ulm university universität
uulm

On the different notions of pseudorandomness

M. Maucher, U. Schöning, H.A. Kestler

Ulmer Informatik-Berichte

Nr. 2008-11
August 2008

On the different notions of pseudorandomness

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

Abstract

This article explains the various notions of pseudorandomness, like Martin-Löf randomness, Kolmogorov complexity, Shannon entropy and quasi-randomness. We describe interconnections between these notions and describe how the non-computable notions among them are relaxed and used in practice, for example in statistics or cryptography. We give examples for pseudorandom generators relating to these notions and list some dependancies between the quality of pseudorandom numbers and its impact on some properties of randomized algorithms using them.

Keywords: pseudorandomness, algorithmic randomness, pseudorandom number generation

1 Introduction

Random numbers play a vital role in many areas of computer science. In cryptography, for example, pairs of public and private keys are chosen at random. In simulation, only statistical facts about a physical phenomenon may be known. In optimization, choices are frequently made at random when the best choice cannot be calculated effectively. And often, probabilistic algorithms are faster than any known deterministic algorithm for the same problem. But random numbers are not an integral part of a computer system. Since computers are purely deterministic systems, random behavior only occurs in the case of a system error. The only way to obtain "real random numbers" in a computer is by using random input from outside the system. Options include using the system time, measuring times between a user's keystrokes, or measuring other physical effects that are supposed to be random. Since input is usually processed much slower than internal data and, depending on the source of randomness, random numbers may only be available at a slow rate, a common approach uses these random numbers as a so-called *seed* for a longer sequence of pseudorandom numbers: From this seed, a much longer sequence is calculated in a deterministic way, assuming that this new sequence leads to a similar result as a sequence of real random numbers of the same length would. One of the first methods of this kind was the linear congruential generator, short lcg. This generator starts with a random number x (the seed) and successively applies a linear function $f(x) = ax + b \bmod m$ to the last value, generating a sequence

*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 Department of Internal Medicine I, hans.kestler@uni-ulm.de, Ulm University, Germany

$(x, f(x), f^2(x), f^3(x), \ldots)$, assuming that this sequence can be used instead of a sequence of random numbers.

Randomness is used with different goals in mind: In cryptography, the main goal is to provide numbers that cannot be guessed by an attacker; in simulation, using the random numbers provided by the computer should lead to the same result as in a real world process; in optimization, expected running times or results should be similar to those when using random numbers. Pseudorandom numbers have another property that is valuable in some cases: By saving the seed, we can efficiently save the whole pseudorandom sequence. This allows to reproduce simulation results, or to synchronize cryptographic processes that need to share sequences of random numbers.

In this paper, we will sum up various methods to define randomness and show the advantages and disadvantages of these notions as well as equivalences between them. We give some examples where pseudorandom numbers are used, how such numbers can be produced and where their quality affects the outcome of algorithms.

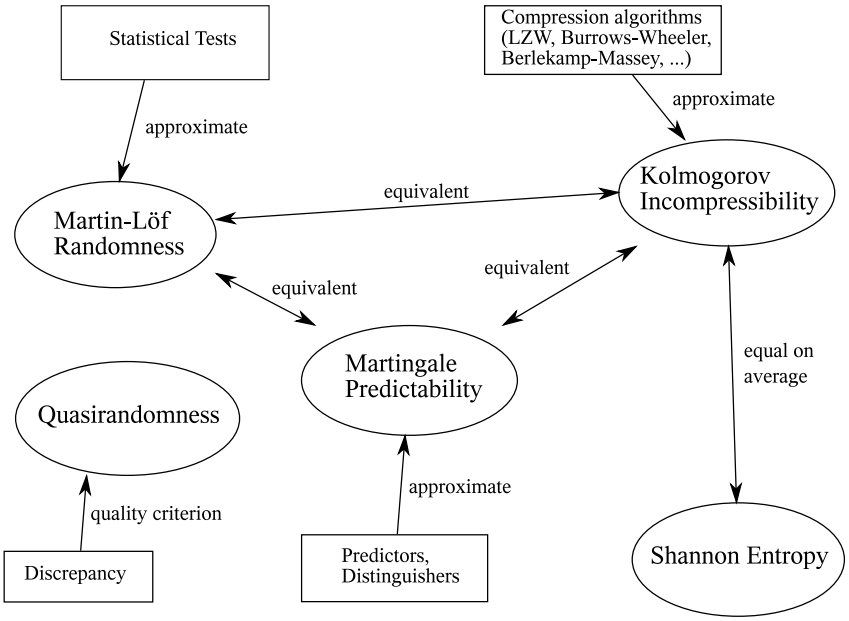

Figure 1: Various notions of pseudorandomness

In Section 2, we will sum up Martin-Löf's definition of algorithmic randomness, Kolmogorov's notion of incompressibility, explain the basics of martingale theory and predictability, and give some insight into Shannon's information theoretic entropy. Additionally, the notion of quasi randomness will be explained. In Section 3, we will list the most commonly used pseudorandom number generators, those used in practice as well as those found in some theoretical results. In section 4, we will give an overview of connections between the quality of pseudorandom generators and their influence on algorithms.

2 Notions of pseudorandomness

In this section, we introduce various methods to measure the "randomness" in a sequence of bits or numbers. After some mathematical preliminaries, we will begin with the notion of Kolmogorov complexity. We will then move on to statistical tests and Per Martin-Löf's definition of algorithmic randomness, describe the theory of martingales and their connection to predictability and distinguishability and explain C.E. Shannon's notion of information-theoretic entropy.

2.1 Preliminaries

In this paper, $\mathbb{N}$ will denote the set of natural numbers, $\mathbb{R}$ the set of real numbers and $\mathbb{R}^{\geq 0}$ the set of non-negative real numbers.

For any finite set Σ, let Σ^* be the set of all finite sequences of elements from Σ. E.g. $\{0,1\}^*$ is the set of all finite bit strings. By Σ^∞ we denote the set of all infinite sequences over Σ. The empty sequence is denoted by $\perp$.

For a finite sequence $x = x_0 x_1 \dots x_{n-1}$, $l(x)$ denotes the *length* of x, in this case $l(x) = n$.

Definition 1 *A random experiment is defined by a sample space S and a probability distribution $D : S \to \mathbb{R}$. A random experiment assigns a random variable, usually denoted by X, Y or Z, a value, drawn from S according to distribution D. If X is D-distributed, short $X \sim D$, then for any $s \in S$, $X = s$ with probability $D(s)$. We write $P_D[X = s]$ to express the probability of $X = s$ under the distribution D. If D is clear from the context, we simply write $P[X = s]$.*

Definition 2 *A sequence of random numbers $X_1, X_2, \dots$ is* k-wise independent *if any subsequence $X_{i_1}, \dots, X_{i_k}$ of length k is independent, i.e. for any $x_1, \dots, x_k$*

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

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

When talking about computability and complexity, a standard model of computation is needed – a model of a simple but powerful computing device. It should be simple enough to allow elegant proofs about what it can or can't do; and it should be powerful enough so that it can compute the same functions that a modern computer can compute. The Turing machine is the standard model of computation that unites these properties.

Definition 3 *A* Turing machine *M is a simplified model of a computer. It consists of several (finitely many) states, where one of these states is the initial state and one or more states are final states. The memory is represented by these states and a one-dimensional tape that is infinitely large in both directions. Each position of the tape may contain an element of the* work alphabet Γ*, which contains a special symbol $\square$, called "blank". The configuration of the Turing machine can be described by the actual state, the position on the tape, and the content of the tape. At the start of a computation, the Turing machine is in the inital state, the tape contains only the input, expressed in the* input alphabet

$\Sigma \subseteq \Gamma - \{\square\}$*, with all other positions on the tape equal to* $\square$*, and the position of the machine is at the leftmost symbol of the input. The behavior of a Turing machine during a computation is determined by its transition rule. For each state and symbol at the actual tape position, the transition rule specifies the new symbol at that position, the new state and the new position at the next time step. The new position may differ from the last one by* -1*,* 0 *or* 1*, i.e the machine can move one position to the left or right, or stay at its current position. The computation ends when the machine reaches a final state. The result of the computation is defined as the content of the tape after the computation, excluding* $\square$ *symbols. If the tape only consists of* $\square$ *symbols, the result is the empty string* $\perp$*. If the machine does not stop in a final state (i.e. runs in an infinite loop of non-final states), the result is* undefined*.*

We say that a Turing machine M *computes a function* $f : \{0,1\}^* \to \{0,1\}^*$*, if for every input* $x \in \{0,1\}^*$*,* M *computes* $f(x)$*.* M *computes a function* $f : \mathbb{N} \to \mathbb{N}$*, if for every* $x \in \mathbb{N}$*,* M *computes the binary representation of* $f(x)$ *if its input is a binary representation of* x*. Computations of functions* $f : \mathbb{N} \to \{0,1\}^*$ *and* $f : \{0,1\}^* \to \mathbb{N}$ *are defined analogously.*

Now that we have defined our model of computation, we can talk about computability:

Definition 4 *A function* $f : \{0,1\}^* \to \{0,1\}^*$ *is* computable *(or* recursive*), if there is a Turing machine* M *that computes* f*. We say a set* $S \subseteq \{0,1\}^*$ *is* computable *(or* recursive*), if its characteristic function* c_S *is* computable *with*

$$c_S(x) := \begin{cases} 0 & \text{if } x \notin L \\ 1 & \text{if } x \in L \end{cases}$$

A set $S \subseteq \{0,1\}^*$ *is* recursively enumerable *if* S *is the range of a total computable function* $f : \{0,1\}^* \to \{0,1\}^*$*, i.e.* $S = \{f(1), f(2), f(3), \ldots\}$*.*

A function $f : \mathbb{N} \to \mathbb{N}$ *is recursively enumerable, if its graph* G_f *is recursively enumerable, with* $G_f := \{(x, y) \mid y \leq f(x)\}$*.*

Since any infinite bit sequence $x \in \{0,1\}^*$ can be interpreted as a function $f : \mathbb{N} \to \{0,1\}$ or set $S \subseteq \mathbb{N}$, the definition of recursive enumerability can be applied to sequences.

Even if we know that a function or set is computable, we might want to be more precise about the difficulty of computing this set or function. To this end, we need the definitions of some standard complexity classes.

Definition 5 *For any Turing machine* M*, define* $\text{time}_M(x)$ *as the number of steps of* M *with input* x *until* M *reaches a terminal state.*

A language $L \subseteq \{0,1\}^*$ *lies in the complexity class* P *if* L *is accepted by a Turing machine* M *and there exists a polynomial* p *such that for every* $x \in \{0,1\}^*$*,* $\text{time}_M(x) \leq p(|x|)$*. I.e.* M*'s running time increases only polynomially in the* length *of the input* x*.*

A language $L \subseteq \{0,1\}^*$ *is in BPP if a Turing machine* M *and two polynomials* p *and* q *exist with the following properties:*

1. *For every* $x \in \{0,1\}^*$ *and every* $y \in \{0,1\}^{q(x)}$*,* $\text{time}_M(x, y) \leq p(|x|)$

2. *If* $Y \in \{0,1\}^{q(x)}$ *is a uniformly distributed random variable,*

$$P[M(x,Y) = c_L(x)] \geq \frac{2}{3} \text{ for every } x \in \{0,1\}^* \ ,$$

i.e. M computes the characteristic function of L, and gives a correct answer with probability at least 2/3.

A language $L \subseteq \{0,1\}^*$ *lies in the complexity class P/poly if there is a Turing machine M and two polynomials p and q with the following properties:*

1. *For every* $x, y \in \{0,1\}^*$, $\text{time}_M(x,y) \leq p(|x|)$,
2. *For every* $n \in \mathbb{N}$ *there exists a* $y_n \in \{0,1\}^{q(n)}$ *such that for every* $x \in \{0,1\}^n$ $M(x,y_n) = c_L(x)$.

Essentially, the languages in all complexity classes in Definition 5 are accepted by some Turing machine M in polynomial time. However, M always (except for the class P) may depend on a so called "advice string" y: For L to be in BPP, a large partof all candidates for y must lead to the correct result; for *P/poly*, there is only one advice string for every *length* of the input; for NP, the existence of one advice string per input is sufficient.

2.2 Kolmogorov complexity and compressibility

Kolmogorov complexity was independently introduced by Solomonoff and Kolmogorov. It measures with how many bits an object, usually a binary string, can be described, where every object has to be described in a given "language". A core principle of Kolmogorov complexity is non-compressibility: If we want to find a short description for a bit string of length n, saving at least l bits, we only have a very limited choice: There are only 2^{n+1-l} strings of length up to $n-l$, so we can compress at most a fraction of 2^{-l+1} of our strings. The rest can't be compressed by those l bits. Kolmogorov complexity uses this fact to disqualify strings as non-random: The probability that a random string can be compressed by l bits is about 2^{-l}. So, if a string can be compressed a lot, it is probably not random.

Definition 6 *Let* $M_1, M_2, M_3, \ldots$ *be a recursive enumeration of all Turing machines. By* $M_i(x)$ *we denote the output of* M_i *when it's run with input x. A* universal Turing machine *is a Turing machine* M_u *with*

$$M_u(i,x) = M_i(x) \ .$$

The Kolmogorov complexity $C(x)$ *of a string x is defined as*

$$C(x) := \min\{l(i) \mid M_i(\epsilon) = x\} \ ,$$

where $l(i)$ *is the length of i's binary representation. An infinite sequence* $s \in \{0,1\}^\infty$ *is Kolmogorov-random, if for every prefix* $s_{1..n}$ *of s the condition* $C(s_{1..n}) \geq n - c$ *holds for a constant c.*

According to this definition, the Turing machine M_u acts as an interpreter of other Turing machines: It is able to simulate any other Turing machine with

any input. That way, every binary string x can be described by describing a Turing machine that outputs x.

In the definition of Kolmogorov complexity, no concrete universal Turing machine is given, so that we could use different universal Turing machines in the definition. Actually, using a different Turing machine wouldn't be a big change. The *invariance theorem* states that using different universal Turing machines only results in a constant difference between the Kolmogorov complexities of any string x, where this constants only depends on the two universal Turing machines, but not on x.

A major disadvantage of Kolmogorov complexity is its non-computability. So it is not possible to exactly compute $C(x)$ for all strings x. It is, however, possible to give an approximation of $C(x)$. Looking for the shortest description d of a string x is equivalent to compressing x, where d can then be decompressed by the universal machine M_u. We can approximate this compression with standard compression algorithms, like `zip` or `bzip2`. For a fixed compression algorithm Z we can then define a string to be (Z, k)-random if Z cannot compress the string by more than k bits.

An interesting approach to describe a sequence of numbers is to assume that the sequence was generated by a linear recursion of some degree k, i.e. for all $i > k$,

$$X_i = a_0 + \sum_{j=1}^{k} a_j X_{i-j}$$

for some coefficients $a_1, \ldots, a_k$. In the case of a binary sequence, all these numbers are 0 or 1 and the recursion corresponds to a linear feedback shift register. A sequence can then be described by $X_1, \ldots, X_k$ and $a_1, \ldots, a_k$. For any given sequence, the linear degree k of that sequence can be efficiently found by the Berlekamp-Massey algorithm [1].

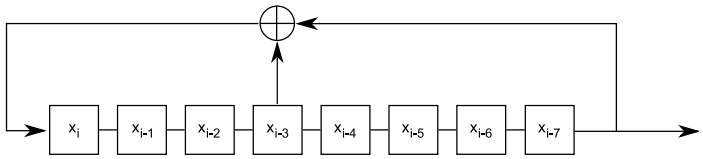

Figure 2: A linear feedback shift register with the recursion $X_i = X_{i-3} + X_{i-7}$

Definition 7 *For a finite sequence $x = (x_1, \ldots, x_n)$ the linear complexity of x, short $\mathrm{lc}(x)$, is the smallest k such that for all $i > k$,*

$$x_i = a_0 + \sum_{j=1}^{k} a_j x_{i-j}$$

for some parameters $a_0, a_1, \ldots, a_k$.

For an infinite sequence x, its linear complexity is a function lc_x with

$$\mathrm{lc}_x(i) = \mathrm{lc}(x_1, \ldots, x_i) \ .$$

For example, the linear complexity of a linear congruential generator is 1. It is easy to see that the linear complexity of every sequence $x \in \{0, \ldots, m-1\}^{2k}$

is at most k: For every $i > k$, suppose x_i is determined by the equation $x_i = a_0 + \sum_{j=1}^{k} a_j x_{i-j}$. Then we can find $a_0, a_1, \ldots, a_k$ by solving the system of all these equations for $x_{k+1}, \ldots, x_{2k}$.

2.3 Statistical tests and Martin-Löf randomness

In statistics, when sampling from a distribution D, a common method to test if a sample X was really drawn from distribution D is to divide the sample space S into two sets: the set S_1 of "typical" outcomes and the set S_0 of "non-typical" outcomes such that $P[X \in S_0] = \epsilon$ for some small ϵ. If $X \in S_0$, that variable could still be sampled according to D, but there is at least a reason to be suspicious.

Definition 8 *Let S be a sample space, D be a probability distribution on S and X a random variable with $X \sim D$.*

A function $f : S \to \{0, 1\}$ with $P[f(X) = 1] = 1 - \epsilon$ is called a statistical test *for D with confidence level ϵ. We say a sample x passes the test f (or f accepts x) if $f(x) = 1$. Otherwise x doesn't pass f (or f rejects x).*

For example, consider a uniform distribution on the set $S = \{0, 1\}^{32}$ of all 32-bit strings. We could just look at the first 10 bits of a random string X and reject X if each of these 10 bits is a zero. Only a fraction of 2^{-10} of all strings in S start with 10 zeros, so we would reject a true random sample with a probability lower than 0.1%. This confidence level could be easily changed by choosing an appropriate number of bits to consider.

Note that in the definition above, f computes the characteristic function of S_1. It is not clear which elements of S should belong to S_0: from a statistical point of view, all sets S_0 with $P[X \in S_0] = \epsilon$ are equally well suited. In practice, S_1 often consists of those sequences that have desirable properties, like equidistribution or pairwise independence, or simply properties that are easy and fast to compute. Ideally, a good pseudorandom sequence would pass all statistical tests, at least with probability of about $1-\epsilon$. However, pseudorandom generators are usually designed to output exponentially many numbers from a small seed, say 2^n numbers from a seed of length n. So when outputting a sequence of length l, only one out of 2^n sequences can be output, while a true random process would output one out of all possible 2^l sequences. By putting all those 2^n sequences into S_0, we can construct a statistical test where ϵ decreases exponentially when the sequence length l is increased. So technically, for every pseudorandom generator g and every confidence level ϵ, there is a statistical test with that confidence level that rejects the output of g. It is not clear, however, if this can be done by an efficient test. To be practically usable, a statistical test's running time should be a polynomial in $1/\epsilon$.

The following properties are commonly tested by statistical tests:

Frequencies of patterns The simplest of these tests just count the number of zeros or ones in a sequence. These should not differ too much. More sophisticated tests count frequencies of certain patterns, either overlapping ones or non-overlapping ones. Instead of counting the frequencies of all patterns up to a certain length, one can also restrict oneself to patterns of special forms. Examples for this kind of test are the run test, which counts the length of runs of zeros or ones (either in the whole sequence or in each

k-bit block) or the poker test, which divides the set of all patterns into classes known from the poker game (like one pair, two pairs, full house, etc.).

Compressibility This kind of test tries to compress a given sequence. A truly random sequence usually isn't compressible (see section 2.2). Compression can be measured by methods like the Lempel-Ziv algorithm, the Burrows-Wheeler transform or the Berlekamp-Massey algorithm. Another approach can be made via non-lossless compression: A cosinus or fourier transform can show if a sequence can be approximated by few base vectors of a given transform.

Random walks Tests from this class use the bits of a sequence to run a simple randomized algorithm. The random excursion test, for example, simulates a random walk on a line, walking in one direction each time a "0" comes up, and walking in the other direction for every "1". This random walk shouldn't wander too far away from the starting point, but it shouldn't stick to it, either. This test is generalized in the cumulative sum test, that interprets bit blocks as integers and examines their cumulative sums.

More There exist many more tests. Actually any property of a sequence can be used to design a statistical test, as long as that property can be measured and something about its stochastic behavior when observed on a random sequence is known. Depending on the distribution of such a property, there are various tests that may be appropriate, like the χ^2 test for testing the variance of a value under a normal distribution.

For further insight into statistical tests, see [2].

A more formal approach concerning statistical tests and randomness has been made by P. Martin-Löf in 1966 [3]. He formally defined algorithmically random sequences of infinite length.

Definition 9 *A recursively enumerable set $T \subseteq \mathbb{N} \times \{0,1\}^*$ is a* Martin-Löf test *if, with $T_n := \{t \in \{0,1\}^* \mid (n,t) \in T\}$,*

$$\sum_{t \in T_n} 2^{-|t|} \leq 2^{-n}.$$

A sequence s in $\{0,1\}^\infty$ passes the test T if

$$s \notin \cap_n \cup_{t \in T_n} \{u \in \{0,1\}^\infty \mid t \text{ is a prefix of } u\} \ .$$

A sequence s is Martin-Löf random, if it passes all *Martin-Löf tests.*

For any $n \in \mathbb{N}$, the set T_n specifies a set of prefixes such that a random string in $\{0,1\}^*$ begins with a prefix from T_n with probability at most 2^{-n}. For example, consider the test $T = \{(n, 0^{n+1}), (n, 1^{n+1}) \mid n \geq 1\}$. Then $T_n = \{0^{n+1}, 1^{n+1}\}$, i.e at "confidence level" n, the test would reject any string where the first $n+1$ bits are equal. According to the definition, this test would only reject the infinite string that only consists of ones and the one that only consist of zeros.

If s is a sequence drawn randomly under the uniform distribution from $\{0,1\}^\infty$ then for any fixed $t \in \{0,1\}^*$, t is a prefix of s with probability $2^{-|t|}$.

Therefore, for any n, the set $\cup_{t\in T_n}\{u \in \{0,1\}^\infty \mid t \text{ is a prefix of } u\}$ is a set of measure at most 2^{-n}. Thus, the set $\cap_n \cup_{t\in T_n} \{u \in \{0,1\}^\infty \mid t \text{ is a prefix of } u\}$ is a set of measure 0. Since there are only countably many Martin-Löf tests, and the union of countably many sets of measure 0 has itself measure 0, the set of sequences that are not Martin-Löf random has measure 0. This means that the set of Martin-Löf random sequences has measure 1, i.e any randomly drawn sequence is Martin-Löf random with probability 1.

It has been shown that a universal Martin-Löf test U exists such that for any Martin-Löf test T, T is included in U, i.e. there is some constant c such that for all n

$$T_{n+c} \subseteq U_n \ ,$$

where c may depend on T. In other words, this universal test on its own is able to detect the non-randomness of any sequence, and a bit sequence $x \in \{0,1\}^\infty$ is Martin-Löf random if and only if it passes the universal test U.

Every computable sequence x can be transformed into a Martin-Löf test $T_x = \{(n,u) \mid u \text{ is a prefix of } x \text{ with } l(u) = n\}$. Now the sequence x will not pass the test T_x. This shows that no computable sequence is Martin-Löw random, analogously to the fact that such a sequence is not Kolmogorov random, since it can be described by a Turing machine of some fixed length. Actually, it was shown by Martin-Löf that the notions of Komogorov randomness and Martin-Löf randomness are equivalent.

Note that there is no efficiency requirement for these statistical tests. Thus only a very restricted version can be used in practice: Efficiently testing for randomness would be limited to a certain number of tests, and only to efficiently computable tests. With these restrictions sequences that are not Martin-Löf random might still look random to a set of efficiently computable statistical tests.

2.4 Martingales and predictability

Another method to look at randomness is the point of view of a gambler: Suppose a betting game where coins are thrown one after the other. A player starts with a capital of c and before each coin is thrown will bet some amount c' on the outcome of "heads" and $c - c'$ on the outcome of "tails". The player's bet on the correct outcome will be doubled, the other is lost. With a perfectly random coin, the game is fair and the expected gain is equal to 0. However, if there exists a strategy that will consistently win, then we might suspect that the sequence is not random.

Definition 10 *A martingale is a function* $m : \{0,1\}^* \to \mathbb{R}^{\geq 0}$ *with*

$$m(w) = \frac{1}{2}\left(m(w0) + m(w1)\right) \ .$$

A martingale m *succeeds on an infinite sequence* s *if*

$$\limsup_{n\to\infty} m(s_{1..n}) = \infty \ .$$

Here, the function $m(w)$ describes a player's capital after the bits of w have been thrown, $m(w0)$ is the player's capital after an additional 0, and $m(w1)$

after an additional 1. This corresponds to a betting strategy where $m(w0)/2$ is bet on a "0", and $m(w1)/2$ is bet on a "1".

It was shown by Schnorr [4] that a sequence s is Martin-Löf random iff s is not succeeded by any recursively enumerable martingale.

A martingale always has access to *all* bits of w. When restricting ourselves to martingales with a limited memory of the last k bits, then a pseudorandom generator that outputs $k+1$-wise independent numbers could not be succeeded by such a limited martingale.

Additionally, we could restrict martingales to efficiently computable functions. This leads to a slightly different view on randomness, mainly found in cryptography:

Definition 11 *Let D_n be a probability distribution on $\{0,1\}^n$ and X a random variable with $X = (X_1, \ldots, X_n) \sim D_n$. An algorithm A is an* ϵ-predictor *for D_n if for some $i < n$, it predicts X_i from $X_1, \ldots, X_{i-1}$ with probability at least $\frac{1}{2} + \epsilon$, i.e.*

$$P[A(X_1, \ldots, X_{i-1}) = X_i] \geq \frac{1}{2} + \epsilon \ .$$

Now if we know that a sequence of random bits is distributed according to a distribution D_n, we can use predictors to guess some bits in advance – if those predictors exist. But suppose the sequence is distributed according to the uniform distribution U_n on n-bit strings. Then no matter how we guess the outcome of an arbitrary bit of that sequence, we can only guess the correct bit with probability $\frac{1}{2}$. That is, if A is an ϵ-predictor for D, it behaves differently for input distributions D_n or U_n. This leads us to the next definition, that of a *distinguisher*.

Definition 12 *Let D_n and $\tilde{D}_n$ be two probability distributions on $\{0,1\}^n$, and X, Y two random variables with $X \sim D_n$ and $Y \sim \tilde{D}_n$. Then an algorithm A is an* ϵ-distinguisher *for D_n and $\tilde{D}_n$, if*

$$|P[A(X) = 1] - P[A(Y) = 1]| \geq \epsilon \ .$$

It can be shown that an ϵ-distinguisher for a distribution and the uniform distribution exists if and only if an ϵ-predictor for that sequence exists. For a proof, see for example [5].

In cryptography, the notions of distinguishers and predictors can be used to define cryptographic security of a sequence. In this setting, a sequence (or the pseudorandom generator that produces it) is defined as cryptographically secure if there is no efficient distinguisher for that sequence. Here it is assumed that the seed is chosen uniformly among all possible seeds. A distinguisher is "efficient" if it runs in polynomial time, i.e. if it's contained in an efficient complexity class like BPP.

Definition 13 *A sequence of distributions $\mathcal{D} = (D_1, D_2, D_3, \ldots)$ is called a* distribution ensemble*, if for any $i \in \mathbb{N}$, D_i is a probability distribution on $\{0,1\}^i$. Let X_i be random variables with $X_i \sim D_i$ for all $i \in \mathbb{N}$, and let U_i denote the uniform distribution on $\{0,1\}^i$ for all $i \in \mathbb{N}$. Then the probability ensemble $\mathcal{D}$ is* cryptographically pseudorandom *if no polynomials $p(n)$ and $q(n)$ exist such that for each n there exists a $\frac{1}{q(n)}$-distinguisher for D_n and U_n with running time $p(n)$.*

Until today, it is not known if pseudorandom distribution ensembles can be efficiently generated. In cryptography, being able to generate pseudorandom sequences from smaller seeds would prove very useful: Sharing the secret seed of such a sequence would allow two parties to efficiently share the whole sequence and thus have access to a common source of bits that can't be distinguished from a source of truly random bits by any efficient algorithm.

2.5 Shannon Entropy

In 1948, C.E. Shannon defined a measure of randomness which was the foundation of information theory, a new scientific discipline [7]. Unlike other measures that define the randomness of single sequences, entropy measures the randomness of a stochastic process, resp. that of a random distribution.

Definition 14 *Let S be a sample space, D a probability distribution on S and X a random variable with $X \sim D$. Then*

$$H(X) := -\sum_{s \in S} P[X = s] \log_2 P[X = s]$$

is the Shannon entropy of X. The minimum entropy of X, $H_{min}(x)$, is defined as

$$H_{min}(X) := \min_{s \in S} \{-\log_2 P[X = s]\} \ .$$

For a given sample space S, we always have $0 \leq H(X) \leq \log_2 |S|$, where $H(X)$ reaches its maximum when X is uniformly distributed in S and its minimum when $P[X = s] = 1$ for an element $s \in S$. It can be shown that Shannon entropy of a random variable X on S is a lower bound for the average code word length for any code over S. On the other hand, there always exists a code that maps every element $s \in S$ to a code word of length $\lceil -\log_2 P[X = s] \rceil$, the so-called Shannon-Fano code. Therefore, for the average codeword length $\overline{L}$ of any optimal code for S the following inequality holds:

$$H(X) \leq \overline{L} < H(X) + 1 \ .$$

Note that for any random variable X, the entropy of X cannot be increased by deterministic methods. I.e. for any function f, $H(X) \geq H(f(X))$. Therefore, the entropy of a pseudorandom sequence will never surpass the entropy of that sequence's seed.

While Kolmogorov complexity and its equivalent notions measure the randomness of single strings, Shannon entropy measures the randomness of a probability distribution on a set of elements. But since both notions are related to the lengths of descriptions or codes of elements, there is an elegant connection between these two notions: If the probability function $p(x) := P[X = x]$ is *computable*, then

$$H(X) \leq \sum_{s \in S} P[X = s] K(s) \leq H(X) + c_p \ ,$$

where c_p is a non-negative constant that only depends on the function p. This means that Kolmogorov complexity gives us codeword lengths of a universal code

that has almost optimal average length (up to a constant added term) for *any* distribution on the set of all strings, as long as this distribution is computable.

Note that while the Shannon entropy of a random variable X can't be increased by applying any function f to X, Kolmogorov complexity of a string x may well be increased by applying a function f to it. As long as f is computable, $f \circ p$ is still computable and the inequality above still holds, but the constant $c_{f \circ p}$ may be greater than c_p.

2.6 Quasirandom sequences

Algorithms following the Monte Carlo method draw many random samples from a given sample space, perform determinstic computations on these samples and then recombine the results. For example, an integral $\int_0^1 f(x)dx$ can be approximated by the sum $\frac{1}{N}\sum_{i=1}^{n} f(x_i)$, where $x_1, x_2, \ldots, x_N$ is a sequence of random numbers in the interval $[0, 1]$. Since the result is largely based on the set of samples, these methods depend on a good quality of the random number generator that is used. On the other hand, due to the large number of samples needed, the random number generator should be very fast. Quasi-Monte Carlo algorithms avoid the usage of random numbers, and instead attempt to generate numbers that are spread over their domain evenly. In the case of the integral above, 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 [8]. Discrepany measures how evenly a set of points in a k-dimensional cube is distributed.

Definition 15 *Let $P := \{x_1, x_2, \ldots, x_N\} \subset [0, 1)^d$. Then the star discrepancy D^* of P is defined as*

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

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

Note that $\frac{|\{x_i \in P | \forall j. x_i^{(j)} < x^{(j)}\}|}{N}$ corresponds to an approximation of the volume of x when using the Monte Carlo method: For each sample, we check if the sample lies within x. At the end, we compute the fraction of these samples among the set of all samples. The expected value of our computation is then equal to the volume of x. I.e. star discrepancy measures the maximum difference between the result of this Monte Carlo computation and the correct result.

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

Definition 16 *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*

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

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

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

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, $\phi_2(11001_2) = 0.10011_2$. A van der Corput sequence X in base b has a star discrepancy of $D_n^*(X) = O\left(\frac{\log N}{N}\right)$.

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

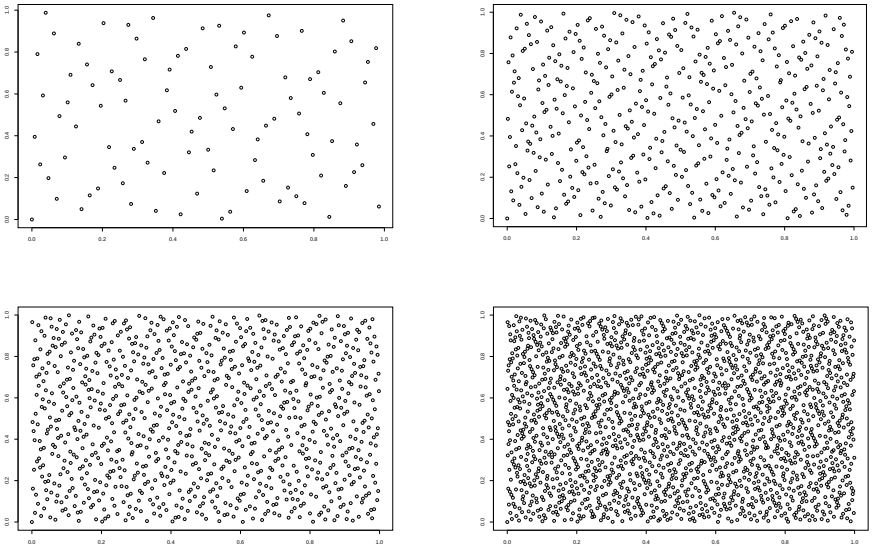

Figure 3: Plots of two-dimensional Halton sequences of lengths 100, 500, 1000 and 2000.

Definition 17 *A* Halton sequence *in the bases* $b_1, \ldots, b_k$ *is defined as*

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

A Halton sequence is a composition of multiple van der Corput sequences with different bases. Some plots of two-dimensional Halton sequences of bases 2 and 3 can be seen in Figure 3. Compare with Figure 5, where uniformly distributed numbers were used.

3 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.

Linear congruential generators

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 [2]. It can be shown that the parameters a and b can be chosen in a way that the linear congruential generator has a period length of m. This property alone, however, does not guarantee that the resulting sequence looks random. For example, the parameters $a = b = 1$ lead to a period length of m, but not to a sequence that looks random. If the parameters do not guarantee a period length of m, the output can have a very short period length. For example, the parameters $a = 3$, $b = 3$ and $m = 17$ lead to a period length of 16 for almost every seed in $\{0, 1, \ldots, 16\}$. Using $X_0 = 7$, however, leads to an output of period length 1, with $X_i = 7$ for all i.

Generalisations include the polynomial congruential generator, which uses a recursion of the form $X_n = \sum_{i=0}^{k} a_i x^i \bmod m$, or the inversive congruential generator, using a recursion of the form $X_n = \left(\sum_{i=0}^{k} a_i x^i\right)^{-1} \bmod m$.

Linear feedback shift registers

A linear feedback shift register is another kind congruential generator. Its binary output sequence is created by the recursion

$$x_i = \bigoplus_{j=1}^{k} a_j x_{i-j} \bmod 2 \ .$$

Such a generator can be implemented in hardware, essentially using a shift register where an internal state $(x_{i-k}, \ldots, x_{i-1})$ is stored and updated.

Many stream ciphers used in cryptography (e.g. Trivium) are based on linear feedback shift registers. Since the recursion of a linear feedback shift register can easily be computed from its output, they add non-linear operations like AND and OR. This makes their cryptanalysis a lot more difficult.

Explicit polynomial generators

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

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

An important property of explicit polynomial generators is k-wise independence. Within one period, any k output numbers of this generator are independent if the parameters $a_0, \ldots, a_k$ are chosen at random. The period length of such a generator is at most m, since $p(x) \equiv_m p(x + m)$ for any polynomial p.

If only pairwise independence is needed, any class of universal hash functions can be used to create a sequence of pairwise independent numbers.

Lagged fibonacci generators

A lagged fibonacci generator produces a sequence similar to the fibonacci sequence, but it usually adds less recent numbers of the output sequence to generate a new number. A lagged fibonacci generator with parameters i_1 and i_2 and operation $\oplus$ produces the sequence defined by

$$X_n = X_{n-i_1} \oplus X_{n-i_2} \bmod m \ .$$

Typically $\oplus$ is implemented as addition, subtraction, multiplication or bitwise XOR. The seed consists of the first $\max(i_1, i_2)$ numbers of the sequence. Choosing a seed for this kind of generator is non-trivial and choosing it at random may lead to output of rather low quality. Generators of this type were used in various programs (e.g. Matlab [9]), but are nowadays replaced by the Mersenne Twister.

Mersenne Twister

The Mersenne Twister [10] 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 [11] or Maple [12]. Any 623 subsequent numbers of its output are independent and uniformly distributed, which makes this generator a good choice in many cases. Note, however, that it is not cryptographically secure and should therefore not be used in security-related algorithms.

Isaac

Isaac (Indirection, Shift, Accumulate, Add, and Count) uses an internal state of 256 bytes and various operations to transform that internal state: Indirection (using a part of the internal state as an address inside the internal state), Shift (rotating parts of the internal state), Accumulate (accumulating a value over various iterations of the algorithm), Add and Count. It has a minimum cycle length of 2^{40} and an expected cycle length of 2^{8295}. It was designed to be cryptographically secure, and as of today, there are no efficient distinguishers or predictors known.

Physical random sources

Instead of using a small seed to produce many numbers, one could also think of ways to rapidly "capture" randomness from physical processes. A few physical sources have already been used to obtain random numbers:

- Radio frequencies where no signal is sent contains only atmospheric noise, which is mainly caused by lightnings all over the world. This noise can be measured with the help of a radio antenna and transformed into a sequence of random numbers.
- When a beam of photons is sent through a so-called beam splitter, every photon has two possible paths to leave that beam splitter. By using fast detectors that can detect single photons, this method can produce random bits at a rate of about 1 Mbit/s [13, 14].

Archived bits

Instead of using a given pseudorandom generator, one can instead use random bits that are available on CD, DVD or the internet. This idea isn't new, however: Back in 1927, a book with the title "Random sampling numbers" was published, containing mostly tables of random numbers [15].

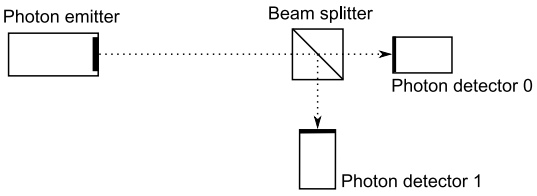

Figure 4: Creating random bits with a beam splitter. Photons are emitted by the photon emitter. With probability 0.5 they pass the beam splitter and are reported by detector 0. With probability 0.5 they are reflected by the beam splitter and are then reported by detector 1.

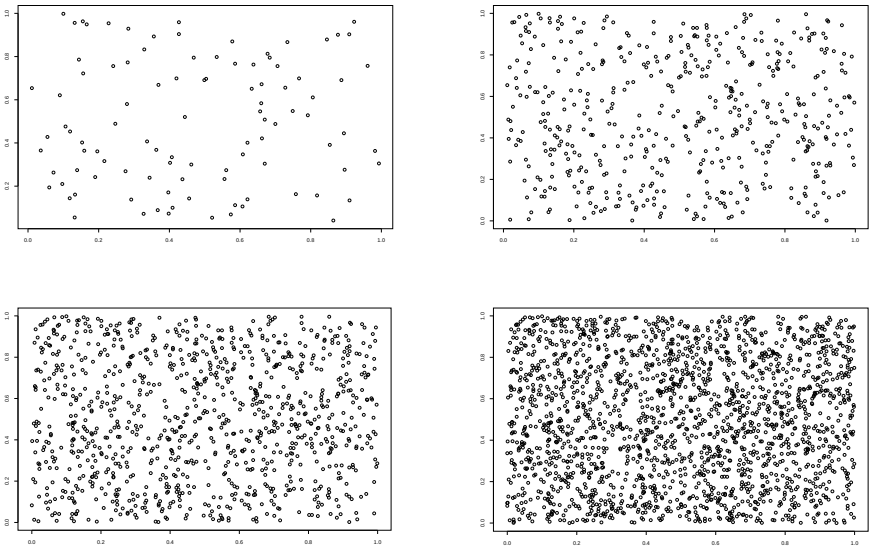

Figure 5: Some sets of random two-dimensional points. Set sizes are 100, 500, 1000 and 2000. The numbers are from Marsaglia's Diehard sequence [16].

Marsaglia's Diehard suite [16] is one of these sources available on the internet. It 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. For some plots of numbers from this source, see Figure 5.

Another source of random bits is the web page random.org [17]. It measures atmospheric noise and converts it into random bits. Currently, downloading a

limited amount of random bits from that source is free.

4 Influence on Algorithms

Pseudorandom numbers are used in various algorithms. For some algorithms, theoretical and empirical results are known about the influence of different pseudorandom generators on the results of these algorithms.

In [18], a simple evolutionary algorithm was run with various pseudorandom generators, but no relevant influences have been found. This indicates that the given evolutionary algorithm does not depend on a very high "quality" of the random numbers involved. In [19] however, an evolutionary algorithm led to better results when quasirandom sequences were used. I.e. a part of the algorithm (namely the mutation phase) was identified where the use of deterministic numbers instead of pseudorandom numbers led to better results. This indicates that in some scenarios, regularity leads to better results than randomness.

Simulated Annealing, another search heuristic, is severely influenced by the period length of the pseudorandom generator. In [19], the Traveling Salesman problem, among others, was solved with the Simulated Annealing heuristic. As long as the period length was long enough, the heuristic was quite tolerant with respective to the "quality" of the generator, i.e. if a linear congruential generator was used or a more sophisticated Mersenne Twister. Shortening the period length of the pseudorandom sequence however, gradually led to worse results.

It is a well-known fact that Shannon's entropy is a lower bound for any sorting method that is based on pairwise comparisons. The entropy of a uniform distribution on the set of all permutations of n elements is equal to $\log_2(n!) = \Theta(n \log n)$. For the randomized version of the QuickSort algorithm, some additional results have been achieved. Karloff et al. [20] showed that QuickSort's worst case complexity can go up to $\Omega(n^2)$ when sorting n numbers with the help of a linear congruential generator. They also showed that QuickSort shows an average case running time of $O(n \log n)$ when using an explicit polynomial generator of degree 4. Their main argument uses the fact that this generator produces 5-wise independent numbers. B. List et al. [21, 22] showed how QuickSort's running time is gradually increased to $\Omega(n^2)$ when the probability of "bad" pivot elements (i.e. very small or large elements) increases.

Chor and Golreich have shown that k-wise independent random numbers are a useful tool for sampling [23]. With this method, only the random bits for a seed are needed and the chance of hitting any subset of the sample space is still good. This can be used to reduce the error probability of RP algorithms with relatively few bits. Usually, such an algorithm's error probability can be reduced by running it multiple times, each time using *new* random numbers. The use of k-wise independent bits allows the same technique to some lesser degree: If the algorithm needs r random bits, these r bits can be used as a seed for a pseudorandom number generator that creates k-wise independent bits for some k. The algorithm can then be run multiple times, using bits from the pseudo random generator. Repeating the algorithm l times, the error probability can be reduced to $(1/l)$, while still only r bits were used. This approach can also be used for BPP algorithms. Note that using true random bits would reduce the error probability exponentially in l, at the cost of a total of lr random bits.

Further uses of pairwise independent numbers can be found in an overview of M. Luby and A. Wigderson [24].

Bach [25] showed that linear congruential generators with a prime modulus are a sufficient source of randomness for computing square roots modulo a prime p (with two probabilistic algorithms from Lehmer [26] and Shanks [27]), for computing q-th roots modulo a prime p (with a probabilistic algorithm by Adleman, manders and Miller [28]), and for testing primality (with the Miller test [29]). For each of these algorithms, the error probability was shown to not increase when a linear congruential generator was used instead of independent, uniformly distributed random numbers.

Hoos et. al. [30] empirically examined the influence of some pseudorandom generators on the result of probabilistic algorithms for the satisfiability problem. They observed that the quality of the random numbers didn't influence the output quality of these algorithms. However, completely derandomizing the algorithms caused them to fail for a few input instances and made parallelization difficult.

Azar et. al. [31] show that random walks can be influenced by a biased source of randomness. They consider random walks on d-regular graphs (i.e. graphs where every node has degree d) and a random source that outputs values $1, 2, \ldots, d$. At each step, with probability $1 - \epsilon$, that number is drawn randomly under a uniform distribution. With probability ϵ, a deterministic process may decide which number is output. They show that the limit probability of any subset $S \subseteq V$ of the graph's vertices can be augmented from $|S|/|V|$ to $(|S|/|V|)^{1-c\epsilon}$. This result also shows that a malevolent random source could decrease a Markov chain's probability to converge to a good solution.

The rho algorithm for factoring numbers is based on the birthday paradox, a statistical fact about independent random numbers. In and of itself, this approach would not be able to beat the running time of the naive approach. The rho algortihm however exploits the fact that it uses a pseudorandom generator. It uses the regularity of that pseudorandom generator to save many computations and thus achieves a much better running time.

References

[1] James L. Massey. Shift-register synthesis and bch decoding. *IEEE Transactions on Information Theory*, 15:122–127, 1969.

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

[3] Per Martin-Löf. The definition of random sequences. *Information and Control*, 9(6):602–619, 1966.

[4] C.P. Schnorr. Zufälligkeit und Wahrscheinlichkeit. In *Lecture Notes in Mathematics*, volume 218. Springer, 1971.

[5] Oded Goldreich. *Foundations of Cryptography*, volume Basic Tools. Cambridge University Press, 2001.

[6] Michael Luby. *Pseudoranomness and Cryptographic Applications.* Princeton University Press, 1996.

[7] Claude E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27:379–423, 623–, july, october 1948.

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

[9] The MathWorks – MATLAB and Simulink for Technical Computing, http://www.mathworks.com/.

[10] Takuji Nishimura Makoto Matsumoto. Mersenne twister: A 623-dimensionally equidistributed uniform pseudo random number generator. *ACM Transactions on Modeling and Computer Simulations: Special Issue on Uniform Random Number Generation*, 1998.

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

[12] Maple, www.maplesoft.com.

[13] Personal correspondance with A. Zeilinger and T. Jennewein, University of Vienna.

[14] T. Jennewein, U. Achleitner, G. Weihs, H. Weinfurter, and A. Zeilinger. A fast and compact quantum random number generator. *Review of Scientific Instruments*, 71:1675–1680, April 2000.

[15] L. H. C. (Leonard Henry Caleb) Tippett. *Random sampling numbers*, volume 15 of *Tracts for computers.* Cambridge University Press, Cambridge, UK, 1927. Reprinted in 1952. Reprinted in 1959 with a foreword by Karl Pearson.

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

[17] RANDOM.ORG – True Random Number Service, http://random.org/.

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

[19] Markus Maucher, Uwe Schöning, and Hans A. Kestler. An empirical assessment of local and population based search methods with different degrees of pseudorandomness. Technical report, Universität Ulm, 2008.

[20] Howard Karloff and Prabhakar 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.

[21] Beatrice List. *Probabilistische Algorithmen und schlechte Zufallszahlen.* PhD thesis, Universität Ulm, 1999.

[22] Beatrice List, Markus Maucher, Uwe Schöning, and Rainer Schuler. Randomized quicksort and the entropy of the random source. In *Computing and Combinatorics - COCOON 2005*, pages 450–460. Springer, 2005.

[23] Benny Chor and Oded Goldreich. On the power of two-point based sampling. *Journal of Complexity*, 5(1):96–106, 1989.

[24] Michael Luby and Avi Wigderson. Pairwise independence and derandomization. Technical Report CSD-95-880, 1995.

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

[26] D. H. Lehmer. Computer technology applied to the theory of numbers. In *Studies in Number Theory*, pages 117–151. Prentice-Hall, 1969.

[27] D. Shanks. Five number-theoretic algorithms. In *Proceedings of the Second Manitoba Conference on Numerical Mathematics*, pages 51–70, 1972.

[28] L. Adleman, K.Manders, and G.Miller. On taking roots in finite fields. In *Proc.18th Annual IEEE Symp. Foundations of Computer Sciences*, pages 175–178, 1977.

[29] Gary L. Miller. Riemann's hypothesis and tests for primality. *Journal of Computer and System Sciences*, 13:300–317, December 1976. invited publication.

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

[31] Yossi Azar, Andrei Z. Broder, Anna R. Karlin, Nathan Linial, and Steven Phillips. Biased random walks. *Combinatorica*, 16(1):1–18, 1996.

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

2008-08 *Henning Wunderlich*
Covers have structure

2008-09 *Karl-Heinz Niggl, Henning Wunderlich*
Implicit characterization of FPTIME and NC revisited

2008-10 *Henning Wunderlich*
On span-P^{CC} and related classes in structural communication complexity

2008-11 *M. Maucher, U. Schöning, H.A. Kestler*
On the different notions of pseudorandomness

Ulmer Informatik-Berichte
ISSN 0939-5091

Herausgeber:
Universität Ulm
Fakultät für Ingenieurwissenschaften und Informatik
89069 Ulm