# Pairwise Independence and Derandomization

# Pairwise Independence and Derandomization

**Michael Luby**

*Digital Fountain*
*Fremont, CA, USA*

**Avi Wigderson**

*Institute for Advanced Study*
*Princeton, NJ, USA*

*avi@ias.edu*

now

the essence of knowledge

Boston – Delft

# Foundations and Trends® in Theoretical Computer Science

*Published, sold and distributed by:*
now Publishers Inc.
PO Box 1024
Hanover, MA 02339
USA
Tel. +1-781-985-4510
www.nowpublishers.com
sales@nowpublishers.com

*Outside North America:*
now Publishers Inc.
PO Box 179
2600 AD Delft
The Netherlands
Tel. +31-6-51115274

A Cataloging-in-Publication record is available from the Library of Congress

The preferred citation for this publication is M. Luby and A. Wigderson, Pairwise Independence and Derandomization, Foundation and Trends® in Theoretical Computer Science, vol 1, no 4, pp 237–301, 2005

*Printed on acid-free paper*

ISBN: 1-933019-76-X

**Foundations and Trends® in Theoretical Computer Science**
Volume 1 Issue 4, 2005

## Editorial Board

# Editorial Scope

**Foundations and Trends® in Theoretical Computer Science** will publish survey and tutorial articles in the following topics:

- Algorithmic game theory
- Computational algebra
- Computational aspects of combinatorics and graph theory
- Computational aspects of communication
- Computational biology
- Computational complexity
- Computational geometry
- Computational learning
- Computational Models and Complexity
- Computational Number Theory
- Cryptography and information security
- Data structures
- Database theory
- Design and analysis of algorithms
- Distributed computing
- Information retrieval
- Operations Research
- Parallel algorithms
- Quantum Computation
- Randomness in Computation

**Information for Librarians**
Foundations and Trends® in Theoretical Computer Science, 2005, Volume 1, 4 issues. ISSN paper version 1551-305X. ISSN online version 1551-3068. Also available as a combined paper and online subscription.

Foundations and Trends® in
Theoretical Computer Science
Vol. 1, No 4 (2005) 237–301

DOI: 10.1561/0400000009

now
the essence of knowledge

# Pairwise Independence and Derandomization

**Michael Luby[1] and Avi Wigderson[2]**

[1] *Digital Fountain, Fremont, CA, USA*
[2] *Institute for Advanced Study, Princeton, NJ, USA, avi@ias.edu*

## Abstract

This article gives several applications of the following paradigm, which has proven extremely powerful in algorithm design and computational complexity. First, design a probabilistic algorithm for a given problem. Then, show that the correctness analysis of the algorithm remains valid even when the random strings used by the algorithm do not come from the uniform distribution, but rather from a small sample space, appropriately chosen. In some cases this can be proven directly (giving "unconditional derandomization"), and in others it uses computational assumptions, like the existence of 1-way functions (giving "conditional derandomization").

The article is based on a series of lectures given by the authors in 1995, where the notes were scribed by the attending students. (The detailed list of scribes and other contributors can be found in the Acknowledgements section at the end of the manuscript.) The current version is essentially the same, with a few minor changes. We note that this publication takes place a decade after the lectures were given. Much has happened in the area of pseudorandomness and derandomization since, and perhaps a somewhat different viewpoint, different material, and different style would be chosen were these lectures given today. Still, the material presented is self contained, and is a prime

manifestation of the "derandomization" paradigm. The material does lack references to newer work though. We recommend the reader interested in randomness, derandomization and their interplay with computational complexity to consult the following books and surveys, as well as their extensive bibliography: [31, 14, 36, 37, 21, 42].

# Contents

# 1

# Pairwise Independence

In this chapter, and indeed in much of the rest of this article, we will be considering randomized algorithms, and their performance when their input is not "purely" random. To set the stage, let us consider an algorithm $A$ that on input $x$ wishes to evaluate some function $f$ at $x$. A randomized algorithm for this task may use an input a sequence of random variables $Z_1, \ldots, Z_n$ where the $Z_i$'s take their value from some finite set $T$. Informally, $A$ would be considered good for computing $f$, if it is very likely to output $f(x)$, when the $Z_i$'s are drawn *independently* and *uniformly* from the set $T$. But what happens, when the random variables are not chosen uniformly, and especially, independently. Depending on the level of independence, the support of the joint distribution on $Z_1, \ldots, Z_n$ could be much smaller than the support of the independent uniform distribution. In turn this allows for efficient calculation of the probability of various events (or to compute the most likely output of $A$ on input $x$). In this section, we introduce definitions that study distributions that are not fully independent, and show some algorithmic uses.

## 1.1 Pairwise independence: Definition

Consider a set of $N$ random variables indexed by a set $U$, i.e., $\{Z_x : x \in U\}$ with $|U| = N$, that take on values from a set $T$, (i.e., $Z_x \in T$). Let $D : T^U \to [0,1]$ denote their joint distribution function, i.e., for $\boldsymbol{\alpha} = \{\alpha_x \in T | x \in U\}$ let $\Pr[\forall x \in U, Z_x = \alpha_x] = D(\boldsymbol{\alpha})$.

For finite $t = |T|$, a uniform distribution (i.e., $D(\boldsymbol{\alpha}) = 1/t^n$) assigns $\Pr[Z_x = \alpha_x] = 1/t$, for all $x \in U$, $\alpha_x \in T$. If this distribution satisfies, for all $x \neq y \in U$ and for all $\alpha, \beta \in T$,

$$\Pr[Z_x = \alpha, Z_y = \beta] = \Pr[Z_x = \alpha] \cdot \Pr[Z_y = \beta] = 1/t^2,$$

then we refer to this distribution as *pairwise independent*.

Pairwise independence is not the same as complete independence. For example, consider following set of three pairwise-independent binary variables ($U = \{1,2,3\}, T = \{0,1\}, t = 2$), where each row gives an assignment to the three variables and the associated probability. (The leftmost column gives an index $s \in \{0,1\}^2$ for each row.)

| $s$ | $Z_1$ | $Z_2$ | $Z_3$ | $D(\cdot)$ |
|---|---|---|---|---|
| 00 | 0 | 0 | 0 | $\frac{1}{4}$ |
| 01 | 0 | 1 | 1 | $\frac{1}{4}$ |
| 10 | 1 | 0 | 1 | $\frac{1}{4}$ |
| 11 | 1 | 1 | 0 | $\frac{1}{4}$ |

The above distribution has its support on only four elements of $T^U$, whereas the uniform distribution has support on all eight elements. (The support of a distribution $D(\,)$ is the set of elements $\boldsymbol{\alpha} \in T^U$ for which $D(\boldsymbol{\alpha}) > 0$.) Shortly, we will see that the support of pairwise independent distrubtions can get *much* smaller than the support of the uniform distribution, as $N = |U| \to \infty$.

The notion of pairwise independence emerged first in the context of "hash functions". To describe this context, notice that each row $s$ above can be thought of as a function $h_s : U \to T$, where $h_s(x) = Z_x$. Let $\mathcal{S}$ be the set of indices $s$ for these functions. So, in this case, $\mathcal{S} = \{0,1\}^2$. For all $x \neq y \in U$, for all $\alpha, \beta \in T$, we have

$$\Pr_{s \in_R \mathcal{S}}[h_s(x) = \alpha \wedge h_s(y) = \beta] = 1/4 = 1/t^2$$

(where the notation $s \in_R \mathcal{S}$ denotes that $s$ is chosen uniformly at random from the set $\mathcal{S}$). (Notice in particular that $\Pr_{s \in_R \mathcal{S}}[h_s(x) = h_s(y)] = 1/2 = 1/t$.) Any set of functions satisfying this condition is a 2-universal family of hash functions. Definitions and explicit constructions of 2-universal hash functions were first given by Carter-Wegman [40]. The original applications described in Carter-Wegman [40] were straightforward, similar to those described in the later section on dictionaries based on hashing. As these notes indicate, subsequently 2-universal hashing has been applied in surprising ways to a rich variety of problems.

## 1.2 Small families of hash functions

In the last section we saw how to construct $N = 3$ pairwise independent random variables with a smaller support than needed for 3 fully independent random variables. (Of course, we couldn't have picked a smaller choice of $N$ to demonstrate this difference!) But to get a true sense of the difference, it is useful to let $N \to \infty$. In this section we will show how to construct $N = 2^n$ random variables, indexed by the set $U = \{0,1\}^n$ taking on values in the set $T$ which is also chosen to be $\{0,1\}^n$.

One simple way to construct a family of hash functions mapping $\{0,1\}^n \to \{0,1\}^n$ is to let $\mathcal{S} = \{0,1\}^n \times \{0,1\}^n$, and then for all $s = (a,b) \in \mathcal{S}$, for all $x \in \{0,1\}^n$ define $h_s(x) = ax + b$, where the arithmetic operations are with respect to the finite field GF$[2^n]$. Thus, each $h_s$ maps $\{0,1\}^n \to \{0,1\}^n$ and $\mathcal{S}$ is the index set of the hash functions. For each $s = (a,b) \in \mathcal{S}$, we can write:

$$\begin{pmatrix} h_s(x) \\ h_s(y) \end{pmatrix} = \begin{pmatrix} x & 1 \\ y & 1 \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix}$$

When $x \neq y$, the matrix is non-singular, so that for any $x, y \in \{0,1\}^n$, the pair $(h_s(x), h_s(y))$ takes on all $2^{2n}$ possible values (as $s$ varies over all $\mathcal{S}$). Thus if $s$ is chosen uniformly at random from $\mathcal{S}$, then $(h_s(x), h_s(y))$ is also uniformly distributed. This property of hash functions is called 2-*universal*.

We can view $\mathcal{S}$ as the set of points in a sample space on the set of random variables $\{Z_x : x \in \{0,1\}^n\}$ where $Z_x(s) = h_s(x)$ for all $s \in S$. With respect to the uniform distribution on $S$, these random variables are pairwise independent, i.e., for all $x \neq y \in \{0,1\}^n$, for all $\alpha, \beta \in \{0,1\}^n$

$$\Pr_{s \in_R \mathcal{S}}[Z_x(s) = \alpha \wedge Z_y(s) = \beta]$$
$$= \Pr_{s \in_R \mathcal{S}}[Z_x(s) = \alpha] \cdot \Pr_{s \in_R \mathcal{S}}[Z_y(s) = \beta] = 1/2^{2n}.$$

To obtain a hash function that maps to $k < n$ bits, we can still use $\mathcal{S}$ as the function index family: The value of the hash function indexed by $s$ on input $x$ is obtained by computing $h_s(x)$ and using the first $k$ bits.

The imporant properties of these hash functions are:

- Pairwise independence.
- Succinctness – each function can be described as a $2n$-bit string. Therefore, randomly picking a function index requires only $2n$ random bits.
- The function $h_s(x)$ can easily be computed (in **LOGSPACE**, for instance) given the function index $s$ and the input $x$.

In the sequel, unless otherwise specified we are referring to this set of pairwise independent hash functions and $\mathcal{S}$ denotes the set of indices for the hash family.

## 1.3 Derandomization applications

Consider, for example, the MAXCUT problem: given a graph $G = (V, E)$, find a two-coloring of the vertices $\chi : V \to \{0,1\}$ so as to maximize $c(\chi) = |\{(x,y) \in E : \chi(x) \neq \chi(y)\}|$. We describe a solution to this problem that is guaranteed to produce a cut where at least half the edges cross the cut.

If the vertices are colored randomly (0 or 1 with probability 1/2) by choosing $\chi$ uniformly from the set of all possible $2^{|V|}$ colorings, then:

$$E[c(\chi)] = \sum_{(x,y) \in E} \Pr[\chi(x) \neq \chi(y)] = \frac{|E|}{2}$$

Thus, there must always be a cut of size at least $\frac{|E|}{2}$. Let $\mathcal{S}$ be the index set for the hash family mapping $V \to \{0,1\}$. Since the summation above only requires the coloring of vertices to be pairwise-independent, it follows that $E[c(h_s)] = \frac{|E|}{2}$ when $s \in_R \mathcal{S}$. Since $|\mathcal{S}| = |V|^2$, we can deterministically try $h_s$ for all $s \in \mathcal{S}$ in polynomial time (even in the parallel complexity class **NC**), and for at least one $s \in \mathcal{S}, h_s$ defines a partition of the nodes where at least $\frac{|E|}{2}$ edges cross the partition.

This derandomization approach was developed and discussed in general terms in the series of papers Chor-Goldreich [10], Luby [28], Alon-Babai-Itai [5]. There, the approach was applied to derandomize algorithms such as witness sampling, a fast parallel algorithm for finding a maximal independent set, and other graph algorithms.

## 1.4 Dictionaries

One application that uses hashing is in building "dictionaries". A dictionary is a data structure that represents a subset $N$ of size $|N| = n$ from some universe of possible words $U$ so as to be able support queries of the form "$x \in N$?", for $x \in U$. (This is a natural abstraction of the classical notion of a "dictionary" for, say, English where $U$ may be thought of as all sequences of upto 20 English letters, while $N$ is the subset which are actually words in English.) Deterministic schemes based on balanced tree data structures build such data structures in time $\mathcal{O}(n \log n)$ and subsequent look-ups in time $\mathcal{O}(\log n)$ each.

Random hashing can speed this up considerably. The simple use of hashing would be as follows. Let $T = \{1, \ldots, t\}$ be a set of sufficiently large cardinality, and let $h$ be some "hash" function mapping $U \to T$. We will store the elements of $N$ in an array $D[1, \ldots, t]$, with elements being indexed by elements of $T$. Our intent would be to set $D[h(x)] = x$ for every $x \in N$. Of course this would not be possible if we had "collisions", i.e., if $h(x) = h(y)$ for $x \neq y \in N$. Assuming $h$ is collision-free on $N$, easy to describe, that $h(x)$ can be computed in "constant" time, and assuming further that $D[1, \ldots, t]$ is initialized properly when we start, insertions and lookups take constant time. It turns out that by picking the hash function $h$ randomly from a small set (as described in Section 1.2) can now be used to remove most of

these assumptions. In particular if we choose $s \in_R \mathcal{S}$, and use $h = h_s$ then the expected number of colliding pairs C may be bounded from above as follows:

$$E[C] = \sum_{x \neq y \in N} \Pr_{s \in_R \mathcal{S}}[h_s(x) = h_s(y)] = \binom{n}{2} \cdot \frac{1}{t}$$

For instance, if $t = n^2$, then $E[C] \leq \frac{1}{2}$ (and so the probability that $h$ is 1-to-1 is $\geq \frac{1}{2}$). (If $t = n$, then $E[C] \leq \frac{n}{2}$.) Thus this yields a simple hashing scheme in which insertions and look-ups cost a unit time, though the size of the data structure representing $N$ is of size $O(n^2)$ (assuming that the table $D$ of size $n^2$ is initialized properly). Below we see how to improve upon the space complexity.

The following two-level hashing scheme, due to Fredman-Komlós-Szemerédi [12], also takes time $\mathcal{O}(n)$ to construct the dictionary and constant time for each look-up, but the advantage is that it uses only $\mathcal{O}(n)$ cells in total. Let $T = \{1, \ldots, n\}$.

(1) Pick $s \in_R \mathcal{S}$ and map $N$ into $T$. For each $i \in T$, let $N_i$ be the subset of $N$ mapped to $i$ by $h_s$, and let $n_i = |N_i|$. Let $C = \sum_{i \in T} \binom{n_i}{2}$ be the number of colliding pairs. If $C > n$ then start over at step (1), else go on to step (2).
(2) For each $i \in T$, if $n_i \geq 1$ then we allocate a table $T_i$ of $n_i^2$ cells, and let $\mathcal{S}_i$ denote the index set for the pairwise independent hash family (as in Section 1.2) mapping $U \to T_i$. Pick $s_i \in_R \mathcal{S}_i$, and use $h_{s_i}$ to map $N_i$ to $T_i$. If $h_{s_i}$ maps $N_i$ 1 to-1 into $T_i$ then this is a good choice for $s_i$, else rechoose $s_i$ independently and try again until $h_{s_i}$ does describe a 1-to-1 mapping.

Because $E[C] \leq n/2$ in step (1), $\Pr[C \leq n] \geq 1/2$, and thus the expected number of times step (1) is repeated is at most 2. Similarly, in step (2), for each $i \in T$, the expected number of times till the mapping of $N_i$ into $T_i$ is 1-*to*-1 is at most 2. Thus, the overall expected time to construct the dictionary is $\mathcal{O}(n)$. The total number of cells used to store $N$ is $D = \sum_{i \in T} n_i^2$. Noting that $D - 2C = |N| = n$, and that $C \leq n$, it

follows that at most $3n$ cells are used to store $N$. Note that we need to also store $s$ and all $s_i$ for all $i \in \{1,\ldots,n\}$, but this takes at most $2(n+1)$ additional cells, since the description of each hash function takes two cells.

Each find operation takes constant time.

# References

**Abbreviations**

- STOC: Proceedings of the ACM Symposium on Theory of Computing
- FOCS: Proceedings of the IEEE Foundations of Computer Science


[1] L. Adleman, "Two theorems on random polynomial time," *FOCS*, pp. 75–83, 1978.

[2] M. Ajtai, "$\sum_1^1$-Formulae on finite structures," *Annals of Pure and Applied Logic*, vol. 24, pp. 1–48, 1983.

[3] M. Ajtai, J. Komlos, and E. Szemeredi, "Deterministic simulation in LOGSPACE," *STOC*, p. 132, 1987.

[4] W. Alexi, B. Chor, O. Goldreich, and C. Schnorr, "RSA/Rabin functions: Certain parts are as hard as the whole," *SIAM J. on Computing*, vol. 17, no. 2, pp. 194–209, April 1988.

[5] N. Alon, L. Babai, and A. Itai, "A fast and simple randomized parallel algorithm for the maximal independent set problem," *Journal of Algorithms*, vol. 7, pp. 567–583, 1986.

[6] N. Alon and F. R. K. Chung, "Explicit construction of linear sized tolerant networks," *Discrete Math*, vol. 72, pp. 15–19, 1989.

[7] C. Bennett and J. Gill, "Relative to a random oracle A, $\mathbf{P}^A \neq \mathbf{NP}^A \neq co - \mathbf{NP}^A$ with probability one," *Siam J. on Computing*, vol. 10, pp. 96–113, 1981.

[8] M. Blum and S. Micali, "How to generate cryptographically strong sequences of pseudo-random bits," *SIAM J. on Computing*, vol. 13, pp. 850–864, A preliminary version appears in *FOCS*, 1982, pp. 112–117, 1984.

[9] J. Cai, "With probability one, a random oracle separates PSPACE from the polynomial-time hierarchy," *J. of Computer and System Sci.*, vol. 38, pp. 68–85, A preliminary version appears in *STOC*, 1986, pp. 21–29, 1989.

[10] B. Chor and O. Goldreich, "On the power of two-point sampling," *Journal of Complexity*, vol. 5, pp. 96–106, 1989.

[11] A. Cohen and A. Wigderson, "Dispersers, deterministic amplification, and weak random sources," *FOCS*, pp. 14–19, 1989.

[12] M. Fredman, J. Komlos, and E. Szemeredi, "Storing a sparse table in O(1) worst case access time," *Journal of the ACM*, vol. 31, pp. 538–544, 1984.

[13] M. Furst, J. Saxe, and M. Sipser, "Parity, circuits and the polynomial time hierarchy," *FOCS*, pp. 260–270, 1981.

[14] O. Goldreich, S. Goldwasser, and S. Micali, "How to construct random functions," *J. of ACM*, vol. 33, no. 4, pp. 792–807, A preliminary version appears in *FOCS*, 1984., 1986.

[15] O. Goldreich and L. Levin, "A hard-core predicate for any one-way function," *STOC*, pp. 25–32, 1989.

[16] S. Goldwasser and S. Micali, "Probabilistic encryption," *J. of Computer and System Sci.*, vol. 28, pp. 270–299, A preliminary version appears in *STOC*, 1982, pp. 365–377., 1984.

[17] S. Goldwasser and M. Sipser, "Private coins vs public coins in interactive proof systems," *STOC*, pp. 59–68, 1986.

[18] J. Håstad, *Computational limitations for small depth circuits.* Ph.D. thesis, 1986. MIT press.

[19] J. Håstad, R. Impagliazzo, L. A. Levin, and M. Luby, "A pseudo-random generator from any one-way function," *SIAM Journal on Computing*, vol. 28, no. 4, pp. 1364–1396, 1999.

[20] S. Hoory, N. Linial, and A. Wigderson, "Expander graphs and their applications," *Bulletin of the AMS*, to appear.

[21] R. Impagliazzo, L. Levin, and M. Luby, "A pseudo-random generator from any one-way function," *STOC*, pp. 12–24, 1989.

[22] R. Impagliazzo and D. Zuckerman, "How to recycle random bits," *FOCS*, pp. 248–253, 1990.

[23] R. Karp and M. Luby, "Monte-carlo algorithms for the planar multiterminal network reliability problem," *J. of Complexity*, vol. 1, pp. 45–64, 1985.

[24] R. Karp, M. Luby, and N. Madras, "Monte-carlo approximation algorithms for enumeration problems," *J. of Algorithms*, vol. 10, no. 3, pp. 429–448, 1989.

[25] R. Karp, N. Pippenger, and M. Sipser, "Expanders, randomness, or time versus space," *First Annual Conference on Structure in Complexity Theory*, pp. 325–329, 1986.

[26] M. Karpinski and M. Luby, "Approximating the number of solutions to a GF[2] formula," *Journal of Algorithms*, vol. 14, no. 2, pp. 280–287, March 1993.

[27] A. Lubotzky, R. Phillips, and P. Sarnak, "Explicit expanders and the ramanujan conjectures," *STOC*, pp. 240–246, (See also: A. Lubotzky, R. Phillips, P. Sarnak. "Ramanujan graphs," *Combinatorica*, vol. 8, 1988, pp. 261–277)., 1986.

[28] M. Luby, "A simple parallel algorithm for the maximal independent set problem," *SIAM J. on Computing*, vol. 15, no. 4, pp. 1036–1053, November 1986.
[29] G. Margulis, "Explicit group-theoretical constructions of combinatorial schemes and their application to the design of expanders and superconcentrators," *Problemy Peredachi Informatsii*, vol. 24, pp. 51–60, (in Russian). (English translation in *Problems of Information Transmission*, vol. 24, 1988, pp. 39–46)., 1988.
[30] N. Nisan, "Pseudorandom bits for constant depth circuits," *Combinatorica*, vol. 1, pp. 63–70, 1991.
[31] N. Nisan, "RL⊆SC," *STOC*, pp. 619–623, 1992.
[32] N. Nisan and A. Wigderson, "Hardness vs. randomness," *J. of Comp. Sci. and Sys.*, vol. 49, no. 2, pp. 149–167, 1994.
[33] N. Nisan and D. Zuckerman, "More deterministic simulation in logspace," *STOC*, pp. 235–244, 1993.
[34] C. H. Papadimitriou, *Computational complexity*. 1993. Addison Wesley.
[35] A. Renyi, *Probability theory*. 1970. North-Holland, Amsterdam.
[36] M. Sipser, "A complexity theoretic approach to randomness," *STOC*, pp. 330–335, 1983.
[37] M. Sipser, *Introduction to the theory of computation*. PWS Publishing, 1997.
[38] L. Valiant, "The complexity of computing the permanent," *Theoretical Computer Science*, no. 8, pp. 189–201, 1979.
[39] L. Valiant and V. Vazirani, "NP is as easy as detecting unique solutions," *Theoretical Computer Science*, vol. 47, pp. 85–93, 1986.
[40] M. Wegman and J. Carter, "New hash functions and their use in authentication and set equality," *Journal of Computer and System Sciences*, vol. 22, no. 3, pp. 265–279, 1981.
[41] A. Yao, "Theory and applications of trapdoor functions," *FOCS*, pp. 80–91, 1982.
[42] A. Yao, "Separating the polynomial-time hierarchy by oracles," *FOCS*, pp. 1–10, 1985.