Time- and Space-Efficient Evaluation of Some Hypergeometric Constants

Howard Cheng, Guillaume Hanrot, Emmanuel Thomé, Eugene Zima, Paul Zimmermann

Time- and Space-Efficient Evaluation of Some Hypergeometric Constants

Howard Cheng
Dept. of Mathematics and Computer Science, Univ. of Lethbridge
Lethbridge, Alberta, Canada
howard.cheng@uleth.ca

Guillaume Hanrot
INRIA Lorraine/LORIA
Villers-lès-Nancy, France
hanrot@loria.fr

Emmanuel Thomé
INRIA Lorraine/LORIA
Villers-lès-Nancy, France
Emmanuel.Thome@normalesup.org

Eugene Zima
Wilfrid Laurier University
Waterloo, Ontario, Canada
ezima@wlu.ca

Paul Zimmermann
INRIA Lorraine/LORIA
Villers-lès-Nancy, France
zimmerma@loria.fr

ABSTRACT

The currently best known algorithms for the numerical evaluation of hypergeometric constants such as $\zeta(3)$ to d decimal digits have time complexity $O(M(d)\log^2 d)$ and space complexity of $O(d \log d)$ or $O(d)$. Following work from Cheng, Gergel, Kim and Zima, we present a new algorithm with the same asymptotic complexity, but more efficient in practice. Our implementation of this algorithm improves over existing programs for the computation of π, and we announce a new record of 2 billion digits for $\zeta(3)$.

Categories and Subject Descriptors

I.1.2 [**Computing methodologies**]: Algorithms—*Symbolic and algebraic manipulation*

General Terms

Algorithms, Performance

Keywords

Hypergeometric constants, high-precision evaluation

1. INTRODUCTION

In this article, we are interested in the high-precision evaluation of constants defined by hypergeometric series of the form

$$\sum_{n=0}^{\infty} a(n) \prod_{i=0}^{n-1} \frac{p(i)}{q(i)}, \qquad (1)$$

where a, p and q are polynomials with integer coefficients. We shall also assume, without loss of generality, that p and q are coprime, have no nonnegative integer as a zero and that $p(n)/q(n)$ tends to a constant $0 < c < 1$ when n goes to infinity.

Under those assumptions, the series converges, and we can compute an approximation to the constant by truncating the series, i.e., by computing

$$\sum_{n=0}^{N-1} a(n) \prod_{i=0}^{n-1} \frac{p(i)}{q(i)} \qquad (2)$$

for an appropriately chosen $N = \Theta(d)$ with d being the number of decimal digits desired. The high-precision evaluation of elementary functions and other constants — including the exponential function, logarithms, trigonometric functions, and constants such as π or Apéry's constant $\zeta(3)$ — is commonly carried out by evaluating such series [11, 13]. For example, we have

$$\frac{1}{\pi} = 12 \sum_{n=0}^{\infty} (-1)^n \frac{545140134n + 13591409}{640320^{3n+3/2}} \frac{(6n)!}{(3n)!n!^3} \qquad (3)$$

or

$$2\zeta(3) = \sum_{n=0}^{\infty} (-1)^n (205n^2 + 250n + 77) \frac{(n+1)!^5 n!^5}{(2n+2)!^5}. \qquad (4)$$

Assuming that $q(n)$ has size $O(\log n)$, the special form of the series (2) implies that the common denominator

$$\prod_{i=0}^{N-2} q(i)$$

has a relatively small size of $O(N \log N)$. An approach to such computations commonly known as "binary splitting" has been independently discovered and subsequently rediscovered by many authors [4, 2, 10, 14, 13]. More general setup of multiprecision evaluation of holonomic functions was investigated in [5] and subsequent papers [20, 21, 22]. Good exposition of the method can be found in [3, 11, 1]. In binary splitting, the use of fast integer multiplication yields a total time complexity of $O(M(d \log d) \log d) = O(M(d) \log^2 d)$, where $M(t) = O(t \log t \log\log t)$ is the complexity of multiplication of two t-bit integers [17]. The space complexity $O(d \log d)$ of the algorithm is the same as the size of the computed numerator and denominator.

The numerator and denominator computed by the binary splitting approach typically have large common factors. For example, it was shown that in the computation of 640,000 digits of $\zeta(3)$, the size of the reduced numerator and denominator is only 14% of the size of the computed numerator and denominator. This suggests possible improvements of the method, by avoiding the unneeded computation of the common divisor between the numerator and denominator. Several approaches have already been taken in that direction. In particular, [7] suggests to use a partially factored form for the computed quantities, in order to efficiently identify and remove common factors, and [23] goes further by explicitly constructing the common divisor and dividing out the numerator and denominator.

The present work builds on top of this strategy and uses a fully factored form in the binary splitting process. We show that the fully factored form yields a time complexity of $O(M(d)\log^2 d)$, and space complexity $O(d)$. This matches the complexity of the standard approaches, but provides a practical speedup confirmed by experiments. Our method appears to be noticeably faster than other optimized binary splitting implementations aimed at the computation of digits of π or other constants. We also show in this article that the exact set of series that are amenable to efficient computation using the fully factored form is characterized by a simple criterion: only the series where $p(n)$ and $q(n)$ are products of linear factors exhibit the large common factor that was observed in the computation of $\zeta(3)$. Therefore our attention is restricted to that case.

This article is organized as follows. Section 2 recalls the binary splitting algorithm, and reviews the different approaches for improving the practical efficiency of the method. Section 3 examines in detail the size of the reduced fraction computed by the binary splitting algorithm. Section 4 presents the alternative of using a fully factored form in the binary splitting approach. In Section 5, the analysis of the algorithm is performed. Section 6 concludes with experimental data, and a comparison with other programs.

2. THE BINARY SPLITTING APPROACH AND ITS VARIANTS

We give a brief description of the binary splitting approach here, following the notations from [7].

Our approximation to the constant to be evaluated can be written $S(0,N)$ where for $0 \le n_1 \le n_2$ we define

$$S(n_1,n_2) = \sum_{n=n_1}^{n_2-1} a(n)\frac{p(n_1)\cdots p(n-1)}{q(n_1)\cdots q(n-1)}.$$

Letting $P(n_1,n_2) = \prod_{n=n_1}^{n_2-1} p(n)$, $Q(n_1,n_2) = \prod_{n=n_1}^{n_2-1} q(n)$, and $T(n_1,n_2) = S(n_1,n_2)Q(n_1,n_2)$, we have for $n_1 < m < n_2$, with $T(n,n+1) = a(n)p(n)$:

$$\begin{aligned} P(n_1,n_2) &= P(n_1,m)P(m,n_2) \\ Q(n_1,n_2) &= Q(n_1,m)Q(m,n_2) \\ T(n_1,n_2) &= T(n_1,m)Q(m,n_2) + P(n_1,m)T(m,n_2). \end{aligned}$$

This leads to a recursive algorithm to evaluate $T(0,N)$ and $Q(0,N)$, which corresponds to the evaluation of a product tree [1]. One then deduces $S(0,N)$ by a division.

Since $p(n)/q(n)$ tends to c, the tail $S(N,\infty)$ of the series is bounded by $O(c^N)$. Therefore, to compute the constant $S(0,\infty)$ with error ϵ, we need $N = \frac{\log \epsilon}{\log c} + O(1)$ terms: the number of terms is proportional to the number d of digits of accuracy desired. The corresponding product tree has height $\log N$, where the leaves have $O(\log N)$ bits and hence the root has $O(N\log N)$ bits. The total evaluation of the truncated series costs $O(M(d\log d)\log d) = O(M(d)\log^2 d)$ with the best-known multiplication algorithms.

Although some constants such as π and $\log 2$ can be computed to d digits with bit complexity of $O(M(d)\log d)$ using the Arithmetic Geometric Mean [2], the $O(M(d)\log^2 d)$ binary splitting algorithm is still competitive up to billions of digits. For example, D. V. and G. V. Chudnovsky held the π record using Formula (3) with 8 billion digits in 1996 [11].

2.1 Improvements of the binary splitting

As mentioned earlier, the binary splitting method suffers from the drawback that the fraction T/Q has size $O(d\log d)$, while an accuracy of only d digits is required. In [11], the authors circumvent this problem by limiting the precision of the intermediate results to $O(d)$ digits. This is used by the `PiFast` program [11] and results in the same time complexity as the binary splitting method but a reduced space complexity of $O(d)$. This truncation, however, implies that the exact reduced fraction is not computed, so that it is not easy to extend the computation to higher precision using results already computed. Further, the truncation only operates on the top levels of the computation tree, since below a depth of order $O(\log\log d)$, the computed integers have size $O(d)$ anyway. Below this depth, the computations performed by the `PiFast` program are expected to be exactly the same as in the classical algorithm above.

Since in the course of the computation of digits of $\zeta(3)$, T and Q have been found to share a large number of common factors, Cheng and Zima [7] worked towards efficiently removing some of these factors from the computation. For this purpose, a partially factored representation was introduced in the binary splitting process. Subsequently, Cheng, Gergel, Kim, and Zima [6] applied modular computation and rational number reconstruction to obtain the reduced fraction. If the reduced numerator and denominator have size $O(d)$, the resulting algorithm has a space complexity of $O(d)$ and the same time complexity as binary splitting. By carefully analyzing the prime divisors of the numerator and denominator of (4), it was shown in [6] that the size of the reduced fraction for $\zeta(3)$ is $O(d)$; it was noted that the analysis was in fact related to using the partially factored representation with all possible prime factors in the binary splitting process. However, it was not practical to use so many primes in the partially factored representation because it was expensive to convert from standard representation by factoring. Additionally practicality of the algorithm depends on the availability of the implementation of the asymptotically fast rational reconstruction algorithm (for example, see [19]).

We also mention the `gmp-chudnovsky` program [23], which uses the binary splitting method to compute digits of π using Formula (3). Two modifications are made to the classical method described above. First, integers $P(n_1,n_2)$ and $Q(n_1,n_2)$ are handled together with their complete factorization. This makes it possible to quickly compute the gcd of $P(n_1,m)$ and $Q(m,n_2)$ by merely comparing the factorizations. Afterwards, the gcd is divided from both numbers. The fraction T/Q is therefore reduced. It should be noted that gmp-chudnovsky still works with expanded integers P, Q, and T (albeit reduced).

The second specificity of the gmp-chudnovsky program lies in the way the leaves $p(n)$ and $q(n)$ are computed. Since the factorization of these numbers is sought, an optimized sieve table is built. Formula (3) implies that the integers to be factored are bounded by $6N$, where N is the number of computed terms. A table of $\lfloor \frac{6N}{2} \rfloor$ entries is built, with the i-th cell containing information on the smallest prime divisor of $2i+1$, its multiplicity, and the integer j such that $2j+1$ is the cofactor. Such a table can be computed very efficiently using a modified Eratosthenes' sieve. This represents a tiny part of the total computing time. Unfortunately, this sieve table is also an impediment to large scale computations, in that it has a space complexity of $O(d \log d)$.

3. SIZE OF THE REDUCED FRACTION

Cheng, Gergel, Kim and Zima showed in [6] that for Formula (4) giving $\zeta(3)$, when removing common factors between T and Q, the reduced fraction $\hat{T}/\hat{Q}$ has size $O(d)$ only. We show here that this fact happens for a large class of hypergeometric constants.

Understanding when the size of the fraction reduces to $O(d)$ is closely linked to a study of the prime divisors of the values $p(i)$ and $q(i)$. Indeed, the fraction being significantly smaller than its expected $O(d \log d)$ size means that there are large cancellations at many primes; it thus means that the primes occurring in $\prod_{i=n_1}^{n_2-1} p(i)$ and $\prod_{i=n_1}^{n_2-1} q(i)$ are mostly the same, and with the same multiplicities.

We first notice that since $p(n)/q(n)$ tends to $c > 0$ when $n \to \infty$, this implies that p and q have the same degree. For a polynomial p, we use the notation $\mathrm{lc}\,(p)$ and $\mathrm{disc}\,(p)$ to denote the leading coefficient and the discriminant of p, respectively. If p is an irreducible polynomial and ℓ a prime (or prime power) coprime to $\Delta(p) := \mathrm{lc}\,(p)\,\mathrm{disc}\,(p)$, we shall denote $\rho_\ell(p)$ the number of roots of p modulo ℓ. If $p = \prod_{i=1}^{k} p_i^{e_i}$, and ℓ is coprime to $\Delta(p) := \prod_{i=1}^{k} \Delta(p_i)$, we shall define $\rho_\ell(p) = \sum_{i=1}^{k} e_i \rho_\ell(p_i)$, which is still the number of roots of p, counted with multiplicities.

The following lemmata lead to estimates of the ℓ-valuation and the size of the quantities $Q(n_1,n_2)$, $T(n_1,n_2)$ and their common divisors when the summation range $[n_1,n_2]$ grows.

Definition 1. *Let $\mathcal{N}_{p,\ell}(n_1,n_2)$ be the number of integer roots of $p(\cdot)$ mod ℓ in $[n_1,n_2[$:*

$$\mathcal{N}_{p,\ell}(n_1,n_2) := \#\{x \in [n_1,n_2[/ p(x) = 0 \bmod \ell\}.$$

Lemma 1. *Let p be a polynomial, and ℓ a prime (or prime power) coprime to $\Delta(p)$. Then,*

$$\mathcal{N}_{p,\ell}(n_1,n_2) = \frac{\rho_\ell(p)}{\ell}(n_2 - n_1) + O(1),$$

where the implied constant in $O(1)$ depends on p only.

Proof. The roots of p modulo ℓ in the interval $[n_1,n_2[$ are exactly integers congruent to one of the $\rho_\ell(p)$ roots of p in $[0,\ell-1]$. The Lemma follows, the precise error term being at most $\rho_\ell(p) \le \deg p$. □

Definition 2. *For an integer m, let $v_\ell(m)$ be the ℓ-valuation of m, i.e., the largest integer j such that ℓ^j divides m.*

Lemma 2. *Let ℓ be a prime not dividing $\Delta(p)$. Then,*

$$v_\ell(P(n_1,n_2)) = \frac{\rho_\ell(p)}{\ell - 1}(n_2 - n_1) + O\left(\frac{\log n_2}{\log \ell}\right).$$

Proof. We shall assume without loss of generality that p is irreducible, since by our definition of $\rho_\ell(p)$, the result in the general case will follow by linearity. Generalizing Legendre's formula

$$v_\ell(n!) = \sum_{j \ge 1} \left\lfloor \frac{n}{\ell^j} \right\rfloor,$$

we find that the ℓ-valuation of $P(n_1,n_2)$ is exactly

$$v_\ell(P(n_1,n_2)) = \sum_{j \ge 1} \mathcal{N}_{p,\ell^j}(n_1,n_2).$$

Since ℓ does not divide $\Delta(p)$, Hensel's Lemma shows that $\rho_\ell(p) = \rho_{\ell^k}(p)$ for all $k \ge 1$.

Further, there exists a constant γ such that the inequality $|p(x)| \le n_2^\gamma$ holds over the interval $[n_1,n_2[$. We then have $\mathcal{N}_{p,\ell^j}(n_1,n_2) = 0$ for $j > J_p := \gamma \frac{\log n_2}{\log \ell}$, and also $\ell^{-J_p} = O(n_2^{-1})$. Lemma 1 yields for $0 \le n_1 \le n_2$:

$$v_\ell(P(n_1,n_2)) = \rho_\ell(p)(n_2 - n_1)\left(\frac{1}{\ell - 1}\right) + O\left(\frac{\log n_2}{\log \ell}\right).$$

The statement follows. □

We need to control (though in a rather rough way) what happens for primes dividing $\Delta(p)$. The following weaker lemma is sufficient; its proof is very close in spirit to that of Lemma 2.

Lemma 3. *For any prime ℓ, we have*

$$v_\ell(P(n_1,n_2)) = O(n_2 - n_1),$$

where the O-constant depends on p only.

Proof. Let $r_1,\ldots,r_k$ be the roots of p in $\overline{\mathbb{Q}}_\ell$, repeated according to their multiplicities. We have

$$v_\ell(p(x)) = v_\ell(\mathrm{lc}\,(p)) + \sum_{j=1}^{k} v_\ell(x - r_j).$$

Hence,

$$\begin{aligned} v_\ell(P(n_1,n_2)) &= (n_2 - n_1)v_\ell(\mathrm{lc}\,(p)) + \sum_{x=n_1}^{n_2-1}\sum_{j=1}^{k} v_\ell(x - r_j) \\ &= \sum_{j=1}^{k}\sum_{x=n_1}^{n_2-1} v_\ell(x - r_j) + O(n_2 - n_1). \end{aligned}$$

Now, as in the proof of Lemmata 1-2, for each j, the number of $x \in [n_1,n_2[$ such that $x - r_j = 0 \bmod \ell^i$ is $O((n_2 - n_1)/\ell^i)$. Hence,

$$\sum_{x=n_1}^{n_2-1} v_\ell(x - r_j) = O\left(\frac{n_2 - n_1}{\ell - 1}\right),$$

from which our claim follows. □

We can now state our main theorem regarding the size of the fraction T/Q in reduced form:

THEOREM 1. *For ℓ a prime not dividing $\Delta(pq)$, one has*

$$v_\ell(\gcd(T(n_1,n_2),Q(n_1,n_2)))$$

$$\geq \frac{\min(\rho_\ell(p),\rho_\ell(q))}{\ell-1}(n_2-n_1)+O\left(\frac{\log n_2}{\log \ell}\right).$$

PROOF. For $0\leq n_1<k<n_2$, put

$$\tau(n_1,k,n_2)=a(k)P(n_1,k)Q(k,n_2).$$

Then we have

$$T(n_1,n_2)=\sum_{n=n_1}^{n_2-1}\tau(n_1,k,n_2).$$

Applying Lemma 2, we see that

$$v_\ell(\tau(n_1,k,n_2))=v_\ell(a(k))+\frac{\rho_\ell(p)}{\ell-1}(k-n_1)$$

$$+\frac{\rho_\ell(q)}{\ell-1}(n_2-k)+O\left(\frac{\log n_2}{\log \ell}\right).$$

As such,

$$v_\ell(T(n_1,n_2))\geq \min_{n_1<k<n_2} v_\ell(\tau(n_1,k,n_2))$$

$$\geq \frac{\min(\rho_\ell(p),\rho_\ell(q))}{\ell-1}(n_2-n_1)+O\left(\frac{\log n_2}{\log \ell}\right).$$

Joined with Lemma 2 for $v_\ell(Q(n_1,n_2))$, this gives the desired statement. □

Note also that it is clear from the proof that this lower bound is generically sharp. Indeed, the proof is a sequence of equalities until the end. The last inequality is an equality up to the $v_\ell(a(k))$ term, which is 0 for almost all ℓ. As for the first inequality, one sees that $v_\ell(u+v)\geq v_\ell(u)+v_\ell(v)+k$ with probability $1/\ell^k$, hence with high probability the difference between the lhs and the rhs is absorbed in the error term.

COROLLARY 1. *The following holds:*

- *If $p(n)$ and $q(n)$ have only linear irreducible factors, the fraction $T(n_1,n_2)/Q(n_1,n_2)$ in reduced form has size $O(\min(n_2,(n_2-n_1)\log n_2))$.*
- *Otherwise, heuristically, as soon as $n_1=o(n_2)$, it is of size $\Theta(n_2\log n_2)$.*

PROOF. In the first case, the second part of the O-estimate is the trivial estimate for the size of $Q(n_1,n_2)$.

Since $S(n_1,n_2)=\frac{T(n_1,n_2)}{Q(n_1,n_2)}$ is a partial sum of a converging series, we have $T(n_1,n_2)=O(Q(n_1,n_2))$. Therefore this second part holds also for the size of the reduced fraction.

The fact that $T(n_1,n_2)=O(Q(n_1,n_2))$ also implies that the size of the reduced fraction is $\log\frac{Q(n_1,n_2)}{\gcd(T(n_1,n_2),Q(n_1,n_2))}+O(1)$.

Now, we have

$$\log\frac{Q(n_1,n_2)}{\gcd(T(n_1,n_2),Q(n_1,n_2))}$$

$$=\sum_{\ell \text{ prime}}[v_\ell(Q(n_1,n_2))-v_\ell(\gcd(T(n_1,n_2),Q(n_1,n_2)))]\log\ell.$$

However, since $q=\prod_{i=1}^k q_i^{e_i}$, we can discard in this sum primes larger than $C(q)n_2$ for some constant $C(q)$ such that $|q_i(x)|\leq C(q)n_2$ for all $i,x\in[n_1,n_2-1]$ (recall that q has only linear factors). Further, the finitely many primes dividing the discriminant Δ of a prime factor of pq contribute for $O(n_2-n_1)$ by Lemma 3. The logarithmic height therefore rewrites as:

$$\sum_{\substack{\ell\leq C(q)n_2,\,\ell \text{ prime}\\(\ell,\Delta(pq))=1}}[\frac{n_2-n_1}{\ell-1}[\rho_\ell(q)-\min(\rho_\ell(p),\rho_\ell(q))]\log\ell$$

$$+O(\log n_2)]+O(n_2-n_1),$$

where we have used Mertens' formula $\sum_{\ell\text{prime}\leq N}\log\ell/.\ell=\log N+O(1)$, see eg. [18].

Under our assumptions, we have $\rho_\ell(p)=\rho_\ell(q)=\deg p=\deg q$, hence this is also

$$\sum_{\substack{\ell\leq C(q)n_2,\,\ell \text{ prime}\\(\ell,\Delta(pq))=1}} O(\log n_2)+O(n_2-n_1)=O(n_2).$$

We now turn to the case where q has irreducible factors of degree greater than 1. In this situation, it is preferable to compute the size of the reduced fraction by subtracting the log of the gcd to the asymptotic value of

$$\log Q(n_1,n_2)=(n_2\log n_2-n_1\log n_1)\deg q+O(n_2-n_1)$$

$$=(n_2-n_1)\deg q\log n_2+O(n_2-n_1),$$

since

$$\begin{aligned}n_2\log n_2-n_1\log n_1 &= (n_2-n_1)\log n_2+n_1\log n_2/n_1\\ &\quad (n_2-n_1)\log n_2+O(n_2-n_1).\end{aligned}$$

Again, write

$$\log\gcd(T(n_1,n_2),Q(n_1,n_2))$$

$$=\sum_{\ell \text{ prime}} v_\ell(\gcd(T(n_1,n_2),Q(n_1,n_2)))\log\ell.$$

Heuristically, almost only primes of the order of magnitude of at most n_2-n_1 should appear both in T and in Q. Joined with the heuristic remark following Theorem 1, this means that we expect the size of the gcd to be of the order of

$$(n_2-n_1)\sum_{\ell\leq n_2-n_1,\,\ell \text{ prime}}\frac{\min(\rho_\ell(p),\rho_\ell(q))}{\ell-1}\log\ell+O(n_2).$$

Recall $\deg p=\deg q$, and let $\mathbb{K}$ be the splitting field of pq. Denote by $\mathcal{P}$ the set of primes ℓ such that $\rho_\ell(p)=\rho_\ell(q)=\deg(p)(=\deg(q))$.

The size of the gcd is

$$\leq(n_2-n_1)\sum_{\substack{\ell\leq n_2-n_1,\,\ell \text{ prime}\\ \ell\notin\mathcal{P}}}\frac{\deg q-1}{\ell-1}\log\ell$$

$$+(n_2-n_1)\sum_{\substack{\ell\leq n_2-n_1,\,\ell \text{ prime}\\ \ell\in\mathcal{P}}}\frac{\deg q}{\ell-1}\log\ell+O(n_2)$$

$$=(n_2-n_1)\log(n_2-n_1)(\deg q-1)$$

$$+(n_2-n_1)\sum_{\substack{\ell\leq n_2-n_1,\,\ell \text{ prime}\\ \ell\in\mathcal{P}}}\frac{\log\ell}{\ell-1}+O(n_2).$$

The last sum over primes evaluates to

$$\sum_{\substack{\ell \le n_2-n_1,\, \ell \text{ prime}\\ \ell\in\mathcal{P}}} \frac{\log \ell}{\ell-1} = \sum_{\substack{\ell \le n_2-n_1,\, \ell \text{ prime}\\ \ell\in\mathcal{P}}} \frac{\log \ell}{\ell} + O(1),$$

which, by the Chebotarev density theorem (see eg. [15]) — notice that the primes of $\mathcal{P}$ are exactly those for which the Artin symbol $(\ell, \mathbb{K}/\mathbb{Q})$ equals 1—, is

$$\frac{\log(n_2-n_1)}{[\mathbb{K}:\mathbb{Q}]} + O(1).$$

Thus, the size of the gcd is at most

$$(n_2-n_1)\log(n_2-n_1)\left(\deg q - 1 + \frac{1}{[\mathbb{K}:\mathbb{Q}]}\right) + O(n_2).$$

Hence, under this heuristic, we see that the size of the reduced fraction can be $O(n_2)$ when $n_1 = o(n_2)$ only if $[\mathbb{K}:\mathbb{Q}] = 1$, which means that p and q are products of linear factors. $\square$

Remark. In the proof of Corollary 1, we show that if p and q have linear factors only, the size of the gcd is[1] $(n_2-n_1)\log(n_2-n_1)\deg q + O(n_2)$. As long as $(n_2-n_1)\log(n_2-n_1) = O(n_2)$, the size of the gcd is negligible with respect to the size of Q, which means that there is almost no compensation between numerator and denominator of the fraction. Thus, compensations start to appear in the evaluation tree only as soon as n_2-n_1 is of the order of $n_2/\log n_2$.

4. FULLY FACTORED REPRESENTATION

We extend in this section the "partially factored representation" of [7, Section 4] to a "fully factored representation" for P, Q, and T.

4.1 Factored representation of integers

We consider a set B of primes. (In practice, it will consist of all primes needed to completely factor $p(n)$ and $q(n)$ up to $n = N-1$.) A *factored representation over* B of an integer z is an expression of the form:

$$z = \prod_{p\in B} p^{\alpha_p}\cdot r,$$

where $\alpha_p \in \mathbb{N}$ and $r \in \mathbb{Z}$. The integer z is represented by the data $((\langle p, \alpha_p\rangle)_p, r)$. For efficiency purposes, the representation skips primes p such that α_p is zero. Note that we do not impose that primes in B do not divide the cofactor r, so different factored representations may correspond to the same integer, like $2^2\cdot 3\cdot 7$ and $2\cdot 3\cdot 14$ over $B = \{2,3\}$. When the cofactor r equals 1, we have the (unique) *fully factored* representation of z.

A binary splitting method using factored representations of integers can be written as in algorithm **FastEval** (see Fig. 1)[2]. We need to define the three operations FullFactor,

```
Algorithm FastEval($n_1$, $n_2$, $B$).
if ($n_1$ == $n_2 - 1$) {                /* leaf computation */
    $P$ = FullFactor($p(n_1)$, $B$);
    $Q$ = FullFactor($q(n_1)$, $B$);
    $T$ = FullFactor($p(n_1)$, $B$) · $a(n_1)$;
} else {
    $m$ = $\lfloor \frac{n_1+n_2}{2} \rceil$;
    $(P_1, Q_1, T_1)$ ← FastEval($n_1$, $m$, $B$);
    $(P_2, Q_2, T_2)$ ← FastEval($m$, $n_2$, $B$);
    $P$ = PartMult($P_1$, $P_2$);
    $Q$ = PartMult($Q_1$, $Q_2$);
    $T$ = PartAdd(PartMult($Q_2$, $T_1$), PartMult($P_1$, $T_2$));
}
```

Figure 1: Binary splitting using factored representation

PartMult and PartAdd. FullFactor computes the full factorization of a given integer over the factor base B, and is obtained by sieving (see below). Further, we define:

$$\mathtt{PartMult}(\prod_{p\in B} p^{\alpha_p}\cdot r, \prod_{p\in B} p^{\beta_p}\cdot s) = \prod_{p\in B} p^{\alpha_p+\beta_p}\cdot(rs).$$

$$\mathtt{PartAdd}(\prod_{p\in B} p^{\alpha_p}\cdot r, \prod_{p\in B} p^{\beta_p}\cdot s)$$

$$= \prod_{p\in B} p^{\gamma_p}\left(\prod_{p\in B} p^{\alpha_p-\gamma_p}\cdot r + \prod_{p\in B} p^{\beta_p-\gamma_p} s\right),$$

where $\gamma_p = \min(\alpha_p, \beta_p)$.

4.2 Leaf computations

We have $P(n, n+1) = p(n)$, $Q(n, n+1) = q(n)$, $T(n, n+1) = a(n)p(n)$. The algorithm in Figure 1 requires computing the fully factored representation of these quantities. This corresponds to the leaves of the evaluation tree.

In order to expect an improvement from the use of the fully factored representation, we must make sure that the gain is not offset by the complexity of the leaf computations. In order to perform this step efficiently, we use a standard window sieving method.

As mentioned in the introduction, we assume here that $p(n)$ and $q(n)$ are polynomials with linear factors only — like in the case of Formulae (3) or (4). Without loss of generality, we illustrate our sieving procedure with the computation of the fully factored representation of the quantity $q(n)$ from Formula (4). This is equivalent to the factorization of $2n+1$. The sieving produces simultaneously the factorization of all the consecutive odd integers in a range $[2n_1+1, 2(n_1+W)+1[$, where W is an arbitrary integer. We proceed as follows.

1. For each odd prime (or prime power) ℓ such that $3 \le \ell < 2N$, we compute the smallest value i_ℓ such that
$$2(n_1+i_\ell)+1 \equiv 0 \mod \ell.$$
2. Sieve using the procedure in Figure 2.

Besides this description, the important observation is that the *next* set of initialization values i_ℓ for the computation of $q(n_1+W), \ldots, q(n_1+2W-1)$ does not have to be computed: the code in Figure 2 has already updated these values correctly.

[1]Taking into account only primes $\le n_2-n_1$, but any compensation occurring for significantly larger primes is bound to be coincidental.
[2]In algorithm **FastEval**, we do not factor the $a(n_1)$ term from T, which will not in general share (additional) common primes with the denominator Q.

```
Algorithm Sieve($n_1$, $n_1 + W - 1$)
factored_representation $\tau[W]$
for all primes $\ell < 2N$
  $i$ = $i_\ell$
  while ($i < W$) { include $\ell$ in $\tau$[i] ; $i$ = $i + \ell$ }
  $i_\ell$ = $i - W$
  for all powers $\ell^k$ of $\ell$, with $\ell^k < 2N$
    $i$ = $i_{\ell^k}$
    while ($i < W$) { increase by one the multiplicity of
      $\ell$ in $\tau$[i]; $i$ = $i + \ell^k$ }
    $i_{\ell^k}$ = $i - W$
```

Figure 2: Pseudo-code for sieving

The translation of this scheme to other factorizations than that of $2n + 1$, as long as we stick to linear polynomials, is straightforward.

5. ANALYSIS OF THE ALGORITHM

5.1 Cost of sieving

Because $p(n), q(n)$ are assumed to have linear factors only, their prime divisors are bounded by cn for some constant c, thus all $p(n), q(n)$ for $n \leq N$ can be completely factored over a set of $O(N/\log N)$ primes or prime powers. Since the initialization of the sieve only has to be done once, the computation of the i_ℓ values is trivial. In total, the sieving code in Figure 2 performs $O(N/\ell)$ sieve updates for each prime (or prime power) ℓ. The number of times the sieve procedure is called is $O(N/W)$, and each time all of the $O(N/\log N)$ primes or prime powers are scanned. This yields a time complexity for sieving which is $O(N \log\log N + N^2/(W \log N))$. The space complexity for sieving is at most $O(W \log N)$. There is some freedom in the choice of W, but it must clearly be between $O(N/(\log N)^4)$ and $O(N/\log N)$, so that the time and space complexities remain below $O(d \log^3 d)$ and $O(d)$, respectively.

5.2 The recursion

Since the factor base B consists of all possible prime divisors of $p(n)$ or $q(n)$, P and Q are always fully factored. Computing the product $P(n_1, n_2) = P(n_1, m)P(m, n_2)$ thus just consists in adding the prime exponents in the lists of factors. If the factored representations of $P(n_1, m)$ and $P(m, n_2)$ have respectively l_1 and l_2 elements, this can be done in $O(l_1 + l_2)$ operations.

At level k — the leaves corresponding to level 0 — the values of P or Q are bounded by $O((N^{\deg p})^{2^k})$, thus have $O(2^k \log N)$ bits. On the other hand, let l be the number of different prime factors in the representation of P (counted with multiplicities), then $P \geq 2^l$, thus P has $\Omega(l)$ bits. It follows that at level k, we have $l = O(2^k \log N)$. (We also have $l = O(\frac{N}{\log N})$ since there are that number of primes up to B.)

The total cost of computing P and Q is thus bounded by $\sum_{k=0}^{\log N} \frac{N}{2^k}(2^k \log N) = O(d \log^2 d)$.

As concerns T, if we could prove that its non-factored part is always $\log N$ times smaller than the factored part, we would get a complexity of $O(M(d) \log d)$ for T. Indeed, at level k, the non-factored part of T would have $O(2^k)$ bits, thus the cost of computing it would be $O(M(2^k))$, since it is obtained from a sum of products $T(n_1, m)Q(m, n_2) + P(n_1, m)T(m, n_2)$, where no cancellation occurs. Thus the total cost would be $\sum_{k=0}^{\log N} \frac{N}{2^k} O(M(2^k)) = O(M(d) \log d)$ for the non-factored part. (The analysis for the factored part is similar to that for P and Q.)

Unfortunately, the above property — the non-factored part of T is $\log N$ times smaller than its factored part — is only true near the root of the product tree, where common factors cancel between P and Q. Thus the computation of T costs $O(M(d) \log^2 d)$ (unless we can do better).

Opteron, 2.4Ghz

digits	our code	gmp-chudnovsky	ratio
2^{25}	60s	61s	0.98
2^{26}	136s	147s	0.93
2^{27}	322s	352s	0.91
2^{28}	768s	853s	0.90
2^{29}	1868s	2059s	0.91
2^{30}	4654s	5328s	0.87

Pentium 4, 3Ghz

digits	our code	PiFast	ratio
28×10^6	127s	135s	0.94
40×10^6	192s	206s	0.93
57×10^6	291s	323s	0.90

Table 1: Comparison with the gmp-chudnovsky and PiFast programs for computing digits of π.

6. EXPERIMENTAL RESULTS

In this section we investigate the benefit of using the fully factored representation for the purpose of computing the sum of series such as (3) or (4).

Compared to the sieve table of the gmp-chudnovsky program mentioned in Section 2, we address the problem of the leaf computation in a different way. The sieving procedure described in Section 4 keeps a space complexity of the order of $O(d)$. We found that our sieving procedure was competitive with the sieve table from the gmp-chudnovsky program.

The fully factored representation is an asset as soon as compensations between prime factors start to appear. However, this is not encountered at the very lowest levels of the computation tree, near the leaves. Quite naturally, a cut-off level appears between the use of the factored representation and the use of the fully expanded integer values. At the lowest levels of the tree, we use the same approach as the gmp-chudnovsky program. For P and Q, both the expanded integer and its factorization are kept. The integer component is dropped above a certain height in the tree. The remark concluding Section 5 suggests that the switch from one algorithm to the other be done when $n_2 - n_1$ is of the order of $\frac{n_2}{\log n_2}$. However the running time is the measurement here, and the precise cut-off is chosen by trial and error.

Another implementation note concerns the PartAdd operation mentioned in Section 4, and also the final expansion of T and Q from the factored form to a flat integer. For this purpose, we use the same kind of algorithm as mentioned in [16] for the computation of $n!$.

We implemented our algorithm in C++ with the GMP and MPFR libraries [12, 8]. We modified the GMP library with an improved FFT multiplication code [9]. We compare our results with the two programs mentioned in Section 2, which compute digits of π using Formula (3). For the purpose of comparison, we focus on the time for the evaluation of the fraction T/Q. Because our program and the `gmp-chudnovsky` program share the GMP library as a common backbone, we are convinced that this comparison gives the most meaningful results. Table 1 gives the relative time spent in the binary splitting process for our program and for the `gmp-chudnovsky` program, measured on a 2.4Ghz Opteron CPU. The gain seems to grow slightly with the number of digits computed.

Table 1 also mentions timings of our program against the `PiFast` program. The comparison has been made on a 3 GHz Pentium 4 CPU (hence the different timings). The lack of source code access for `PiFast` mandates some caution for the interpretation of the timings, since the operating system overhead seems to be included. Nonetheless, it seems that our fully factored binary splitting provides a growing improvement.

Finally, we used our program to establish a new record-size computation for 2 billion decimal digits of $\zeta(3)$. The series of Formula (4) was evaluated up to $N = 664385619$ terms. The computation of Q and T was first spanned over 16 distinct 2.4Ghz Opteron processors. The cumulative CPU time spent by these processors to compute their fraction of the result was 20 hours. These results were gathered to form the final fraction T/Q (in factored form) on a single computer in 3 hours. Converting the fraction T and Q to integers took 18 minutes. The division took 53 minutes, and the decimal conversion took 2 hours and 37 minutes (these timings are suboptimal, since GMP does not yet implement Newton's division).

Error analysis. Using the inequality $\frac{n}{2n+1} < \frac{1}{2}$ and a rough bound on $a(n)$, it can be shown that Formula (4) gives an error of at most $2^{-10N+2\log_2 N+2}$, which is less than $2^{-6643856129}$ here. Using a precision of $p = 6643856189$ bits, we converted T and Q to floating-point numbers, divided both, and converted the binary quotient to a decimal string of $2 \cdot 10^9 + 1$ digits, all those operations being made in rounding to nearest mode with the MPFR library. An error analysis yields a maximal absolute error of 2^{1-p}, which together with the above truncation error gives a maximal absolute error of $10^{-1999999981}$.

Acknowledgement. The authors thank Richard Brent for his clever remarks on a first draft of this paper, and the three anonymous referees for their very detailed reviews.

7. REFERENCES

[1] Bernstein, D. J. Fast multiplication and its applications. `http://cr.yp.to/papers.html`, 2004.

[2] Borwein, J. and Borwein, P. Pi and the AGM. John Wiley and Sons, 1987.

[3] Borwein, J. and Bradley, D. and Crandall, R. Computational stratgies for the Riemann zeta function. *Journal of Computational and Applied Mathematics 121* (2000), 247–296.

[4] Brent, R. Fast multiple-precision evaluation of elementary functions. *Journal of the ACM 23*, 2 (1976), 242–251.

[5] Chudnovsky, D., Chudnovsky G. Computer algebra in the service of mathematical physics and number theory. *Computers in mathematics (Stanford, CA, 1986), 109–232, Lecture Notes in Pure and Appl. Math.*, 125, Dekker, New York, 1990.

[6] Cheng, H., Gergel, B., Kim, E., and Zima, E. Space-efficient evaluation of hypergeometric series. *SIGSAM Bulletin, Communications in Computer Algebra 39*, 2 (2005), 41–52.

[7] Cheng, H., and Zima, E. On accelerated methods to evaluate sums of produts of rational numbers. In *Proceedings of ISSAC'00* (2000), pp. 54–61.

[8] Fousse, L., Hanrot, G., Lefèvre, V., Pélissier, P., and Zimmermann, P. MPFR: A multiple-precision binary floating-point library with correct rounding. *ACM Trans. Math. Softw. 33*, 2 (2007).

[9] Gaudry, P., Kruppa, A., and Zimmermann, P. A GMP-based implementation of Schönhage-Strassen's large integer multiplication algorithm. Proceedings of ISSAC'07, Waterloo, Ontario, Canada, 2007.

[10] Gosper, R. Strip mining in the abandoned orefields of nineteenth century mathematics. *Computers in Mathematics* (1990), pp. 261–284.

[11] Gourdon, X., and Sebah, P. Binary splitting method. `http://numbers.computation.free.fr/Constants/Algorithms/splitting.html`, 2001.

[12] *GMP: The GNU Multiple Precision Arithmetic Library*, 4.2.1 ed., 2006. `http://gmplib.org/`.

[13] Haible, B., and Papanikolaou, T. Fast multiprecision evaluation of series of rational numbers. Algorithmic Number Theory, Third International Symposium, ANTS-III, volume 1423 of *Lecture Notes in Computer Science*, Springer, 1998.

[14] Karatsuba, E. A. Fast evaluation of transcendental functions. In *Problems of Information Transmission*, 27: 339-360, 1991.

[15] Lang, S. Algebraic Number Theory. Springer-Verlag, 1994.

[16] Schönhage, A., Grotefeld, A. F. W., and Vetter, E. *Fast Algorithms, A Multitape Turing Machine Implementation.* BI-Wissenschaftsverlag, 1994.

[17] Schönhage, A. and Strassen, V. Schnelle Multiplikation großer Zahlen. *Computing 7* (1971), 281–292.

[18] Tenenbaum, G. Introduction to Analytic and Probabilistic Number Theory. Cambridge University Press, 1995.

[19] Wang, X. and Pan, V. Y. Acceleration of Euclidean algorithm and rational number reconstruction. *SIAM Journal on Computing*, 32(2):548–556, 2003.

[20] van der Hoeven, J. Fast evaluation of holonomic functions. *Theoret. Comput. Sci.* 210 (1999), no. 1, 199–215.

[21] van der Hoeven, J. Fast evaluation of holonomic functions near and in regular singularities. *J. Symbolic Comput.* 31 (2001), no. 6, 717–743.

[22] van der Hoeven, J. Efficient accelero-summation of holonomic functions. *J. Symbolic Comput.* 42 (2007), no. 4, 389–428.

[23] Xue, H. `gmp-chudnovsky.c` code for computing digits of π using the GNU MP library. Available at `http://gmplib.org/pi-with-gmp.html`, 2002.