I N F O R M A T I K

On Wallace's Method for the Generation of Normal Variates

Christine Rüb

MPI–I–98–1–020 September 1998

FORSCHUNGSBERICHT RESEARCH REPORT

M A X - P L A N C K - I N S T I T U T
F Ü R
I N F O R M A T I K

Im Stadtwald 66123 Saarbrücken Germany

Author's Address

Max-Planck-Institut für Informatik Im Stadtwald D-66123 Saarbrücken
email: rueb@mpi-sb.mpg.de

Acknowledgements

Supported by the Deutsche Forschungsgemeinschaft, Research–Cluster "Efficient Algorithms for Discrete Problems and their Applications, grant LE 952/1–2.

Abstract

A method proposed by Wallace for the generation of normal random variates is examined. His method works by transforming a pool of numbers from the normal distribution into a new pool of number. This is in contrast to almost all other known methods that transform one or more variates from the uniform distribution into one or more variates from the normal distribution. Unfortunately, a direct implementation of Wallace's method has a serious flaw: if consecutive numbers produced by this method are added, the resulting variate, which should also be normally distributed, will show a significant deviation from the expected behavior. Wallace's method is analyzed with respect to this deficiency and simple modifications are proposed that lead to variates of better quality. It is argued that more randomness (that is, more uniform random numbers) is needed in the transformation process to improve the quality of the numbers generated. However, an implementation of the modified method has still small deviations from the expected behavior and its running time is much higher than that of the original.

Keywords

Random number generators, Normal distribution, Gaussian distribution

1 Introduction

In [8] Wallace introduced a novel method of generating pseudo–random numbers from the unit–normal and the unit–exponential distribution. His algorithms work by transforming a pool of numbers from the desired distribution into a new pool of numbers. This is in contrast to almost all other known methods that transform one or more variates from the uniform distribution into one or more variates from the target distribution. (For a description of several of these methods see [3] or [6]). Since Wallace's algorithms work directly on variates with the desired distribution they are very fast – specifically, their speed is comparable to the speed of random number generators for the uniform distribution.

Wallace also reports on the outcome of empirical tests performed with a specific implementation for the normal distribution [7]. This program passed all of the tests used. However, all of these tests were for single numbers produced by the generator. If v successive numbers from the generator for the normal distribution are added, we expect to obtain a normal variate with variance v. Unfortunately, this is not the case here: in many cases the variance of the numbers produced is much too small. In this paper we present test results that demonstrate this deficiency and analyze its cause. We also propose modifications of Wallace's method that lessen this deficiency. Unfortunately, there are still small but measurable deviations from the expected behavior that seem to be difficult to get rid of. Additionally, the proposed changes lead to a much reduced speed and lessen one of the big advantages of this method.

One might argue that there is no need to add up several normal variates, since the resulting sum is a normal variate with a larger variance, which could as well have been obtained from a single $N(0;1)$–variate by scaling ($N(e;v)$ stands for the normal distribution with expectation e and variance v). However, many applications are such that, although random variates are not added explicitly, sums of variates are implicitly formed in the course of time. In fact, the deficiency became apparent to us while checking the result of a molecular dynamics simulation of a synthetic polymer [5], where such implicit summing occurs in the simulation of a random walk taken by a gas atom.

This paper is organized as follows: Section 2 describes Wallace's method for the generation of normal variates and Section 3 presents test results for an implementation of this method provided by Wallace. Section 4 analyzes Wallace's method with respect to this deficiency and proposes modifications to lessen the deficiency. Section 5 briefly discusses two similar algorithms and Section 6 draws some conclusions.

2 Wallace's method for the normal distribution

In the following we describe Wallace's method for the generation of $N(0;1)$–variates [8]. It is based on the following facts.

Let $x_1, ..., x_k$ be k independent $N(0;1)$–variates and let $\beta_1, .., \beta_k$ be real numbers. Then $y = \sum_{i=1}^{k} \beta_i x_i$ is $N(0;\sigma^2)$–distributed where $\sigma^2 = \sum_{i=1}^{k} \beta_i^2$. Thus, if $\sum_{i=1}^{k} \beta_i^2 = 1$, y is $N(0;1)$–distributed.

Let $x_1, ..., x_k$, and y be defined as above and let $z = \sum_{i=1}^{k} \gamma_i x_i$ where $\gamma_1, ..., \gamma_k$ are real numbers. Then y and z are normal distributed with expectation 0. We have $\mathrm{Exp}(yz) = \mathrm{Exp}(\sum_{i=1}^{k} \beta_i x_i \sum_{i=1}^{k} \gamma_i x_i) = \mathrm{Exp}(\sum_{i=1}^{k} \beta_i \gamma_i x_i^2) + \sum_{i \neq j} (\beta_i \gamma_j \mathrm{Exp}(x_i x_j)) = \mathrm{Exp}(\sum_{i=1}^{k} \beta_i \gamma_i x_i^2)$ since the x_i's are independent. Thus $\mathrm{Exp}(yz) = 0 = \mathrm{Exp}(y)\mathrm{Exp}(z)$, which means that y and z are independent, if $\sum_{i=1}^{k} \beta_i \gamma_i = 0$. This implies that we can take a vector $X = (x_1, ..., x_k)$

of k independent $N(0;1)$–distributed variates, multiply it from the left by an orthonormal matrix B, and obtain thus a vector $Y = (y_1, ..., y_k)$ of k independent $N(0;1)$–distributed variates. Note that the y_i's are not independent from the x_i's since it can be shown that $\sum_{i=1}^{k} x_i^2 = \sum_{i=1}^{k} y_i^2$.

Wallace's method starts by generating a pool of $N = kn$ normal variates using a conventional method like the Box–Muller method (see, e.g., [3] or [6]). The numbers in this pool are normalized such that the sum of their squares is N. In one **pass** these N numbers are transformed into N new numbers. This is done by taking k old numbers at a time, treating them as a k–vector X and forming k new variates by computing $Y^T = BX^T$ where B is an orthonormal $k \times k$ matrix. Each old pool value is used exactly once. It can be shown that this transformation preserves the sum of squares.

Since the sum of the squares of N $N(0;1)$–variates is chi–squared distributed with N degrees of freedom (for short, χ_N^2 distributed), preserving the sum of squares would obviously be a defect of the generator. Thus, for each pass, a variate ν from the χ_N^2 distribution is generated and each number generated in the pass is multiplied by $(\nu/N)^{0.5}$ before it is returned to the user.

To make sure that the numbers in the old pool are mixed, the order in which the elements are treated is somewhat randomized. Also, the orthonormal matrix used in a pass is chosen randomly out of a small set of matrices that allow for a fast computation of the new variates.

Thus, in this method, uniform random numbers are used to generate the initial pool of normal variates, to choose a matrix for each pass and to choose an order in which the elements are treated. This is in contrast to conventional methods, which use at least one uniform variate per normal variate, and allows for a very fast implementation.

We next describe the details of the implementation made available by Wallace [7], where $N = 1024$, $k = 4$, and $n = 256$.

To generate a variate ν from the χ_{1024}^2 distribution, the following approximation is used. Let y be $N(0;1)$ distributed. Then $z = 0.5(C + Ay)^2$ is approximately χ_k^2 distributed, where $A = (1 + 1/(8k))$ and $C^2 = 2k - A^2$. The 1,024th element of the new pool is used for y. This element is not returned to the user, but is used to generate the elements of the next pool.

The procedure is implemented in–place, that is, there is only one array of size 1,024 for the old and the new pool. Four types of scanning patterns are used to mix the values in the old pool. In **Type 1** the first half of the pool is treated as a 4×128 matrix stored in row major order and the second half as a 4×128 matrix stored in column major order (a matrix is stored in row (column) major order if the rows (columns, respectively) are stored one after the other, starting from the top (left, respectively) of the matrix). Depending on which matrix is used in a pass, one of the rows of the first matrix will be read from right to left. The columns of these matrices are the vectors that are multiplied by the transformation matrix B. In each step, one column of the first matrix and one column of the second matrix are used to generate eight new variates. The four variates generated by the column of the first matrix are stored at the positions of the column of the second matrix and vice versa. The order in which the columns of the second matrix are picked is randomized; before the pass, a start column and an odd increment are picked at random and determine the order. The columns of the first matrix are used consecutively.

The **Type 2** scanning pattern is similar to the Type 1 pattern, except that the first (second) half of the array is now treated in the way the second (first, respectively) half of the array is treated with the Type 1 pattern.

For the **Type 3** and the **Type 4** scanning pattern, the array is divided into odd and even indexed positions. In Type 3 the even (odd) indexed elements play the role the first (second, respectively) half of the array plays in Type 1, and in Type 4 it is the other way round. The four scanning types are used circularly.

The transformation matrices come in pairs. A transformation matrix B^1 (B^2) is applied to the columns of the above mentioned first (second, respectively) matrix. For each pass, one of the following four matrix pairs is selected at random.

$$B_1^1 = \frac{1}{2}\begin{bmatrix} 1 & -1 & 1 & 1 \\ -1 & 1 & 1 & 1 \\ -1 & -1 & -1 & 1 \\ -1 & -1 & 1 & -1 \end{bmatrix} \quad B_1^2 = \frac{1}{2}\begin{bmatrix} -1 & -1 & -1 & 1 \\ -1 & 1 & 1 & 1 \\ -1 & 1 & -1 & -1 \\ 1 & 1 & -1 & 1 \end{bmatrix}$$

$$B_2^1 = \frac{1}{2}\begin{bmatrix} 1 & 1 & 1 & -1 \\ -1 & -1 & 1 & -1 \\ -1 & 1 & -1 & -1 \\ -1 & 1 & 1 & 1 \end{bmatrix} \quad B_2^2 = \frac{1}{2}\begin{bmatrix} 1 & 1 & -1 & 1 \\ -1 & -1 & -1 & 1 \\ 1 & -1 & 1 & 1 \\ 1 & -1 & -1 & -1 \end{bmatrix}$$

$$B_3^1 = \frac{1}{2}\begin{bmatrix} -1 & -1 & 1 & -1 \\ 1 & 1 & 1 & -1 \\ 1 & -1 & -1 & -1 \\ 1 & -1 & 1 & 1 \end{bmatrix} \quad B_3^2 = \frac{1}{2}\begin{bmatrix} 1 & 1 & -1 & 1 \\ 1 & -1 & -1 & -1 \\ 1 & 1 & 1 & -1 \\ -1 & 1 & -1 & -1 \end{bmatrix}$$

$$B_4^1 = \frac{1}{2}\begin{bmatrix} -1 & 1 & -1 & -1 \\ 1 & -1 & -1 & -1 \\ 1 & 1 & 1 & -1 \\ 1 & 1 & -1 & 1 \end{bmatrix} \quad B_4^2 = \frac{1}{2}\begin{bmatrix} -1 & 1 & -1 & -1 \\ -1 & 1 & 1 & 1 \\ 1 & 1 & 1 & -1 \\ -1 & -1 & 1 & -1 \end{bmatrix}$$

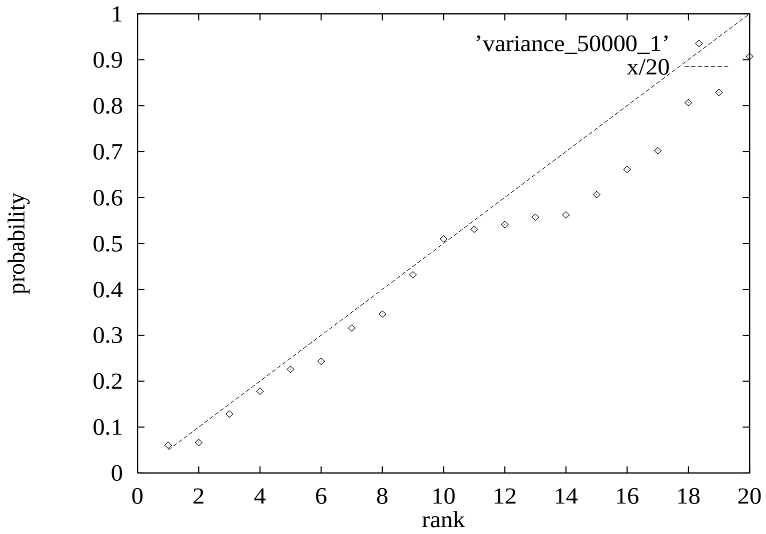

Figure 1: 20 tests for the variance of 50,000 numbers

3 Some Test Results

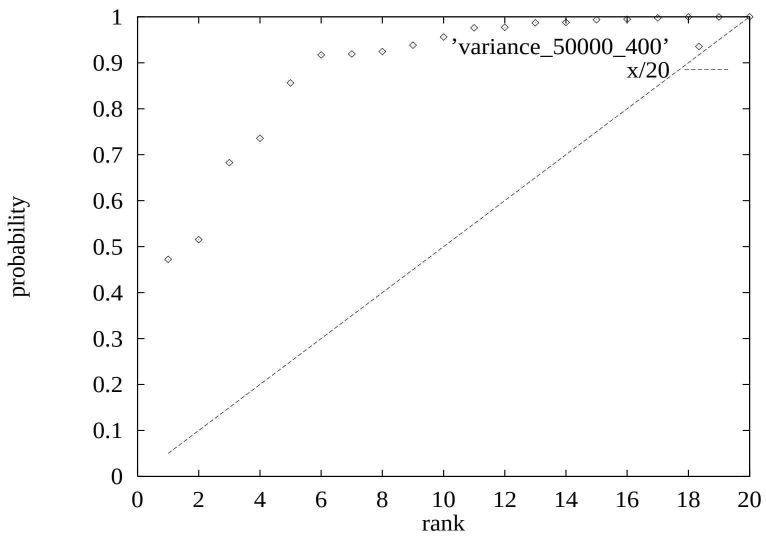

Figure 2: 20 tests for the variance of 50,000 sums of 400 numbers

We have performed a variance test for the numbers generated by this program as well as for sums of generated numbers. This test inputs m supposedly independent random variates from a common normal distribution $N(0;\sigma^2)$ and computes $S = \sum_{i=1}^{m} X_i^2/\sigma^2$. The test outputs the tail probability $\Pr(T \geq S)$, where T is χ^2–distributed with m degrees of freedom. Since for any continuous random variable X with distribution function F the random variable $F(X)$ is uniformly distributed in $[0,1]$, the same applies for the output of our variance test if the assumption about the input numbers is correct. Figures 1 and 2 show the outcome of 20 runs of this test for 50,000 variates each; for the first test each such variate is one produced by Wallace's generator, and for the second test, it is the sum of 400 variates produced consecutively by the generator. In both cases, the probabilities are sorted and plotted against their rank. A comparison with the (ideal) line with slope 1/20 shows that the probabilities in Figure 2 are much too large, indicating that the sample variances are too small. The test results in Figure 1 are not significant.

The strength of this effect depends on the number of variates summed. It can become very strong if a few numbers at the beginning are discarded. If, e.g., the first 128 generated numbers are discarded and after this 1,023 consecutive numbers are summed, a figure would only show a straight line at 1. (Remember that the 1,024th number of each pool is not returned to the user.) We performed the variance test for this case 1,000 times for 50,000 (sums of 1,023) variates each and the smallest outcome of the test was larger than 0.999999. Figure 3 shows the outcome of the variance test for one seed value if the number of discarded variates ranges from 0 to 1,023. Again, 50,000 times 1,023 numbers were added. This figure is typical in that there are always values close to 1 with an offset of 128 and values close to 0 with an offset of 640.

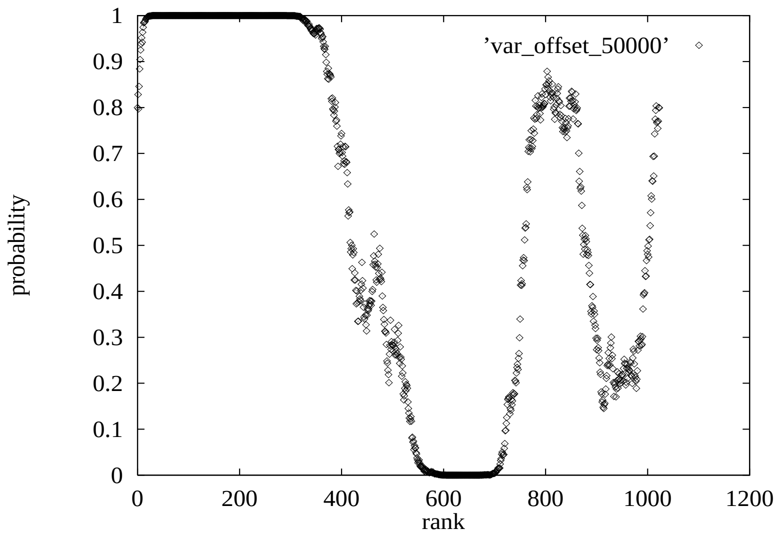

Figure 3: Outcome of the variance test for all offsets for one seed value. For each run 50,000 sums of 1023 variates were used.

4 The Deficiencies

We have seen above that adding consecutive numbers generated by the algorithm leads to a variance that is much too low. We will not analyze the algorithm in detail since this seems to be difficult to achieve, but we are going to use this example to point out possible causes of problems in an implementation of Wallace's method. For now we ignore the normalization which in itself causes some (smaller) problems. This means that we assume that the 1,024th element is returned to the user and that the proper behavior of the sum of squares is somehow guaranteed.

The algorithm works by computing $B(a, b, c, d)^T = (a', b', c', d')^T$ where B is an orthonormal matrix. Since B is orthonormal, a', b', c', and d' will be $N(0;1)$–distributed and independent given that a, b, c, and d are independent $N(0;1)$ variates. However, the new values and the old values are not independent.

Suppose that an algorithm adds a', a, b, c, and d. Then $S = a' + a + b + c + d$ should be $N(0;5)$–distributed. Let us assume that a, b, c, and d are independent and in fact $N(0;1)$–distributed (without this assumption it is difficult to prove anything). Since $a' = \beta_{11}a + \beta_{12}b + \beta_{13}c + \beta_{14}d$ and $\mathrm{Exp}(\beta_{ij}) = 1/4\sum_{i=1}^{4}\beta_{ij}^k$ where β_{ij}^k is an entry of the kth transformation matrix, we have

$$\mathrm{Exp}(S) = \frac{1}{4}\sum_{k=1}^{4}(1+\beta_{11}^k)\mathrm{Exp}(a) + (1+\beta_{12}^k)\mathrm{Exp}(b) + (1+\beta_{13}^k)\mathrm{Exp}(c) + (1+\beta_{14}^k)\mathrm{Exp}(d).$$

Since the old values are $N(0;1)$–distributed,

$$\mathrm{Exp}(S) = 0.$$

Let us now examine the variance of S. Since the old values are independent and $N(0;1)$–distributed,

$$\begin{aligned}
\mathrm{Exp}(S^2) = \quad & \frac{1}{4}\sum_{k=1}^{4}\Big((1+\beta_{11}^k)^2\mathrm{Exp}(a^2) + (1+\beta_{12}^k)^2\mathrm{Exp}(b^2) + (1+\beta_{13}^k)^2\mathrm{Exp}(c^2) \\
& \qquad + (1+\beta_{14}^k)^2\mathrm{Exp}(d^2)\Big) \\
= \quad & \frac{1}{4}\sum_{k=1}^{4}\Big((1+\beta_{11}^k)^2 + (1+\beta_{12}^k)^2 + (1+\beta_{13}^k)^2 + (1+\beta_{14}^k)^2\Big) \\
= \quad & \frac{1}{4}\sum_{k=1}^{4}\Big(4 + (\beta_{11}^k)^2 + (\beta_{12}^k)^2 + (\beta_{13}^k)^2 + (\beta_{14}^k)^2 + 2\beta_{11}^k + 2\beta_{12}^k + 2\beta_{13}^k + 2\beta_{14}^k\Big).
\end{aligned}$$

Since the B's are orthonormal, we arrive at

$$\mathrm{Exp}(S^2) = 5 + \frac{1}{4}\sum_{k=1}^{4}\Big(2\beta_{11}^k + 2\beta_{12}^k + 2\beta_{13}^k + 2\beta_{14}^k\Big). \tag{1}$$

From this follows that

$$\sum_{i=1}^{4}\beta_{11}^i + \beta_{12}^i + \beta_{13}^i + \beta_{14}^i = 0,$$

since S has to be $N(0;5)$–distributed. That means that the entries of the first rows of all matrices, summed over all matrices used, must cancel out. The same condition holds for **any** row or column of the matrices used if the appropriate values are added. The condition for the columns follows since $B^{-1} = B^T$ for an orthonormal matrix B and we could, say, add a and a', b', c', and d'.

We can derive even more severe restrictions than this. Assume that we add a and a' into S. Then we get for the variance

$$\begin{aligned}
\mathrm{Exp}(S^2) &= \frac{1}{4}\sum_{k=1}^{4}\Big((1+\beta_{11}^k)^2 + (\beta_{12}^k)^2 + (\beta_{13}^k)^2 + (\beta_{14}^k)^2\Big) \\
&= 2 + \frac{1}{4}\sum_{k=1}^{4}2\beta_{11}^k
\end{aligned}$$

which means that

$$\sum_{k=1}^{4}\beta_{11}^k = 0$$

must hold. We can argue similarly for all other entries of the transformation matrices.

Let us now see what happens in the case of the tested algorithm. First observe that we have to treat the B^1's and B^2's separately since they are applied in different ways. We can observe that in the B^2's, the third rows and the third columns add up to $+1$, i.e., the entries corresponding to the new c's and the old c's, respectively, add up to 1. Likewise, in the B^2's the rows and columns corresponding to the new and old d's add up to -1 and in the B^1's the old and new c's, the new d's and the old b's are affected.

If we also take the four scanning types into account, it is easy to devise a test that the generator will fail. Consider, e.g., scanning type 1. If scanning type 1 is used, the elements of two pools will be used in the following way. Each pool is stored as an array of length

1,024. The first (second) half of this array is interpreted as a 4 by 128 matrix stored in row (column, respectively) major order. We divide the entries of the two matrices into four classes denoted by $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$, and $\mathcal{D}$. $\mathcal{A}$ ($\mathcal{B}$, $\mathcal{C}$, $\mathcal{D}$) contains all entries in the first (second, third, fourth, respectively) row of one of the matrices. That is, we denote the role an element will play when the transformation matrices are applied. The letters in the figure below indicate to which classes the numbers in the pool belong if the first scanning pattern is used. Note that the old as well as the new pool is used in this way.

$$\overbrace{\mathcal{A} \ldots \mathcal{A}}^{128}\ \overbrace{\mathcal{B} \ldots \mathcal{B}}^{128}\ \overbrace{\mathcal{C} \ldots \mathcal{C}}^{128}\ \overbrace{\mathcal{D} \ldots \mathcal{D}}^{128}\ \overbrace{\mathcal{A}\ \mathcal{B}\ \mathcal{C}\ \mathcal{D}\ \mathcal{A}\ \mathcal{B}\ \mathcal{C}\ \mathcal{D} \ldots}^{128\times 4}$$

The numbers in the first half of the new pool are generated by applying the B^2s. Also, the entries in the last rows of the B^2s add up to -1. Thus, it follows from Equation (1) and the following discussion that we will get a variable with too small a variance if we add the numbers from the second half of the old pool to the fourth row (of 128 numbers) of the new pool.

In our variance test, none of the scanning types used in the algorithm directly leads to summing the problematic values (in fact, $a' + b' + c' + d' + a + b + c + d$ will have the correct variance). However, the four scanning types together seem to be able to concentrate numbers that are computed from these problematic values into the first 128 elements of the new matrices (the specific, deterministic order in which the scanning types are used does not seem to play a crucial role; randomizing this order does not help). The choice of the set of orthonormal matrices together with the simple scanning pattern causes the problem; the randomization of the order in which the rows of the second matrix are used makes no difference.

Note that the condition that the entries cancel out if summed over all matrices has to hold for all rows and all columns if such a simple scanning pattern is used: an application might just use the generated numbers in a way that the difference in variance shows up.

Three possible ways to take care of this problem spring to mind. One is to choose matrices that confirm to the above stated condition, a second one is to use only variates of every second pool, and a third one is to totally randomize the order in which the elements of the pools are treated.

The second method could be thought of as taking care of other dependencies between the old pool elements and the new pool elements as well. However, it does not work. We have tested versions where up to three pools of numbers were discarded, but none of them passed the variance test. Figure 4 shows results from variance tests where one, two, or three consecutive pools from the generator are skipped. Again, the first 128 numbers generated were discarded and then 1,023 consecutive numbers were added to generate one number with variance 1,023.

The cheapest method (with respect to the running time) of the three above uses matrices that have the above required property. We tried this approach with the following pairs of matrices. The first (third) pair of matrices is just the first (third, respectively) pair of matrices suggested by Wallace. The second (fourth) pair was obtained by replacing each entry r in the first (third, respectively) matrix by $-r$. These matrices fulfill all conditions derived above to ensure the correct variances. Additionally, the order in which the scanning patterns are used was randomized.

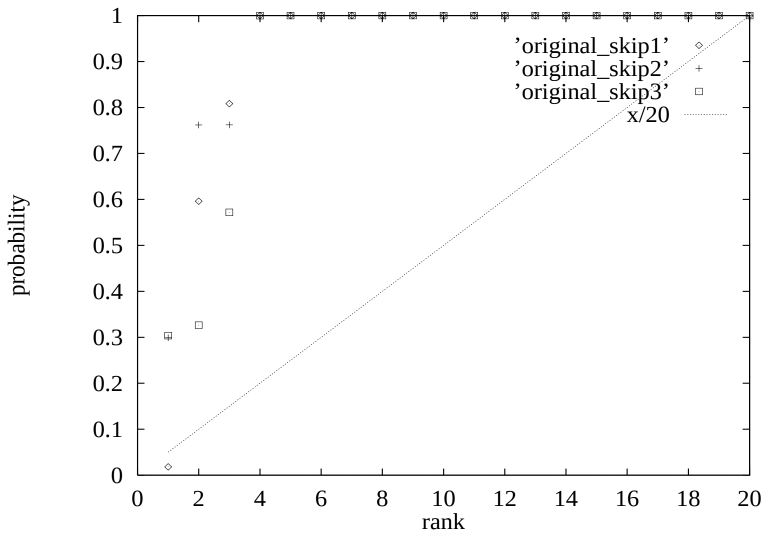

Figure 4: The effect of omitting pools. The results of 20 runs of the variance test are shown where between one (skip1) and three (skip3) consecutive pools are skipped. For each run, the first 128 numbers were discarded and after this 50,000 times 1,023 consecutive numbers were added.

$$B_1^1 = \frac{1}{2}\begin{bmatrix} 1 & -1 & 1 & 1 \\ -1 & 1 & 1 & 1 \\ -1 & -1 & -1 & 1 \\ -1 & -1 & 1 & -1 \end{bmatrix} \quad B_1^2 = \frac{1}{2}\begin{bmatrix} -1 & -1 & -1 & 1 \\ -1 & 1 & 1 & 1 \\ -1 & 1 & -1 & -1 \\ 1 & 1 & -1 & 1 \end{bmatrix}$$

$$B_2^1 = \frac{1}{2}\begin{bmatrix} -1 & 1 & -1 & -1 \\ 1 & -1 & -1 & -1 \\ 1 & 1 & 1 & -1 \\ 1 & 1 & -1 & 1 \end{bmatrix} \quad B_2^2 = \frac{1}{2}\begin{bmatrix} 1 & 1 & 1 & -1 \\ 1 & -1 & -1 & -1 \\ 1 & -1 & 1 & 1 \\ -1 & -1 & 1 & -1 \end{bmatrix}$$

$$B_3^1 = \frac{1}{2}\begin{bmatrix} -1 & -1 & 1 & -1 \\ 1 & 1 & 1 & -1 \\ 1 & -1 & -1 & -1 \\ 1 & -1 & 1 & 1 \end{bmatrix} \quad B_3^2 = \frac{1}{2}\begin{bmatrix} 1 & 1 & -1 & 1 \\ 1 & -1 & -1 & -1 \\ 1 & 1 & 1 & -1 \\ -1 & 1 & -1 & -1 \end{bmatrix}$$

$$B_4^1 = \frac{1}{2}\begin{bmatrix} 1 & 1 & -1 & 1 \\ -1 & -1 & -1 & 1 \\ -1 & 1 & 1 & 1 \\ -1 & 1 & -1 & -1 \end{bmatrix} \quad B_4^2 = \frac{1}{2}\begin{bmatrix} -1 & -1 & 1 & -1 \\ -1 & 1 & 1 & 1 \\ -1 & -1 & -1 & 1 \\ 1 & -1 & 1 & 1 \end{bmatrix}$$

With these matrices, the generator showed an acceptable behavior in the variance test. Unfortunately, this is not true for the fourth moments of summed variates. This can be explained as follows.

Consider the first scanning pattern and assume we add the second half of the old pool and the first 128 values of the new pool into S. Assume further that the values in the old pool are independent and $N(0;1)$–distributed and that the transformation matrix B used in this step is fixed. Then S is normally distributed with expectation 0 and variance $128((1+\beta_{11})^2+(1+\beta_{12})^2+(1+\beta_{13})^2+(1+\beta_{14})^2)$. However, if we choose the transformation matrix randomly from of a set of matrices, S is **not** normally distributed. To see this, consider, e.g., the fourth moment of S. Remember that $\mathrm{Exp}(X^4)=3$ if X is $N(0;1)$–distributed. We get

$$\mathrm{Exp}(S^4)=\frac{1}{4}\sum_{k=1}^{4}3\cdot 128^2\left((1+\beta_{11}^k)^2+(1+\beta_{12}^k)^2+(1+\beta_{13}^k)^2+(1+\beta_{14}^k)^2\right)^2$$

where β_{ij}^k is an entry of the kth matrix. In the case of the matrices used here we get

$$\mathrm{Exp}(S^4)=\frac{1}{2}\cdot 3\cdot 128^2(3^2+7^2)=3\cdot 128^2\cdot 29$$

instead of the required $3\cdot 128^2\cdot 25$. Note that the variance of S has the correct value.

As was the case for the variance, we can derive conditions on the transformation matrices that ensure that the fourth moment will have the correct value in this case. Since the transformation matrices are orthonormal and row entries have to cancel out to ensure the correct variance we obtain

$$\mathrm{Exp}(S^4)=\frac{1}{4}\sum_{i=1}^{4}3\cdot 128^2\left(5+2(\beta_{11}^i+\beta_{12}^i+\beta_{13}^i+\beta_{14}^i)\right)^2$$

$$=\frac{1}{4}\sum_{i=1}^{4}3\cdot 128^2\left(25+20(\beta_{11}^i+\beta_{12}^i+\beta_{13}^i+\beta_{14}^i)+4\left(\beta_{11}^i+\beta_{12}^i+\beta_{13}^i+\beta_{14}^i\right)^2\right)$$

$$=3\cdot 128^2\left(25+\frac{1}{4}\sum_{i=1}^{4}4\left(\beta_{11}^i+\beta_{12}^i+\beta_{13}^i+\beta_{14}^i\right)^2\right).$$

This means we get the condition

$$\frac{1}{4}\sum_{i=1}^{4}\left(\beta_{11}^i+\beta_{12}^i+\beta_{13}^i+\beta_{14}^i\right)^2=0$$

which can only be achieved if

$$\beta_{11}^i+\beta_{12}^i+\beta_{13}^i+\beta_{14}^i=0$$

for every matrix B^i. We can derive even more severe restrictions than this. Consider again the first scanning pattern and assume that we add the a's of the second half of the old pool and the a's of the first half of the new pool into S. We know already that $\sum_{k=1}^4\beta_{11}^k=0$ has to hold to ensure the correct variance for S.
On the other hand,

$$\mathrm{Exp}(S^4)=\frac{1}{4}\sum_{i=1}^{4}3\cdot 128^2\left((1+\beta_{11}^i)^2+(\beta_{12}^i)^2+(\beta_{13}^i)^2+(\beta_{14}^i)^2\right)^2$$

$$= \frac{1}{4}\sum_{i=1}^{4} 3 \cdot 128^2 \left(2 + 2\beta_{11}^i\right)^2$$

$$= 3 \cdot 128^2 \left(4 + \frac{1}{4}\sum_{i=1}^{4} 4 \left(\beta_{11}^i\right)^2\right)$$

which implies that $\beta_{11}^i = 0$ has to hold for every matrix B^i. Since we can derive the same requirement for each entry of the transformation matrix, we will arrive at the 0 matrix. Clearly this is not a possible choice.

The above discussion shows that it is not possible to achieve the correct distribution for arbitrary sums of generated variates if simple scanning types are used and the transformation matrices are fixed in each pass. Depending on the number of scanning types used, we can expect deviations in the moments of several per cent. Let us now turn to a version where the order in which the elements of the old pool and the new pool are treated is totally randomized. We will see that this helps to lessen the deficiencies described above; however, they will not vanish totally. Note that it is not possible to randomize only the order in which the elements of one pool are treated: if only for one pool the order is randomized, we can add all numbers from this pool and specific numbers from the non–randomized pool and will again obtain a variate with a fourth moment that is too large.

Let us first consider the case in which we add x and x' into S where x' has been computed using x from the old pool. If we choose the matrices such that S has the correct variance, $\mathrm{Exp}(S^4)$ will be too large if not $\beta_{jj}^i = 0$ for each matrix B^i and $1 \leq j \leq 4$. However, it will not often happen that we add x and x' and nothing else: if we, e.g., add one value from a fixed position in the old pool and one value from a fixed position in the new pool, we will only get matching values in about $1/n$ of all cases. Since the deviation for one pair of matching values is not very large (with the matrices used here it is 25%), we will get a very small deviation overall (in our case, where $n = 256$, the deviation will be less than 0.1%).

Let us next assume that we add $p = p_1 + p_2$ numbers into S, where p_1 (p_2) numbers come from fixed positions of the old (new, respectively) pool. Assume further that the matrix B used for the transformation is fixed. Unlike the case in which we use a fixed scanning pattern, S will not be normally distributed: here we have to average over all possible ways the p_2 new pool values depend on the p_1 old pools values. On the other hand, the deviation from the expected moments can be quite small since not all of the p_2 new pool values depend on the old p_1 pool values and since deviations can cancel out. For ease of explanation, consider the case where S consists of $p/2$ pairs (x_i, y_i) where y_i has been computed using x_i and no other number used or created in that step is present in S. Let β_i be the matrix entry of B used to compute y_i from x_i. Then the k-th moments of S will contain entries of the form

$$\mathrm{Exp}\left(\left(\sum_{j=1}^{p/2}(1+\beta_j)^2\right)^k\right) = \mathrm{Exp}\left(\left(p/2 + \sum_{j=1}^{p/2} 2\beta_j + (\beta_j)^2\right)^k\right)$$

where the expectation is taken over all possible ways to choose the β_j. This value should be close to p^k. If the absolute values of the matrix entries are all identical, as is the case with the matrices used here, and since the matrices are orthonormal, this leads to

$$\text{Exp}\left(\left(p+\sum_{j=1}^{p/2} 2\beta_j\right)^k\right).$$

If for each entry with value z in B there is an entry with value $-z$ and the β_j are chosen at random, they can cancel each other mostly out and the probability that we get a large deviation from p^k is small. This also means that for each value $p+x$ there is a value $p-x$ that occurs with the same probability. Altogether this means that there will be deviations from the correct moments; however, the deviations will be small.

Thus the transformation matrices used in Wallace's program will produce variates where the deviation from the expected moments is small if the order in which the pool elements are treated is totally randomized. The deviation from the expected values is inversely proportional to the size of the pool used and can thus be further reduced by making the pool larger. The reason the randomization helps is that we average over all entries of the transformation matrices. Another modification that might work is to choose a transformation matrix at random whenever one is needed. In the implementation considered here, this would reduce the number of random choices by a factor of four.

Finally we consider the normalization procedure. Remember that the 1,024th element of the pool is not returned to the user but used to normalize all other elements of this pool. Let z be this element. Then all 1,023 other elements of this pool are multiplied by $c_1(c_2+Gz)$ where $c_1 \approx 0.0221$, $c_2 \approx 45.239$, and G is the normalization factor used for the old pass. This method leads to too high moments for elements of the new pool that are computed using z. The error produced here is very small; however, to improve the quality, it would be better to generate a normal variate with a traditional method.

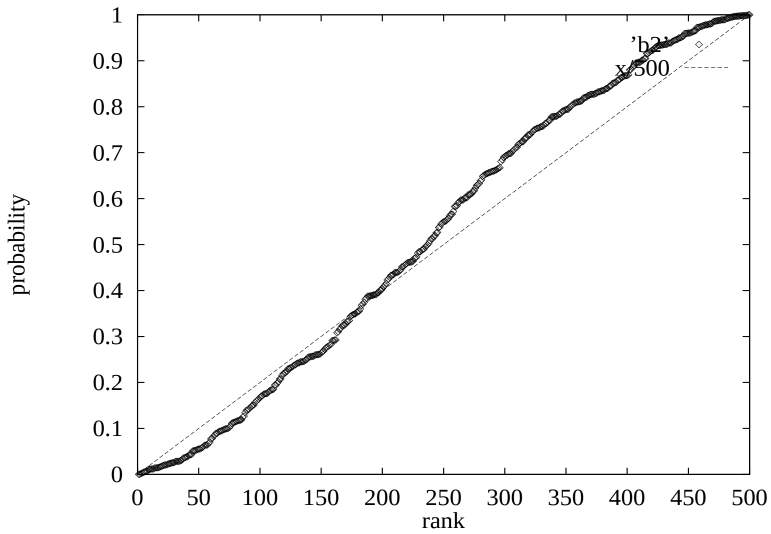

Figure 5: Test results of 500 runs of a b_2 test for 50,000 generated numbers each

At the end of this section we include some tests that even the randomized version of the algorithm failed. Figure 5 shows the result of 500 runs of a test for the standardized fourth moment (b_2 test), each for 50,000 single numbers created by the generator. For these tests

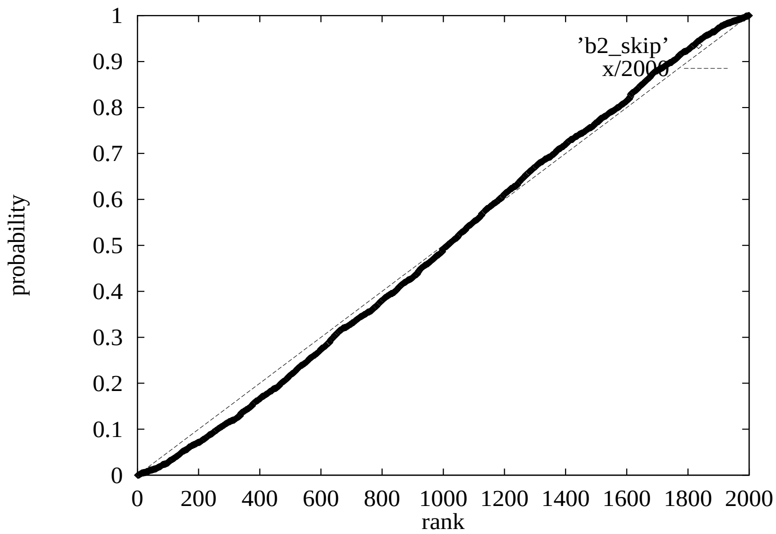

Figure 6: Test results of 2,000 runs of a b_2 test for 50,000 generated numbers each where every second pool was skipped

we used the Anscombe and Glynn approximation, see, e.g., [2]. We can see that the curve produced has an S shape instead of being approximately linear. The values shown in Figure 5 are significant at the 0.5% level if an Anderson Darling test [2] is used.

Figure 6 shows similar results for the case where only the variates from every second pool were used; here 2,000 runs were used. As we have seen before, omitting whole pools does not seem to change the behavior of the generator much. Note that we get very similar results for the original implementation without randomization. Again, the values shown are significant at the 0.5% level.

We also conducted tests for sums of two generated numbers from different pools. More precisely, we summed the ith number of a pool and the ith number of the following pool to obtain normal variates with variance two. Figure 7 shows the result of the b_2 test for this case. Here we used 10,000,000 numbers for each test run. As we can see, the fourth moment is too small. The deviation here is about 0.1%. This is the same order as predicted above; however, the deviation is in the other direction. At the moment it is not clear why this is the case. Figure 7 also shows results of the b_2 test for sums of four numbers where two numbers are from one pool and two numbers are from the following pool. As one can see, the fourth moment is still too small, but the deviation is smaller. As we add more numbers according to a similar rule, the deviation becomes even smaller.

5 Related algorithms

In this section we briefly discuss two variants of Wallace's algorithm that have been proposed recently. Note that we did not implement and test these algorithms but base our comments on the theory developed in Section 4.

In [1] Brent suggested a modification of Wallace's algorithm and in [4] Fernández and Rivero proposed an algorithm based on similar ideas. Both of these algorithms work by using

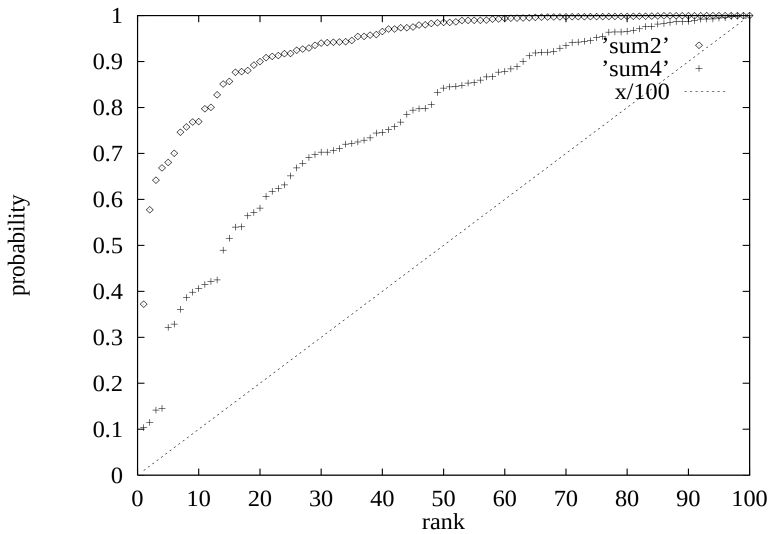

Figure 7: Test results of 100 runs of a b_2 test for 10,000,000 sums of numbers. The upper curve (sum2) is for the sum of two numbers and the lower curve (sum4) is for the sum of four numbers.

2×2 transformation matrices of the form

$$\begin{pmatrix} \cos\Theta & \sin\Theta \\ -\sin\Theta & \cos\Theta \end{pmatrix},$$

and [1] imposes the restriction that $\pi/6 \leq |\Theta| \leq \pi/3$ or $2\pi/3 \leq |\Theta| \leq 5\pi/6$.

The details of Brent's modifications are as follows. The size of the pool is a power of 2 and at least 512. Let the old pool consist of $x_0, .., x_{n-1}$ and $y_0, ..., y_{n-1}$. Then the new pool consists of pairs (x_i', y_i') where

$$\begin{pmatrix} x_j' \\ y_j' \end{pmatrix} = A_j \begin{pmatrix} x_{\alpha j+\gamma \bmod n} \\ y_{\beta j+\delta \bmod n} \end{pmatrix},$$

A_j is of the form given above, and α, β, γ, and δ are chosen randomly in a way to ensure that each x_i and each y_i is used exactly once. The transformation matrices are not fixed during a pass, but a random Θ is picked at the beginning of each inner loop (whose length is not specified) and used to compute the transformation matrix for this loop.

Since the x's and the y's are stored in fixed places in the pools, it is easy to add, say, all x's from the old pool and all x's from the new pool into a number S. Because of the choice of the transformation matrices (the expected value of each entry is 0), the expectation and the variance of S will have the correct value. However, this is not the case for the fourth moment of S. Assume for the moment that a single transformation matrix is used for one pass. Then $\mathrm{Exp}(S^4)$ will be about 26% too large (the calculations are similar to those in Section 4). Since actually the transformation matrix is changed more often, the deviation will be smaller and can be expected to be about $26/x$% where x is the number of transformation matrices used in one pass.

The algorithm in [4] was developed independently and the numbers in the pool are treated differently. In one pass, the pool is parsed from start to end. Denote the elements of the pool by $P_1, ..., P_N$. In the ith step, P_i plays the role of x. An index $j \neq i$ is picked at random and P_j plays the role of y. Then a transformation matrix is applied to x and y, and the two new values are output and stored as P_i and P_j. In each step, the transformation matrix is chosen randomly from a precomputed set of matrices (the use of 16 different, randomly generated, matrices is suggested). Thus, in this algorithm two additional uniform variates are used per two normal variates.

In [4] it is also argued that the deviation from the expected behavior is of the order of $1/N$ where N is the size of the pool. Thus, for the suggested N =1,024, the deviation would be about 0.1%. However, the analysis does not seem to cover all cases.

Consider, e.g., the case where only every second number produced is used and where $2N$ of these are added up into S. By picking only every second number we make sure that we only use x's. Also, we can expect that for about half of the numbers, the number that has been used as x in its production is also present. One of these pairs contributes

$$(1 + \cos\Theta)^2 + (\sin\Theta)^2 = 2 + 2\cos\Theta$$

to the variance of S, where Θ is one of, say, r randomly chosen values between 0 and 2π. Denote these values by $\Theta_1, ..., \Theta_r$. Since $\mathrm{Exp}(\cos^2\Theta) = \pi$ if Θ is chosen uniformly between 0 and 2π, we have

$$\mathrm{SD}\left(\sum_{i=1}^{r} \cos(\Theta_i)\right) = \sqrt{\pi r},$$

where SD denotes the standard deviation. Thus we can expect a deviation of the variance of S from the correct value of the order of $1/\sqrt{r}$. The magnitude and direction of this deviation depends on the choice of random matrices made initially. Summarizing, many more randomly chosen matrices have to be used or the matrices have to be chosen more carefully.

6 Conclusions

We have seen that Wallace's method, when used with simple scanning patterns and with a fixed transformation matrix in a pass, will not produce adequate normal variates and that for sums of variates the deviation from the expected moments can be quite large. This deficiencies can be lessened by randomizing the order in which the pool elements are treated or, perhaps, by choosing a transformation matrix at random for each tuple of numbers generated. Thus, to increase the quality of the output, more uniform numbers are needed which increases the running time of the algorithms considerably. However, there will still be small deviations for certain sums of variates. These deviations can be made smaller by increasing the size of the pool which might again reduce the speed of the algorithm. It also seems to be the case that these deficiencies cannot be overcome by omitting (a small number of) whole pools of generated numbers.

In a case where the speed of the proposed method is most important, namely, if many normal variates are needed in an application, these deficiencies are most serious. Thus, we feel that at the moment Wallace's method cannot be recommended for use in applications where a high quality of normal variates is required.

Addendum

In the mean time, Wallace has proposed a modification of his algorithm (see ftp.cs.monash.edu.au in directory pub/csw) that needs only a few additional uniform variates per pass, is only slightly slower than the original version and seems to have the same quality as the version where the elements of the pool are treated in a totally randomized order. Thus, this version avoids the "big" deviations reported above, but smaller deviations from the expected behavior are still observable.

References

[1] R. P. Brent. A fast vectorised implementation of Wallace's normal random number generator. Technical Report TR-CS-97-07, Department of Computer Science, The Australian National University, Canberra 0200 ACT, Australia, Apr. 1997.

[2] R. B. D'Agostino and M. A. Stephens. *Goodness–of–fit techniques.* Marcel Dekker, 1986.

[3] L. Devroye. *Non-Uniform Random Variate Generation.* Springer-Verlag, 1986.

[4] J. F. Fernández and J. Rivero. Fast algorithms for random numbers with exponential and normal distributions. *Computers in Physics*, 10(1), Jan/Feb 1996.

[5] B. Jung, H. Lenhof, P. Müller, and C. Rüb. Langevin dynamics simulations of macromolecules on parallel computers. *Macromol. Theory Simul.*, pages 507–521, 1997.

[6] D. E. Knuth. *The Art of Computer Programming, Volume 2: Seminumerical Algorithms.* Second Edition, Addison-Wesley, Reading, 1981.

[7] C. S. Wallace. Source files are available by anonymous ftp at ftp.cs.monash.edu.au in directory pub/csw.

[8] C. S. Wallace. Fast pseudorandom generators for normal and exponential variates. *ACM Transactions on Mathematical Software*, 22(1):119–127, Mar. 1996.

I N F O R M A T I K

Below you find a list of the most recent technical reports of the Max-Planck-Institut für Informatik. They are available by anonymous ftp from `ftp.mpi-sb.mpg.de` under the directory `pub/papers/reports`. Most of the reports are also accessible via WWW using the URL `http://www.mpi-sb.mpg.de`. If you have any questions concerning ftp or WWW access, please contact `reports@mpi-sb.mpg.de`. Paper copies (which are not necessarily free of charge) can be ordered either by regular mail or by e-mail at the address below.

Max-Planck-Institut für Informatik
Library
attn. Birgit Hofmann
Im Stadtwald
D-66123 Saarbrücken
GERMANY
e-mail: `library@mpi-sb.mpg.de`

MPI-I-98-2-012	G. Delzanno, A. Podelski	Model Checking Infinite-state Systems in CLP
MPI-I-98-2-011	A. Degtyarev, A. Voronkov	Equality Reasoning in Sequent-Based Calculi
MPI-I-98-2-010	S. Ramangalahy	Strategies for Conformance Testing
MPI-I-98-2-009	S. Vorobyov	The Undecidability of the First-Order Theories of One Step Rewriting in Linear Canonical Systems
MPI-I-98-2-008	S. Vorobyov	AE-Equational theory of context unification is Co-RE-Hard
MPI-I-98-2-007	S. Vorobyov	The Most Nonelementary Theory (A Direct Lower Bound Proof)
MPI-I-98-2-006	P. Blackburn, M. Tzakova	Hybrid Languages and Temporal Logic
MPI-I-98-2-005	M. Veanes	The Relation Between Second-Order Unification and Simultaneous Rigid E-Unification
MPI-I-98-2-004	S. Vorobyov	Satisfiability of Functional+Record Subtype Constraints is NP-Hard
MPI-I-98-2-003	R.A. Schmidt	E-Unification for Subsystems of S4
MPI-I-98-1-019		2nd Workshop on Algorithm Engineering WAE '98 - Proceedings
MPI-I-98-1-018	D. Dubhashi, D. Ranjan	On Positive Influence and Negative Dependence
MPI-I-98-1-017	A. Crauser, P. Ferragina, K. Mehlhorn, U. Meyer, E. Ramos	Randomized External-Memory Algorithms for Some Geometric Problems
MPI-I-98-1-016	P. Krysta, K. Loryś	New Approximation Algorithms for the Achromatic Number
MPI-I-98-1-015	M.R. Henzinger, S. Leonardi	Scheduling Multicasts on Unit-Capacity Trees and Meshes
MPI-I-98-1-014	U. Meyer, J.F. Sibeyn	Time-Independent Gossiping on Full-Port Tori
MPI-I-98-1-013	G.W. Klau, P. Mutzel	Quasi-Orthogonal Drawing of Planar Graphs
MPI-I-98-1-012	S. Mahajan, E.A. Ramos, K.V. Subrahmanyam	Solving some discrepancy problems in NC*
MPI-I-98-1-011	G.N. Frederickson, R. Solis-Oba	Robustness analysis in combinatorial optimization
MPI-I-98-1-010	R. Solis-Oba	2-Approximation algorithm for finding a spanning tree with maximum number of leaves
MPI-I-98-1-009	D. Frigioni, A. Marchetti-Spaccamela, U. Nanni	Fully dynamic shortest paths and negative cycle detection on diagraphs with Arbitrary Arc Weights
MPI-I-98-1-008	M. Jünger, S. Leipert, P. Mutzel	A Note on Computing a Maximal Planar Subgraph using PQ-Trees

MPI-I-98-1-007	A. Fabri, G. Giezeman, L. Kettner, S. Schirra, S. Sch'onherr	On the Design of CGAL, the Computational Geometry Algorithms Library
MPI-I-98-1-006	K. Jansen	A new characterization for parity graphs and a coloring problem with costs
MPI-I-98-1-005	K. Jansen	The mutual exclusion scheduling problem for permutation and comparability graphs
MPI-I-98-1-004	S. Schirra	Robustness and Precision Issues in Geometric Computation
MPI-I-98-1-003	S. Schirra	Parameterized Implementations of Classical Planar Convex Hull Algorithms and Extreme Point Compuations
MPI-I-98-1-002	G.S. Brodal, M.C. Pinotti	Comparator Networks for Binary Heap Construction
MPI-I-98-1-001	T. Hagerup	Simpler and Faster Static AC^0 Dictionaries
MPI-I-97-2-012	L. Bachmair, H. Ganzinger, A. Voronkov	Elimination of Equality via Transformation with Ordering Constraints
MPI-I-97-2-011	L. Bachmair, H. Ganzinger	Strict Basic Superposition and Chaining
MPI-I-97-2-010	S. Vorobyov, A. Voronkov	Complexity of Nonrecursive Logic Programs with Complex Values
MPI-I-97-2-009	A. Bockmayr, F. Eisenbrand	On the Chvátal Rank of Polytopes in the 0/1 Cube
MPI-I-97-2-008	A. Bockmayr, T. Kasper	A Unifying Framework for Integer and Finite Domain Constraint Programming
MPI-I-97-2-007	P. Blackburn, M. Tzakova	Two Hybrid Logics
MPI-I-97-2-006	S. Vorobyov	Third-order matching in $\lambda \rightarrow$-Curry is undecidable
MPI-I-97-2-005	L. Bachmair, H. Ganzinger	A Theory of Resolution
MPI-I-97-2-004	W. Charatonik, A. Podelski	Solving set constraints for greatest models
MPI-I-97-2-003	U. Hustadt, R.A. Schmidt	On evaluating decision procedures for modal logic
MPI-I-97-2-002	R.A. Schmidt	Resolution is a decision procedure for many propositional modal logics
MPI-I-97-2-001	D.A. Basin, S. Matthews, L. Viganò	Labelled modal logics: quantifiers
MPI-I-97-1-028	M. Lermen, K. Reinert	The Practical Use of the $\mathcal{A}^*$ Algorithm for Exact Multiple Sequence Alignment
MPI-I-97-1-027	N. Garg, G. Konjevod, R. Ravi	A polylogarithmic approximation algorithm for group Steiner tree problem
MPI-I-97-1-026	A. Fiat, S. Leonardi	On-line Network Routing - A Survey
MPI-I-97-1-025	N. Garg, J. Könemann	Faster and Simpler Algorithms for Multicommodity Flow and other Fractional Packing Problems
MPI-I-97-1-024	S. Albers, N. Garg, S. Leonardi	Minimizing Stall Time in Single and Parallel Disk Systems
MPI-I-97-1-023	S.A. Leonardi, A.P. Marchetti-Spaccamela	Randomized on-line call control revisited
MPI-I-97-1-022	E. Althaus, K. Mehlhorn	Maximum Network Flow with Floating Point Arithmetic
MPI-I-97-1-021	J.F. Sibeyn	From Parallel to External List Ranking
MPI-I-97-1-020	G.S. Brodal	Finger Search Trees with Constant Insertion Time
MPI-I-97-1-019	D. Alberts, C. Gutwenger, P. Mutzel, S. Näher	AGD-Library: A Library of Algorithms for Graph Drawing
MPI-I-97-1-018	R. Fleischer	On the Bahncard Problem
MPI-I-97-1-017	S. Albers, M.R. Henzinger	Exploring Unknown Environments
MPI-I-97-1-016	M. Thorup	Faster deterministic sorting and priority queues in linear space
MPI-I-97-1-015	M. Jünger, S. Leipert, P. Mutzel	Pitfalls of using PQ–Trees in automatic graph drawing