formula

$$\tau(1) = 1, \quad \tau(n) = -\frac{24}{n-1} \cdot \sum_{k=1}^{n-1} \sigma_1(n-k)\tau(k) \quad \text{for} \quad n > 1 \; . \tag{2–2}$$

Once the values $\sigma_1(n)$ are known, it is obvious how to evaluate and implement Equation (2–2) for $n \geq 2$. First, we are not aware of any reasonable estimation of the computational complexity to compute $\sigma_1(n)$. However, if we use trial division and reasonable values of n, say $n \leq 50000$, we may use machine types and hence fast arithmetic for the computation of $\sigma_1(n)$. Indeed, as explained below, our practical test shows that only about 5% of the CPU time is spent computing the values $\sigma_1(n)$.

Second, we assume that the division by $n-1$ in Equation (2–2) takes the same time as a multiplication. Let $R(N)$ denote the total number of multiplications to compute $\tau(n)$ for $2 \leq n \leq N$ using Ramanujan's recursion formulae. Then it is easy to see that

$$R(N) = \sum_{n=2}^{N} (n+1) = \frac{N^2}{2} + \frac{3N}{2} - 2 \; . \tag{2–3}$$

Finally, if we set $N = 50000$, this approach takes us 6 minutes, 15 seconds in practice to compute $\tau(n)$ for $2 \leq n \leq N$. The CPU time to get all relevant values $\sigma_1(n)$ is 19 seconds.

Let us turn to the second method of this section. It is due to Niebur ([Niebur 75], [Gouvêa 97]). Niebur shows

$$\tau(n) = n^4 \sigma_1(n) - 24 \cdot \sum_{k=1}^{n-1} \left(35k^4 - 52k^3 n + 18k^2 n^2\right) \sigma_1(k)\sigma_1(n-k) \; . \tag{2–4}$$

We describe how we evaluate Equation (2–4). As above, we leave out the cost of the computation of $\sigma_1(n)$ in our following discussion. Let n and k be given. We compute (in this order) k^2, kn, $k^4 = k^2 \cdot k^2$, $k^3 n = k^2 \cdot kn$, and $k^2 n^2 = (kn)^2$. In all, we have to perform 10 multiplications to compute an addend in the sum of Equation (2–4). Thus, the number of multiplications to compute $\tau(n)$ is $10(n-1) + 4 = 10n - 6$. Let $Ni(N)$ be the total amount

of multiplications to compute $\tau(n)$ for $2 \le n \le N$ using Niebur's formula. Then we have

$$Ni(N) = \sum_{n=2}^{N} (10n - 6) = 5N^2 - N - 4 \; . \qquad (2\text{–}5)$$

Compared to $R(N)$, the number of multiplications of this method is about 10 times larger. Indeed, the run time in practice is much slower. If we compute $\tau(n)$ for n up to 50000, the CPU time is 22 minutes, 21 seconds.

We will see in Section 6, that if we use a method due to Kaneko and Zagier, the whole computation of $c(n)$ up to $n = 50000$ takes less than 9 minutes in practice. Thus, we skip the further computation of the $c(n)$ by the methods of this section.

3. APPROACHES DUE TO RADEMACHER AND MAHLER

This section deals with two further methods to compute the coefficients $c(n)$. The first one is due to Rademacher [Rademacher 38]; the second one is due to Mahler [Mahler 76].

H. Rademacher [Rademacher 38] used the circle method to prove a formula which expresses $c(n)$ as a convergent infinite series in terms of Bessel functions and Kloosterman sums. He realized, however, that the convergence of the series is rather slow and that "the coefficients ... can be found [from the formula] by troublesome computations, which for higher n are practically inexecutable ..." For this reason, we did not check how Rademacher's formula performs using the computing power now available. A modern account of Rademacher's and related later work is given in [Knopp 90].

We next discuss Mahler's approach. In [Mahler 76, page 91], K. Mahler proved a system of recursion formulas for $c(n)$. They read

$$\begin{aligned} c(4n) &= c(2n+1) + \frac{1}{2}(c(n)^2 - c(n)) \\ &\quad + \sum_{k=1}^{n-1} c(k)c(2n-k) \; , \qquad (3\text{–}1) \\ c(4n+1) &= c(2n+3) - c(2)c(2n) + \frac{1}{2}(c(n+1)^2 \\ &\quad - c(n+1)) + \frac{1}{2}(c(2n)^2 + c(2n)) \\ &\quad + \sum_{k=1}^{n} c(k)c(2n-k+2) - \sum_{k=1}^{2n-1} (-1)^{k-1} c(k)c(4n-k) \\ &\quad + \sum_{k=1}^{n-1} c(k)c(4n-4k) \; , \qquad (3\text{–}2) \end{aligned}$$

$$c(4n+2) = c(2n+2) + \sum_{k=1}^{n} c(k)c(2n-k+1) \; , \qquad (3\text{–}3)$$

$$\begin{aligned} c(4n+3) &= c(2n+4) - c(2)c(2n+1) - \frac{1}{2}(c(2n+1)^2 \\ &\quad - c(2n+1)) + \sum_{k=1}^{n+1} c(k)c(2n-k+3) \\ &\quad - \sum_{k=1}^{2n} (-1)^{k-1} c(k)c(4n-k+2) \\ &\quad + \sum_{k=1}^{n} c(k)c(4n-4k+2) \; . \qquad (3\text{–}4) \end{aligned}$$

As soon as the values $c(-1), \ldots, c(5)$ are known, the sequence of $c(n)$ is uniquely determined by Mahler's recursion formulas.

We next investigate the number of multiplications to evaluate Mahler's equations. Let $N \in \mathbb{N}, 4 \mid N$. By $M(N)$ we denote the number of multiplications to compute the Fourier coefficients $c(n)$ up to $n = N$ by Mahler's approach. We do not consider a factor $\frac{1}{2}$, as this operation is only a right shift. We fix $1 \le n \le \frac{N}{4}$. Obviously, Equations (3–1)–(3–4) yield a contribution of n, $4n+1$, n, and $4n+3$ multiplications to $M(N)$, respectively. Thus, for fixed n, the contribution is $10n + 4$ multiplications. As we assume $c(-1), \ldots, c(5)$ to be known, we conclude

$$M(N) = \sum_{n=1}^{\frac{N}{4}-1} (10n + 4) + \frac{N}{4} - 6 = \frac{5N^2}{16} - 10 \; . \qquad (3\text{–}5)$$

In this way, a table of $c(n)$ for $n \le 50000$ was computed in the first author's thesis [Baier 02]. The run time in [Baier 02] compared to the method of Section 6. is much larger. Although both hardware and libraries in use of [Baier 02] are inferior to our environment, we expect an implementation of Mahler's Equations (3–1)–(3–4) to be inferior to the method of Kaneko on our platform, too.

4. HERRMANN'S METHOD

We next present two methods for the computation of the $c(n)$ which are both bases on an article by O. Herrmann [Herrmann 73]. The first method is due to Herrmann himself. In an early work in the field of algorithmic number theory, Herrmann computed a table of $c(n)$ for $n \le 6002$ as explained below. The second approach is our variant of Herrmann's algorithm. It turns out that our algorithm is slightly faster in practice than the original method.

The crucial observation is that we may write the discriminant function Δ in terms of Dedekind's η-function. More precisely, we have $\Delta = \eta^{24}$, where

$$\eta(z) = e\left(\frac{z}{24}\right) \cdot \prod_{n=1}^{\infty}(1-q^n) = \sum_{n=1}^{\infty}\left(\frac{12}{n}\right) \cdot e\left(\frac{n^2 z}{24}\right)$$
$$= e\left(\frac{z}{24}\right) \cdot \sum_{n=-\infty}^{\infty}(-1)^n q^{n(3n+1)/2}\,, \qquad (4\text{–}1)$$

where $\left(\frac{12}{n}\right)$ is a quadratic residue symbol. Herrmann ([Herrmann 73]) used Equation (4–1) to compute the values of $c(n)$ for $n \le 6002$ as follows. He avoided the computation of the power E_4^3 in Equation (1–2) by means of the identity

$$E_4^3 = E_{12} + \frac{432000}{691}\Delta\,, \qquad (4\text{–}2)$$

where

$$E_{12}(z) = 1 + \frac{65520}{691} \cdot \sum_{n=1}^{\infty}\sigma_{11}(n)q^n \qquad (4\text{–}3)$$

is the normalized Eisenstein series of weight 12. Then he divided $E_{12}(z)$ repeatedly 24 times by $\eta(z)$. This works well since Euler's series $\sum_{n=-\infty}^{\infty}(-1)^n q^{n(3n+1)/2}$ is sparse with very few coefficients ± 1 and all others equal to 0.

In order to implement Herrmann's proposal, we mention the following observations. First, using the relation $\Delta = \eta^{24}$ and Equations (1–2), (4–2), we get $\left(j - \frac{432000}{691}\right)\cdot\eta^{24} = E_{12}$. Second, it is obvious that we may write the right sum in Equation (4–1) as $\sum_{n=0}^{\infty} e(n)q^n$ with $e(n) \in \{-1; 0; 1\}$. Thus, we get

$$\left(c(-1) + \left(c(0) - \frac{432000}{691}\right)q + \sum_{n=1}^{\infty} c(n)q^{n+1}\right)$$
$$\cdot \left(\sum_{n=0}^{\infty} e(n)q^n\right)^{24} = 1 + \frac{65520}{691} \cdot \sum_{n=1}^{\infty}\sigma_{11}(n)q^n\,. \qquad (4\text{–}4)$$

Once the coefficients $e(n)$ and $\sigma_{11}(n)$ are known, Equation (4–4) shows how to recover the Fourier coefficients of the modular function j. The computation of $\sigma_{11}(n)$ is straightforward. In addition, the computation of the coefficients $e(n)$ is very fast. An algorithm may be found, for instance, like Algorithm 7.3 in the first author's thesis [Baier 02]. We remark that in our implementation, we multiply both sides of Equation (4–4) by 691 to work with integers.

We estimate the number of multiplications to get the $c(n)$ up to $n = N$ by this method. It is obvious that one division by the series $\sum_{n=0}^{\infty} e(n)q^n$ in Equation (4–4) takes $\sum_{k=1}^{N} k = \frac{N(N+1)}{2}$ multiplications. Thus, in all, the number of multiplications using Herrmann's method is at least $12N(N+1)$. However, the multiplications are trivial, as one factor is a coefficient $e(n)$ and therefore in $\{-1; 0; 1\}$. Additionally, the case $e(n) = 0$ is by far the most common. Thus we cannot compare this number of multiplications directly to the number $M(N)$ of Section 3.. The CPU time of this method to compute $c(n)$, $-1 \le n \le 50000$, was 39 minutes, 39 seconds.

We next explain our similar method. The fundamental difference to Herrmann's approach is that instead of successively dividing by $\sum_{n=0}^{\infty} e(n)q^n$ in Equation (4–4), we first compute a series representing the 24th power of $\sum_{n=0}^{\infty} e(n)q^n$. More precisely, let $e_{24}(n)$ denote the Fourier coefficients of the series $\left(\sum_{n=0}^{\infty} e(n)q^n\right)^{24}$. Thus, we set $\sum_{n=0}^{\infty} e_{24}(n)q^n = \left(\sum_{n=0}^{\infty} e(n)q^n\right)^{24}$. Again, Equation (4–4) yields an obvious recursion formula for the $c(n)$, once the values $e_{24}(n)$ and $\sigma_{11}(n)$ are known. In contrast to the coefficients $e(n)$, the computation of the values $e_{24}(n)$ is more burdensome.

In Section 5, we use a Hecke representation of η^8 to get the Fourier coefficients of the series $\left(\sum_{n=0}^{\infty} e(n)q^n\right)^8$. Similar to above, we denote these coefficients as $e_8(n)$. Then we use standard exponentiation methods to compute the coefficients $e_{24}(n)$.

Unfortunately, we are not able to count the number of multiplications of this method to compute the coefficients $c(n)$, $n \le N$. Hence, we cannot present a theoretical estimation of the complexity of our approach. Nevertheless, our practical tests give evidence that our method is slightly faster than Harrmann's original method. For example, $N = 50000$ yields a run time of 35 minutes, 39 seconds. Furthermore, this approach seems to be faster than using Mahler's formula of Section 3.

5. COMPUTATIONS VIA HECKE SERIES

The method in this section is similar to the approach in Section 4. We use the formula

$$j(z) = \gamma_2^3(z) \qquad \text{with} \qquad \gamma_2(z) = \frac{E_4(z)}{\eta^8(z)}\,. \qquad (5\text{–}1)$$

However, it is known from Schoeneberg [Schoeneberg 53] and later writers ([Serre 85], [Köhler 88]), that several powers of $\eta(z)$ are represented by theta series with a Hecke character on an imaginary quadratic number field and that, therefore, their Fourier expansion is lacunary. Specifically, we have (see [Köhler 88, page 84])

$$\eta^8(z) = \frac{1}{6} \cdot \sum_{\mu \in \mathbb{Z}[\omega]} \chi(\mu)\mu^3 e\left(\frac{1}{3}\mu\overline{\mu}z\right),$$

where $\omega = e\left(\frac{1}{6}\right) = \frac{1}{2}(1+\sqrt{-3})$ and

$$\chi(x+y\omega) = \left(\frac{x-y}{3}\right)$$

for $x, y \in \mathbb{Z}$, with a quadratic residue symbol on the right-hand side. We collect the contribution of associated and conjugate elements in $\mathbb{Z}[\omega]$ and obtain the expansion

$$\eta^8(z) = \sum_{\substack{n>0 \\ n\equiv 1 \bmod 3}} a_8(n) e\left(\frac{nz}{3}\right) , \tag{5–2}$$

$$a_8(n) = \sum_{\substack{x>0, \\ x^2=n}} \left(\frac{x}{3}\right)\cdot x^3 + \sum_{\substack{1\le y<x, \\ x^2+xy+y^2=n}} \left(\frac{x-y}{3}\right)\cdot (x-y)(2n+3xy) \tag{5–3}$$

In Section 4, we introduced coefficients $e_8(n)$ defined as Fourier coefficients of the series $\left(\sum_{n=0}^{\infty} e(n)q^n\right)^8$. The relation $e_8((n-1)/3) = a_8(n)$ is obvious from Equation (5–2). Thus, Equation (5–3) yields an efficient algorithm to compute a table of the coefficients $e_8(n)$.

It is well known that $\gamma_2(z) = q^{-1/3}\sum_{n=0}^{\infty} g(n)q^n$ with integers $g(n)$. We combine Equations (5–1) and (1–4) to get a recursion formula for the coefficients $g(n)$. More precisely, it is easy to see that for $n \ge 1$, we have

$$g(n) = 240\cdot\sigma_3(n) - \sum_{k=0}^{n-1} g(k)e_8(n-k). \tag{5–4}$$

Again, we then make use of standard exponentiation techniques to compute the values of $c(n)$ from the relation $j = \gamma_2^3$.

Although this method is very similar to our variant of Herrmann's algorithm, it turns out to be much slower for $N = 50000$. The run time is 202 minutes, 33 seconds. The reason is that using the power function is rather slow for the large coefficients $g(n)$ compared to the coefficients $e_8(n)$. The CPU time to compute the values $g(n)$, $n \le N+1$, is only 8 minutes, 16 seconds. Hence, 95.9% of the run time is spent computing $c(n)$ from $g(n)$.

There is a variant of this method. We observe that

$$j = \frac{E_4^3 - E_6^2 + E_6^2}{\Delta} = 12^3 + \frac{E_6^2}{\Delta} = 12^3 + \gamma_3^2 \tag{5–5}$$

with

$$\gamma_3 = \frac{E_6}{\eta^{12}}. \tag{5–6}$$

There is no representation of η^{12} as a theta series with Hecke character. But we have ([Köhler 88, page 88])

$$\eta^6(z) = \frac{1}{4}\cdot\sum_{\mu\in\mathbb{Z}[\sqrt{-1}]} \chi(\mu)\mu^2 e\left(\frac{1}{4}\mu\overline{\mu}z\right), \tag{5–7}$$

where $\chi(x+y\sqrt{-1}) = (-1)^y$ if $x \not\equiv y \bmod 2$ and $\chi(x+y\sqrt{-1}) = 0$ otherwise. Thus, we can tabulate η^6 as efficiently as η^8. Squaring η^6 yields η^{12}, a division gives γ_3, and squaring again gives the coefficients $c(n)$ of $j(z)$. In practice we observe that this variant is far more efficient than the first one via γ_2. The total CPU time is 95 minutes, 53 seconds.

6. A FORMULA OF KANEKO AND ZAGIER

In this section, we describe a method which was discovered by D. Zagier [Zagier 96] and M. Kaneko [Kaneko 99]. The main Formula (6–4) makes use of coefficients $t(n)$ introduced by Zagier. Once the $t(n)$ are known, Equation (6–4) promises to be highly efficient since it requires just additions, but essentially no multiplications. Indeed we will see that this method turns out to be the most efficient one to compute the coefficients $c(n)$.

Zagier defined the sequence of numbers $t(n)$ using certain singular values of $j(z)$. We call the numbers $t(n)$ *Zagier coefficients*. Zagier gave an equivalent definition of the $t(n)$ by means of the Fourier expansion of a meromorphic modular form of weight $\frac{3}{2}$, namely,

$$g(z) = -\frac{E_4(4z)\theta_1(z)}{\eta(4z)^6} = \sum_d t(d)q^d , \tag{6–1}$$

where $\theta_1(z) = \sum_{n=-\infty}^{\infty}(-1)^n q^{n^2}$ is one of Jacobi's theta series. We have

$$t(-1) = -1, \qquad t(0) = 2, \qquad t(d) = 0 \tag{6–2}$$

if $d < -1$ or $d \equiv 1, 2 \bmod 4$.

Zagier proved the recursion formulas

$$\sum_{r\in\mathbb{Z}} r^2 t(4n-r^2) = -480\sigma_3(n), \qquad \sum_{r\in\mathbb{Z}} t(4n-r^2) = 0 \tag{6–3}$$

for $n \ge 1$. It is obvious that the relations of Equation (6–3) uniquely determine the values $t(3)$, $t(4)$, $t(7)$, ... Using this, Kaneko proved

$$c(n) = \frac{1}{n}\left(\sum_{r\in\mathbb{Z}} t(n-r^2) + \sum_{r>0,\ r\,\text{odd}} \left((-1)^n t(4n-r^2) - t(16n-r^2)\right)\right) \tag{6–4}$$

for $n \ge 1$. When we use this formula to compute a table of $c(n)$ for $n \le N$, we need to compute a table of the Zagier coefficients $t(d)$ for $d \le 16N$. As $t(d) = 0$ for

$$
\begin{aligned}
t(800000) = {} & 216426099970105280458598122748826247119874847295 2\backslash \\
& 973369697514365708060103573179061817399052614353 3\backslash \\
& 365651228274236563611995661746319645905061382458 7\backslash \\
& 675044029577699879047042389879299295760190404920 5\backslash \\
& 297340399654675595987752778433901428877523674883 5\backslash \\
& 286347601663414649424960960240291730809757225478 8\backslash \\
& 797331964148573451431850042966834851852581282402 6\backslash \\
& 007489073102041683099836477809939029102739464118 3\backslash \\
& 243053338700634900532403034699904708361713785198 8\backslash \\
& 121016060027178879161021020721499119883262077914 2\backslash \\
& 516151050390250082071772312528346867905976978563 8\backslash \\
& 362544102646927709061971035321629714205855184500 1\backslash \\
& 253635205951482939413997733047289870416933528269 9\backslash \\
& 265558685408731999891078383045282826561117015656 6\backslash \\
& 461037867284807840636551450383754964376961396740 3\backslash \\
& 648651095344497417251042707752112920429418098086 7\backslash \\
& 999232325006113667235308836622260443478219319028 1\backslash \\
& 255099429274454819330136583310650785056595254228 8\backslash \\
& 994696571727561832935341471978408911437171454611 8\backslash \\
& 164082107740627551894991011281297940604677303714 1\backslash \\
& 590798765306188782515004204799655695415960940559 6\backslash \\
& 417805454475974509514145272497733840598476055733 9\backslash \\
& 499367651468883978300909005850222654781788210978 1\backslash \\
& 574503670911215656586655024089047510079114529798 2\backslash \\
& 528846788862573350090531733289276113660937680
\end{aligned}
$$

FIGURE 1. A Sample Zagier Coefficient.

$d \equiv 1, 2 \bmod 4$, this is essentially a table of length $8N$. Once the Zagier coefficients are known, we just have to do additions to compute the $c(n)$ using Equation (6–4).

We explain how to recursively compute the Zagier coefficients. It is obvious from the formulas in Equation (6–3) that if some $n \geq 1$ is given, we get the following recursion:

$$
\begin{aligned}
t(4n-1) &= -240\sigma_3(n) - \sum_{r=2}^{\sqrt{4n+1}} r^2 t(4n-r^2)\ , \\
t(4n) &= -2 \sum_{r=1}^{\sqrt{4n+1}} t(4n-r^2)\ .
\end{aligned}
$$

Most of the CPU time is spent to compute the values $t(d)$. If $N = 50000$, their computation takes us 8 minutes, 19 seconds. The run time of the whole computation of the $c(n)$, $n \leq N$, is 8 minutes, 43 seconds. Thus, 95% of the CPU time is spent computing the table of the Zagier coefficients.

In Figure 1, we list the coefficient $t(800000)$. We choose this coefficient, as for $N = 50000$, we have to compute the values $t(d)$ up to $d = 799999$. We remark that $t(800000)$ is an integer of bitlength 4056.

Finally, we remark that if N is of order of magnitude 50000, this method assumes that a large quantity of main memory is to our disposal, say more than 500 MByte. For example, we terminated this algorithm on a PC having about 100 MByte of main memory after 15 minutes. At this point, the CPU usage of our process was less than 5%, while the swapping process took almost all of the time.

REFERENCES

[Baier 02] H. Baier. "Efficient Algorithms for Generating Elliptic Curves over Finite Fields Suitable for Use in Cryptography." PhD thesis, Darmstadt University of Technology, 2002.

[Gouvêa 97] F.Q. Gouvêa. "Non-Ordinary Primes: A Story." *Exp. Math.* 6:3 (1997), 195–205.

[Herrmann 73] O. Herrmann. "Über die Berechnung der Fourierkoeffizienten der Funktion $j(\tau)$." *J. f. d. reine u. angew. Math.* 274 (1973), 187–195.

[Kaneko 99] M. Kaneko. *Traces of Singular Moduli and the Fourier Coefficients of the Elliptic Modular Function* $j(\tau)$. Volume 19 of Number Theory. Fifth Conf. Canad. Number Theory Assoc., Ottawa, Ontario, Canada, Aug. 1996. AMS, CRM Proc. Lect. Notes, 1999.

[Köhler 88] G. Köhler. "Theta Series on the Hecke Groups $G(\sqrt{2})$ and $G(\sqrt{3})$." *Math. Z.* 197:1 (1988), 69–96.

[Knopp 90] M. Knopp. "Rademacher on $J(\tau)$, Poincaré Series of Nonpositive Weights and the Eichler Cohomology." *Notices Amer. Math. Soc.* 37 (1990), 385–393.

[Mahler 76] K. Mahler. "On a Class of Non-Linear Functional Equations Connected with Modular Functions." *Journal of the Australian Mathematical Society* 22: Series A (1976), 65–120.

[Niebur 75] D. Niebur. "A Formula for Ramanujan's τ-Function." *Ill. J. Math.* 19 (1975), 448–449.

[Rademacher 38] H. Rademacher. "The Fourier Coefficients of the Modular Invariant $J(\tau)$." *Amer. J. Math.* 60 (1938), 501–512.

[Ramanujan 27] S. Ramanujan. *Collected Papers.* Cambridge, UK: Cambridge University Press, 1927. Reprinted New York, 1962.

[Schoeneberg 53] B. Schoeneberg. "Über den Zusammenhang der Eisensteinschen Reihen und Thetareihen mit der Diskriminante der elliptischen Funktionen." *Math. Ann.* 126 (1953), 177–184.

[Serre 85] J. P. Serre. "Sur la lacunarite des puissances de η." *Glasg. Math. J.* 27 (1985), 203–221.

[Zagier 96] D. Zagier. "Traces of Singular Moduli." preprint, 1996. Preprint.

Harald Baier, Darmstadt Center of IT-Security, Darmstadt University of Technology, D-64283 Darmstadt, Germany (hbaier@dzi.tu-darmstadt.de)

Günter Köhler, Department of Mathematics, University of Würzburg, D-97074 Würzburg, Germany (koehler@mathematik.uni-wuerzburg.de)

Received July 17, 2002; accepted in revised form May 21, 2003.

Product Replacement in the Monster

Petra E. Holmes, Stephen A. Linton, and Scott H. Murray

CONTENTS

We show that the product replacement algorithm can be used to produce random elements of the Monster group. These random elements are shown to have the same distribution of element orders as uniformly distributed random elements after a small number of steps.

1. INTRODUCTION

Computing in finite groups often requires a supply of random elements. There are several known methods for producing them. The best known practical method is the product replacement algorithm which is given in Section 2.

The Monster is the largest of the 26 sporadic simple groups. It has order

808 017 424 794 512 875 886 459 904 961 710 757 005 754 368 000 000 000

its minimal faithful permutation and matrix representation degrees are respectively 97 239 461 142 009 186 000 and 196 882. This makes it far harder to work with than the 25 smaller sporadics.

In Section 3, we describe techniques which allow limited computation in the Monster. In Section 4, we describe our experiments to assess the effectiveness of combining these techniques with a suitable version of the product replacement algorithm, and in Section 5, we present our results and conclusions.

2. THE PRODUCT REPLACEMENT ALGORITHM

Let G be a finite group generated by the set X. The product replacement algorithm uses an array $s = (s_1, \ldots, s_m)$ of elements of G satisfying the property that $\langle s_1, \ldots, s_m \rangle = G$. We require that m be larger than the size of X. Initially we take the entries in s to be the elements of X, with repetitions to fill out the array. Then at each stage of the algorithm, we choose distinct random integers i and j between 1 and m; and then replace s_i by the product $s_i s_j$. We then return the new value of s_i as

2000 AMS Subject Classification: Primary 20-04, 20D08
Keywords: Monster group, randomised algorithms

1058-6458/2001 $0.50 per page

our random element. It is known from [Celler et al. 95] that the random elements returned converge in the long term to a fixed distribution. While this distribution is often close to uniform, [Pak 01] has shown that in certain cases, it can be very far from uniform.

A more recent variant of this algorithm [Leedham-Green and Murray 02] does converge to the uniform distribution. In this variant, we have an extra group element s_0, called the *accumulator*. Then at each step we do product replacement as above, but in addition we choose a third random integer k and multiply s_0 by s_k. Then s_0 is returned as our random element.

We have run 100 tests of the product replacement algorithm (with and without an accumulator) with different random seeds. We assess the randomness of the elements produced using the χ^2 statistic at the 0.9 probability level, applied to a test value derived from the orders of the random elements. We consider that the algorithm has converged at step t if, for at least nine out of the subsequent ten steps, the χ^2 value is below the 0.9 level, as would be expected from genuinely random elements. These methods for testing product replacement are based on [Babai et al. 01, Celler et al. 95].

3. COMPUTING IN THE MONSTER

Computing in the Monster is very different to computing in smaller groups. In most of the groups in which we work, generating elements can be stored as permutations or matrices. In the Monster we must take a different approach, as seen in the three constructions of [Linton et al. 98], [Holmes and Wilson], and [Wilson 00]. The construction of [Linton et al. 98], which uses 3-local subgroups and linear algebra in characteristic 2, gives us the fastest way of computing in $\mathbb{M}$ and so we choose it for our computations.

All three of the constructions use three generators, two of which generate a local subgroup. In the construction of [Linton et al. 98], two of the generators, C and D, generate a subgroup isomorphic to $3^{1+12}{\cdot}2\mathrm{Suz}{:}2$, the normaliser of an element in class $3B$. The third generator, T, normalizes a subgroup of $\langle C, D\rangle$ isomorphic to $3^{2+5+10}{:}(\mathrm{M}_{11} \times 2^2)$, the centraliser of two commuting elements of class $3B$. The element T extends this group to $3^{2+5+10}{:}(\mathrm{M}_{11} \times \mathrm{D}_8)$. The dimension 196882 module for $\mathbb{M}$ over $GF(2)$ restricts to $3^{1+12}{\cdot}2\mathrm{Suz}{:}2$ with shape

$$142 \oplus 32760 \oplus (3^6 \otimes 90),$$

where all dimensions are over $GF(4)$ except 142. So C and D are represented as files each containing four matrices, one for each piece of the representation. Similarly, the module structure for $3^{2+5+10}{:}(\mathrm{M}_{11} \times \mathrm{D}_8)$ allows T to be stored as a collection of small matrices.

There are two main programs: one which calculates the image of any vector in the 196882-dimensional space under any of the generators, and one which can multiply together the elements in the local subgroup and return the product in the same format as the generators. In this paper, we use the vector-image program to calculate orders of words in the generators.

It takes approximately 60 ms for each occurrence of T in a word to multiply a vector by a word using a Pentium II/450MHz processor with 384 MB of RAM. This operation is about 100 times faster than when using the construction of [Holmes and Wilson], although that construction is the more frequently used as it gives easy access to 2-local subgroups and comes equipped with a method for shortening words [Holmes 02].

4. PRODUCT REPLACEMENT IN THE MONSTER

We performed 100 independent incarnations of the product replacement algorithm, both with and without an accumulator, running them for 20 steps with an accumulator and 25 without.

To assess the uniformity of the distribution of random elements that we obtained using the χ^2 test, we need some easily calculated property of these elements which will take a reasonably small number of values and whose distribution in the whole Monster is known. The χ^2 test can then be used to compare the distribution of the values which appear in our sample with the true distribution of the value. In the Monster, the only suitable property which it appears feasible to calculate is element order, which has also been used in studying product replacement in other groups [Celler et al. 95]. To meet the requirements of the χ^2 test, that each outcome have expected frequency at least 1 for the sample size being used, we group the orders $\{1, 2, \ldots, 23, 25, 28, 33, 34, 44, 45, 55, 105, 110\}$ together to form a single test value.

In fact, we do not strictly use element order, but instead use the order of the action of the elements on a fixed test vector. It is highly likely, but not proven, that this vector lies in a faithful orbit of $\mathbb{M}$. Otherwise, there will be a very small chance of obtaining a divisor of the correct order. Since this would make the orders look less, rather than more, random, it does not invalidate our conclusions.

The product replacement algorithm was actually performed in the free group on three generators using an

Number of steps	χ^2 with accumulator	χ^2 no accumulator
1	670.051	419.923
2	390.119	367.669
3	249.562	300.125
4	163.061	252.426
5	142.792	247.926
6	62.536	274.574
7	65.525	223.413
8	**36.882**	94.410
9	50.313	131.215
10	37.671	56.936
11	37.680	76.733
12	35.448	**52.527**
13	57.899	35.812
14	48.167	33.932
15	37.387	38.222
16	34.160	44.395
17	37.068	43.748
18	47.394	49.358
19	30.610	40.715
20	40.744	38.068
21		45.162
22		36.205
23		38.297
24		46.629
25		43.727

TABLE 1. χ^2 values. The entries in bold show where convergence has occurred.

array of length 4, resulting, for each random seed, in two sequences of words (one obtained using the accumulator, and the other not). This part of the calculation and the computation of χ^2 values was done in GAP [GAP 02].

A C program, using subroutines derived from [Linton et al. 98], was then used to compute the orders of these words evaluated at the three generators described in Section 3.

5. CONCLUSIONS

Table 1 gives the χ^2 values for each step. It can be seen that product replacement with an accumulator converges after 8 steps, and for product replacement, we need 12.

One use of random element generation in any group is to obtain elements of specific conjugacy classes or orders, for use in subsequent calculations, such as finding standard generators [Wilson 96]. To assess the suitability of product replacement in the Monster for this purpose, we tabulate in Table 2 the number of steps of product replacement with an accumulator that we have to perform before we see elements of multiples of each possible order amongst our hundred runs. We are only interested in multiples of element orders because we can get the element of the required order by powering up. It is clear

Number of steps	Element orders
1	1, 2, 3, 4, 5, 6, 7, 8, 10, 12, 14, 15, 20, 23, 28, 30, 56, 60
2	9, 13, 31, 39, 41, 45, 46
3	11, 17, 19, 21, 24, 34, 35, 42, 55, 57, 68, 70, 84, 119
4	16, 22, 25, 26, 32, 33, 40, 47, 50, 52, 66, 78, 95, 104
5	18, 27, 36, 48, 51, 54, 62, 69, 92
6	29, 59, 71, 87, 93, 94
7	38, 44
8	88
9	105, 110

TABLE 2. Element orders occurring in the output of product replacement.

from the table that short words are not suitable for generating random elements as there are some orders of elements which we were unable to find without using many steps. To assess the cost of computing with random elements produced in this way, we graph in Figure 1 the length of the words generated against the number of steps of product replacement. Applying a word of length n to

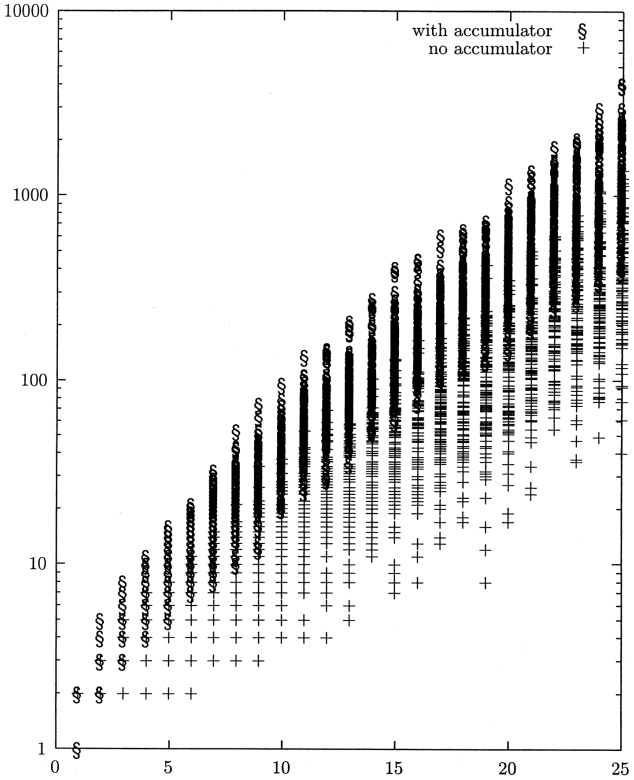

FIGURE 1. Word lengths for product replacement in the free group.

a single vector takes about 60 $n$ms on a Pentium II/450 processor.

We conclude that, while it is not practical to use a single product replacement calculation to produce a series of random elements of the Monster as is done in other groups, since the words involved would rapidly grow too long, it would appear that, at least as far as element orders are concerned, starting a fresh product replacement algorithm and running it for between 8 and 15 steps (depending on the quality of randomness required) is both a feasible and an effective way to generate pseudorandom elements of the Monster.

Note that it is easy to see that the elements obtained in this way must be far from uniformly distributed on $\mathbb{M}$. By counting the possible random choices, we see that n step product replacement with (without) accumulator can produce at most $(4 * 3 * 4)^n$ $((4 * 3)^n)$ distinct outcomes, and this reaches the order of the Monster only for $n \geq 32$ $(n \geq 49)$. What our experiments show is that product replacement approaches the same distribution of element orders as the uniform distribution of elements much sooner than this.

ACKNOWLEDGMENTS

The first author is supported by EPSRC grant GR/R95265/01. The computations described in this paper were performed on equipment provided by EPSRC grant GR/M32351/01.

REFERENCES

[Babai et al. 01] László Babai, Walter Kim, Scott H. Murray, and Rebecca Vessenes. "Quality of Random Elements in Finite Groups." Unpublished.

[Celler et al. 95] Frank Celler, Charles R. Leedham-Green, Scott H. Murray, Alice C. Niemeyer, and E. A. O'Brien. "Generating Random Elements of a Finite Group." *Comm. Algebra* 23 (1995), 4931–4948.

[Holmes 02] P. E. Holmes. "Computing in the Monster." PhD diss., Birmingham, 2002.

[Holmes and Wilson] P. E. Holmes and R. A. Wilson. "A New Computer Construction of the Monster Using 2-Local Subgroups." *J. LMS.* To appear.

[Leedham-Green and Murray 02] C. R. Leedham-Green and Scott H. Murray. "Variants of Product Replacement." In *Computational and Statistical Group Theory*, pp. 97–104, Contemporary Mathematics 298, Providence, RI: American Mathematical Society, 2002.

[Linton et al. 98] S. A. Linton, R. A. Parker, P. G. Walsh, and R. A. Wilson. "Computer Construction of the Monster." *J. Group Theory* 1 (1998), 307–337.

[Pak 01] Igor Pak. "What Do We Know about the Product Replacement Algorithm?" In *Groups and Computation, III (Columbus, OH, 1999)*, pp. 301–347. Berlin: de Gruyter, 2001.

[GAP 02] The GAP Group. *GAP–Groups, Algorithms, and Programming*, Version 4.3. Available from World Wide Web (http://www.gap-system.org), 2002.

[Wilson 96] R. A. Wilson. "Standard Generators for Sporadic Simple Groups." *J. Algebra* 184 (1996), 505–515.

[Wilson 00] R. A. Wilson. "A Construction of the Monster Group over GF(7), and an Application." Preprint, 2000.

Petra E. Holmes, School of Mathematics and Statistics, University of Birmingham, Edgbaston, Birmingham B15 2TT, United Kingdom (P.E.Holmes@bham.ac.uk)

Stephen A. Linton, School of Computer Science, University of St. Andrews, North Haugh, St. Andrews, Fife, KY16 9SS, United Kingdom (sal@dcs.st-and.ac.uk)

Scott H. Murray, Faculteit Wiskunde en Informatica, Technische Universiteit Eindhoven, Postbus 513, 5600 MB Eindhoven, The Netherlands (smurray@win.tue.nl)

Received June 12, 2002; accepted March 20, 2003.

Instructions for Authors

Experimental Mathematics is devoted to experimental aspects of mathematical research. It publishes results inspired by experimentation, conjectures suggested by experiments, surveys of certain areas from the experimental point of view, descriptions of algorithms and software for mathematical exploration, and general articles of interest to the community. A more detailed statement of philosophy and of the publishability criteria is available on the Web at http://www.expmath.org, or by request from the publisher (see address below, or send e-mail to editorial@expmath.org).

How to Submit an Article

To submit a contribution, you may either send e-mail to editorial@expmath.org with an attachment (pdf file) or address from which the paper (pdf file) can be downloaded, or send four printed copies of the material to

Experimental Mathematics
A K Peters, Ltd.
63 South Avenue
Natick, MA 01760-4626
phone: 508-655-9933
fax: 508-655-5847

In either case, you must include a note stating that the paper is intended for publication in *Experimental Mathematics* and contact information for each author, consisting of (at least) full name, postal address, electronic address and phone number.

Conditions of Submission

By submiting a paper, authors agree and confirm that: substantially the same work has not been published elsewhere (in a journal or proceedings, though it may have appeared in the form of an abstract or as part of a lecture, review, or thesis); substantially the same work is not under consideration for publication elsewhere; if and when the manuscript is accepted for publication, substantially the same work will not be published elsewhere, except that each author retains the right of republication in any book of which he/she is the author or editor; publication has been approved by all authors and, if required, by the institution where the work was carried out.

Submissions will be acknowledged, but not returned.

Charges

There are no page charges for publications, but authors are expected to contribute toward the cost of color illustrations in their articles. Rates will take into account funding available to authors and editorial necessity.

Offprints

Authors will receive 25 free offprints of their work. At production time authors may order up to 75 additional offprints at cost.

Manuscript Requirements

Manuscripts must be written clearly and concisely. We reserve the right to edit contributions for style and format, with changes subject to the authors' approval.

All submissions must include the following elements:

1. title and (if title exceeds 75 characters) alternative short title for running heads;

2. postal address, affiliation (if appropriate) and electronic address (if available) for each author;

3. an abstract of at most 150 words, in the same language as the article, and an English translation if the article is not in English.

References

References should include full information: author or institution; full title; publisher, city and year (for books, manuals, etc.), or full journal name, volume, year and page range (for papers). References to software should contain complete manufacturer's or distributor's names and addresses. All references in the bibliography should be cited in the text, or accompanied by comments stating their relevance. Reference tags in the text should include author's last name and year of publication, in brackets [Poincaré 1901]. Use a comma to separate a tag from a subsequent page or section number, and semicolons to separate several tags in the same brackets.

1058-6458/2001 $0.50 per page

Code and Tables

Experimental Mathematics does not publish computer programs in printed form. You can include short illustrative excerpts from your programs, either within the text itself (if at most three lines) or as a separate display. Please supply a caption and a number for each displayed listing. Keep in mind that many readers will not be familiar with the programming language in which your program is written; it is almost always better to explain what a program does in words than to let the program speak for itself.

Similar considerations apply to program output and interactive sessions.

Tables should be kept to a minimum, and generally serve an illustrative rather than archival purpose. Very short tables can be embedded in the text; all others should be able to float and have a caption.

Figures

All figures should be available in electronic form. For electronically-generated figures, you can use photographs or printouts for hard-copy submission, but you must supply the electronic source files if your article is accepted for publication. Under no circumstances will we reproduce printouts, low-resolution scans, or screen photographs.

Figure source files should be in Encapsulated PostScript (EPS). All art files must be supplied in either grayscale, or in CMYK (if color will be included in your article); RGB files should not be used. Halftone images should have a resolution of 300 dpi for best printing quality. Line art (black and white with no grays) should ideally have a resolution of 1200 dpi, but definitely no less than 800 dpi. Please be sure to check and make sure that your line art is truly black and white and not RGB in disguise.

When in doubt whether your figure source is in an acceptable format, check with the editors by sending electronic mail to editorial@expmath.org.

For each figure, please supply a caption and a number by which the figure is referred to in the text. If possible, integrate the figures with the text; otherwise, indicate their optimal placement by means of a comment such as "Place Figure 1 here." In referring to the figure, avoid constructions ("the curve looks like this:") that require the exact placement to be known in advance.

See also the section on Charges on the preceeding page.

Electronic Text

If your article is accepted, we request that you submit the text in electronic form. You can transfer it by e-mail, ftp, or diskette. Send e-mail to editorial@expmath.org for details. We also require that you send us a printed copy of the final version of your article.

Experimental Mathematics is typeset in LaTeX. If you have prepared your manuscript in a form other than LaTeX or TeX, please save your files as text-only or ASCII.

- If possible, please use LaTeX article style.
- Do not redefine existing LaTeX commands.
- Do not embed any new definitions in your text.
- Avoid using explicit vertical spacing commands such as `\vskip`, `\medskip`, `\bigbreak`.
- All user-defined macros must be placed in a separate file which is input at the top of the document.
- Avoid using specialized style files which may work only on your installation—if you use other style files, they must be submitted with your article
- Avoid using explicit horizontal spacing commands. If you must use extra spacing, do it consistently, by means of macros.
- Do not, under any circumstances, insert forced line breaks or page breaks in your document.
- Use double-column format if possible; otherwise single column is acceptable. Since Experimental Mathematics is set in double-column format, your preparation of the electronic files in this way will ensure that your mathematical formulas are broken correctly. It will also help in the sizing of your illustrations.