ulm university universität
uulm

An empirical assessment of local and population based search methods with different degrees of pseudorandomnes

M. Maucher, U. Schöning, H.A. Kestler

Ulmer Informatik-Berichte

Nr. 2008-07
Juni 2008

An empirical assessment of local and population based search methods with different degrees of pseudorandomness

M. Maucher* U. Schöning† H.A. Kestler‡

Abstract

When designing and analyzing randomized algorithms, one usually assumes that a sequence of uniformly distributed, independent random variables is available as a source of randomness. Implementing these algorithms, however, one has to use pseudorandom numbers. The quality of the used pseudorandom number generator may severely influence the quality of an algorithm's output. We examined the effect of using low quality pseudorandom numbers on the performance of different search heuristics like Simulated Annealing and a basic evolutionary algorithm.

Keywords: pseudorandom number generators, simulated annealing, evolutionary algorithms

1 Introduction

Many search heuristics use randomness to find solutions. Karloff et al. [7] and Bach [4] showed that some algorithms, like QuickSort or primality testing, have bad running times or even yield wrong results when used with unsuitable pseudo-random number generators. Karloff et al. also showed that a good average case behavior can be guaranteed by the right kind of pseudorandom generator. In the case of QuickSort, they prove a good average case behavior when using an explicit polynomial generator (see Section 3). Meysenburg showed that a simple evolutionary algorithm's solution did not significally depend on the choice of the random number generator [14]. Tompkins and Hoos showed that stochastic local search methods for the satisfiability problem seem not to be influenced by the quality of the pseudorandom number generator [17].

We were interested in the effects of generators with very low quality on local search heuristics, especially Simulated Annealing, and population based heuristics like evolutionary algorithms. To this end, we conducted some experiments where we gradually decreased the quality of our pseudorandom number generator, and tested if this decrease in quality directly affected the output of our search heuristics.

*Institute of Theoretical Computer Science, markus.maucher@uni-ulm.de, †Institute of Theoretical Computer Science, uwe.schoening@uni-ulm.de, ‡Institute of Neural Information Processing and Internal Medicine I, hans.kestler@uni-ulm.de, Ulm University, Germany

2 Search Heuristics

2.1 Simulated Annealing

The Simulated Annealing heuristic has been popularized by Kirkpatrick in 1983 [9] and is based on the Metropolis-Hastings algorithm [13] that dates back to 1953. Since then, it has been widely used for various optimization problems. It simulates the cooling of physical matter, where a state changes to a state with higher energy only with a certain probability that decreases during the proceeding of the algorithm. When optimizing a function f, we interpret $f(x)$ as the energy of state x. Beginning in a randomly chosen state, this state slightly changes step by step, i.e. transforms into a neighboring state. This transformation prefers new states with lower energy. The chance that a state with higher energy is accepted depends on a temperature parameter T, which is gradually decreased. This way, the acceptance probability for higher energy states decreases and the system gradually tends to stick to lower energy states.

Simulated Annealing is a typical local search heuristic: Beginning at one point in the search space, the algorithm moves through the search space, trying to find a global minimum.

Input: Function f
Output: x with f(x) as small as possible
Initialize temperature T;
$x \leftarrow$ random;
m $\leftarrow$ x;
while $T > T_0$ **do**
  y $\leftarrow$ random neighbour of x;
  if $f(y) < f(x)$ **then**
    x $\leftarrow$ y;
  else
    x $\leftarrow$ y with probability $e^{-\frac{f(y)-f(x)}{T}}$;
  end
  if $f(x) < f(m)$ **then**
    m $\leftarrow$ x;
  end
  decrease T;
end
output m;

Algorithm 1: Pseudocode of the Simulated Annealing heuristic.

For fixed temperature T, the algorithm simulates a Markov process where the variable x holds the random state. It can be shown [6] that this process limits to the Gibbs distribution where an element x occurs with probability

$$P(x) = \frac{e^{-\frac{f(x)}{T}}}{\sum_x e^{-\frac{f(x)}{T}}} \quad .$$

For $T \to \infty$, this distribution limits to the uniform distribution. For $T \to 0$, it limits to the uniform distribution on $\Omega = \{\mathrm{x} |\ f(x) = min_x f(x)\}$, the set of all global minima.

For low temperature T, however, such a Markov chain remains at *local* optima for many steps and only converges to the stationary distribution very slowly. To speed up convergence, Simulated Annealing starts with a high value of T and gradually lowers T. This random process does not have a stationary transition matrix like a Markov chain, but still converges to the uniform distribution on the global minima if $T(t) \to 0$ slowly enough. More precisely, let $T(t)$ be the temperature at iteration $t = 0, 1, 2, \ldots$ and let Ω_0 be the set of all global minima as defined above. Then the simulated annealing process converges to the uniform distribution on Ω_0 if $\lim_{t\to\infty} T(t) \to 0$ and $T(t) \in \Omega(\frac{1}{\log n})$ (see [6]).

2.2 Population based heuristics

Population based search heuristics try to optimize a function by searching at many objects of the search space simultaneously. They are usually inspired by populations found in nature. Genetic algorithms, for example, are inspired by evolution: An initial population changes during the course of time with the help of some basic operations:

- Selection: Each object of the population is evaluated with the help of a fitness function. Objects with a higher fitness are more likely to survive.
- Mutation: Some objects are slightly changed.
- Crossover: Pairs of parent objects are combined into new objects that resemble both parent objects.

Evolutionary algorithms use randomness at various places: The initial population is often chosen at random. The crossover and mutation operators are often applied to random objects, their probability usually depending on their individual fitness. When replacing unfit objects, we may choose these objects randomly, too. This way, the population is kept diverse and is prevented from concentrating around a local optimum; e.g. crossover with an unfit object might lead away from such a local optimum and help find the global optimum. The crossover itself requires random choices, too: Usually there are many ways to combine two parent objects. See Algorithm 2 for pseudocode of a basic genetic search heuristic.

Input: Function f
Output: x; the goal is to output an x with minimal $f(x)$.
Initialize population P;
repeat
  Replace unfit objects by mutations of fit objects;
  Replace unfit objects by crossover of fit objects;
until *termination condition* ;
Output best object found so far;

Algorithm 2: Pseudocode for a simple evolutionary algorithm.

In our experiments, we usually replaced the whole population in the mutation step. The individuals were chosen by a roulette wheel algorithm, an algorithm that chooses elements with a probability that depends on their fitness (for an implementation in *R*, see the appendix). In each loop, we checked that the best object so far was not removed from the population. In the crossover step, we replaced about 10% of the population by crossover offsprings. The algorithm terminated after a fixed number of loops.

3 Pseudorandom Generators

A pseudorandom number generator (short PRNG) is an algorithm that outputs a new number each time it is called. Its output depends on an internal state that is changed in a deterministic way each time a number is output. A PRNG is initialized with the help of a number or sequence called the seed, which is usually chosen at random (when reproducing a result however, we might as well reuse an experiment's seed). We will call the *output* $X_0, X_1, X_2, \ldots$ of a PRNG a *pseudorandom sequence*. A pseudorandom sequence, although deterministically created, should look like a random sequence of numbers. One way to formalize the notion of "looking random" is a statistical test.

A statistical test is based on a *null hypothesis* H_0 and a statistic $f : \mathbb{R}^k \to \mathbb{R}$. It measures the plausibility of the statistic's outcome $s = f(X_0, X_1, \ldots, X_k)$ under the assumption that the null hypothesis is true. To this end, the probability $p_0 = P[f(X_0, X_1, \ldots, X_k) > s \mid H_0]$ is computed. The input is accepted if p_0 is not too close to 0 or 1, i.e. if p_0 lies within a confidence intervall $[a, b] \subset [0, 1]$. Otherwise, the input is rejected. Typical values for the size of the confidence interval are 0.9, 0.95 or 0.99. In the case of pseudorandom number generators, H_0 is an assumption like "the numbers $X_0, X_1, \ldots, X_k$ are uniformly and independently distributed in $[0, 1)$", and if this sequence is accepted by a statistical test, we can say it "looks random", at least to that specific statistical test. Since every pseudorandom generator produces only a tiny fraction of the sequences a random process would produce, there is always a statistical test that rejects the output of a given pseudorandom generator (and accepts almost all random sequences). So it is only possible to construct a pseudorandom generator that can fool a given, fixed set of statistical tests. There is no solid criteria for which tests a "good" PRNG should pass. Commonly, one constrains to tests that are easy to implement and that check properties that are rather simple but considered important.

Examples for popular statistical tests are the Kolmogorov-Smirnov test, χ^2 test, poker test, run test or gap test. Many more popular tests exist, and depending on the application, some more exotic tests may be suited even better to determine which pseudorandom generator will work well with that application.

Two of the most desirable properties for pseudorandom numbers are *uniformity* (i.e. every single pseudorandom number should be distributed evenly among all possible values) and *k-wise independence* (see definition below). For these measures, we assume that the seed of the pseudorandom generator was chosen under the uniform distribution from the set of all possible seeds.

Definition 1 *A sequence of random numbers $X_1, X_2, \ldots$ is* k-wise independent *if any subsequence $X_{i_1}, \ldots, X_{i_k}$ of length k is independent, i.e. for any $x_1, \ldots, x_k$*

$$P[X_{i_1} = x_1, \ldots, X_{i_k} = x_k] = \prod_{j=1}^{k} P[X_{i_j} = x_j] \ .$$

For $k = 2$ we call such a sequence pairwise independent.

3.1 Classical Pseudorandom Generators

There exist many different kinds of pseudorandom number generators. We here list some that are or were used intensively in practice or that were used to theoretically analyze algorithmic behavior.

- A *linear congruential generator* with parameters *a*,*b* and *m* and seed $X_0 \in \{0, \ldots, m-1\}$ is defined by the recursion

  $$X_{n+1} = aX_n + b \bmod m \ .$$

  For further reference, see [10]. In our experiments, we always chose the parameters a and b in a way that the linear congruential generator had a period length of m. Additionally, we chose the parameters that resulted in the least compressible sequences. Compressibility was checked with the bzip2 software.

- An *explicit polynomial generator* of degree k with parameters $a_0, a_1, \ldots, a_k$ and m is defined by

  $$X_n = \sum_{i=0}^{k} a_i n^i \bmod m \ .$$

  Its use with the QuickSort algorithm is analyzed in [7]. An important property of explicit polynomial generators is k-wise independence. Within one period, any k output numbers of this generator are independent. The period length of such a generator is at most m, since $p(x) \equiv_m p(x+m)$ for any polynomial p.

- The Mersenne Twister [12] is a relatively recent pseudorandom number generator with an extremely huge period length of $2^{19937} - 1$ in the most commonly used version. It is based on a combination of linear recurrences and is currently used as the standard source of random numbers in many mathematical software projects like R or Maple [1]. Any 623 subsequent numbers of its output are independent and uniformly distributed.

- Marsaglia's Diehard suite [2] is a set of statistical tests that was published on CDROM in 1995, along with several files of bit sequences that pass these tests. These sequences were obtained by the bitwise XOR of several sequences, some of them obtained from physical devices, some of them from other sources like pseudorandom number generators or even an audio CD. This approach is based on the following fact: Let $X, Y \in \{0, 1\}$ be two independent random variables. Then $X \oplus Y$ is *uniformly* distributed if at least one of the two variables is uniformly distributed. This way, the bitwise XOR of several sequences is uniformly

distributed among the set of all bit sequences of the same length, if at least one of those sequences was uniformly distributed. Thus, one could hope to obtain a good pseudorandom sequence when forming the bitwise XOR of several sequences that are supposed to behave like true random numbers.

3.2 Quasi-random sequences

Quasi-Monte Carlo algorithms and their analysis do not depend on uniformity and independence of the "random" numbers they use. They rather need numbers that are spread over their domain evenly. A major discipline in the field of Quasi-Monte Carlo algorithms is the numerical integration of a function f. Here, the integral $\int_0^1 f(x)dx$ is approximated by the sum $\frac{1}{N}\sum_{i=1}^{n} f(x_i)$ (see [16]), where $x_1, x_2, \ldots, x_N$ is s sequence of random numbers. It can be shown that the difference between the integral and the approximating sum can be bounded from above by $V(f)D^*(x_1, \ldots, x_n)$, where $V(f)$ is the variation of f and D^* is the star discrepancy. Discrepany measures how evenly a set of points in a k-dimensional cube is distributed.

Definition 2 *Let $P := \{x_1, x_2, \ldots, x_N\} \subset [0,1)^d$. then the star discrepancy D^* of P is defined as*

$$D^*(P) := \sup_{x \in [0,1)^d} \left(\frac{|\{x_i \in P \mid \forall j . x_i^{(j)} < x^{(j)}\}|}{N} - \prod_{i=1}^{d} x^{(i)} \right) \quad ,$$

where $x^{(j)}$ denotes the j-th component of the vector x.

An example of a sequence with low star discrepancy is the van der Corput sequence.

Definition 3 *Let $n_k, \ldots, n_0$ be the b-ary representation of a number n, i.e. $n = \sum_{i=0}^{k} n_i b^i$, with $0 \leq n_i < b$ for all i. We then define*

$$\phi_b(n) = \sum_{j=0}^{\infty} n_i b^{-i-1} \quad .$$

The van der Corput sequence *in base b is defined as*

$$X_n = \phi_b(n) \quad .$$

Intuitively, ϕ_b takes the digits of a number n in b-ary representation, reverses their order and places them behind a decimal point. For example, $\phi_2(11001_2) = 0.10011_2$.

When tupels of higher dimension are needed, the van der Corput sequences can be generalized to sequences of k-tuples:

Definition 4 *A* Halton sequence *in the bases $b_1, \ldots, b_k$ is defined as*

$$X_n = (\phi_{b_1}(n), \ldots, \phi_{b_k}(n)) \quad .$$

A Halton sequence is just a simple composition of multiple van der Corput sequences with different bases.

4 Implementation

Our test cases were implemented in the statistical programming language *R* [3] and run on a system with *R* version 2.6.1 installed. We used R because it allows the quick development of code as well as an easy visualisation of data. Disadvantages of R include the absence of a call by reference and the fact that R is an interpreter. Both these facts make it somehow difficult to develop fast software. Due to these limitations, we implemented some time-consuming parts, e.g. the crossover operation for the evolutionary algorithm, in *C*.

5 Experimental Setup and Results

In our experiments, we were interested in the dependency between the quality of the pseudorandom generator and the quality of the solution given by two search heuristics, namely Simulated Annealing and an evolutionary algorithm.

One problem we used for our analysis was the Traveling Salesman Problem (short TSP).

Definition 5 *The* Traveling Salesman Problem *is defined as follows:*
Given a quadratic $n \times n$ matrix D of positive values (a distance matrix), what is the permutation $\pi \in S_n$ where

$$D_{\pi_n \pi_1} + \sum_{i=1}^{n-1} D_{\pi_i \pi_{i+1}}$$

is minimized?

Intuitively, The Traveling Salesman Problem asks for a tour that visits each of n nodes exactly once, then goes back to the initial node and minimizes the total cost of the tour. Costs for moving from any node i to a node j are given by the matrix entry D_{ij}, and to obatin the total cost of a tour, we can just add up the costs of the individual steps.

Another interesting set of test functions was published by DeJong [5] in 1975. The set was specially designed to measure the performance of search heuristics and consists of the following 5 test functions:

- $f_1(x_1, \ldots, x_k) = \sum_{i=1}^{k} x_i^2$, with $-5.12 \leq x_i \leq 5.12$.

  This function should not be a problem for an optimization algorithm. The function is very smooth and has only one local minimum, which is also the global minimum.

- $f_2(x_1, \ldots, x_k) = \sum_{i=1}^{k-1} \left(100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2\right)^2$, with $-5.12 \leq x_i \leq 5.12$.

  f_2 is rather hard to optimize: the points with good values lie on a thin line, with the function growing rapidly when moving away from that line.

- $f_3(x_1, \ldots, x_k) = 6k + \sum_{i=1}^{k} \lfloor x_i \rfloor$, with $-5.12 \leq x_i \leq 5.12$.

  This function consists of many plateaus, where almost every point is a local minimum. Search algorithms with small step sizes could have problems optimizing this function, due to difficulties finding a good direction.

- $f_4(x_1,\ldots,x_k) = \sum_{i=1}^{k} i x_i^4 + \mathcal{N}(0,1)$, with $-5.12 \leq x_i \leq 5.12$.

  Here, each time the function is evaluated, a new term is added, drawn from a normal distribution with mean 0 and variance 1.

- $f_5(x_1,x_2) = \left(0.002 + \sum_{j=1}^{25} \frac{1}{j + \sum_{i=1}^{2} (x_i - a_{ij})^6}\right)^{-1}$,
  with $-65536 \leq x_i \leq 65536$ and matrix
  $(a_{ij}) = \begin{pmatrix} -32 & -16 & 0 & 16 & 32 & \cdots & 0 & 16 & 32 \\ -32 & -32 & -32 & -32 & -32 & \cdots & 32 & 32 & 32 \end{pmatrix}$.

  This function has 25 local minima. It was designed to "trap" optimization algorithms in one local optimum to research if they could still find a global optimum after being trapped. In our experiments, neither Simulated Annealing nor the evolutionary strategy consistently found a value smaller than 500. So this function is not well suited to compare the influence of the random number generator.

One aspect of "quality" of a pseudorandom generator is its period length. In order to achieve pseudorandom sequences with scalable period length, we artificially shortened period lengths of PRNGs by counting the output numbers and resetting the seed after a fixed amount of output numbers. On the one hand, this method enabled us to compare various pseudorandom sequences with equal period lengths. On the other hand, it allowed us to scale the period length of pseudorandom sequences that had otherwise a very long period length. Corresponding *R* code can be found in Listing 2 in the appendix.

Our results will be shown in boxplot diagrams: The three horizontal lines of a box represent first quantile, median and third quantile. The ends of the whiskers represent minimum and maximum values, where a whisker's maximum length is 1.5 times the interquartile range (distance between first and third quartile). Any values outside of that range are plotted as individual points.

5.1 Simulated Annealing

Experiment 1

In Experiment 1, we ran the Simulated Annealing heuristic on a traveling salesman instance with 50 cities and measured the quality of the solutions, i.e. the length of the optimal tour found. The distance matrix D was symmetric, i.e. $D_{ij} = D_{ji}$ for all i, j, and its entries were chosen at random, distributed uniformly in $\{1,2,\ldots,40\}$. As our source of randomness, we used a Mersenne Twister, where we artificially reduced the period length to values ranging from 1003 to 512009.

The period lengths were chosen as prime numbers because we wanted to prevent moving in cycles as much as possible. Each run started at temperature 20. For each temperature, we executed 3000 iterations, then slightly decreased the actual temperature by multiplying it with the factor 0.97. The program ended after 100 different temperature values had been used. For each pseudorandom generator we used, the algorithm was run with 50 different seeds.

The pseudorandom numbers were used at two places: To compute a random neighbor permutation, we randomly swapped two elements of the current permutation, and to compute if a new permutation is accepted, we tested if a random number was smaller than $e^{-\Delta f/T}$.

Parameter	Value
Heuristic	Simulated Annealing
Input	symmetric 50×50 distance matrix
Run time	100 temperature values, 3000 iterations per temperature
Generators	Mersenne Twister with reduced period length (denoted by r-x with x=period length)
Seeds per generator	50

Table 1: Parameters of Experiment 1.

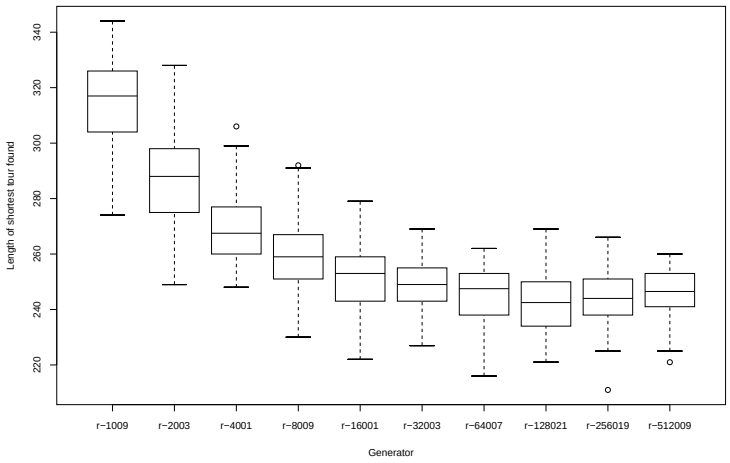

Figure 1: Simulated Annealing and the Traveling Salesman Problem (Experiment 1). Horizontal axis: Generator used. Vertical axis: Length of the shortest tour found with the help of that generator. For further parameters, see Table 1.

The optimal tour lengths we achieved can be seen in Figure 1. Increasing the period length had a remarkable effect on the quality of our solutions. Note that the total number of random numbers that were used in each run lies between 900,000 and 910,000.

Experiment 2

In Experiment 2, we tested the performance of Simulated Annealing on DeJong's test function 2 [5], which is defined as

$$f_2(x_1,\ldots,x_k) := \sum_{i=1}^{k-1} 100(x_{i+1}-x_i^2)^2 + (x_i-1)^2 \quad .$$

This function has a global minimum $f(1,\ldots,1) = 0$. In our experiment, we used dimension $k = 20$. The function was optimized in the range $[-5.12, 5.12]^{20}$.

The Simulated Annealing heuristic started at temperature 20. For some common parameters of this experiment, see Table 2. To obtain a neighbor of the actual state, we changed every component of the 20-dimensional vector by a pseudorandom number in the interval $[-0.5, 0.5]$. The source of randomness was a modified Mersenne

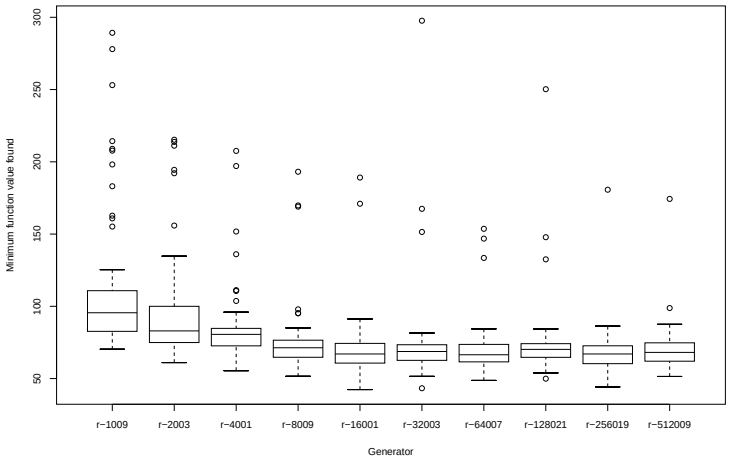

Figure 2: Solving DeJong's test function f_2 with Simulated Annealing (Experiment 2). Horizontal axis: Generator used. Vertical axis: Minimum value of f_2 found with the help of that generator. For further parameters, see Table 2.

Parameter	Value
Heuristic	Simulated Annealing
Run time	100 temperature values, 500 iterations per temperature
Generators	Mersenne Twister with reduced period length (denoted by r-x)
Seeds per generator	50

Table 2: Parameters of Experiment 2.

Twister, where the period length was artificially reduced to values ranging from 1003 to 512009. As in Experiment 1, we used primes as period lengths to avoid moving in cycles. For each run we measured the minimum function value $f_2(x)$ that was found in the interval $[-5.12, 5.12]^{20}$. The results of this experiment can be seen in Figure 2.

Experiment 3

In Experiment 3, the Simulated Annealing heuristic was again used to solve an instance of the Traveling Salesman Problem. The algorithm started at temperature 20, which was then gradually decreased by the factor 0.97.

This time we used van der Corput sequences and Halton sequences of dimension 2 as sources of randomness. Each of the van der Corput and Halton sequences was started at 50 different points, the Mersenne Twister was used with 50 different seeds. Since the roulette wheel algorithm does not need tuples from the random source, we used the Halton sequences in a simplified way and flattened them: Let $((x_{11}, x_{12}), (x_{21}, x_{22}), (x_{31}, x_{32}), \ldots)$ be a Halton sequence of dimension 2. Then we used the 1-dimensional sequence $(x_{11}, x_{12}, x_{21}, x_{22}, x_{31}, x_{32}, \ldots)$ instead. For the van der Corput sequences in base 8, we additionally permuted the digits of n, i.e instead of

Parameter	Value
Heuristic	Simulated Annealing
Input	symmetric 50×50 distance matrix
Run time	100 temperature values, 3000 iterations per temperature
Generators	van der Corput sequences of base 2 (vdc2) van der Corput sequences of base 8 (vdc8-px) 2-dimensional Halton sequences (Hal-xy) Mersenne Twister (MT)
Seeds per generator	50

Table 3: Parameters of Experiment 3.

ϕ_b in Definition 3, we used

$$\tilde{\phi}_b(n) = \sum_{j=0}^{\infty} p(n_i) b^{-i-1} \quad ,$$

where p was a permutation of $\{0,\ldots,7\}$. We used the permutations
$p_1 = (2\ 4\ 6\ 0\ 1\ 3\ 5\ 7)$,
$p_2 = (0\ 3\ 6\ 1\ 4\ 7\ 2\ 5)$,
$p_3 = (2\ 5\ 6\ 4\ 1\ 0\ 3\ 7)$ and
$p_4 = (3\ 6\ 4\ 5\ 1\ 7\ 2\ 0)$
(meaning that p_1 maps 0 to 2, 1 to 4, 2 to 6, ...).

The results of Experiment 3 can be seen in Figure 3. Here, "vdc" denotes the van der Corput sequence in base 2, "vdc8p1" to "vdc8p4" denote van der Corput sequences in base 8, with permutations 1 to 4, and "hal-xy" denote Halton sequences of bases x and y.

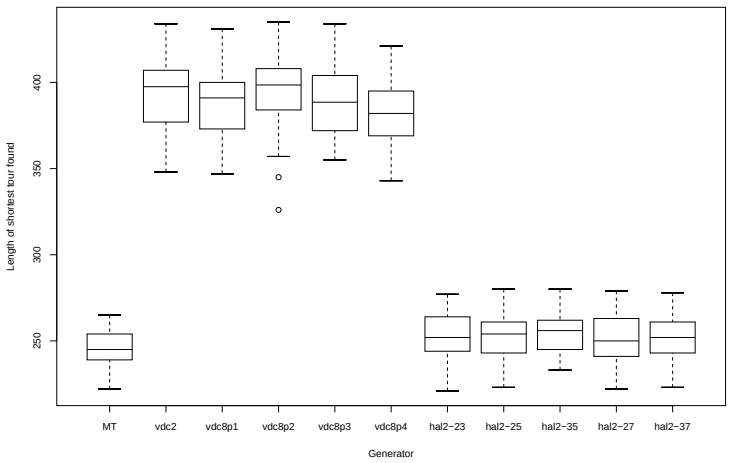

Figure 3: Traveling Salesman Problem with Simulated Annealing and quasirandom sequences (Experiment 3). Horizontal axis: Pseudorandom number generator we used. Vertical axis: Length of the shortest tour found with the help of that generator. For further parameters, see Table 3.

With the use of van der Corput sequences, the algorithm consistently found worse solutions than with the Mersenne Twister, whereas the use of Mersenne Twister and Halton sequences both led to good solutions that did not significantly differ from each other.

Experiment 4

In Experiment 4, we tried to measure the influence of k-wise independence on the quality of the Simulated Annealing heuristic. We fixed the range m of an explicit polynomial generator at $m = 10000$ and varied its degree from $k = 2$ up to $k = 10$.

Parameter	Value
Heuristic	Simulated Annealing
Input	symmetric 50×50 distance matrix
Run time	100 temperature values, 3000 iterations per temperature
Generators	Polynomial generators with period length 10000 and degree $k \in \{2, \ldots, 10\}$ (denoted by p-k) Mersenne Twister
Seeds per generator	60 (20 for each parameter set)

Table 4: Parameters of Experiment 4.

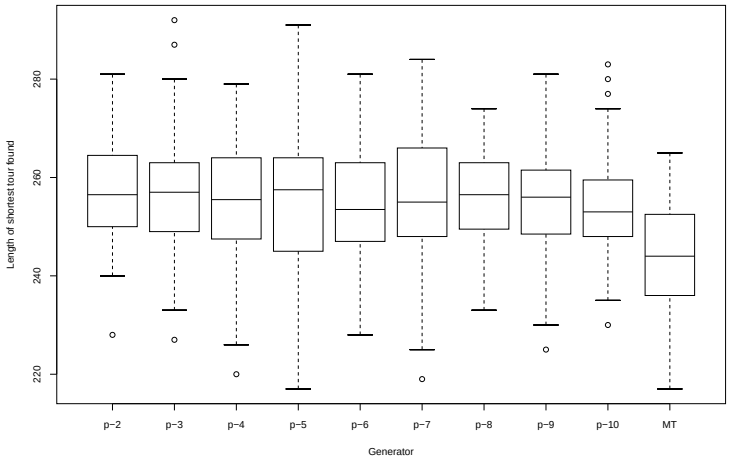

Figure 4: Traveling Salesman Problem with Simulated Annealing and polynomial generators (Experiment 4). Horizontal axis: Pseudorandom number generator we used. Vertical axis: Length of the shortest tour found with the help of that generator. For further parameters, see Table 4.

To find good coefficient sets for that generator, we used the following approach: For each degree k, we created 200 coefficient sets at random, i.e. we chose the coefficients $a_0, \ldots, a_k$. For each of these coefficient set, we then output 100000 numbers with a polynomial generator that used these coefficients. Each of these output

sequences was then compressed individually with the bzip2 algorithm. For our experiment, we only chose the three coefficient sets that lead to the three longest files after compression. That way, we tried to avoid sequences with obvious regularities.

Most other parameters were chosen as for Experiment 1. For an overview of the parameters, see Table 4.

The results of the experiment are shown in Figure 4. Increasing the degree of the polynomial generator did not significantly increase the quality of the solution. Comparing with the results of Experiment 1, the solutions achieved with the polynomial generators are comparable to those achived when using a linear congruential generator with period length of 8000 or 16 thousand. Since our polynomial generators had a modulus $m = 10000$, and thus most likely a period length of 10000, period length seems to have more influence on the result than k-wise independence of the pseudo-random numbers.

5.2 Population based heuristics

Experiment 5

In Experiment 5 we ran a simple evolutionary algorithm on the same instance of the TSP problem as in Experiment 1. In each step, the fitness values of the population were calculated and the next population then chosen by a roulette wheel algorithm (see Listing 3 in the appendix). Then every element of the population was mutated, switching two permutation elements. From this new population, we then created 10 new elements by combining two random elements with the edge-3 crossover operator [8]. These new elements replaced one of their parents each.

Parameter	Value
Heuristic	Evolutionary Algorithm
Input	symmetric 50×50 distance matrix
Population size	100
Run time	3000 iterations
Generators	linear congruential generators (l-x) Mersenne Twister (MT) Marsaglia's Diehard sequence (D)
Seeds per generator	60 (20 for each of three parameter sets for lcg)

Table 5: Parameters of Experiment 5.

As source of randomness, we used linear congruential generators that had maximum period lengths from 1000 up to 512000. Their parameters were chosen such that their output could not be well compressed by the bzip2 program. To this end, we randomly chose parameters that could guarantee maximum period lengths, then compressed sequences obtained from a linear congruential generator with these parameters and compressed them. For each period length, we used the three parameter sets that resulted in the 3 biggest files after compression. For each of these parameter sets, we used 20 different seeds. For comparison, we also used the Mersenne Twister and Marsaglia's random sequence.

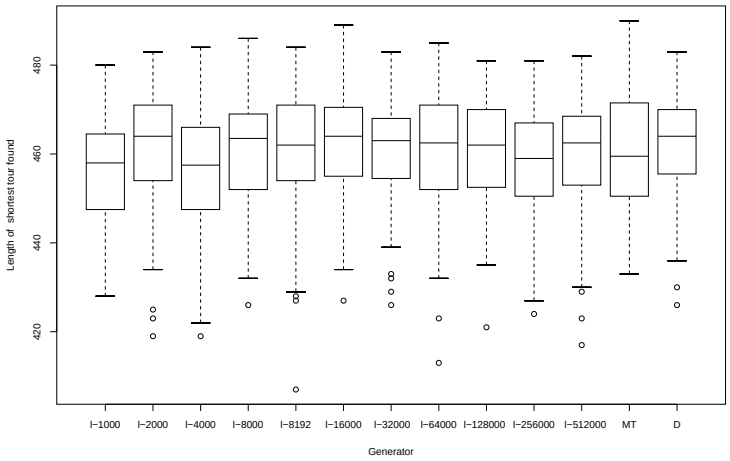

Figure 5: Solving the Traveling Salesman Problem wit a population based approach (Experiment 5). Horizontal axis: Pseudorandom number generator we used. Vertical axis: Length of the shortest tour found with the help of that generator. For further parameters, see Table 5.

Figure 5 shows the results of this experiment. The solution quality does not seem to depend on the period length of the linear congruential generators.

Experiment 6

Experiment 6 was very similar to Experiment 5: We used the same input, the same population size and the same mutation/crossover procedure. The fitness function for the roulette wheel algorithm was normalized such that the object with the worst fitness was chosen with probability 0. In this experiment, we used van der Corput sequences and Halton sequences of dimension 2 as sources of randomness.

Parameter	Value
Heuristic	Evolutionary Algorithm
Input	symmetric 50×50 distance matrix
Population size	100
Run time	10,000 iterations
Generators	van der Corput sequences of base 2 (vdc2) van der Corput sequences of base 8 (vdc8-px) 2-dimensional Halton sequences (hal-xy) Mersenne Twister (MT)
Seeds per generator	50

Table 6: Parameters of Experiment 6.

Since the roulette wheel algorithm does not need tuples from the random source, we flattened the Halton sequences as described in Experiment 3. The van der Corput sequences in base 8 used permutations, see Experiment 3 for further details.

The results of Experiment 6 can be seen in Figure 6. With the use of van der Corput sequences, the algorithm consistently found better solutions than with any other

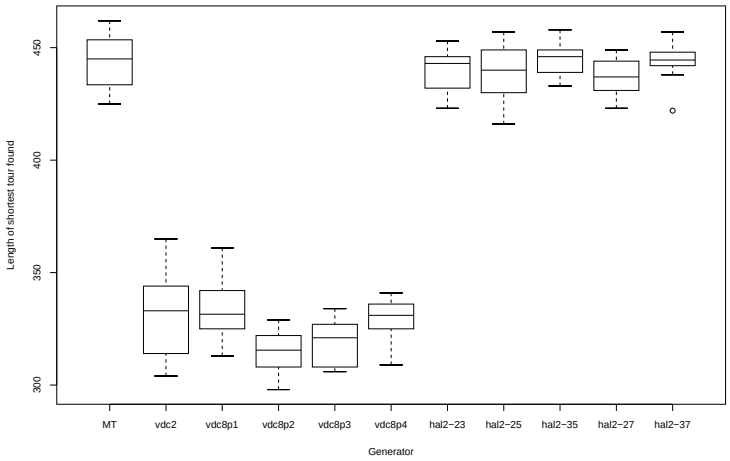

Figure 6: Solving the TSP with a population based approach, using quasi-random sequences (Experiment 6). Horizontal axis: Pseudorandom number generator we used. Vertical axis: Length of the shortest tour found with the help of that generator. For further parameters, see Table 6.

sources of randomness we used.

Experiment 7

In Experiment 7, we wanted to find an explanation why the use of van der Corput sequences in Experiment 6 lead to better solutions. Especially, we wanted to know if the crossover/mutation steps or the roulette wheel were improved by using quasirandom sequences. To this end, we provided the roulette wheel algorithm and the mutation/crossover steps with numbers from different generators. In a first test, the mutation and crossover steps used a van der Corput sequence while the roulette wheel used numbers from a Mersenne Twister. In a second test, mutation and crossover used numbers from a Mersenne Twister and the roulette wheel was run with a van der Corput sequence.

Parameter	Value
Heuristic	Evolutionary Algorithm
Input	symmetric 50×50 distance matrix
Population size	100
Run time	10,000 iterations
Generators	Mersenne Twister (MT) Combination of van der Corput sequences and Mersenne Twister: M-vdc... : van der Corput seq. for mutation/crossover R-vdc... : van der Corput seq. for roulette wheel
Seeds per generator	50

Table 7: Parameters of Experiment 7.

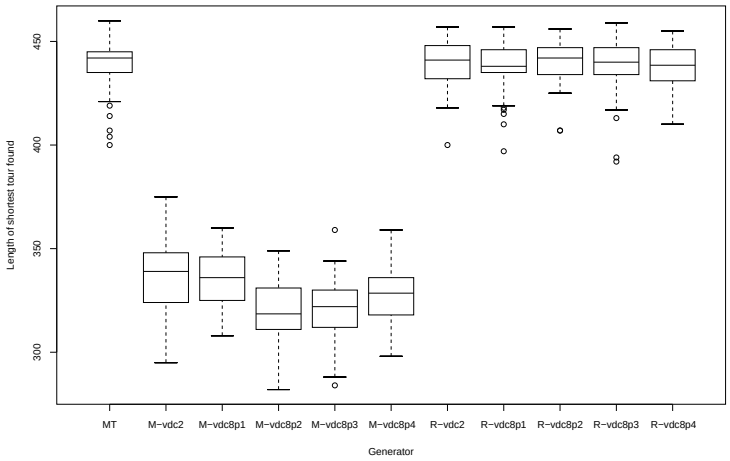

Figure 7: A population based approach, using different pseudorandom generators (quasirandom sequences and a Mersenne Twister) for mutation and selection (Experiment 7). Horizontal axis: Pseudorandom number generator we used – "M-.." used quasiradnom sequences for mutation and crossover, "R-.." used quasirandom sequences for roulette wheel selection. Vertical axis: Length of the shortest tour found with the help of that combination of generators. For further parameters, see Table 7.

The results of Experiment 7 can be seen in Figure 7. Cases where the mutation and crossover steps used van der Corput sequences are denoted by "M-...", cases where the roulette wheel used van der Corput sequences are denoted by "R-...", and *MT* denotes the case where only the Mersenne Twister was used. The roulette wheel seems not to be sensitive to the generator choice. Providing the mutation and crossover steps with the quasirandom numbers lead to better results.

More experiments with DeJong's test functions

We conducted some more experiments with the functions f_1 to f_4 from DeJong's test function suite.

Both Simulated Annealing and the evolutionary algorithm were run with two different neighbor resp. mutation heuristics: Adding a uniformly distributed value to each component of a vector or adding a normally distributed value to each vector component. Simulated Annealing was run with a Mersenne Twister with reduced period lengths, while the evolutionary algorithm was run with linear congruential generators with varying period lengths from 1000 to 512000. Functions 1 to 4 were solved for dimension $k = 20$.

- For solving f_1, the evolutionary algorithm had a population size of 20 and was run for 1000 iterations. Simulated Annealing was run for 500 steps per temperature and 100 different temperature values.
- When solving f_2, the evolutionary algorithm had a population size of 40 and was run for 2000 iterations. Simulated Annealing was run for 500 steps per temperature and 100 different temperature values.

Heuristic	Test function	Range of means	Range of mean/sd
SA(unif.)	1	2.86 - 3.10	0.16 - 0.20
SA(norm.)	1	2.92 - 3.15	0.16 - 0.21
SA(unif.)	2	68 - 117	0.25 - 0.50
SA(norm.)	2	28.9 - 42.6	0.076 - 0.60
SA(unif.)	3	11.7 - 18.4	0.15 - 0.55
SA(norm.)	3	11.60 - 18.7	0.15 - 0.65
SA(unif.)	4	1.05 - 1.38	0.31 - 0.49
SA(norm.)	4	1.13 - 1.70	0.30 - 0.51
EA(unif.)	1	4.14 - 4.81	0.13 - 0.19
EA(norm.)	1	3.76 - 5.51	0.14 - 0.20
EA(unif.)	2	3.63 - 3.92	0.13 - 0.18
EA(norm.)	2	3.72 - 5.08	0.13 - 0.18
EA(unif.)	3	7.65 - 8.88	0.10 - 0.15
EA(norm.)	3	6.85 - 8.38	0.08 - 0.17
EA(unif.)	4	2.02 - 2.30	0.30 - 0.41
EA(norm.)	4	2.07 - 2.50	0.29 - 0.38

Table 8: Experiments with Dejong's test suite. Values are rounded to 2 decimal digits. Heuristics include Simulated Annealing (SA) and an evolutionary algorithm (EA) with uniformly resp. normally distributed neigbor/mutation function. For each generator, the mean result and the quotient mean result/standard deviation were calculated. In the table, we list the minimum and maximum values of these characteristics.

- For f_3, the evolutionary algorithm had a population size of 40 and was run for 2000 iterations. Simulated Annealing was run for 500 steps per temperature and 100 different temperature values.
- For f_4, the evolutionary algorithm had a population size of 40 and was run for 2000 iterations. Simulated Annealing was run for 3000 steps per temperature and 100 different temperature values.

The results are summarized in Table 8.

5.3 Discussion

For the Simulated Annealing heuristic, the quality of the solution for the traveling salesman problem depended mainly on the period length of the pseudorandom generators we used. The results we received when using simple linear congruential generators were comparable with those we received when using "high end" generators like the Mersenne Twister or Marsaglia's diehard sequence, when we artificially reduced the period lengths of those high-end generators to match those of the LCGs.

When using quasirandom numbers, Halton sequences of dimension 2 lead to results comparable to the results when using a Mersenne Twister. Van der Corput sequences however led to drastically inferior results and should thus not be used in conjunction with Simulated Annealing and the Traveling Salesman Problem.

The population-based heuristic seemed to be very robust with respect to the quality of the pseudorandom generator we used (Fig. 5). It's a common approach to restart a search heuristic a fixed number of times and then take the best result achieved so far. Therefore, it is always interesting to look at the best result achieved for a specific pseudorandom generator. Considering only the minimum value over the 50 repetitions, the renowned Mersenne Twister was outclassed (although not by much) by most linear congruential generators, even by that with period length 1000.

An astonishing result has been achieved when running the evolutionary algorithm with quasi-random sequences: When solving the traveling salesman problem, the use of van der Corput sequences lead to much better results than the other generators we used (Fig. 6). They were even better than Halton sequences, although Halton sequences are just a generalized version of van der Corput sequences. This result was not expected, especially because the code for finding a neighboring state in the Simulated Annealing heuristic in Experiment 3 and the code for mutating an individual in Experiment 6 where identical.

When optimizing the TSP with an evolutionary strategy, the mutation and crossover steps were identified in Experiment 6 as the parts that benefit from the use of quasirandom sequences, whereas the roulette wheel selection part seemed not to be influenced significantly by this choice of the generator. This gives rise to the assumption that for the elements of a new generation, equidistribution is a very important property. Note that in the case of evolutionary algorithms, this means equidistribution in the neighborhood of the old generation, not equidistribution in the whole search space.

6 Outlook

So far, we investigated the effect of a pseudorandom number generator's period length on the quality of two different random search heuristics. However, there are still a few things left that we aim to do in the near future.

Period length of the PRNG seems to influence the quality of the Simulated Annealing heuristic, at least for some optimization problems, while k-wise independence seems to have no influence. On the other hand, evolutionary algorithms seem to be susceptible to the use of quasi-random sequences, while period length seemed to have no influence. We want to examine more criteria that can be used to measure the quality of a pseudorandom source, and investigate their effects, and we will also try to find some theoretical foundations as to why these effects happen.

It appears that some functions need "better" random numbers for finding a good solution, while other functions can also be optimized with low quality randomness. We want to further investigate this topic and isolate some characteristics that make a function easy or hard to optimize with somehow limited randomness.

Furthermore, we want to investigate the effects of bad random number quality on other search heuristics, like swarm algorithms or ant colony optimization.

Acknowledgements

We thank A. Winterhof for interesting and helpful discussions during his stay in Ulm. The support of the Graduate School on Mathematical Analysis of Evolution, Informa-

tion and Complexity funded by the State of Baden-Württemberg is gratefully acknowledged. We further acknowledge the funding by the Stifterverband für die Deutsche Wissenschaft (HAK) and by the Deutsche Forschungsgemeinschaft (US, HAK).

References

[1] Maple, www.maplesoft.com.

[2] The Marsaglia Random Number CDROM including the Diehard Battery of Tests of Randomness, http://stat.fsu.edu/pub/diehard/.

[3] The R Project for Statistical Computing, http://www.r-project.org/.

[4] E. Bach. Realistic analysis of some randomized algorithms. *J. Comput. Syst. Sci.*, 42(1):30–53, 1991.

[5] K. DeJong. *An Analysis of the Behavior of a Class of Genetic Adaptive Systems*. PhD thesis, Dept. of Electrical Eng. and Computer Science, Univ. of Michigan, 1975.

[6] S. Geman and D. Geman. Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6:721–741, 1984.

[7] H. Karloff and P. Raghavan. Randomized algorithms and pseudorandom numbers. In *STOC '88: Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 310–321, New York, NY, USA, 1988. ACM.

[8] M. Keith and W. Darrell. Genetic operators, the fitness landscape and the traveling salesman problem, 1992.

[9] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science, Number 4598, 13 May 1983*, 220, 4598:671–680, 1983.

[10] D. E. Knuth. *Art of Computer Programming, Volume 2: Seminumerical Algorithms (3rd Edition)*. Addison-Wesley Professional, November 1997.

[11] J. B. Kruskal. Extremely portable random number generator. *Commun. ACM*, 12(2):93–94, 1969.

[12] M. Matsumoto and T. Nishimura. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Trans. Model. Comput. Simul.*, 8(1):3–30, January 1998.

[13] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller. Equation of state calculations by fast computing machines. *The Journal of Chemical Physics*, 21(6):1087–1092, 1953.

[14] M. M. Meysenburg. The Effect of Pseudo-Random Number Generator Quality on the Performance of a Simple Genetic Algorithm. Master's thesis, University of Idaho, 1997.

[15] K. Mulmuley. *Computational Geometry: An Introduction through Randomized Algorithms*. Prentice-Hall, 1994.

[16] H. Niederreiter. *Random number generation and quasi-Monte Carlo methods*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1992.

[17] D. A. D. Tompkins and H. H. Hoos. On the quality and quantity of random decisions in stochastic local search for sat. In L. Lamontagne and M. Marchand, editors, *Canadian Conference on AI*, volume 4013 of *Lecture Notes in Computer Science*, pages 146–158. Springer, 2006.

A Implementation Details

Listing 1 shows the basic implementation of a class `rng`, along with a subclass `reference`. The global object `rng.reference` is then created and a function `reference.getNext` defined, which gives us one random number each time we call it. The last output is stored in the global object, just in case we want to reuse it.

```
setClass("rng",
  representation(
    seed="numeric", range="numeric", state="numeric"
  )
);
setClass("reference",representation(),contains="rng")

rng.reference <- new("reference");

reference.getNext <- function() {
  rng.reference@state <<- runif(1,min=0,max=1);
  return(rng.reference@state);
}
```

Listing 1: The reference RNG, which uses R's builtin Mersenne Twister.

In order to see if the heuristics depended on the PRNG's period length, we artificially shortened the period lengths of some sophisticated PRNGs like the Mersenne Twister or Marsaglia's CD-ROM sequence. An example for the Mersenne Twister is shown in listing 2. We simply count the number of output numbers and reset the seed when the output has reached a given length.

```
setClass("mrepeater",representation(length="numeric"),contains="rng");

rng.mrepeater <- new("mrepeater");

mrepeater.getNext <- function() {
  rng.mrepeater@state <<- (rng.mrepeater@state%%rng.mrepeater@length)+1;
  if(rng.mrepeater@state == 1)
    set.seed(rng.mrepeater@seed);
  return(runif(1));
}
```

Listing 2: A Mersenne random number repeater.

Listing 3 shows how we implemented the roulette wheel used in the evolutionary algorithm. `fitt` is a vector of fitness values, *number* is the number of indexes we need, and the optional argument `getNext` is a function that gives us pseudorandom

numbers. `roulette` returns a vector of indexes, which we use to select objects from our population.

```
roulette <- function(fitt, number, getNext=reference.getNext){
  fit       <- fitt / sum(fitt)
  fit       <- cumsum(fit)
  selection <- double(number)

  for (i in 1:number){
    selection[i] <- which(fit >= getNext() )[1]
  }

  return(selection)
}
```

Listing 3: The roulette wheel selection procedure.

Liste der bisher erschienenen Ulmer Informatik-Berichte

Einige davon sind per FTP von `ftp.informatik.uni-ulm.de` erhältlich
Die mit * markierten Berichte sind vergriffen

List of technical reports published by the University of Ulm

Some of them are available by FTP from `ftp.informatik.uni-ulm.de`
Reports marked with * are out of print

91-01 *Ker-I Ko, P. Orponen, U. Schöning, O. Watanabe*
Instance Complexity

91-02* *K. Gladitz, H. Fassbender, H. Vogler*
Compiler-Based Implementation of Syntax-Directed Functional Programming

91-03* *Alfons Geser*
Relative Termination

91-04* *J. Köbler, U. Schöning, J. Toran*
Graph Isomorphism is low for PP

91-05 *Johannes Köbler, Thomas Thierauf*
Complexity Restricted Advice Functions

91-06* *Uwe Schöning*
Recent Highlights in Structural Complexity Theory

91-07* *F. Green, J. Köbler, J. Toran*
The Power of Middle Bit

91-08* *V.Arvind, Y. Han, L. Hamachandra, J. Köbler, A. Lozano, M. Mundhenk, A. Ogiwara, U. Schöning, R. Silvestri, T. Thierauf*
Reductions for Sets of Low Information Content

92-01* *Vikraman Arvind, Johannes Köbler, Martin Mundhenk*
On Bounded Truth-Table and Conjunctive Reductions to Sparse and Tally Sets

92-02* *Thomas Noll, Heiko Vogler*
Top-down Parsing with Simulataneous Evaluation of Noncircular Attribute Grammars

92-03 *Fakultät für Informatik*
17. Workshop über Komplexitätstheorie, effiziente Algorithmen und Datenstrukturen

92-04* *V. Arvind, J. Köbler, M. Mundhenk*
Lowness and the Complexity of Sparse and Tally Descriptions

92-05* *Johannes Köbler*
Locating P/poly Optimally in the Extended Low Hierarchy

92-06* *Armin Kühnemann, Heiko Vogler*
Synthesized and inherited functions -a new computational model for syntax-directed semantics

92-07* *Heinz Fassbender, Heiko Vogler*
A Universal Unification Algorithm Based on Unification-Driven Leftmost Outermost Narrowing

92-08* *Uwe Schöning*
On Random Reductions from Sparse Sets to Tally Sets

92-09* *Hermann von Hasseln, Laura Martignon*
Consistency in Stochastic Network

92-10 *Michael Schmitt*
A Slightly Improved Upper Bound on the Size of Weights Sufficient to Represent Any Linearly Separable Boolean Function

92-11 *Johannes Köbler, Seinosuke Toda*
On the Power of Generalized MOD-Classes

92-12 *V. Arvind, J. Köbler, M. Mundhenk*
Reliable Reductions, High Sets and Low Sets

92-13 *Alfons Geser*
On a monotonic semantic path ordering

92-14* *Joost Engelfriet, Heiko Vogler*
The Translation Power of Top-Down Tree-To-Graph Transducers

93-01 *Alfred Lupper, Konrad Froitzheim*
AppleTalk Link Access Protocol basierend auf dem Abstract Personal Communications Manager

93-02 *M.H. Scholl, C. Laasch, C. Rich, H.-J. Schek, M. Tresch*
The COCOON Object Model

93-03 *Thomas Thierauf, Seinosuke Toda, Osamu Watanabe*
On Sets Bounded Truth-Table Reducible to P-selective Sets

93-04 *Jin-Yi Cai, Frederic Green, Thomas Thierauf*
On the Correlation of Symmetric Functions

93-05 *K.Kuhn, M.Reichert, M. Nathe, T. Beuter, C. Heinlein, P. Dadam*
A Conceptual Approach to an Open Hospital Information System

93-06 *Klaus Gaßner*
Rechnerunterstützung für die konzeptuelle Modellierung

93-07 *Ullrich Keßler, Peter Dadam*
Towards Customizable, Flexible Storage Structures for Complex Objects

94-01 *Michael Schmitt*
On the Complexity of Consistency Problems for Neurons with Binary Weights

94-02 *Armin Kühnemann, Heiko Vogler*
A Pumping Lemma for Output Languages of Attributed Tree Transducers

94-03 *Harry Buhrman, Jim Kadin, Thomas Thierauf*
On Functions Computable with Nonadaptive Queries to NP

94-04 *Heinz Faßbender, Heiko Vogler, Andrea Wedel*
Implementation of a Deterministic Partial E-Unification Algorithm for Macro Tree Transducers

94-05 *V. Arvind, J. Köbler, R. Schuler*
On Helping and Interactive Proof Systems

94-06 *Christian Kalus, Peter Dadam*
Incorporating record subtyping into a relational data model

94-07 *Markus Tresch, Marc H. Scholl*
A Classification of Multi-Database Languages

94-08 *Friedrich von Henke, Harald Rueß*
Arbeitstreffen Typtheorie: Zusammenfassung der Beiträge

94-09 *F.W. von Henke, A. Dold, H. Rueß, D. Schwier, M. Strecker*
Construction and Deduction Methods for the Formal Development of Software

94-10 *Axel Dold*
Formalisierung schematischer Algorithmen

94-11 *Johannes Köbler, Osamu Watanabe*
New Collapse Consequences of NP Having Small Circuits

94-12 *Rainer Schuler*
On Average Polynomial Time

94-13 *Rainer Schuler, Osamu Watanabe*
Towards Average-Case Complexity Analysis of NP Optimization Problems

94-14 *Wolfram Schulte, Ton Vullinghs*
Linking Reactive Software to the X-Window System

94-15 *Alfred Lupper*
Namensverwaltung und Adressierung in Distributed Shared Memory-Systemen

94-16 *Robert Regn*
Verteilte Unix-Betriebssysteme

94-17 *Helmuth Partsch*
Again on Recognition and Parsing of Context-Free Grammars:
Two Exercises in Transformational Programming

94-18 *Helmuth Partsch*
Transformational Development of Data-Parallel Algorithms: an Example

95-01 *Oleg Verbitsky*
On the Largest Common Subgraph Problem

95-02 *Uwe Schöning*
Complexity of Presburger Arithmetic with Fixed Quantifier Dimension

95-03 *Harry Buhrman,Thomas Thierauf*
The Complexity of Generating and Checking Proofs of Membership

95-04 *Rainer Schuler, Tomoyuki Yamakami*
Structural Average Case Complexity

95-05 *Klaus Achatz, Wolfram Schulte*
Architecture Indepentent Massive Parallelization of Divide-And-Conquer Algorithms

95-06 *Christoph Karg, Rainer Schuler*
Structure in Average Case Complexity

95-07 *P. Dadam, K. Kuhn, M. Reichert, T. Beuter, M. Nathe*
ADEPT: Ein integrierender Ansatz zur Entwicklung flexibler, zuverlässiger kooperierender Assistenzsysteme in klinischen Anwendungsumgebungen

95-08 *Jürgen Kehrer, Peter Schulthess*
Aufbereitung von gescannten Röntgenbildern zur filmlosen Diagnostik

95-09 *Hans-Jörg Burtschick, Wolfgang Lindner*
On Sets Turing Reducible to P-Selective Sets

95-10 *Boris Hartmann*
Berücksichtigung lokaler Randbedingung bei globaler Zieloptimierung mit neuronalen Netzen am Beispiel Truck Backer-Upper

95-12 *Klaus Achatz, Wolfram Schulte*
Massive Parallelization of Divide-and-Conquer Algorithms over Powerlists

95-13 *Andrea Mößle, Heiko Vogler*
Efficient Call-by-value Evaluation Strategy of Primitive Recursive Program Schemes

95-14 *Axel Dold, Friedrich W. von Henke, Holger Pfeifer, Harald Rueß*
A Generic Specification for Verifying Peephole Optimizations

96-01 *Ercüment Canver, Jan-Tecker Gayen, Adam Moik*
Formale Entwicklung der Steuerungssoftware für eine elektrisch ortsbediente Weiche mit VSE

96-02 *Bernhard Nebel*
Solving Hard Qualitative Temporal Reasoning Problems: Evaluating the Efficiency of Using the ORD-Horn Class

96-03 *Ton Vullinghs, Wolfram Schulte, Thilo Schwinn*
An Introduction to TkGofer

96-04 *Thomas Beuter, Peter Dadam*
Anwendungsspezifische Anforderungen an Workflow-Mangement-Systeme am Beispiel der Domäne Concurrent-Engineering

96-05 *Gerhard Schellhorn, Wolfgang Ahrendt*
Verification of a Prolog Compiler - First Steps with KIV

96-06 *Manindra Agrawal, Thomas Thierauf*
Satisfiability Problems

96-07 *Vikraman Arvind, Jacobo Torán*
A nonadaptive NC Checker for Permutation Group Intersection

96-08 *David Cyrluk, Oliver Möller, Harald Rueß*
An Efficient Decision Procedure for a Theory of Fix-Sized Bitvectors with Composition and Extraction

96-09 *Bernd Biechele, Dietmar Ernst, Frank Houdek, Joachim Schmid, Wolfram Schulte*
Erfahrungen bei der Modellierung eingebetteter Systeme mit verschiedenen SA/RT–Ansätzen

96-10 *Falk Bartels, Axel Dold, Friedrich W. von Henke, Holger Pfeifer, Harald Rueß*
Formalizing Fixed-Point Theory in PVS

96-11 *Axel Dold, Friedrich W. von Henke, Holger Pfeifer, Harald Rueß*
Mechanized Semantics of Simple Imperative Programming Constructs

96-12 *Axel Dold, Friedrich W. von Henke, Holger Pfeifer, Harald Rueß*
Generic Compilation Schemes for Simple Programming Constructs

96-13 *Klaus Achatz, Helmuth Partsch*
From Descriptive Specifications to Operational ones: A Powerful Transformation Rule, its Applications and Variants

97-01 *Jochen Messner*
Pattern Matching in Trace Monoids

97-02 *Wolfgang Lindner, Rainer Schuler*
A Small Span Theorem within P

97-03 *Thomas Bauer, Peter Dadam*
A Distributed Execution Environment for Large-Scale Workflow Management Systems with Subnets and Server Migration

97-04 *Christian Heinlein, Peter Dadam*
Interaction Expressions - A Powerful Formalism for Describing Inter-Workflow Dependencies

97-05 *Vikraman Arvind, Johannes Köbler*
On Pseudorandomness and Resource-Bounded Measure

97-06 *Gerhard Partsch*
Punkt-zu-Punkt- und Mehrpunkt-basierende LAN-Integrationsstrategien für den digitalen Mobilfunkstandard DECT

97-07 *Manfred Reichert, Peter Dadam*
$ADEPT_{flex}$ - Supporting Dynamic Changes of Workflows Without Loosing Control

97-08 *Hans Braxmeier, Dietmar Ernst, Andrea Mößle, Heiko Vogler*
The Project NoName - A functional programming language with its development environment

97-09 *Christian Heinlein*
Grundlagen von Interaktionsausdrücken

97-10 *Christian Heinlein*
Graphische Repräsentation von Interaktionsausdrücken

97-11 *Christian Heinlein*
Sprachtheoretische Semantik von Interaktionsausdrücken

97-12 *Gerhard Schellhorn, Wolfgang Reif*
Proving Properties of Finite Enumerations: A Problem Set for Automated Theorem Provers

97-13 *Dietmar Ernst, Frank Houdek, Wolfram Schulte, Thilo Schwinn*
Experimenteller Vergleich statischer und dynamischer Softwareprüfung für eingebettete Systeme

97-14 *Wolfgang Reif, Gerhard Schellhorn*
Theorem Proving in Large Theories

97-15 *Thomas Wennekers*
Asymptotik rekurrenter neuronaler Netze mit zufälligen Kopplungen

97-16 *Peter Dadam, Klaus Kuhn, Manfred Reichert*
Clinical Workflows - The Killer Application for Process-oriented Information Systems?

97-17 *Mohammad Ali Livani, Jörg Kaiser*
EDF Consensus on CAN Bus Access in Dynamic Real-Time Applications

97-18 *Johannes Köbler,Rainer Schuler*
Using Efficient Average-Case Algorithms to Collapse Worst-Case Complexity Classes

98-01 *Daniela Damm, Lutz Claes, Friedrich W. von Henke, Alexander Seitz, Adelinde Uhrmacher, Steffen Wolf*
Ein fallbasiertes System für die Interpretation von Literatur zur Knochenheilung

98-02 *Thomas Bauer, Peter Dadam*
Architekturen für skalierbare Workflow-Management-Systeme - Klassifikation und Analyse

98-03 *Marko Luther, Martin Strecker*
A guided tour through *Typelab*

98-04 *Heiko Neumann, Luiz Pessoa*
Visual Filling-in and Surface Property Reconstruction

98-05 *Ercüment Canver*
Formal Verification of a Coordinated Atomic Action Based Design

98-06 *Andreas Küchler*
On the Correspondence between Neural Folding Architectures and Tree Automata

98-07 *Heiko Neumann, Thorsten Hansen, Luiz Pessoa*
Interaction of ON and OFF Pathways for Visual Contrast Measurement

98-08 *Thomas Wennekers*
Synfire Graphs: From Spike Patterns to Automata of Spiking Neurons

98-09 *Thomas Bauer, Peter Dadam*
Variable Migration von Workflows in *ADEPT*

98-10 *Heiko Neumann, Wolfgang Sepp*
Recurrent V1 – V2 Interaction in Early Visual Boundary Processing

98-11 *Frank Houdek, Dietmar Ernst, Thilo Schwinn*
Prüfen von C–Code und Statmate/Matlab–Spezifikationen: Ein Experiment

98-12 *Gerhard Schellhorn*
Proving Properties of Directed Graphs: A Problem Set for Automated Theorem Provers

98-13 *Gerhard Schellhorn, Wolfgang Reif*
Theorems from Compiler Verification: A Problem Set for Automated Theorem Provers

98-14 *Mohammad Ali Livani*
SHARE: A Transparent Mechanism for Reliable Broadcast Delivery in CAN

98-15 *Mohammad Ali Livani, Jörg Kaiser*
Predictable Atomic Multicast in the Controller Area Network (CAN)

99-01 *Susanne Boll, Wolfgang Klas, Utz Westermann*
A Comparison of Multimedia Document Models Concerning Advanced Requirements

99-02 *Thomas Bauer, Peter Dadam*
Verteilungsmodelle für Workflow-Management-Systeme - Klassifikation und Simulation

99-03 *Uwe Schöning*
On the Complexity of Constraint Satisfaction

99-04 *Ercument Canver*
Model-Checking zur Analyse von Message Sequence Charts über Statecharts

99-05 *Johannes Köbler, Wolfgang Lindner, Rainer Schuler*
Derandomizing RP if Boolean Circuits are not Learnable

99-06 *Utz Westermann, Wolfgang Klas*
Architecture of a DataBlade Module for the Integrated Management of Multimedia Assets

99-07 *Peter Dadam, Manfred Reichert*
Enterprise-wide and Cross-enterprise Workflow Management: Concepts, Systems, Applications. Paderborn, Germany, October 6, 1999, GI–Workshop Proceedings, Informatik ’99

99-08 *Vikraman Arvind, Johannes Köbler*
Graph Isomorphism is Low for ZPP^{NP} and other Lowness results

99-09 *Thomas Bauer, Peter Dadam*
Efficient Distributed Workflow Management Based on Variable Server Assignments

2000-02 *Thomas Bauer, Peter Dadam*
Variable Serverzuordnungen und komplexe Bearbeiterzuordnungen im Workflow-Management-System ADEPT

2000-03 *Gregory Baratoff, Christian Toepfer, Heiko Neumann*
Combined space-variant maps for optical flow based navigation

2000-04 *Wolfgang Gehring*
Ein Rahmenwerk zur Einführung von Leistungspunktsystemen

2000-05 *Susanne Boll, Christian Heinlein, Wolfgang Klas, Jochen Wandel*
Intelligent Prefetching and Buffering for Interactive Streaming of MPEG Videos

2000-06 *Wolfgang Reif, Gerhard Schellhorn, Andreas Thums*
Fehlersuche in Formalen Spezifikationen

2000-07 *Gerhard Schellhorn, Wolfgang Reif (eds.)*
FM-Tools 2000: The 4th Workshop on Tools for System Design and Verification

2000-08 *Thomas Bauer, Manfred Reichert, Peter Dadam*
Effiziente Durchführung von Prozessmigrationen in verteilten Workflow-Management-Systemen

2000-09 *Thomas Bauer, Peter Dadam*
Vermeidung von Überlastsituationen durch Replikation von Workflow-Servern in ADEPT

2000-10 *Thomas Bauer, Manfred Reichert, Peter Dadam*
Adaptives und verteiltes Workflow-Management

2000-11 *Christian Heinlein*
Workflow and Process Synchronization with Interaction Expressions and Graphs

2001-01 *Hubert Hug, Rainer Schuler*
DNA-based parallel computation of simple arithmetic

2001-02 *Friedhelm Schwenker, Hans A. Kestler, Günther Palm*
3-D Visual Object Classification with Hierarchical Radial Basis Function Networks

2001-03 *Hans A. Kestler, Friedhelm Schwenker, Günther Palm*
RBF network classification of ECGs as a potential marker for sudden cardiac death

2001-04 *Christian Dietrich, Friedhelm Schwenker, Klaus Riede, Günther Palm*
Classification of Bioacoustic Time Series Utilizing Pulse Detection, Time and Frequency Features and Data Fusion

2002-01 *Stefanie Rinderle, Manfred Reichert, Peter Dadam*
Effiziente Verträglichkeitsprüfung und automatische Migration von Workflow-Instanzen bei der Evolution von Workflow-Schemata

2002-02 *Walter Guttmann*
Deriving an Applicative Heapsort Algorithm

2002-03 *Axel Dold, Friedrich W. von Henke, Vincent Vialard, Wolfgang Goerigk*
A Mechanically Verified Compiling Specification for a Realistic Compiler

2003-01 *Manfred Reichert, Stefanie Rinderle, Peter Dadam*
A Formal Framework for Workflow Type and Instance Changes Under Correctness Checks

2003-02 *Stefanie Rinderle, Manfred Reichert, Peter Dadam*
Supporting Workflow Schema Evolution By Efficient Compliance Checks

2003-03 *Christian Heinlein*
Safely Extending Procedure Types to Allow Nested Procedures as Values

2003-04 *Stefanie Rinderle, Manfred Reichert, Peter Dadam*
On Dealing With Semantically Conflicting Business Process Changes.

2003-05 *Christian Heinlein*
Dynamic Class Methods in Java

2003-06 *Christian Heinlein*
Vertical, Horizontal, and Behavioural Extensibility of Software Systems

2003-07 *Christian Heinlein*
Safely Extending Procedure Types to Allow Nested Procedures as Values
(Corrected Version)

2003-08 *Changling Liu, Jörg Kaiser*
Survey of Mobile Ad Hoc Network Routing Protocols)

2004-01 *Thom Frühwirth, Marc Meister (eds.)*
First Workshop on Constraint Handling Rules

2004-02 *Christian Heinlein*
Concept and Implementation of C+++, an Extension of C++ to Support User-Defined Operator Symbols and Control Structures

2004-03 *Susanne Biundo, Thom Frühwirth, Günther Palm(eds.)*
Poster Proceedings of the 27th Annual German Conference on Artificial Intelligence

2005-01 *Armin Wolf, Thom Frühwirth, Marc Meister (eds.)*
19th Workshop on (Constraint) Logic Programming

2005-02 *Wolfgang Lindner (Hg.), Universität Ulm , Christopher Wolf (Hg.) KU Leuven*
2. Krypto-Tag – Workshop über Kryptographie, Universität Ulm

2005-03 *Walter Guttmann, Markus Maucher*
Constrained Ordering

2006-01 *Stefan Sarstedt*
Model-Driven Development with ACTIVECHARTS, Tutorial

2006-02 *Alexander Raschke, Ramin Tavakoli Kolagari*
Ein experimenteller Vergleich zwischen einer plan-getriebenen und einer leichtgewichtigen Entwicklungsmethode zur Spezifikation von eingebetteten Systemen

2006-03 *Jens Kohlmeyer, Alexander Raschke, Ramin Tavakoli Kolagari*
Eine qualitative Untersuchung zur Produktlinien-Integration über Organisationsgrenzen hinweg

2006-04 *Thorsten Liebig*
Reasoning with OWL - System Support and Insights –

2008-01 *H.A. Kestler, J. Messner, A. Müller, R. Schuler*
On the complexity of intersecting multiple circles for graphical display

2008-02 *Manfred Reichert, Peter Dadam, Martin Jurisch,l Ulrich Kreher, Kevin Göser, Markus Lauer*
Architectural Design of Flexible Process Management Technology

2008-03 *Frank Raiser*
Semi-Automatic Generation of CHR Solvers from Global Constraint Automata

2008-04 *Ramin Tavakoli Kolagari, Alexander Raschke, Matthias Schneiderhan, Ian Alexander*
Entscheidungsdokumentation bei der Entwicklung innovativer Systeme für produktlinien-basierte Entwicklungsprozesse

2008-05 *Markus Kalb, Claudia Dittrich, Peter Dadam*
Support of Relationships Among Moving Objects on Networks

2008-06 *Matthias Frank, Frank Kargl, Burkhard Stiller (Hg.)*
WMAN 2008 – KuVS Fachgespräch über Mobile Ad-hoc Netzwerke

2008-07 *M. Maucher, U. Schöning, H.A. Kestler*
An empirical assessment of local and population based search methods with different degrees of pseudorandomness

Ulmer Informatik-Berichte
ISSN 0939-5091

Herausgeber:
Universität Ulm
Fakultät für Ingenieurwissenschaften und Informatik
89069 Ulm