

Randomized Algorithms

A short course on randomized algorithms and the probabilistic method

Wolfgang Merkle
Ruprecht-Karls-Universität Heidelberg *

October 30, 2001

Contents

1	Randomized Algorithms and the Probabilistic Method	2
1.1	Introduction	2
1.2	Basic Facts from Probability Theory	2
1.3	Interactive Proofs	7
1.4	Fingerprinting	10
1.5	Self-correcting programs	12
1.6	Byzantine Agreement	14
2	Derandomization	18
2.1	An Example: Finding Cuts	18
2.2	Derandomization by Conditional Expectation	19
2.3	Derandomization by k -Wise Independent Random Variables	21
3	Randomized Sorting	25
3.1	Sorting Lists	25
3.2	Sorting Nuts and Bolts	27
4	Applications in Logic	36
4.1	The Rado-graph and a 0-1-law for graph theory	36
4.2	A Fragment of First-order Logic with Decidable Satisfiability Problem	39

*Universität Heidelberg, Mathematisches Institut, Im Neuenheimer Feld 294, D-69120 Heidelberg, Germany, merkle@math.uni-heidelberg.de

1 Randomized Algorithms and the Probabilistic Method

1.1 Introduction

These lecture notes provide an introduction to randomized algorithms and the probabilistic method in general. For more detailed accounts of these areas we refer the reader to Alon and Spencer [3], to Habib, McDiarmid et al. [19], and to Motwani and Raghavan [24].

A randomized algorithm is an algorithm such that the behavior of the algorithm depends on an internal or external random source. I.e., on the same input but with different random sources, the algorithm might generate different outputs or might run for a different number of steps. Of course, we will usually require that with sufficiently high probability, the randomized algorithm terminates within a reasonable number of steps and outputs the desired result. Randomized algorithms can be viewed as a special case of the probabilistic method, which roughly amounts to the following. In order to demonstrate that a certain combinatorial object exists (say, a certain subgraph of a given graph), set up an appropriate chance experiment and show that with non-zero probability it results in an object as desired.

In this first section, after reviewing some facts from elementary probability theory, we consider examples of randomized algorithms, which include a toy example of a zero-knowledge protocol, fingerprinting, self-correcting programs and a randomized algorithm that solves the Byzantine agreement problem. In Section 2, we describe two standard techniques for derandomization, the method of conditional expectation and the method of k -wise independent random variables. In Section 3 we consider the randomized version of quicksort and its variant for sorting nuts and bolts and we give a derandomization of the latter algorithm by means of expander graphs. Finally, Section 4 features applications of the probabilistic method in logic.

1.2 Basic Facts from Probability Theory

Probability spaces and random variables. In this section we review some facts from elementary probability. For an introduction to probability theory in general, see the textbooks cited in the bibliography [9, 17, 18, 24]. In the sequel, we consider chance experiments where the set Ω of possible outcomes is finite (e.g., $\Omega = \{0, 1\}$ when tossing a coin) or is countably infinite (e.g., $\Omega = \mathbb{N}$ when counting the number of trials that are necessary to achieve a certain goal). Such chance experiments can be modelled by discrete probability spaces as introduced in Definition 1. The term discrete refers to the fact that the outcomes are well-distinguished and are apart from each other, as opposed to chance experiments like measuring a physical parameter, where the possible outcomes can be viewed

as forming a continuum.

Definition 1 Let Ω be a finite or countably infinite set and assume that we are given a real-valued function

$$\text{Prob}_0 : \Omega \rightarrow [0, 1]$$

such that the values $\text{Prob}_0[\omega]$ add up to 1 (i.e., $\sum_{\omega \in \Omega} \text{Prob}_0[\omega] = 1$). Then Prob_0 determines a function Prob that is defined on all subsets of Ω by

$$\text{Prob}[S] = \sum_{\omega \in S} \text{Prob}_0[\omega].$$

In this situation, the pair (Ω, Prob) is called a discrete probability space and Prob is called a discrete probability measure. Furthermore, $\text{Prob}[S]$ is called the probability of S .

If Prob is applied to singletons, we write $\text{Prob}[\omega]$ instead of $\text{Prob}[\{\omega\}]$, hence for example we have $\text{Prob}[\omega] = \text{Prob}_0[\omega]$ for all ω in Ω .

Example 2 Probability spaces can be used to model chance experiments, i.e., experiments leading to an outcome that is considered as being random. For example, the throw of a fair dice can be modelled by the probability space

$$(\Omega, \text{Prob}) \text{ where } \Omega = \{1, \dots, 6\} \text{ and } \text{Prob}[i] = \frac{1}{6} \text{ for } i = 1, \dots, 6.$$

A probability space where Ω has m elements and each element is assigned probability $1/m$ is called the uniform distribution on Ω . When describing a corresponding chance experiment we speak of picking an elements of Ω uniformly at random.

Definition 3 Let (Ω, Prob) be a discrete probability space.

Any mapping from Ω to the reals is called a (real-valued) random variable on the probability space (Ω, Prob) (or a random variable on Ω , when Prob is understood from the context).

Any random variable $X : \Omega \rightarrow \mathbb{R}$ determines a discrete probability measure Prob_X on its range $\Omega_X = \{X(\omega) : \omega \in \Omega\}$ where for any value x in the range we have

$$\text{Prob}_X[x] = \sum_{\{\omega \in \Omega : X(\omega) = x\}} \text{Prob}[\omega].$$

The probability measure Prob_X is called the distribution of X .

We write $\text{Prob}[X = x]$ instead of $\text{Prob}_X[x]$ and expressions like $\text{Prob}[X \geq x]$ or $\text{Prob}[X \in S]$ have the obvious meaning, e.g.,

$$\text{Prob}[X \geq x] = \sum_{\{\omega \in \Omega : X(\omega) \geq x\}} \text{Prob}[\omega].$$

Example 4 In Example 2, we have modelled the throw of a fair dice by a probability space (Ω, Prob) . Now consider the random variable $X : \Omega \rightarrow \mathbb{R}$ where

$$X(i) = \begin{cases} 1 & \text{in case } i \text{ is prim} \\ 0 & \text{otherwise} \end{cases}$$

Random variables of this type are called indicator variables because their values indicate whether a certain event (say, i being prime) takes place. Concerning the distribution of X , we have

$$\text{Prob}[X = 0] = \text{Prob}[\{1, 4, 6\}] = \frac{1}{2} \quad \text{and} \quad \text{Prob}[X = 1] = \text{Prob}[\{2, 3, 5\}] = \frac{1}{2}.$$

Mutual and Pairwise Independence. When there are several random variables that are defined on the same probability space, we might be interested in the probability that a certain combination of values occurs for the variables. These probabilities are given by the joint distribution of the random variables.

Definition 5 Let X_1, \dots, X_m be random variables on a discrete probability space Ω . Then their joint distribution $\text{Prob}_{X_1, \dots, X_m}$ is given by

$$\text{Prob}_{X_1, \dots, X_m}[r_1, \dots, r_m] = \sum_{\{\omega \in \Omega : X_1(\omega) = r_1, \dots, X_m(\omega) = r_m\}} \text{Prob}[\omega].$$

Again we write $\text{Prob}[X_1 = r_1, \dots, X_m = r_m]$ instead of $\text{Prob}_{X_1, \dots, X_m}[r_1, \dots, r_m]$.

Example 6 shows that the joint distribution of a random variable is not determined by the distributions of the random variables involved.

Example 6 Consider again the probability space from Example 2, which models the throw of a fair dice. Define indicator variables X , Y , and Z for the events that i is prime, is even, and is odd, respectively (i.e., for example $Z(i) = 1$ if and only if i is odd). Then all three random variables have the same distribution because each of them assumes the values 0 and 1 with equal probability $1/2$. However, the joint distribution of X and Y differs from the joint distribution of Y and Z , as we have

$$\text{Prob}[X = 1, Y = 1] = \frac{1}{6} \quad \text{and} \quad \text{Prob}[Y = 1, Z = 1] = 0.$$

This shows that in general the joint distribution is not determined by the distributions of the considered random variables.

If we toss m fair coins and the individual tosses are independent of each other, we can assume that each sequence $b_1 \dots b_m$ of possible outcomes occurs with probability 2^{-m} , i.e., the probability of $b_1 \dots b_m$ is simply the product of the probabilities that the first toss shows b_1 , the second toss results in b_2 , and so on. So for random variables that are in a certain sense mutually independent, the joint distribution can be written as the product of the distributions of the single random variables.

Definition 7 Let X_1, \dots, X_m be random variables on the same discrete probability space. These variables are called mutually independent if for any combination of values r_1, \dots, r_m in the range of X_1, \dots, X_m , respectively, we have

$$\text{Prob}[X_1 = r_1, \dots, X_m = r_m] = \text{Prob}[X_1 = r_1] \cdot \dots \cdot \text{Prob}[X_m = r_m] .$$

The variables X_1, \dots, X_m are called pairwise independent if any pair X_i and X_j with $i \neq j$ is mutually independent, i.e., if for all possible values r_i and r_j we have

$$\text{Prob}[X_i = r_i \text{ and } X_j = r_j] = \text{Prob}[X_i = r_i] \cdot \text{Prob}[X_j = r_j] .$$

If we are given random variables that depend on mutually disjoint “parts” of the underlying probability space (e.g., each random variable depends on a different coin toss), then often we will be able to argue that these random variables are indeed mutually independent and that hence their joint distribution can be obtained easily from the distributions of the single variables.

It can be shown that mutual independence implies pairwise independence. However, for sets of at least three random variables, in general the reverse implication is false. In Section 2, we construct sets of random variables that are pairwise independent but can easily be shown not to be mutually independent. An even simpler example of such a set of random variables, taken from Feller [17, Section IX.1], is the following.

Example 8 Consider the probability space that is given by the uniform distribution on the set

$$\begin{aligned} \Omega = \{ & (1, 2, 3), (1, 3, 2), (2, 1, 3), (2, 3, 1), (3, 1, 2), (3, 2, 1), \\ & (1, 1, 1), (2, 2, 2), (3, 3, 3) \} \end{aligned}$$

i.e., Ω contains all permutations of 1, 2 and 3 plus three tuples of the form (i, i, i) and each of these tuples has probability $1/9$. Now given a tuple ω in Ω , let the random variable X_i , $i = 1, 2, 3$, be the projection to the i th component, e.g., $X_1(2, 1, 3) = 2$. Then it is easily verified that the random variables X_1 , X_2 and X_3 are pairwise independent, however they cannot be mutually independent because the values of any two of them determine the value of the third one.

The Expectation of a Random Variable. Is it reasonable to accept a gamble where with probabilities 0.1 and 0.2 one gains 10 and 5 euro, respectively, while with probability 0.7 one loses 3 euro? When playing this gamble, we can expect to gain

$$0.1 \cdot 10 + 0.2 \cdot 5 - 0.7 \cdot 3 = -0.1$$

euro per game, i.e., the game is slightly disadvantageous. In general, we might ask for the *expected value* of a random variable (also called *value on average*). Before we introduce the corresponding concept of expectation of a random variable, we recall the concept of an absolutely converging infinite sum.

Remark 9 By definition, an infinite sum $\sum_{i \in \mathbb{N}} a_i$, $a_i \in \mathbb{R}$, converges to s if and only if the partial sums $a_0 + \dots + a_n$ converge to s when n goes to infinity. The concept of expectation of a random variable X on a discrete probability space (Ω, Prob) is defined as the infinite sum over the terms $\text{Prob}[\omega]X(\omega)$ with $\omega \in \Omega$. Of course, we do not want the expectation of a random variable to depend on the order in which we sum up this terms. It can be shown that an infinite sum $\sum_{i \in \mathbb{N}} a_i$ converges to the same value no matter in which order we sum up the a_i if and only if this sum is absolutely convergent [25], i.e., if and only if the sum $\sum_{i \in \mathbb{N}} |a_i|$ converges (where $|a_i|$ is the absolute value of a_i).

Definition 10 Let X be a random variable that is defined on a discrete probability space Ω . The expectation of X is

$$\mathbf{E}[X] = \sum_{\omega \in \Omega} \text{Prob}[\omega]X(\omega),$$

provided that this sum converges absolutely. If this condition is satisfied, we say the expectation of X exists, otherwise we say the expectation does not exist.

By Remark 9, the condition on absolute convergence in Definition 10 ensures that the expectation of a random variable X does not depend on the way we order the underlying probability space. Note that the condition is automatically satisfied for random variables that are defined on a finite probability space or are restricted to positive values. An example of a random variable for which the expectation does not exist is given by the random variable $X : i \mapsto 2^{(i+1)}$ that is defined on the probability space $(\mathbb{N}, \text{Prob})$ with $\text{Prob}[i] = 2^{-(i+1)}$.

Example 11 Consider again the probability space (Ω, Prob) from Example 2, which models the throw of a fair dice. Let the random variable X be the identity mapping on $\Omega = \{1, \dots, 6\}$. Then we have

$$\mathbf{E}[X] = \sum_{i \in \{1, \dots, 6\}} \text{Prob}[i] i = \frac{1}{6} + \frac{2}{6} + \dots + \frac{6}{6} = \frac{21}{6} = 3.5.$$

This means that when throwing a fair dice, we can expect to obtain 3.5 points or, put differently, we will obtain 3.5 points on average.

Propositions 12 and 13 are handy tools when trying to calculate the expectation of a random variable. In order to obtain the expectation of X , e.g., we can exploit linearity of expectation in order to express $\mathbf{E}[X]$ as the sum of the expectations of random variables X_1 through X_m such that $\mathbf{E}[X_j]$ can be computed easily. For the proofs of both propositions, we refer to Feller [17, Section IX.2].

Proposition 12 (Linearity of expectation) Let r be any real number and let X and X_1, \dots, X_n be random variables on a discrete probability space Ω such that their expectations all exist. Then the expectation of rX and of $X_1 + \dots + X_m$ exists and we have

$$\mathbf{E}[rX] = r\mathbf{E}[X] \quad \text{and} \quad \mathbf{E}[X_1 + \dots + X_n] = \mathbf{E}[X_1] + \dots + \mathbf{E}[X_n].$$

Proposition 13 *Let X_1, \dots, X_n be random variables on a discrete probability space Ω that are mutually independent and such that their expectations all exist. Then the expectation of the product $X_1 \cdot \dots \cdot X_n$ exists and is given by*

$$\mathbf{E}[X_1 \cdot \dots \cdot X_n] = \mathbf{E}[X_1] \cdot \dots \cdot \mathbf{E}[X_n].$$

The next proposition can be used to bound probabilities of the form $\text{Prob}[X \geq r]$ in terms of the expectation of X , where often the latter is easier to compute than the probability itself.

Proposition 14 (Markov Inequality) *Let X be a random variable that assumes only non-negative values. Then for every positive real number r , we have*

$$\text{Prob}[X \geq r] \leq \frac{\mathbf{E}[X]}{r}.$$

Proof. Let (Ω, Prob) be the probability space on which X is defined. We have

$$\begin{aligned} \mathbf{E}[X] &= \sum_{\omega \in \Omega} \text{Prob}[\omega] X(\omega) && \geq \sum_{\{\omega \in \Omega: X(\omega) \geq r\}} \text{Prob}[\omega] X(\omega) \\ &\geq r \sum_{\{\omega \in \Omega: X(\omega) \geq r\}} \text{Prob}[\omega] && \geq r \text{Prob}[X \geq r], \end{aligned}$$

where the equation is just the definition of expectation and the inequalities hold because X is non-negative, because $X(\omega) \geq r$ for all ω that are considered in the sum, and finally by definition of $\text{Prob}[X \geq r]$. \square

1.3 Interactive Proofs

Consider the following game of two players, which is played in several rounds. At the beginning of a round, each of the players puts down secretly a binary value that is either 0 or 1. Then both values are revealed, Player 1 wins if the values are distinct and Player 2 wins if the values are the same. Now assume that Player 1 follows a deterministic strategy that is known to and can be simulated by Player 2. Then Player 2 knows in advance which value Player 1 will choose and can simply duplicate the value, hence Player 1 will lose all the time. On the other hand, in case Player 1 determines his secret value by independent tosses of a fair coin that are not known to the other player (more precisely, that are independent of the behavior of Player 2), then on average, Player 2 will win exactly half of the rounds, no matter how dumb Player 1 and how smart Player 2 is.

The ability to pursue a randomized strategy indeed makes a big difference in game-theoretical situations. A similar remark holds with respect to cryptographic settings [22, 26, 27], where a certain task has to be completed in such a way that a possible malign adversary cannot achieve his goals. For example,

parties **A** and **B** might want to exchange certain information while at the same time they try to prevent an eavesdropping adversary from getting a clue what this information might be. There is a large variety of solutions to this and related problems and virtually all solutions involve some sort of randomization.

As an example how randomization helps in cryptographic settings, we consider the problem of identifying oneself without revealing any relevant information. Typically, if **A** wants to prove her identity to **B**, this is done such that **A** knows some secret information (say a password) that she sends to **B**, who is able to check whether this is really **A**'s secret. The problem with this simple approach is that anyone who overhears the communication from then on knows **A**'s secret and thus can identify himself as being **A**. Is there a way for **A** to prove her identity without exposing any relevant information about her secret? This question can be answered affirmatively and one possible solution is given by special protocols ruling the communication between the two parties that are called *interactive proof systems with the zeroknowledge property*. We present such an interactive proof system that is based on the problem of legally coloring a graph. While currently this protocol might not be suitable for applications, it shows nicely the essential role of randomization, both for verifying the information sent by **A**, as well as for keeping **A**'s secret.

Definition 15 *Let $G = (V, E)$ be a graph. A k -coloring of G is a mapping*

$$g : V \rightarrow \{1, \dots, k\},$$

and a coloring of G is a k -coloring of G for some k . Such a coloring is legal if for every edge $\{u, v\}$ in E , the "colors" $g(u)$ and $g(v)$ are distinct.

For a given pair of a graph G and a number of colors k it is considered to be hard to tell whether G has a legal k -coloring and if so, to compute such a coloring. Accordingly it is currently believed that for graphs that are large enough and for appropriate values of the parameter k , in general it is infeasible to find a legal k -coloring of G . Now, for the sake of the argument, assume that there are parameters k and n and a randomized procedure that yields a graph G with n nodes together with a legal k -coloring of G such that given just G but not its coloring, according to the current state of the art it is impossible to legally color G within any reasonable resource-bounds.

Then **A** is given a graph G of size n and a legal k -coloring of G . The coloring is only known to **A**, but the graph G and the parameter k are also known to **B**. By assumption, for anyone other than **A** it is virtually impossible to come up with a legal k -coloring of G , hence **A** might identify herself by convincing **B** that **A** knows a legal coloring. In order to satisfy the additional requirement that **A** must not reveal any relevant information about the coloring, **A** can not just send the coloring to **B**. Instead, **A** initiates the following protocol.

Protocol Coloring

Assumption: **A** knows a legal coloring g of G and has access to a random source not known to **B**.

- Step 1 (**A**) Pick uniformly at random a permutation π of $\{1, \dots, k\}$.
Commit to the list of colors $\pi(g(1)), \dots, \pi(g(n))$.
- Step 2 (**B**) Among all edges of G , pick an edge $\{u, v\}$ uniformly at random.
- Step 2 (**B**) Send u and v to **A**.
- Step 3 (**A**) Reveal the colors $\pi(g(u))$ and $\pi(g(v))$ to **B**.
- Step 4 (**B**) Accept if the two nodes are colored with distinct colors from $\{1, \dots, k\}$ and reject, otherwise.

In Protocol Coloring, the expression *committing to a color* refers to putting down the color in such a way that it cannot be changed afterwards and is only accessible after granting permission. In real life, this might be achieved by placing a locked opaque box that holds a token of the color committed to. In a setting of interaction between computers, virtually the same can be achieved by cryptographic primitives known as commitment schemes [4, 8, 20], where the details of these techniques are beyond the scope of this course.¹

When analyzing Protocol Coloring, we find that in case the colors committed to are not a legal coloring of G , then this will be detected with probability at least $1/m$, i.e., the probability of erroneously accepting an illegal coloring is less than $1 - 1/m$. In order to achieve a smaller error probability, we consider an iterated protocol where m copies of Protocol Coloring are run successively and independently, and **B** accepts in the iterated protocol if and only if all copies accept.

Without giving a formal definition of the concept of interactive proof system with the zeroknowledge property, we state that in order to demonstrate that the iterated protocol is such a proof system we have to show that

- (i) **A** can always verify correctly her identity (i.e., in case in all iterations the colors committed to form a legal k -coloring, then **B** accepts).
- (ii) If at all iterations, the colors committed to do not form a legal k -coloring of G (say, because there is no legal k -coloring or because **A** does not know one), then **B** accepts with probability of at most $1/2$.
- (iii) **B** is not able to extract any relevant information while communicating with **A** during the protocol.

¹In order to get an idea how commitment schemes might work, consider the following example. For the sake of the argument, assume that we are given an easy-to-compute one-to-one function $f : \mathbb{N} \rightarrow \mathbb{N}$ such that when we choose x at random, it is not only hard to compute x from $f(x)$, but in fact given $f(x)$ it is infeasible to decide whether x is even or odd. Then we could commit to a value in $\{0, 1\}$ by picking randomly an even or an odd number and publish $f(x)$. Afterwards we can open the box $f(x)$ by revealing x . As f is one-to-one, there is now way to change the value we have committed to.

Assertion (i) is easily verified by inspection of Protocol Coloring. Concerning Assertion (ii), observe that if during any iteration the colors committed to do not form a legal k -coloring, then there is at least one edge of G that has its two endpoints colored illegally. Now any of the m edges of G is selected with equal probability, hence during each iteration \mathbf{B} selects an illegally colored edge and hence detects the illegal coloring with probability at least $1/m$. Since the iterations are assumed to be mutually independent (i.e., the random bits used for the different iterations are mutually independent), the probability of accepting in case none of the colorings committed to is legal can be bounded from above by

$$\left(1 - \frac{1}{m}\right)^m \leq \frac{1}{e} \leq \frac{1}{2},$$

(Recall that $(1 - 1/m)^m$ converges nondecreasingly to $1/e$ with $e = 2,71\dots$) Finally, consider the communication between \mathbf{A} and \mathbf{B} . No matter how \mathbf{B} determines the edge of which she wants to know the colors of the endpoints, i. e., even if \mathbf{B} deviates from the protocol and does not pick the edge at random, in each iteration \mathbf{B} just receives two distinct random colors in $\{1, \dots, k\}$. Furthermore, the pairs of colors obtained during the different iterations are mutually independent. These colors reveal to \mathbf{B} some partial information about \mathbf{A} 's random source, however, intuitively it is obvious that \mathbf{B} does not learn anything relevant about colorings of G . In order to formalize the latter argument, observe that \mathbf{B} easily could produce on its own a sequence of pairs of colors that has the same distribution as the sequence evolving during the protocol, hence the actual communication that occurs does not improve \mathbf{B} 's knowledge. The latter formalization is a special form of the formal definition of the zeroknowledge property, where it is required that \mathbf{B} can generate on its own sequences of exchanged messages that have the same distribution as the messages actually exchanged with \mathbf{A} .

1.4 Fingerprinting

Consider the following situations where we want to compare two possibly large binary words (say, a large file), while comparing them bit by bit is infeasible.

- (i) When transmitting a file over a possibly noisy channel, we might want to check the correctness of the transmission without transmitting the whole file twice.
- (ii) We might want to check whether the current version of a file differs from an older version without storing a separate copy of the older version.

In both situations, instead of comparing the files themselves, we could compare *fingerprints* of them. Roughly, a fingerprint of a word w is obtained by mapping the word to a much smaller word $f(w)$ in such a way that with high probability, distinct words have distinct fingerprints. For example, for all

words $w = w_1 \dots w_n$ of length n and any natural number $k > 0$, let

$$\tilde{f}(w) = 2^n w_n + 2^{n-1} w_{n-1} + \dots + 2w_1 \quad \text{and} \quad f_k(w) = \tilde{f}(w) \pmod{k}. \quad (1)$$

We may use $f_k(w)$ as a fingerprint of w . The value $f_k(w)$ requires approximately $\log k$ bits of storage, so if we choose $k < n$, then $f_k(w)$ will be considerably smaller than w itself. Thus in the first situation above it is reasonable to transmit the fingerprint $f_k(w)$ together with the transmitted word w , while in the second situation we can compare the fingerprint of the current version of the file with the stored fingerprint of the old version.

Trivially, for any number k the fingerprints $f_k(u)$ and $f_k(v)$ of identical words u and v will be the same. When are these fingerprints the same for distinct words u and v ? This happens if and only if $f(u)$ and $f(v)$ leave the same remainder modulo k , i.e., if and only if

$$k \text{ divides } |f(u) - f(v)|. \quad (2)$$

In case the changes we want to detect are caused by a random process, say, by random transmission errors, using a fixed value of k might be good enough. However, this deterministic strategy fails against deliberate changes by a clever adversary who knows the parameter k . In the second situation above, such an adversary might first change relevant parts of the file (say, information related to an account), then disguising the changes by changing less relevant parts (say, garbage data remaining from an already deleted file) in such a way that (2) holds when u and v denote the old and the new version of the file. We argue now that by choosing a prime modulus at random, there is only a small probability that the changes of the adversary are not reflected in the fingerprints.²

Algorithm Fingerprint

(Parameter: a natural number t)

Input: Two words u and v of length $n \geq 3$.

Choose p uniformly at random among the first tn prime numbers.

If $f_p(u) = f_p(v)$ then accept, else reject.

Proposition 16 *Assume that Algorithm Fingerprint with parameter t is applied to words u and v . In case both words are identical, the algorithm accepts with probability 1, in case they differ, the algorithm accepts with probability at most $1/t$.*

Proof. In case u and v are the same, then $f_p(u) = f_p(v)$ for all values of p , hence the algorithm accepts with probability 1. Next assume, on the other hand, that u and v differ and let

$$d = |f(u) - f(v)|.$$

²Of course, we have to assume that the adversary does not know the parameter k . Moreover, the adversary must not be able to change the stored fingerprint. The latter requirement is reasonable because the fingerprint is much smaller than the file itself and hence might be stored by different means or in a different location.

Then the algorithm accepts if and only if the randomly chosen prime p divides d . Moreover, $1 \leq d \leq 2^{n+1} \leq 2 \cdot 3^{n-1}$ for $n \geq 3$, hence d differs from 0 and has at most n distinct prime factors. So the probability that p is equal to one of these factors, i.e., the probability that p divides d , is at most $n/(tn) = 1/t$. \square

Remark 17 *Algorithm Fingerprint requires to choose a prime at random among the first tn primes. This is no problem in case tn is so small that we can store a table of all this primes. For larger values of tn we can proceed as follows. By the prime number theorem (see Aigner and Ziegler [1, Chapter 1] and the references cited there), we have*

$$\lim_{m \rightarrow \infty} \frac{|\{p \leq m : p \text{ is prime}\}|}{m/\log m} = 1,$$

i.e., for large m , there are approximately $m/\log m$ prime numbers less than or equal to m . Thus we can choose a number m such that there are approximately tn prime numbers less than or equal to m . Then we pick successively numbers less than or equal to m and run a randomized primality test against them, until we find a number p that passes the test.

If we use p in Algorithm Fingerprint as before, the probability of error will be roughly the same, except that we have to take into account that with small probability, the number p we have chosen is not a prime (in which case our analysis of the error probability is no longer valid). Note that even if p is not a prime, Algorithm Fingerprint will always accept any pair of identical words. For details and for an introduction to primality tests see for example Motwani and Raghavan [24, Sections 7 and 14.6].

1.5 Self-correcting programs

The existing methods for ensuring correctness of hard- or software are not fully satisfying, e.g., they are likely to overlook certain bugs, as it is the case for simple testing, or they are impracticable in most situations, as it is the case for the various approaches to formal verification. As a consequence, current hard- and software is apt to produce erroneous results under certain circumstances or at certain inputs. For example, the well-known division bug in the Pentium micro-processor resulted in incorrect computation for certain rare inputs. Blum and Wasserman [5], after discussing the Pentium division bug and similar failures, investigate into general methods for checking and correcting programs. Among others, they propose a method for correcting a program where the result for the actual input is expressed in terms of the result for a randomly chosen input. This is done in such a way that even in case the actual input leads to an erroneous computation, this is unlikely for the random input. Similarly, a program can be checked by relating the result for the actual input to randomly chosen, less complex inputs. As an example for these methods, we present checkers and correctors for multiplication. While Blum and Wasserman deal with multipli-

cation of floating point numbers, we consider the simpler setting of multiplying two natural numbers that are represented by a word of a fixed length n .

In order to check whether the product of a and b is indeed c , we might use Algorithm MultiplicationCheck.

Algorithm MultiplicationCheck

(Parameter: a natural number t)

Input: three natural numbers a , b and c of size at most 2^n .

Choose p uniformly at random among the first tn prime numbers.

Let $d_1 = c \bmod p$.

Let $d_2 = ((a \bmod p)(b \bmod p)) \bmod p$

If $d_1 = d_2$ then accept, else reject.

Algorithm MultiplicationCheck accepts inputs where $ab = c$ with probability 1. For other inputs, the algorithm accepts with probability at most $1/t$, as can be shown by essentially the same proof as for Proposition 16. Similarly, Remark 17 on the choice of p in Algorithm Fingerprint applies also to the choice of p in Algorithm MultiplicationCheck. For values of t that are not too large, the computation of d_1 and d_2 can be considered to be less complex and hence to be less likely to be incorrect than the computation of ab .

Next we deal with correcting a program that is meant to be used for multiplying two n -bit natural numbers but produces incorrect results on certain input pairs (i.e., on the same input, the program is either always correct or is always incorrect). The correcting routine requires to multiply $(n + 1)$ -bit natural numbers, so we assume that the multiplication program actually takes pairs of such numbers as input. Furthermore, we assume that an ε -fraction of these pairs are multiplied incorrectly.

Algorithm MultiplicationCorrection

Input: two natural numbers a and b , each represented by n bits.

Determine n -bit natural numbers r_1 and r_2 by $2n$ independent tosses of a fair coin.

Let $c = (a + r_1)(b + r_2) - r_1(b + r_2) - r_2(a + r_1) + r_1r_2$.(*)

Output: c (where c is assumed to be equal to ab .)

Proposition 18 *Assume that Algorithm MultiplicationCorrection is run on a processor where addition and subtraction are always correct, while multiplication is incorrect for an ε -fraction of all pairs of $(n + 1)$ -bit natural numbers. Then for any pair a and b of n -bit natural numbers, the probability that the algorithm fails to compute the product of a and b correctly is at most 16ε .*

Proof. By multiplying out the right-hand side of (*) it is easy to see that c is indeed equal to the product of a and b , provided that the arithmetical operations involved in the calculation of c have all been evaluated correctly. By assumption,

addition and subtraction are always correct, so when calculating the probability for an error, it suffices to consider the four multiplications in (*). If we can argue that for each of them, the probability for an incorrect computation is at most 4ε , then we are done because by adding up the four error probabilities we obtain that the overall error probability is at most 16ε .

We calculate a bound for the error probability of the second product, i.e., for the probability that the product of r_1 and $(b + r_2)$ is evaluated incorrectly, and we omit the virtually identical considerations for the three remaining cases. By construction, the words r_1 and r_2 are chosen uniformly and independently among all the 2^n words of length n , hence the pair $(r_1, b + r_2)$ is chosen uniformly from a set P of 2^{2n} pairs. The set P comprises a $1/4$ -fraction of the $2^{2(n+1)}$ pairs of $(n+1)$ -bit integers (where, for example, we consider the n -bit natural number r_1 as an $(n+1)$ -bit natural number in the natural way). Furthermore, the fraction of pairs of $(n+1)$ -bit natural numbers that are multiplied incorrectly is ε . As a consequence, the fraction of pairs in P that lead to an incorrect multiplication is at most 4ε . \square

Remark 19 *Blum and Wasserman report that the pentium division bug results in an incorrect computation for less than one in eight billions inputs. By applying Algorithm MultiplicationCheck, we could transform the processor into a device that for any input computes the correct result with extremely high probability, i.e., for any input, only about two in a billion invocations will not produce the correct result. Furthermore, by repeatedly applying the corrected program to the same input and then determining the output by a majority vote among the possibly non-identical results, the probability of an error that is solely caused by the division bug can be made so small that an error due to other failures is much more likely.*

1.6 Byzantine Agreement

In this section we describe a randomized protocol that solves the *Byzantine agreement problem* from the theory of distributed computing. We follow the exposition by Motwani and Raghavan [24], see there for details and further references.

In the Byzantine agreement problem, we have n processors (or, say, Byzantine generals, ...) that communicate with each other in order to reach an admissible agreement on a binary value b , where admissible agreement will be defined shortly. For a fraction of strictly less than $\delta = 1/8$ of the processors we cannot assume collaboration, in fact, these processors might even deliberately proceed in such a way as to prevent the group from reaching such an agreement. Depending on the actual application, these processors might be considered as being faulty or being traitors and we call them the bad processors. On the other hand, a fraction of at least $1 - \delta = 7/8$ of the processors collaborate in reaching an admissible agreement, these are the good processors.

Each processor has an initial binary value and the byzantine agreement problem

asks for a protocol that results in an agreement on a binary value b that reflects to a certain extent the majority among the initial values. However, as not all processors collaborate, there cannot be a protocol such that b is always the majority vote with respect to the initial values. Instead, we require that the protocol results in an admissible agreement in the following sense.

- (i) Finally, the good processors must agree on the same value b (in particular, each good processor must know b).
- (ii) In case all the good processors have the same initial value, then b must be equal to this value.

The communication between the processors is done in rounds $s = 1, 2, \dots$. At the beginning of each round, each processor sends messages to all other processors. The messages sent by a processor to different receivers might differ. Before the communication starts, the bad processors may agree on a common strategy that is not known to the good processors. This strategy can be arbitrarily complex and may determine the behavior of the processors in all situations that may possibly occur in the course of the communication. Furthermore, at the beginning it is not known to the good processors which processors are bad.

A natural strategy for reaching an agreement on a binary value between several processors is the following. Each processor votes for either 0 or 1 and the agreement is on 1 if and only if the total count of votes for 1 exceeds a certain threshold t . However, as there is no centralized trusted authority that might decide whether the threshold is exceeded, each processor has to reach its own decision. Now assume that the number of good processors that vote for 1 is below the threshold, while in case all the bad processors would also vote for 1, the threshold would be exceeded. Then in the local view of any processor the threshold is exceeded or not, depending on the number of 1's this processor receives from the bad processors. So by sending different messages to the different processors, the bad processors can arrange any pattern of locally exceeded thresholds they like. In the Protocol ByzantineAgreement, this problem is handled by a simple trick. In each round, a common threshold is chosen randomly and uniformly among one of two possible values. The point in choosing a random threshold is that the threshold is only determined after all the messages have been sent and that the difference between the two possible thresholds is larger than the number of bad processors. Hence no matter what the actual count of 1's among the votes of the good processors is, at least for one of the thresholds the messages sent by the bad processors do not make a difference with respect to the question whether the threshold is exceeded or not. Thus with probability at least $1/2$, the decisions of all good processors are the same, hence all good processors vote the same in the next round, which then ends the protocol.

Protocol ByzantineAgreement

(There are $n \geq 2$ processors, the following is the algorithm for Processor i .)

Input: A binary value b_i .

Fix constants $t_0 = \frac{5}{8}n$ and $t_1 = \frac{6}{8}n$.

Let $v_i(1) = b_i$.

For rounds $s = 1, 2, \dots$ do the following.

Send $v_i(s)$ to all others.

For all $j \neq i$, receive $v_j(s)$ from Processor j .

For $l = 0, 1$, let $c_l = |\{j : 1 \leq j \leq m \text{ and } v_j(s) = l\}|$.

If $c_0 \geq c_1$ then $u(s) = 0$ and $c(s) = c_0$,

else $u(s) = 1$ and $c(s) = c_1$.

(The most frequent value among $v_1(s), \dots, v_n(s)$ is $u(s)$
and $c(s)$ is its count.)

Let $\tau(s) \in \{0, 1\}$ be obtained by tossing a fair coin

(The random bit $\tau(s)$ is the same for and is known to all processors.)

If $c(s) > t_{\tau(s)}$, then $v_i(s+1) = u(s)$, else $v_i(s+1) = 0$.

If $c(s) > \frac{7}{8}n$, then assume that the agreement is on $u(s)$

and let $v_i(s+1) = v_i(s+2) = v_i(s+3) = \dots = u(s)$.

Proposition 20 *Let a group of n processors communicate with each other such that all but a fraction of at most $\delta = 1/8$ of the processors obey the Protocol ByzantineAgreement. Then an admissible agreement is reached with probability 1 and in an expected number of rounds that is constant.*

Proof. First observe that in case all the good processors share the same initial value b , then more than $7/8$ of the processors send message b in the first round, hence the bad processors cannot prevent that at the end of the second round all the good processors assume an agreement on b . So in case all good processors have the same initial values, an agreement on this value is reached. Thus it suffices to show that with probability 1, an agreement on some value is reached.

Fix any round s and suppose no processor has assumed an agreement during any of the rounds 1 through $s-1$. Then with probability at least $1/2$ some processor assumes an agreement in round $s+1$. Let c be the number of good processors that vote for 1 at the beginning of the round, i.e., each processor receives a number of votes for 1 of at least c and of less than $c + \delta n$ votes. This range contains at most one of the possible thresholds t_1 and t_2 because the difference between t_1 and t_2 is δn . Thus for at least one of the thresholds t_1 and t_2 , all numbers in this range are either below or above the threshold, hence such a threshold is chosen with probability at least $1/2$. In this case, as no processor has assumed an agreement before, all the good processors will vote the same at the beginning of round $s+1$, and as there are more than $(7/8)n$ of them, at the end of round $s+1$ all good processors will assume an agreement on the same value.

Next fix any round s and suppose that s is the least round such that at the end of the round some good processor assumes that an agreement has been reached. Then all good processors assume an agreement on the same value in round $s+1$. For a proof, consider round s . Suppose that at the end of the round, Processor j assumes an agreement on the binary value b . Then in the local view of Processor j , more than a fraction of $7/8$ of all processors have voted for b at the beginning of the round. Hence more than $(6/8)n$ good processors must have been voted for b , and as all of them obey the protocol and send always the same message to all others, at the beginning of the round, all processors have received more than $(6/8)n$ votes for b . So for all processors both of the possible thresholds are exceeded and, as by assumption no processor has assumed an agreement on a different value before, in the next round all good processors vote for b , hence from then on all good processors assume an agreement on b .

By the discussion in the preceding paragraphs, an admissible agreement is reached two rounds after some good processor assumes an agreement for the first time and in case the latter has not happened up to round s , this happens with probability at least $1/2$ during round $s+1$. From this it is easy to see that an admissible agreement is reached with probability 1. Furthermore, the expected number of rounds required to reach an agreement is constant because it can be bounded from above by the converging infinite sum $\frac{2}{2} + \frac{3}{4} + \frac{4}{8} + \frac{5}{16} \dots$. \square

For the version of the Byzantine agreement problem stated above, there are deterministic protocols that reach an admissible agreement within $n+1$ rounds and it can be shown that any deterministic protocol requires that many rounds in worst case [24]. So by Proposition 20, we have an example of a problem where randomization is provably better than determinism.

2 Derandomization

2.1 An Example: Finding Cuts

Many randomized algorithms can be derandomized in the sense that they can be transformed into a deterministic algorithm that solves the same task in a way similar to the original algorithm. In general, the deterministic algorithm may be less elegant or may perform worse, e.g., may consume more resources or may yield a solution that is not quite as good as the one obtained from the randomized algorithm.

In this section we present derandomization by the method of conditional expectation and derandomization by k -wise independent random variables. These standard techniques are explained by applying them to the following randomized algorithm Cut, which takes a graph with m edges and returns a cut with at least $m/2$ edges crossing the cut.

Definition 21 Let $G = (V, E)$ be a graph. A cut of G is a partition of V into two disjoint subsets V_0 and V_1 . The weight of a cut is the number of edges between V_0 and V_1 (i.e., the cardinality of the set $\{\{u, v\} \in E : u \in V_0, v \in V_1\}$).

Algorithm Cut

Input: A graph $G = (V, E)$ where $V = \{1, \dots, n\}$.

Choose random bits r_1, \dots, r_n by independent tosses of a fair coin.

Let $V_0 = \{i : r_i = 0\}$.

Let $V_1 = \{i : r_i = 1\}$.

Output: The cut (V_0, V_1) .

Proposition 22 Let G be a graph with m edges. Then on input G , the expected weight of the cut returned by Algorithm Cut is $m/2$. In particular, the graph G has a cut with weight at least $m/2$.

Proof. Fix any graph $G = (V, E)$ and assume that Algorithm Cut is run on this graph. Assume that E has m edges e_1, \dots, e_m . For any edge $e_i = \{u_i, v_i\}$, define a random variable \widehat{e}_i that has value 1 in case edge e_i crosses the cut returned by the algorithm and has value 0, otherwise. Then the expected value of this random variable is

$$\begin{aligned} \mathbf{E}[\widehat{e}_i] &= 1 \cdot \text{Prob}[\widehat{e}_i = 1] + 0 \cdot \text{Prob}[\widehat{e}_i = 0] \\ &= \text{Prob}[u_i \in V_0 \text{ and } v_i \in V_1] + \text{Prob}[u_i \in V_1 \text{ and } v_i \in V_0] \quad (3) \\ &= \text{Prob}[u_i \in V_0] \text{Prob}[v_i \in V_1] + \text{Prob}[u_i \in V_1] \text{Prob}[v_i \in V_0] \\ &= \frac{1}{2} \cdot \frac{1}{2} + \frac{1}{2} \cdot \frac{1}{2} = \frac{1}{2} \end{aligned}$$

In (3) we exploit that the assignments of nodes to the two sides of the cut are mutually and hence pairwise independent, thus for any pair u and v of distinct nodes and all j_u, j_v in $\{0, 1\}$,

$$\text{Prob}[u \in V_{j_u} \text{ and } v \in V_{j_v}] = \text{Prob}[u \in V_{j_u}] \cdot \text{Prob}[v \in V_{j_v}] \quad (4)$$

Let \widehat{w}_G be the weight of the cut obtained by applying Algorithm Cut to the graph G , i.e., \widehat{w}_G is just the sum of the \widehat{e}_i . By linearity of expectation and (3) we obtain

$$\mathbf{E}[\widehat{w}_G] = \mathbf{E}[\widehat{e}_1 + \dots + \widehat{e}_m] = \mathbf{E}[\widehat{e}_1] + \dots + \mathbf{E}[\widehat{e}_m] = \frac{m}{2} \quad (5)$$

Now, if all possible outcomes of the coin tosses of Algorithm Cut resulted in a cut with weight strictly less than $m/2$, then (4) would be wrong. Hence we can pick a sequence of coin tosses that yields a cut with weight at least $m/2$ and thus in particular there is such a cut. \square

2.2 Derandomization by Conditional Expectation

For any word α of length $s \leq n$, let Cut_α be the algorithm that works essentially like Algorithm Cut, except that $r_1 \dots r_s$ is set equal to α (while the bits r_{s+1} through r_n are again chosen by independent coin tosses). For example, Algorithm Cut_{01} will always assign the first node to V_0 , the second node to V_1 , and the remaining nodes according to independent coin tosses. Furthermore, let $\widehat{w}_G(\alpha)$ be the weight of the random cut returned by Algorithm Cut_α on input G .

Let λ denote the empty word, i.e., the unique word of length 0. Then the Algorithms Cut_λ and Cut coincide, hence so do the expected values of their results $\widehat{w}_G(\lambda)$ and \widehat{w}_G . Thus we obtain from (5)

$$\mathbf{E}[\widehat{w}_G(\lambda)] = \mathbf{E}[\widehat{w}_G] = \frac{m}{2}. \quad (6)$$

Now consider Algorithm Cut_α where α has length s , when applied to a graph G with at least $s + 1$ nodes. Then with probability $1/2$, the random bit r_{s+1} is set to 0 and in this situation, the algorithm behaves like $\text{Cut}_{\alpha 0}$. Likewise, with probability $1/2$ the random bit r_{s+1} is set to 1 and the algorithm behaves like $\text{Cut}_{\alpha 1}$. In terms of expected weight of the resulting cuts, this means that

$$\mathbf{E}[\widehat{w}_G(\alpha)] = \frac{1}{2}\mathbf{E}[\widehat{w}_G(\alpha 0)] + \frac{1}{2}\mathbf{E}[\widehat{w}_G(\alpha 1)], \quad (7)$$

hence it can not be that both expected values on the right-hand side are strictly smaller than the expected value on the left-hand side, i.e., we have

$$\mathbf{E}[\widehat{w}_G(\alpha)] \leq \mathbf{E}[\widehat{w}_G(\alpha 0)] \quad \text{or} \quad \mathbf{E}[\widehat{w}_G(\alpha)] \leq \mathbf{E}[\widehat{w}_G(\alpha 1)] \quad (8)$$

This leads to the following derandomized version of Algorithm Cut. Successively determine values r_1, r_2, \dots such that $\mathbf{E}[\widehat{w}_G(r_1 \dots r_s)]$ is non-decreasing

in s and, accordingly, $\mathbf{E}[\widehat{w}_G(r_1 \dots r_s)]$ is always at least as large as the initial value $\mathbf{E}[\widehat{w}_G(\lambda)] = m/2$. (For the sake of simplicity, in the condition of the if-clause of the algorithm, we write $r_1 r_0$ and $r_1 r_1$ for the empty word and for r_1 .)

Derandomized Algorithm Cut (Method of Conditional Expectation)

Input: A graph $G = (V, E)$ where $V = \{1, \dots, n\}$.

For $s = 1, \dots, n$

If $\mathbf{E}[\widehat{w}_G(r_1 \dots r_{s-1} 0)] \geq \mathbf{E}[\widehat{w}_G(r_1 \dots r_{s-1} 1)]$ (*)
then $r_s = 0$ else $r_s = 1$

Let $V_0 = \{i : r_i = 0\}$.

Let $V_1 = \{i : r_i = 1\}$.

Output: The cut (V_0, V_1) .

If applied to a graph G with m edges, the derandomized version of Algorithm Cut yields a cut of weight w_G of at least $m/2$. For a proof, it suffices to observe that

$$\begin{aligned} \frac{m}{2} = \mathbf{E}[\widehat{w}_G(\lambda)] &\leq \mathbf{E}[\widehat{w}_G(r_1)] \leq \mathbf{E}[\widehat{w}_G(r_1 r_2)] \leq \dots \\ &\dots \leq \mathbf{E}[\widehat{w}_G(r_1, \dots, r_{n-1})] \leq \mathbf{E}[\widehat{w}_G(r_1, \dots, r_n)] = w_G \end{aligned} \quad (9)$$

where the first equation is just (5), the second equation is immediate from the definition of both sides, and the inequalities hold because of (8) and the choice of the r_i .

Concerning the efficiency of the derandomized version of Algorithm Cut, it remains to show that Condition (*) can be evaluated efficiently. Consider iteration s of the for-loop of the algorithm, assuming that $r_1 \dots r_{s-1}$ have already been defined. Partition E into four sets E_1 , E_2 , E_3^0 , and E_3^1 defined by

$$\begin{aligned} E_1 &= \{ \{j_1, j_2\} \in E : j_1 < s \text{ and } j_2 < s \} \\ E_2 &= \{ \{j_1, j_2\} \in E : j_1 > s \text{ or } j_2 > s \} \\ E_3^i &= \{ \{j, s\} \in E : j < s \text{ and } r_j = i \} . \end{aligned}$$

In order to compare the values of $\mathbf{E}[\widehat{w}_G(r_1 \dots r_{s-1} r_s)]$ for $r_s = 0$ and $r_s = 1$, we ask for each of the four sets which of its edges will be in the constructed cut. The choice of r_s makes no difference with respect to the edges in E_1 and E_2 . For the edges in E_1 , it is determined by $r_1 \dots r_{s-1}$ which edges will cross the constructed cut and accordingly, let

$$E'_1 = \{ \{j_1, j_2\} \in E_1 : r_{j_1} \neq r_{j_2} \} ,$$

i.e., E'_1 contains all edges in E_1 that connect two nodes that have already been assigned to different sides of the cut. Any edge in E_2 will cross the constructed cut with probability $1/2$ because after having assigned the lesser endpoint of the edge to one side of the cut, the larger endpoint will be assigned to the other side with probability $1/2$. On the other hand, the choice of r_s makes a difference with

respect to the edges in the sets E_3^0 and E_3^1 . For $r_s = i$, all edges in E_3^{1-i} but no edge in E_3^i will cross the constructed cut. In summary we have for $i = 0, 1$,

$$\mathbf{E}[\widehat{w}_G(r_1 \dots r_{s-1} i)] = |E'_1| + \frac{1}{2}|E_2| + |E_3^{1-i}|.$$

So the comparison of $\mathbf{E}[\widehat{w}_G(r_1 \dots r_{s-1} r_s)]$ for $r_s = 0$ and $r_s = 1$ amounts to compare the sizes of E_3^0 and E_3^1 , i.e., we simply have to count for the nodes 1 through $s - 1$ that are neighbors of node s , how many of the corresponding bits r_j are equal to 0 and to 1.

The derandomized version of Algorithm Cut then can be rephrased as follows. The new and the old version of the algorithm are identical except for the formulation of the condition in the if-clause.

Derandomized Algorithm Cut (Method of Conditional Expectation)

Input: A graph $G = (V, E)$ where $V = \{1, \dots, n\}$.

For $s = 1, \dots, n$

If $|\{j < s : \{j, s\} \in E \text{ and } r_j = 1\}| \geq |\{j < s : \{j, s\} \in E \text{ and } r_j = 0\}|$
then $r_s = 0$ else $r_s = 1$.

Let $V_0 = \{i : r_i = 0\}$.

Let $V_1 = \{i : r_i = 1\}$.

Output: The cut (V_0, V_1) .

So we end up with an efficient and extremely simple deterministic algorithm, while the randomized version of the algorithm is still used for verifying that the deterministic version works as required.

2.3 Derandomization by k -Wise Independent Random Variables

For any randomized algorithm we obtain a *trivial derandomization* by just simulating the algorithm for every possible value of its random source. In general, however, the deterministic algorithm obtained this way will run slower than the randomized algorithm by an exponential factor. More precisely, a randomized algorithm that runs for t steps might use up to t random bits, hence the trivial derandomization must simulate the algorithm for 2^t possible random sources. For example, the Algorithm Cut from Section 2.2 uses n random bits on a graph with n nodes, hence its trivial derandomization runs for at least 2^n steps. This time complexity is usually considered to be infeasible and is indeed roughly the same as the complexity of an exhaustive search through all possible cuts.

The trivial derandomization can be useful when applied to randomized algorithms where the number of random bits used is small compared to the running time. For example in case the number of random bits is logarithmic in the running time, then the number of distinct values of the random source is approximately the same as the running time, hence the trivial derandomization will approximately square the running time.

Next we consider a variant of Algorithm Cut where the number of random bits is logarithmic in the number of nodes of the input graph. The new algorithm works like the old one except that the random bits $r_1 \dots r_n$ are not obtained by independent tosses of a fair coin but by a randomized mapping

$$s_\tau : i \mapsto r_i$$

that depends on a random word τ . The idea underlying the construction of s_τ is the following. In the verification of Algorithm Cut we have used that the r_i are pairwise independent but not that they are mutually independent. Thus the verification goes through as long as we ensure that the r_i obtained from s_τ are pairwise independent. The point in generating the r_i via the mapping s_τ is that the set of all possible assignments to τ is considerably smaller than the set of all possible assignments to the r_i , hence the trivial derandomization is feasible with respect to τ . Informally, we call this type of derandomization the *method of small sample spaces* or the *method of pairwise (k-wise) independent variables*.

Definition 23 *Let D and R be two finite sets. Let τ be a random variable that assumes values in $\{a_1, \dots, a_l\}$. Suppose that for each a_i , we have fixed a mapping s_{a_i} and let s_τ be the random mapping from D to R where $s_\tau(x) = s_{a_i}(x)$ in case a_i is the actual outcome of τ .*

Then s_τ is said to have pairwise independent values if for all $x_1 \neq x_2$ in D and for all y_1 and y_2 in R , we have

$$\text{Prob}[s_\tau(x_1) = y_1 \text{ and } s_\tau(x_2) = y_2] = \text{Prob}[s_\tau(x_1) = y_1] \cdot \text{Prob}[s_\tau(x_2) = y_2] \quad (10)$$

(where the probabilities refer to the ones induced by the distribution of τ .)

Mappings that have pairwise independent values and use a number of random bits that is logarithmic in the size of their domain can be constructed by algebraic methods. Recall that for natural numbers y and $p > 0$, the term $y \bmod p$ denotes the remainder of x when divided by y . For example $(2p \bmod p)$ is 0 and $(3p - 1 \bmod p)$ is $p - 1$ for all $p > 0$.

Proposition 24 *Let p be any prime number and consider the random experiment where $\tau = (\hat{a}, \hat{b})$ is determined by picking two numbers \hat{a} and \hat{b} uniformly and independently from $\{0, \dots, p - 1\}$. Then the mapping*

$$\begin{aligned} s_\tau : \{0, \dots, p - 1\} &\rightarrow \{0, \dots, p - 1\} \\ x &\mapsto (\hat{a}x + \hat{b}) \bmod p \end{aligned}$$

has pairwise independent values and for any fixed input x , the value $s_\tau(x)$ is uniformly distributed in $\{0, \dots, p - 1\}$.

Proof. Fix any numbers x_1, x_2, y_1, y_2 in $\{0, \dots, p - 1\}$ where $x_1 \neq x_2$. Then no matter what value is assumed by \hat{a} , there is exactly one choice of \hat{b} that

results in $s_\tau(x_1) = y_1$. But \widehat{b} is chosen uniformly and independently from a set of size p , hence the probability that s_τ maps x_1 to y_1 is just $1/p$ (and a similar statement holds for x_2 and y_2). On the other hand, for a prime p it follows from basic algebra that there is exactly one choice of \widehat{a} and \widehat{b} that yields $s_\tau(x_1) = y_1$ and $s_\tau(x_2) = y_2$, hence the probability that both equations become simultaneously true is $1/p^2$. In summary, we obtain that the values of s_τ are pairwise independent and are uniformly distributed in $\{0, \dots, p-1\}$. \square

Next we use the generation of pairwise independent random variables according to Proposition 24 for reducing the size of the sample space of Algorithm Cut. For simplicity, we assume that the algorithm receives as additional input the least prime $p \geq n$. Then we argue that with the derandomized version of the algorithm we do not require to know p and that in fact there are ways to generate the r_i that are slightly more involved but avoid using p at all.

Algorithm Cut_{pi} (Using pairwise independent random bits)

Input: A graph $G = (V, E)$ where $V = \{1, \dots, n\}$ and the least prime $p \geq n$.

Choose \widehat{a} and \widehat{b} uniformly and independently in $\{0, \dots, p-1\}$.

For $i = 1, \dots, n$, let $r_i = (\widehat{a}i + \widehat{b}) \bmod p$.

Let $V_0 = \{i : r_i \text{ is even}\}$.

Let $V_1 = \{i : r_i \text{ is odd}\}$.

Output: The cut (V_0, V_1) .

Proposition 25 *Let G be a graph with m edges. Then on input G , the expected weight of the cut returned by Algorithm Cut_{pi} is at least $(1 - \frac{1}{m^2}) \frac{m}{2}$. In particular, the algorithm returns a cut of weight at least $m/2$ for some assignment of its random source.*

Proof. By Proposition 24, the values $(\widehat{a}i + \widehat{b}) \bmod p$ are pairwise independent and hence so are the events that these values are even or odd. Furthermore, each node is assigned to V_0 with probability $(p-1)/2p$ and to V_1 with probability $(p+1)/2p$. Hence we infer like in the proof of Proposition 22 that the expected value of the variable that indicates whether edge $e_i = \{u_i, v_i\}$ crosses the constructed cut is

$$\begin{aligned} \mathbf{E}[\widehat{e}_i] &= \text{Prob}[u_i \in V_0 \text{ and } v_i \in V_1] + \text{Prob}[u_i \in V_1 \text{ and } v_i \in V_0] \\ &= \text{Prob}[u_i \in V_0] \text{Prob}[v_i \in V_1] + \text{Prob}[u_i \in V_1] \text{Prob}[v_i \in V_0] \\ &= \frac{p-1}{2p} \cdot \frac{p+1}{2p} + \frac{p+1}{2p} \cdot \frac{p-1}{2p} = \frac{(p-1)(p+1)}{2p^2} \\ &= \frac{p^2-1}{p^2} \cdot \frac{1}{2} = \left(1 - \frac{1}{p^2}\right) \cdot \frac{1}{2} \end{aligned} \tag{11}$$

By linearity of expectation, the expected weight of the constructed cut is

$$\mathbf{E}[\widehat{w}_G] = \mathbf{E}[\widehat{e}_1] + \dots + \mathbf{E}[\widehat{e}_m] = \left(1 - \frac{1}{p^2}\right) \frac{m}{2} \geq \left(1 - \frac{1}{m^2}\right) \frac{m}{2}, \tag{12}$$

where the inequality holds by $p \geq m$. In particular, the expected weight $\mathbf{E}[\widehat{w}_G]$ differs from $m/2$ by at most $1/(2m)$. By inspection of the cases m even and m odd, we infer that for all $m > 1$, the least integer larger than or equal to $\mathbf{E}[\widehat{w}_G]$ is at least $m/2$. As usual, we can argue that some assignment to the random source results in a cut of weight at least equal to the expected value and, as such a cut must have integer weight, the weight is at least $m/2$. \square

The following derandomized version of Algorithm Cut simulates Algorithm Cut_{p_i} for all possible choices of $(\widehat{a}, \widehat{b})$ and among all cuts computed by the simulations outputs one with maximum weight. By Proposition 25, for the Algorithm Cut_{p_i} there is at least one choice of τ that results in a cut of weight at least $m/2$, hence the derandomized version computes a cut of weight at least $m/2$.

Derandomized Algorithm Cut (Method of k -wise Independent Random Variables)

Input: A graph $G = (V, E)$ where $V = \{1, \dots, n\}$ and the least prime $p \geq n$.

For all pairs (a, b) in $\{0, \dots, p-1\}$

For $i = 1, \dots, n$, let $r_i = (ai + b) \bmod p$.

Let $V_0^{(a,b)} = \{i : r_i \text{ is even}\}$.

Let $V_1^{(a,b)} = \{i : r_i \text{ is odd}\}$.

Output: A cut $(V_0^{(a,b)}, V_1^{(a,b)})$ of maximum weight.

By Bertrand's postulate [1, Chapter 2], for all $n \geq 1$, the least prime $p > n$ is not larger than $2n$. Thus the derandomized version of Algorithm Cut has to check at most $4n^2$ pairs of numbers (a, b) . In addition, the assumption that the algorithm knows p can be discarded by running the algorithm for all values of p between $n+1$ and $2n$, then outputting among all cuts found the one with maximum weight. For non-prime values of p , our analysis of the performance of the algorithm might be incorrect, but by Bertrand's postulate we know that at least one of these values is prime and for each such value the algorithms performs as expected.

Finally, we can avoid the usage of p by using a little bit more of algebra. Instead of evaluating $\widehat{a}i + \widehat{b} \bmod p$ over \mathbb{N} , which essentially amounts to evaluating $\widehat{a}i + \widehat{b}$ over the finite field \mathbb{F}_p with p elements, we evaluate the latter function over \mathbb{F}_{2^k} , the field with 2^k elements, where k is the least number such that $n \leq 2^k$. Furthermore, we obtain the r_i by considering the images of the first n elements of this field. The verification of the algorithm works as before and becomes even slightly simpler because the term $(1 - \frac{1}{m^2})$ in Proposition 25 disappears. The technical details required for this construction can be found in the section on universal hash functions in the survey by Miltersen [23].

3 Randomized Sorting

3.1 Sorting Lists

The sorting problem asks for sorting a list of n elements according to their size. In the general version of the sorting problem we can compare any pair of two elements to each other but we have no additional information about the sizes of the elements. Any algorithm that solves this problem has a worst-case complexity of at least $r = c(n \log n)$ comparisons for some constant c , i.e., necessarily that many comparisons have to be used on some inputs. In a nutshell, the reason is that by r comparisons we can distinguish at most 2^r orderings, and it can be shown that the latter number is approximately equal to the total number $n! = n \cdot (n-1) \cdot \dots \cdot 2 \cdot 1$ of distinct orderings of n elements. This worst-case lower bound is tight up to the choice of the constant c , as there are algorithms that achieve a worst-case complexity of $O(n \log n)$ comparisons [13].

The following algorithm Quicksort solves the sorting problem. The algorithm is a randomized version of the usual deterministic quicksort algorithm, for details and further references see Motwani and Raghavan [19, Chapter 3] and Cormen et. al.[13]. Algorithm Quicksort receives as input a list of n elements, where for the sake of simplicity we assume that the elements in the list are of mutually distinct size. Then an element s from the list to be sorted is picked uniformly at random. By comparison with s , the remaining elements are split into two lists that consist of all elements that are smaller than s and are larger than s , respectively. Then Algorithm Quicksort is applied recursively to each of these lists and the results of the recursive calls are combined in order to get an ordered version of the initial list.

Algorithm Quicksort (Randomized Quicksort.)

Input: A list $S = (s_1, \dots, s_n)$ of n elements of mutually distinct size.

Pick an element s of S uniformly at random.

$S_{\text{small}} = (s_i)_{s_i < s}$ (i.e., the sequence S_{small} contains the s_i with $s_i < s$).

$S_{\text{large}} = (s_i)_{s_i > s}$

If $|S_{\text{small}}| > 1$, then $S_{\text{small}} = \text{Quicksort}(S_{\text{small}})$.

If $|S_{\text{large}}| > 1$, then $S_{\text{large}} = \text{Quicksort}(S_{\text{large}})$.

Output: $(S_{\text{small}} \circ s \circ S_{\text{large}})$ (where \circ denotes concatenation).

Proposition 26 states that the randomized algorithm Quicksort uses $O(n \log n)$ comparisons on average. This bound does not depend on the actual input and indeed there are no particular bad inputs for the randomized algorithm. However, we might have bad luck with the random choices, e.g., it might happen that we always choose the least element of the list to be sorted and in this case S_{small} is always empty, the depth of recursion is about n , and the total number of comparisons is about $c \cdot n^2$ for some constant c . The picture is different for the deterministic version of Algorithm Quicksort, where the element used to split the current list is not picked at random but deterministically. It can be shown

that in case the input list is chosen uniformly at random from the set of all permutations of n elements, then the expected number of comparisons required by the deterministic algorithm is $O(n \log n)$. However, for the deterministic version there are certain inputs on which the algorithm always behaves much worse. For example, consider a list that is already in increasing order and assume that we always split by the first element in the list, hence for the given list we always split by the least element in the list to be sorted. Then we obtain as above that the total number of comparisons is about $c \cdot n^2$ for some constant c .

Proposition 26 *In case Algorithm Quicksort is run on a list of n elements of mutually distinct size, then the expected number of comparisons required to sort the list is $O(n \log n)$.*

When demonstrating Proposition 26, we follow Motwani and Raghavan [19, Chapter 3]. Their elegant proof shows that the probabilistic method can be applied when analyzing the performance of probabilistic (or other) algorithms.

Proof. Let $S = (s_1, \dots, s_n)$ be any list of n elements of mutually distinct size where the size of s_i is less than the size of s_j whenever $i < j$. Assume that Algorithm Quicksort is applied to any permutation of this list. We show the assertion on the expected number of comparisons and omit the easy proof that the algorithm indeed terminates and outputs the sorted list S .

By an easy induction argument, one infers that the arguments of the recursive calls to Quicksort are always lists that contain only elements that have not yet been compared to each other. Furthermore, during the execution of a any particular call to Algorithm Quicksort, any pair of elements is compared to each other at most once. As a consequence, any pair of elements from S is compared to each other at most once. Let X_{ij} be the indicator variable for the event that s_i is ever compared to s_j , i.e., $X_{ij} = 1$ if and only if this event occurs and $X_{ij} = 0$, otherwise. Then by the preceding discussion, the actual number of comparisons is just the sum over the X_{ij} with $i < j$, and the expected number of comparisons can be written as

$$\mathbf{E} \left[\sum_{1 \leq i < j \leq n} X_{ij} \right] = \sum_{1 \leq i < j \leq n} \mathbf{E}[X_{ij}] = \sum_{1 \leq i < j \leq n} p_{ij}, \quad (13)$$

where we write p_{ij} for the probability that X_{ij} is equal to 1. The first equation in (13) holds by linearity of expectation and the second one follows because by definition we have

$$\mathbf{E}[X_{ij}] = p_{ij} \cdot 1 + (1 - p_{ij}) \cdot 0 = p_{ij}. \quad (14)$$

In order to bound the probabilities p_{ij} , fix any pair i and j where $i < j$. In the sequence of recursive calls to Algorithm Quicksort, the $j - i + 1$ elements s_i, \dots, s_j always stick together until for the first time one of these elements is picked for

splitting the current list. Before, each time such a splitting element is picked, the elements s_i through s_j have equal chances to be picked, hence also the overall probability for being picked first among these $j-i+1$ elements is just $1/(j-i+1)$ for each of them. In case the first element picked differs from s_i and s_j , then these two elements are assigned to different sublist and are never compared. On the other hand, the two elements are obviously compared in case one of them is picked first. So the probability that s_i and s_j are compared at all is just $2/(j-i+1)$. Substituting in (13), the expected number of comparisons can be bounded by

$$\sum_{1 \leq i < j \leq n} p_{ij} = \sum_{1 \leq i < j \leq n} \frac{2}{j-i+1} = 2 \sum_{i=1}^{n-1} \sum_{j=i+1}^n \frac{1}{i-j+1} \leq 2n \sum_{k=1}^n \frac{1}{k} = 2nH_n .$$

where $H_n = 1/1 + 1/2 + \dots + 1/n$ is the n th Harmonic number. But for all n ,

$$H_n \leq 1 + \ln n, \tag{15}$$

where $\ln n$, the logarithm of n to base $e = 2,71\dots$, is in $O(\log n)$. As a consequence, the expected number of comparisons is in $O(n \log n)$.

For a proof of (15), let h be the step-like function h that for all $k \geq 2$, maps every real x with $k-1 \leq x < k$ to $1/k$. Consider the integral over h between 1 and n . By construction, this integral is just $H_n - 1$. Furthermore, $h(x)$ is always at most $1/x$, hence the integral is at most equal to the integral over the function $x \mapsto 1/x$ between 1 and n , which is equal to $\ln n - \ln 1 = \ln n$. Inequality (15) follows. \square

3.2 Sorting Nuts and Bolts

In Section 3.1, we have considered the randomized algorithm Quicksort, which solves the general sorting problem and uses an expected number of $O(n \log n)$ comparisons. In this section, we consider the related but apparently more involved problem of sorting nuts and bolts. The input to this problem is a pile of n bolts of mutually distinct sizes together with a pile of n matching nuts and the goal is to sort both piles. However, while sorting we are only allowed to compare nuts to bolts (but not nuts to nuts or bolts to bolts). The result of such a comparison is either nut too small, nut fits, or nut too large.

There is a simple randomized algorithm for sorting nuts and bolts, which is pretty similar to Algorithm Quicksort from Section 3.1. Initially, we pick randomly and uniformly a nut n . We compare n to all bolts. This way, we find the bolt b that matches n and at the same time, split the remaining bolts into two lists that contain the bolts smaller and the bolts larger than b . Similarly, we use b in order to split the remaining nuts into two lists that contain the nuts that are smaller and are larger than n . We end up with a list of small nuts and a matching list of small bolts, as well as with a list of large nuts and a matching list of large bolts. Then we apply the same procedure recursively to the small nuts and bolts and to

the large nuts and bolt, and the outputs of these recursive calls are concatenated in order to get the sorted list of nuts and the sorted list of bolts.

Algorithm NutsandBolts (Randomized sorting of nuts and bolts.)

Input: Two lists $N = (n_1, \dots, n_n)$ and $B = (b_1, \dots, b_n)$ of nuts and of bolts such that each nut matches exactly one bolt.

Pick an element n of N uniformly at random.

Let b be the bolt that matches b_i .

$B_{\text{small}} = (b_i)_{b_i < n}$ (i.e., the list B_{small} contains the b_i with $b_i < n$).

$B_{\text{large}} = (b_i)_{b_i > n}$

$N_{\text{small}} = (n_i)_{n_i < b}$

$N_{\text{large}} = (n_i)_{n_i > b}$

If $|N_{\text{small}}| > 1$, then $(N_{\text{small}}, B_{\text{small}}) = \text{NutsandBolts}(N_{\text{small}}, B_{\text{small}})$.

If $|N_{\text{large}}| > 1$, then $(N_{\text{large}}, B_{\text{large}}) = \text{NutsandBolts}(N_{\text{large}}, B_{\text{large}})$.

Output: $(N_{\text{small}} \circ n \circ N_{\text{large}}, B_{\text{small}} \circ b \circ B_{\text{large}})$ (\circ denotes concatenation).

Proposition 27 *Consider two arbitrary lists of n bolts and of n nuts where each nut matches exactly one bolt. Then Algorithm NutsandBolts sorts these list while using an expected number of comparisons in $O(n \log n)$.*

We omit the proof of Proposition 27, which is pretty similar to the proof of the corresponding Proposition 26 for Algorithm Quicksort.

An argument similar to the one used in connection with the general sorting problem shows that any deterministic algorithm that solves the sorting nuts and bolts problem uses at least $c(n \log n)$ comparisons in worst case for some constant $c > 0$. Bradford and Fleischer [7] give an deterministic algorithm that uses $O(n \log^2 n)$ comparisons in worst case, thus improving on a previous algorithm due to Alon et. al [2] that uses $O(n \log^4 n)$ comparisons. Kómlós, Ma, and Szemerédi [21] describe an algorithm that uses just $O(n \log n)$ comparisons, however, they remark that their algorithm relies on the construction of certain graphs that are known to exist but cannot be constructed efficiently by current techniques [21].

In the remainder of this section, we present the algorithm by Bradford and Fleischer. Before, we review some notation and facts related to expander graphs.

Definition 28 *A graph (V, E) is bipartite if V is the disjoint union of two sets V_0 and V_1 such that E does not contain any edges between nodes in V_0 or between nodes in V_1 . In this situation, we write (V_0, V_1, E) for the bipartite graph (V, E) .*

With a graph (V, E) understood, the neighborhood $\text{Nb}(x)$ of a node x in V is the set of all nodes that are connected to x by an edge in E . The neighborhood $\text{Nb}(U)$ of a set $U \subseteq V$ is the union of the neighborhoods $\text{Nb}(x)$ of all x in U .

A node of a graph has degree d if the node has exactly d neighbors. A graph is d -regular if every node of the graph has degree d .

As graphs do not contain loops, i.e., edges between a node and itself, the neighborhood $\text{Nb}(x)$ of a node x does never contain x . Furthermore, as there is at most one edge between any pair of nodes, the neighborhood of a node x is in one-to-one correspondence with the set of edges to which x is incident. In particular, the degree of a node is just the number of edges to which this node is incident. Observe that for a bipartite graph (V_0, V_1, E) the neighborhood of any subset of V_i is contained in V_{1-i} .

Definition 29 For any rational $\delta \geq 0$, a bipartite graph (V_0, V_1, E) is a δ -expander if $|V_0| = |V_1|$ and for any subset U of V_0 of cardinality at most $|V_0|/2$, we have

$$|\text{Nb}(U)| \geq (1 + \delta)|U| .$$

In the literature there are various, slightly different definitions of the concept of expander, which all feature the idea that any subset of nodes that is not too large has a neighborhood that is larger than the set itself. In particular, expanders in the sense of Definition 29 that have degree at most d are called $(n, d, 1/2, 1 + \delta)$ OR-concentrators in Motwani and Raghavan [24, Section 5.3].

In general, it is desirable to obtain δ -expanders of degree at most d where d is small and δ is large. Good expanders can be obtained by sophisticated methods from algebraic graph theory, we refer to Motwani and Raghavan [24] and Miltersen [23] for a survey and to Alon and Spencer [3] for a more technical account.

The Gabber-Galil expanders introduced in Definition 30 have a particularly simple structure, however, there are more recent constructions that yield expanders with significantly better expansion factors.

Definition 30 For any m , let the Gabber-Galil expander of order m be the bipartite graph (V_0, V_1, E) where

$$V_0 = V_1 = \{(i, j) : 1 \leq i, j \leq m\} ,$$

(i.e., V_0 and V_1 have m^2 nodes and can hence be identified with $\{1, \dots, m^2\}$) and where for any node (i, j) in V_0 , the set E contains an edge between (i, j) and each of the following nodes of V_1

$$(i, j), (i, i + j), (i, i + j + 1), (i + j, j), (i + j + 1, j)$$

where all additions are meant to be modulo m .

While the construction of the Gabber-Galil-expanders is rather simple, the verification of the expansion properties as stated in Proposition 31 is more involved and we omit the corresponding proof.

Proposition 31 *There is a rational $\delta_{\text{GG}} > 0$ such that for any m , the Gabber-Galil-expander of order m is a 5-regular $2\delta_{\text{GG}}$ -expander.*

For further use in the verification of the algorithm by Bradford and Fleischer, we construct in Remark 32 a graph that satisfies a somewhat technical expansion property. This graph is obtained by iterating copies of the Gabber-Galil expander (or any other expander with similar properties).

Remark 32 *For any rational $\alpha > 0$ there is a constant q such that on input m we can construct in time polynomial in m a bipartite graph $G_m = (V_0, V_1, E)$ with $|V_0| = |V_1| = m^2$ such that*

- (i) *each node in V_0 has degree at most q ,*
- (ii) *for any subset U of V_0 of cardinality at least $\alpha|V_0|$, the neighborhood $\text{Nb}(U)$ has cardinality*

$$|\text{Nb}(U)| \geq \left(\frac{1}{2} + \delta_{\text{GG}}\right) |V_1|, \quad (16)$$

where $\delta_{\text{GG}} > 0$ is the constant from Proposition 31.

In order to construct G_m for given $\alpha > 0$, let l be the least integer such that

$$\alpha(1 + 2\delta_{\text{GG}})^{l-1} > 1/2.$$

Note that l depends on α but not on m .

Let the graph $H = (W, E_H)$ be defined as follows. The node set of H is the disjoint union of sets W_0, W_1, \dots, W_l of size m^2 . The edges of H are only between nodes in W_i and nodes in W_{i+1} and the subgraph induced by any set of the form $W_i \cup W_{i+1}$ is a copy of the Gabber-Galil expander of order m (i.e., the graph H consists of l copies of the Gabber-Galil expander of order m such that the right-hand nodes of copy i coincide with the left-hand nodes of copy $i + 1$).

Say a path in H is legal if it contains at most one edge from any copy of the Gabber-Galil expander, i.e., intuitively speaking, a legal path connects to nodes such that when going from one node to the other, one always goes in the same direction. Let $G_m = (W_0, W_l, E)$ where E contains edges exactly between those pairs of nodes that are connected by a legal path of H .

It remains to show that the graphs G_m have the required properties. The Gabber-Galil expanders are 5-regular, hence any node in W_0 has at most $q = 5^l$ neighbors in W_l . In order to show (ii), fix any subset U of W_0 of size at least $\alpha|W_0|^2$ and let c_i be the number of nodes in W_i that can be reached from U by a legal path. If for some $i < l$, c_i is at least $|W_0|/2$, then we are done because then c_j is at least $(1/2 + \delta_{\text{GG}})|W_0|$ for all $j > i$ by the expansion properties of the Gabber-Galil expander. Assuming otherwise, the expansion properties ensure for all i that c_{i+1} is at least $(1 + 2\delta_{\text{GG}})c_i$. Contrary to our assumption, we obtain

$$c_{l-1} \geq (1 + 2\delta_{\text{GG}})^{l-1} c_0 \geq (1 + 2\delta_{\text{GG}})^{l-1} \alpha |W_0| > |W_0|/2$$

by choice of l and because $c_0 = |U| \geq \alpha|W_0|$.

The deterministic algorithm for sorting nuts and bolts by Bradford and Fleischer works exactly the same as Algorithm NutsandBolts, except that the random choice of the nut n used for splitting the input list is replaced by a deterministic procedure that for some constant $\gamma > 0$ guarantees to find a nut n that is a γ -approximate median in the sense that it splits the list into two parts that each contain fraction of at least γ of the whole list.

Definition 33 Let D be any totally ordered domain, let x be an element of D and let $S = \{y_1, \dots, y_n\}$ be any list of elements of D .

The (relative) rank $\text{rank}(x, S)$ of x in S is the fraction of elements of S that are at most as large as x , i.e.,

$$\text{rank}(x, S) = \frac{|\{i : y_i \leq x \text{ and } 1 \leq i \leq |S|\}|}{|S|}.$$

The element x is called a γ -approximate median for S if $\gamma \leq \text{rank}(x, S) \leq 1 - \gamma$.

Consider the deterministic version of Algorithm NutsandBolts where the random choice of the nut n used for splitting list is replaced by a deterministic procedure that for some rational $\gamma > 0$ guarantees to find a nut n that is a γ -approximate median. In case such a median can be found while using $O(n \log n)$ comparisons in worst case, then the deterministic sorting algorithm requires $O(n \log^2 n)$ comparisons in worst case. For a proof, fix any input (N, B) where N and B both have n elements and picture the recursive calls to the deterministic version of Algorithm NutsandBolts as a binary tree where the two subtrees of any node correspond to the arguments of the recursive calls. Then the depth of this tree is in $O(\log n)$ because each recursive calls diminishes the size of the inputs by a factor of at least $(1 - \gamma)$. Furthermore, the inputs to the recursive calls at the same depth have a total size of at most n , hence summing up the assumed complexity of $O(n \log n)$ comparisons over a single depth yields again $O(n \log n)$ comparisons. So the tree has at most $O(\log n)$ levels while each level requires at most $O(n \log n)$ comparisons, hence the overall complexity is at most $O(n \log^2 n)$ comparisons.

By the preceding discussion, the construction of a deterministic algorithm for sorting nuts and bolts that uses $O(n \log^2 n)$ comparisons in worst case can be reduced to the problem of devising an appropriate deterministic algorithm for finding an approximate median. The Algorithm Median below solves the latter problem, i.e., the algorithm uses $O(n \log n)$ comparisons in worst case in order to find a γ -approximate median for some constant $\gamma > 0$.

Initially, Algorithm Median uses Algorithm CandidateList below in order to compute a list n_1, \dots, n_l of nuts and associated two-element lists of bolts (b_i^-, b_i^+) such that $b_i^- \leq n_i \leq b_i^+$. For the application in Algorithm Median, we require that the nuts in this list comprise a non-zero constant fraction of all nuts and that each bolt appears at most constantly often in one of the two-element lists.

Algorithm CandidateList (Computing a list of candidates.)

Input: Two lists $N = (n_1, \dots, n_n)$ and $B = (b_1, \dots, b_n)$ of nuts and of bolts such that each nut matches exactly one bolt.

Let m be the maximum natural number such that $m^2 \leq n$.

Let $N_0 = (n_1, \dots, n_{m^2})$ and $B_0 = (b_1, \dots, b_{m^2})$.

Let $G = (B_0, N_0, E)$ be isomorphic to the graph G_m from Remark 32 where α is chosen as $\delta_{GG}/4$.

(I.e., G is obtained from $G_m = (V_0, V_1, E)$ by identifying V_0 with B_0 and V_1 with N_0 .)

Let M and L both be equal to the empty list.

For $i = 1$ to m^2

If there are bolts b^- and b^+ in $\text{Nb}(n_i)$ such that $b^- < n_i < b^+$ then

pick any pair of such bolts and let $b_i^- = b^-$ and $b_i^+ = b^+$,
append n_i to M and the pair (b_i^-, b_i^+) to L .

Output: The lists $M = (n_1, \dots, n_l)$ and $L = ((b_1^-, b_1^+), \dots, (b_l^-, b_l^+))$.

Lemma 34 *Let $N = (n_1, \dots, n_n)$ and $B = (b_1, \dots, b_n)$ be lists of nuts and of bolts such that each nut matches exactly one bolt.*

When applied to N and B , Algorithm CandidateList uses $O(n)$ comparisons in worst case and eventually outputs lists M and L such that $b_i^- \leq n_i \leq b_i^+$ for $i = 1, \dots, n$. Furthermore, there are rationals $\gamma_M > 0$ and $\gamma_L > 0$ such that M has length at least $\gamma_M n$ and L contains at least $\gamma_L n$ mutually distinct bolts, i.e.,

$$|M| \geq \gamma_M n \quad |\{b_i^-, b_i^+, \dots, b_l^-, b_l^+\}| \geq \gamma_L n. \quad (17)$$

Proof. Inspection of Algorithm CandidateList shows that the algorithm always terminates and that $b_i^- \leq n_i \leq b_i^+$ holds for all i . By construction, each of the m^2 bolts in B_0 has degree at most q in G , hence G has at most qm^2 edges. Exactly those nuts and bolts that are connected by an edge are compared during the algorithm, hence the total number of comparisons is at most $qm^2 \leq qn$ and is hence in $O(n)$.

The number m has been chosen maximum such that $m^2 \leq n$ and thus $m^2 \geq n/3$. For $n = 1, 2, 3$ this is obviously true while for all $n > 3$, the closed interval between $n/3$ and n contains a square because the closed interval bounded by the square roots of $n/3$ and of n has size at least 1 and hence contains some natural number. Next observe that we can assume $n \geq 24/\delta_{GG}$, hence

$$m^2 > \frac{8}{\delta_{GG}}.$$

The finitely many inputs where n is smaller can be handled by any algorithm that computes the desired output (say, by a brute-force sorting algorithm) without changing the asymptotic complexity $O(n \log n)$.

In order to show the assertion on the number of mutually distinct bolts, let $\varepsilon \geq \alpha$ be the least rational such that εm^2 is a natural number. This implies

$$\varepsilon \leq \frac{3}{8} \delta_{GG}$$

because εm^2 and αm^2 differ by at most 1, hence ε and α differ by at most $1/m^2$, while $\alpha = \delta_{GG}/4$ and $1/m^2 \leq \delta_{GG}/8$.

Let B^- and B^+ be the sets of the least and of the largest εm^2 many bolts in B_0 . By Remark 32 and the definition of G , for both sets the neighboring nuts comprise at least a fraction of $1/2 + \delta_{GG}$ of all nuts in N_0 . As a consequence, the intersection of their neighborhoods must comprise at least $\delta_{GG} m^2$ many nuts in N_0 , i.e., that many nuts are compared to a bolt in B^- and in B^+ . If we subtract all nuts that match a bolt in at least one of the two latter sets, we are left with a fraction of at least

$$\delta_M = \delta_{GG} - 2\varepsilon \geq \frac{2}{8} \delta_{GG} > 0$$

of all nuts in N_0 . By construction, all this nuts are compared to a strictly smaller and to a strictly larger bolt and are thus contained in M . Furthermore, each nut in M is compared to two bolts that appear in L while each bolt in L is compared to at most q nuts, thus L must contain at least a fraction of $\delta_L = 2\delta_M/q$ of the m^2 bolts in N_0 . By $m^2 \geq n/3$ and $|N| = |B| = n$, this implies that M contains a fraction of at least $\gamma_m = \delta_M/3$ of the bolts in B , while L contains a fraction of at least $\gamma_L = \delta_L/3$ of the bolts in B . \square

In Algorithm Median, first we obtain lists M and L as they are provided by algorithm CandidateList. Then the algorithm proceeds in rounds. At the beginning of each round the nuts in M are paired and from each pair one nut survives to the next round while the other nut is discarded. The criteria for surviving is which of the two nuts has rank closer to $1/2$ with respect to the combined list of bolts of both nuts. The combined list, which might contain multiple copies of a single bolt, is then assigned to the surviving nut. For technical reasons that will become clear in the verification of the algorithm, we pair only nuts where both have rank strictly less than $1/2$ or both have rank at least $1/2$ with respect to their current list. Furthermore, in case the total number of nuts to be paired is odd, the single remaining nut is simply discarded together with its list of bolts.

Algorithm Median (Computing a γ -approximate median.)

Input: Two sequences $N = (n_1, \dots, n_n)$ and $B = (b_1, \dots, b_n)$ of nuts and of bolts such that each nut matches exactly one bolt.

Apply Algorithm CandidateList to N and B in order to obtain lists $M = (n_1, \dots, n_l)$ and $L = ((b_1^-, b_1^+), \dots, (b_l^-, b_l^+))$.

For $i = 1$ to l , let $L_i = (b_i^-, b_i^+)$.

While $|M| > 2$

Let $M_{\text{low}} = \{\mathbf{n}_i \in M : \text{rank}(\mathbf{n}_i, L_i) < 1/2\}$.
 Let $M_{\text{high}} = \{\mathbf{n}_i \in M : \text{rank}(\mathbf{n}_i, L_i) \geq 1/2\}$.
 If $|M_{\text{low}}|$ is odd, pick a nut in M_{low} and discard it from M_{low} and M .
 If $|M_{\text{high}}|$ is odd, pick a nut in M_{high} and discard it from M_{high} and M .
 Pair the nuts in M_{low} and pair the nuts in M_{high} .
 For all pairs $(\mathbf{n}_i, \mathbf{n}_j)$ obtained this way
 if $\text{rank}(\mathbf{n}_i, L_i \circ L_j)$ is closer to $1/2$ than $\text{rank}(\mathbf{n}_j, L_i \circ L_j)$
 then let $L_i = L_i \circ L_j$ and discard \mathbf{n}_j from M ,
 else let $L_j = L_i \circ L_j$ and discard \mathbf{n}_i from M .

Let n be any nut \mathbf{n}_i remaining in M .

Output: The nut n .

Proposition 35 *There is a rational $\gamma > 0$ such that when Algorithm Median is applied to lists $N = (\mathbf{n}_1, \dots, \mathbf{n}_n)$ and $B = (\mathbf{b}_1, \dots, \mathbf{b}_n)$ of nuts and of bolts where each nut matches exactly one bolt, the algorithm outputs a γ -approximate median for N while using $O(n \log n)$ comparisons.*

Proof(Sketch). Let k be the number of iterations of the while-loop. Then we have $k \leq \log |M| + 1$ because during each iteration of the while-loop at least half of the nuts in M are discarded. On the other hand, $k \geq \log |M| - 2$ for sufficiently large n because in total at most $2k$ nuts are discarded that have not been paired before (again we can argue that we can apply an arbitrary algorithm that produces the desired output for the finitely many inputs where n is smaller). Furthermore, an easy induction argument shows that after the while-loop has been executed j times, then for all nuts that have not been discarded from M , the corresponding list of bolts has length 2^{j+1} . In particular, the nuts that survive all k iterations have a list of bolts of length of at least

$$2^{k+1} \geq 2^{(\log |M|)-1} = \frac{|M|}{2} \geq \frac{\gamma_M n}{2},$$

where the inequalities and equations follow, from left to right, by the preceding discussion, by definition of the function \log , and by Lemma 34. As each bolt appears at most q times in any list, each of these final lists contains at least $\delta_1 n$ mutually distinct bolts where $\delta_1 = \gamma_M n / (2q) > 0$. We show next that at any time any nut is a $1/4$ -approximate median of its list of bolts. This then finishes the proof of the proposition because then each nut surviving all iterations is a $1/4$ -approximate median for a fraction of δ_1 of all bolts and hence is a $\delta_1/4$ -approximate median with respect to all bolts.

It remains to show that at any time any nut \mathbf{n}_i in M is a $1/4$ -approximate median of its list of bolts. For a proof by induction, observe that initially this assertion is true by Lemma 34 and the choice of the \mathbf{b}_i^- and \mathbf{b}_i^+ . In order to show that this property is maintained during the execution of the algorithm, consider two nuts \mathbf{n}_i and \mathbf{n}_j that are paired and assume by induction hypothesis that they are $1/4$ -approximate medians of their lists of bolts L_i and L_j . We

assume that both nuts are in M_{high} and omit the symmetrical case where both nuts are in M_{low} . Furthermore, let n_i be smaller than n_j . We have to show that the nut that survives is a $1/4$ -approximate median of the combined list $L_i \circ L_j$. The nut n_i has rank at least $1/4$ with respect to the combined list because it has rank at least $1/2$ with respect to L_i and the lists L_i and L_j are of equal length. By induction hypothesis, nut n_j and hence also the smaller nut n_i have rank at most $3/4$ with respect to L_j . But n_i has also rank at most $3/4$ with respect to L_i and hence also with respect to the combined list. Thus n_i is a $1/4$ -approximate median for the combined list and the assertion follows because n_i survives unless the rank of n_j with respect to the combined list is even closer to $1/2$. \square

4 Applications in Logic

4.1 The Rado-graph and a 0-1-law for graph theory

In this section, we investigate into the properties of random graphs in a setting of first-order logic. For an introduction to first-order logic, we refer to the textbooks by Ebbinghaus, Flum, and Thomas [15] and by Enderton [16].

The language L_G of graph theory is the vocabulary that contains just a single binary predicate E . A structure over this vocabulary is called a graph if it satisfies the axioms of graph theory

$$\delta_1 \equiv (\forall x)[\neg E(x, x)] \quad \text{and} \quad \delta_2 \equiv (\forall x, y)[E(x, y) \rightarrow E(y, x)] ,$$

which assert that graphs do not contain loops (i.e., there are no edges between a node and itself) and are undirected (i.e., the edge relation is symmetric).

We consider graphs where the node set V and hence also the graph itself are either finite or countably infinite. Unless explicitly stated otherwise, we assume for simplicity that V is either equal to $\{1, \dots, n\}$ for some natural number $n > 0$ or is equal to \mathbb{N} . For any such node set V , the random graph

$$\widehat{G} = (V, \widehat{E})$$

is obtained by the chance experiment where the edge relation \widehat{E} is determined by independent tosses of a fair coin, i.e., each possible edge is added with probability $1/2$ and independent of all other edges. Note that this is indeed equivalent to picking a random graph according to the uniform distribution on the set of all graphs with node set V .

Example 36 *The edge relation of the random graph $\widehat{G} = (\mathbb{N}, \widehat{E})$ can be viewed as being obtained in stages $s = 1, 2, \dots$ where the set of edges is initially empty and*

at stage s , it is determined by independent tosses of a fair coin for all $j < s$, whether an edge between j and s should be added

(say, an edge is added if and only if the corresponding toss comes up head).

What can be said about the graph $(\mathbb{N}, \widehat{E})$? It comes as a slight surprise that with probability 1, this graph is isomorphic to some fixed graph, called the *Rado graph*. In order to prove the latter assertion we introduce the following notation.

Definition 37 *For any natural number $k > 0$ and any subset I of $\{1, \dots, k\}$,*

the extension axiom φ_k^I is defined by

$$\varphi_k^I \equiv \forall v_1 \dots \forall v_k \left[\bigwedge_{1 \leq i < j \leq k} v_i \neq v_j \rightarrow \exists v_{k+1} \left[\bigwedge_{1 \leq i \leq k} v_i \neq v_{k+1} \right. \right. \\ \left. \left. \wedge \bigwedge_{i \in I} (\mathbf{E}(v_i, v_{k+1}) \wedge \mathbf{E}(v_{k+1}, v_i)) \wedge \bigwedge_{i \notin I} (\neg \mathbf{E}(v_i, v_{k+1}) \wedge \neg \mathbf{E}(v_{k+1}, v_i)) \right] \right]$$

In words, the extension axiom φ_k^I asserts that for any k -tuple v_1, \dots, v_k of mutually distinct nodes there is a node v_{k+1} that is distinct from all the nodes in this tuple and is connected to the j th node in the tuple if and only if $j \in I$.

Proposition 38 *With probability 1, the random graph $(\mathbb{N}, \widehat{E})$ is a model of*

$$\mathsf{T}_{\text{rand}} = \{\varphi_k^I : k > 0 \text{ and } I \subseteq \{1, \dots, k\}\} \cup \{\delta_1, \delta_2\}.$$

Proof. By construction, $(\mathbb{N}, \widehat{E})$ satisfies δ_1 and δ_2 , hence it suffices to show that with probability 1 all extension axioms are satisfied. By σ -additivity of the underlying probability measure, it suffices in turn to show that each of the countably many extension axioms is satisfied with probability 1. So fix any extension axiom φ_k^I . Then for any k -tuple of natural numbers, the probability that this k -tuple witnesses that the given extension axiom is not satisfied, i.e., the probability that there is no node v_{k+1} as required, is just 0. For a proof, observe that every node added subsequent to the nodes in the tuple will have the properties required for v_{k+1} with probability $1/2^k$, hence the probability that none of the infinitely many additional nodes has these properties is 0. Furthermore, by σ -additivity of the underlying probability measure, the probability that for any of the countable many k -tuples there is no node v_{k+1} as required by φ_k^I is again 0. As a consequence, the graph \widehat{G} satisfies any given extension axiom with probability 1. \square

Theorem 39 *The theory T_{rand} is ω -categorical (i.e., all countably infinite models of this theory are isomorphic).*

Proof. Given two countable models (\mathbb{N}, E_1) and (\mathbb{N}, E_2) of T_{rand} , we construct an isomorphism g between them by the back-and-forth method. We build g in stages $s = 1, \dots$ where during stage $s > 1$, we extend an already defined partial isomorphism g_{s-1} to a partial isomorphism g_s with finite domain. Initially, we let g_1 be the mapping with domain $\{0\}$ where $g_1(0) = 0$. For all $i > 0$, during the stages $s = 2i$ and $s = 2i + 1$, we ensure that i is in the domain and in the range of g_s , respectively. Assuming that g_s has already been defined, stage $s + 1$ is defined as follows. First assume $s + 1 = 2i$. In case i is not already in the domain of g_s , let $g_{s+1}(i) = y$ for the least y that is not in the range of g_s and where for all x in the domain of g_s ,

$$(x, i) \in E_1 \quad \text{if and only if} \quad (g_s(x), y) \in E_2.$$

There is such a y because (\mathbb{N}, E_2) satisfies the corresponding extension axiom. Secondly, assume $s + 1 = 2i + 1$. In case i is not already in the range of g_s , let $g_{s+1}(y) = i$ for the least y that is not in the domain of g_s and satisfies for all x in the domain of g_s ,

$$(x, y) \in E_1 \quad \text{if and only if} \quad (g_s(x), i) \in E_2 .$$

Again such y exists because (\mathbb{N}, E_1) satisfies an appropriate extension axiom. \square

Definition 40 *The (up to isomorphism) unique countable model of T_{rand} is called Rado graph.*

From Proposition 38 and Theorem 39 the following corollary is then immediate.

Corollary 41 *With probability 1, the random graph $(\mathbb{N}, \widehat{E})$ is (isomorphic to) the Rado graph.*

So with high probability, any countably infinite random graph will satisfy all extension axioms. What's about finite models of the extension axioms? Of course, no finite graph can satisfy all extension axioms. But given any finite set of extension axioms, is it true in some finite graph? The answer to this question is affirmative and the proof works again by considering random graphs.

Definition 42 *For any sentence φ , let $p_n(\varphi)$ be the probability that φ is true in the random graph with n nodes.*

Lemma 43 *For any extension axiom φ holds $\lim_{n \rightarrow \infty} p_n(\varphi) = 1$.*

Proof. Fix any extension axiom φ_k^I . Then the probability that any given k -tuple in the random graph with $n > k$ nodes witnesses that φ_k^I does not hold is

$$\left(1 - \frac{1}{2^k}\right)^{n-k} .$$

By summing up over all k tuples, the probability that the random graph with n nodes does not satisfy φ_k^I can be bounded from above by

$$n^k \left(1 - \frac{1}{2^k}\right)^{n-k} \xrightarrow{n \rightarrow \infty} 0 ,$$

and the lemma follows. \square

Proposition 44 *Let T be any ω -categorical theory. Then T is complete (i.e., for all L_G -sentences φ , either φ is true in all models of T or $\neg\varphi$ is true in all models of T).*

Proof. Assume for a proof by contradiction that there were an L_G -sentence φ where the sets $T \cup \{\varphi\}$ and $T \cup \{\neg\varphi\}$ both have models. Then both sets have countable models by the theorem of Löwenheim and Skolem, i.e., there are two countable models of T that differ with respect to the truth value of φ and hence are not isomorphic. But this contradicts the assumption that T is ω -categorical \square

Theorem 45 (0-1-law of graph theory) *For any L_G -sentence φ we have either*

$$\lim_{n \rightarrow \infty} p_n(\varphi) = 1 \quad \text{or} \quad \lim_{n \rightarrow \infty} p_n(\varphi) = 0$$

Proof. Fix any L_G -sentence φ . By Theorem 39 and Proposition 44, T_{rand} either implies φ or $\neg\varphi$. We assume the former and omit the almost identical considerations for the latter case. By compactness of first-order logic, there is a finite subset T_φ of T_{rand} that already implies φ . Each sentence in T_φ is either a graph axiom, which hold in any graph, or is an extension axiom, which by Lemma 43 holds with probability arbitrary close to 1 for sufficiently large n . As a consequence, the probability that T_φ and hence φ holds goes to 1 when n goes to infinity. \square

The proof of Theorem 45 shows that for any L_G -sentence φ that is true in the Rado graph, $p_n(\varphi)$ converges to 1 while for any L_G -sentence that is false in the Rado graph, $p_n(\varphi)$ converges to 0. Results similar to the 0-1-law of graph theory can be derived for more comprehensive vocabularies and for more powerful logics, see for example the monograph by Ebbinghaus and Flum [14] and Compton [10, 11, 12].

4.2 A Fragment of First-order Logic with Decidable Satisfiability Problem

In this section we consider the problem of deciding satisfiability of a given first-order sentence. For first-order sentences in general this problem is undecidable, i.e., cannot be solved effectively, however the problem is decidable for certain fragments of first-order logic. As an example we consider the decidable fragment of first-order logic that contains sentences that can be written in the form $\forall^2 \exists^* \psi$ where ψ is quantifier-free and contains neither the equality sign, constant symbols, nor function symbols. We refer to Ebbinghaus, Flum and Thomas [15] for a discussion of undecidability of first-order logic. The monograph by Börger, Grädel and Gurevich [6] gives a comprehensive account of decidable and undecidable fragments of first-order logic that can be defined by restricting the form of quantifier-prefixes and the vocabulary.

There are finite axiom systems in which one can derive exactly those sentences of first-order logic that are valid, i.e., that hold in all structures, and accordingly there are effective procedures that enumerate exactly the valid sentences of

first-order logic (over some suitable vocabulary). However, there is no effective procedure that decides validity of first-order sentences, where deciding validity means that on any input the procedure eventually terminates and correctly tells whether the input is valid or not. Now a sentence is valid if and only if its negation is not satisfiable, i.e., does not hold in any model, hence, as we can not decide validity of sentences, we cannot decide satisfiability, either.

In general, we say the *satisfiability problem* of a given set of sentences is *decidable* if there is an effective procedure that correctly decides for any given sentence from this set whether the sentence is satisfiable (while the procedure might behave arbitrarily on inputs that are not in the given set). By the preceding discussion, the satisfiability problem of the set of all sentences is undecidable. However, for example the set of all relational $\forall^2\exists^*$ -sentences without equality introduced in Definition 46 has a decidable satisfiability problem.

Definition 46 *A $\forall^2\exists^*$ -sentence is a sentence of the form $\forall x_1\forall x_2\exists x_3\dots\exists x_{l+2}\psi$ for some $l \geq 0$ and a quantifier-free formula ψ .*

A formula is relational if in addition to the logical connectives it contains only relation symbols (but no constant or function symbols). A formula without equality is a formula that does not contain the equality symbol.

Theorem 47 *The set of relational $\forall^2\exists^*$ -sentences without equality has a decidable satisfiability problem.*

Theorem 47 is an immediate consequence of Propositions 49 and 50, i.e., the proof works by showing that the set of sentences under consideration has the finite model property. This property is shared by other fragments of first-order logic, where by Proposition 50 these fragments all have a decidable satisfiability problem, too [6, 14].

Definition 48 *A set of sentences has the finite model property if every satisfiable sentence in this set has a finite model.*

Proposition 49 *The set of relational $\forall^2\exists^*$ -sentences without equality has the finite model property.*

Proposition 50 *Every set of sentences with the finite model property has a decidable satisfiability problem.*

Proof of Proposition 50. In order to decide the satisfiability of an input φ , run the following two processes in parallel. First, we look for a finite model of φ by a possibly infinite exhaustive search. Secondly, we try to verify that $\neg\varphi$ is valid by enumerating all the valid sentences. In the verification of the algorithm, we can assume that φ is indeed from the given set. In case the formula φ is satisfiable, then it has a finite model and the first process eventually halts while the second

one runs forever. In case the formula φ is not satisfiable, then its negation $\neg\varphi$ is valid and consequently the second process eventually halts and the first one runs forever. So we can simply wait until one of the processes halts. Note that this decision procedure might fail to stop on inputs that are not in the given set of sentences. \square

Remark 51 *For any relational formula φ , let L_φ be the finite vocabulary that contains exactly the relation symbols occurring in φ . Then in order to show that a given set S of relational sentences has the finite model property, it suffices to show that any φ in S that is true in some L_φ -structure is in fact true in some finite L_φ -structure.*

For a proof, recall that in order to show that S has the finite model property, we have to demonstrate that any sentence φ in S has a finite model in case it is satisfiable at all, i.e., in case φ is true in some L -structure, where L must be a superset of L_φ . But given such an L -structure, by reducing it to the interpretations of the relation symbols in L_φ , we obtain an L_φ -structure that satisfies φ , hence by assumption φ is also true in some finite L_φ -structure. If wanted, we can even expand the latter L_φ -structure to a finite L -structure that satisfies φ by interpreting all relations in $L \setminus L_\varphi$ in an arbitrary way.

The still missing proof of Proposition 49 is split into two lemmata. First, in Lemma 53, we show that any satisfiable relational $\forall^2\exists^*$ -sentence without equality has a model without kings. A king is any element with a unique 1-table and the lemma is shown by simply duplicating elements in any given model of the sentence. Second, in Lemma 54, we conclude the proof of Proposition 49. Given any relational sentence without equality that is true in an L_φ -structure without kings, we give a probabilistic construction of a finite model of φ .

Definition 52 *Let L be a vocabulary that contains only relation symbols and let \mathfrak{A} be an L -structure with universe A .*

For $k > 0$, a k -table over vocabulary L is an L -structure with domain $\{1, \dots, k\}$. For mutually distinct elements a_1, \dots, a_k in A , the k -table induced by a_1, \dots, a_k in \mathfrak{A} is the unique k -table that is isomorphic to the substructure of \mathfrak{A} induced by the a_i via the mapping $i \mapsto a_i$. A k -table is realized in \mathfrak{A} if it is induced by some k -tuple over A .

An element of A is a king if it induces a 1-table that differs from all 1-tables induced by other elements of A . The structure \mathfrak{A} is a structure without kings if any 1-table that is realized in \mathfrak{A} at all is induced by at least two distinct elements of A .

In connection with Definition 52 and the following Lemma 53, observe that the concept of a model without kings is meant to be applied to L -structures where L is finite. For a vocabulary L with infinitely many relation symbols, any satisfiable relational sentence φ is true in an L -structure without kings because

the relation symbols in $L \setminus L_\varphi$ can be used to transform any given L_φ -structure that satisfies φ into an L -structure that satisfies φ and does not have kings.

Lemma 53 *Every satisfiable relational sentence φ without equality is true in an L_φ -structure without kings.*

Proof. Fix any satisfiable relational sentence φ without equality. By Remark 51, we can assume that φ is true in some L_φ -structure \mathfrak{A} . Let A be the universe of this structure. For any set I , let $\mathfrak{A} \times I$ be the unique L_φ -structure with universe $A \times I$ such that for every k -ary relation symbol in L_φ , all a_1, \dots, a_k in A and all i_1, \dots, i_k in I ,

$$R(a_1, \dots, a_k) \text{ is true in } \mathfrak{A} \iff R((a_1, i_1), \dots, (a_k, i_k)) \text{ is true in } \mathfrak{A} \times I.$$

If I has at least two elements, then the structure $\mathfrak{A} \times I$ does not have kings. Furthermore, a routine proof by induction over the construction of L_φ -formulae shows that any sentence without equality is true in \mathfrak{A} if and only if it is true in $\mathfrak{A} \times I$. \square

Lemma 54 *Let φ be a relational $\forall^2\exists^*$ -sentence. If φ is true in an L_φ -structure without kings, then φ has a finite model.*

Proof. Fix any L_φ -structure \mathfrak{A} without kings in which φ is true. We describe a randomized construction that for any given n yields an L_φ -structure \widehat{B}_n with universe $B_n = \{1, \dots, n\}$. Lemma 54 then follows by arguing that for sufficiently large n , with non-zero probability the construction results in a model of φ .

For $r = 1, 2$, let T_r be the set of all r -tables that are realized in \mathfrak{A} . Both sets are finite because L_φ is finite. For given n , the random L_φ -structure \widehat{B}_n is obtained as follows:

- (i) to each i in B_n , assign a 1-table \mathfrak{T}_i that is chosen uniformly at random from T_1 ,
- (ii) to each subset $\{i_1, i_2\}$ of B_n where $i_1 < i_2$, assign a 2-table $\mathfrak{T}_{\{i_1, i_2\}}$ that is chosen uniformly at random from the set of all 2-tables in T_2 where the 1-tables induced by the elements 1 and 2 are \mathfrak{T}_{i_1} and \mathfrak{T}_{i_2} , respectively,
- (iii) for any relation symbol R in L_φ of arity k and any k -tuple i_1, \dots, i_k over B_n where the truth value of $R(i_1, \dots, i_k)$ has not already been determined during steps (i) and (ii) (i.e., for any k -tuple that has at least 3 mutually distinct components), let $R(i_1, \dots, i_k)$ be true with probability $1/2$.

It is to be understood that the random choices made during this construction are done such that their outcomes are mutually independent. Observe that the construction requires that the initial model \mathfrak{A} has no kings because otherwise a

king's 1-table might be assigned to more than one element during step (i), thus leaving no choice for the 2-table of any pair of such elements in step (ii).

It remains to show that for n sufficiently large, φ is true in $\widehat{\mathfrak{B}}_n$ with non-zero probability. Assume that φ can be written in the form

$$\varphi \equiv \forall x_1 \forall x_2 \exists x_3 \dots \exists x_{l+2} \psi$$

where ψ is a quantifier-free formula ψ in variables x_1, \dots, x_{l+2} . Then φ is true in any given L_φ -structure if for every non-empty subset $\{i_1, i_2\}$ of this structure's universe there are not necessarily distinct elements i_3, \dots, i_{l+2} in the universe such that $\psi[i_1, \dots, i_{l+2}]$ is true.

We show in a minute, that there is a rational $\delta < 1$ such that for any fixed subset $\{i_1, i_2\}$ of B_n , the probability that in $\widehat{\mathfrak{B}}_n$ there are no elements i_3, \dots, i_{l+2} as required is at most δ^n . By summing up this bound on the "error probability" over the less than n^2 non-empty sets $\{i_1, i_2\}$ in $\widehat{\mathfrak{B}}_n$, the probability that φ is not true in $\widehat{\mathfrak{B}}_n$ then can be bounded from above by

$$n^2 \delta^n \xrightarrow{n \rightarrow \infty} 0.$$

As a consequence, the probability that φ is true in $\widehat{\mathfrak{B}}_n$ is not just non-zero for large n but even tends to 1 when n goes to infinity.

It remains to show that there is a rational δ as required. Let \mathfrak{T} be any 1- or 2-table in $T_1 \cup T_2$. Then \mathfrak{T} is realized in the structure \mathfrak{A} , which satisfies φ , hence we can extend \mathfrak{T} to an $(l+2)$ -table $\text{ext}(\mathfrak{T})$ that is realized in \mathfrak{A} and where $\psi[1, 2, i_3, \dots, i_{l+2}]$ is true in $\text{ext}(\mathfrak{T})$ for not necessarily distinct numbers i_3, \dots, i_{l+2} in $\{1, \dots, l+2\}$ (i.e., in general not all numbers i_3 through i_{l+2} have to be considered for verifying that the existential formula φ is true).

Now consider the randomized construction of \widehat{B}_n . Fix n and any subset $\{i_1, i_2\}$ of B_n of cardinality $r \in \{1, 2\}$. Furthermore, assume that during step (ii) of the construction of $\widehat{\mathfrak{B}}_n$, we have assigned to $\{i_1, i_2\}$ some r -table \mathfrak{T} from $T_1 \cup T_2$. Now let i_3, \dots, i_{l+2} be any numbers in B_n that are mutually distinct and differ from i_1 and i_2 . Then the table induced by $\{i_1, \dots, i_{l+2}\}$ in $\widehat{\mathfrak{B}}_n$ will be equal to $\text{ext}(\mathfrak{T})$ with probability $\varepsilon(\mathfrak{T}) > 0$ that does not depend on n or on the i_j . For a proof, observe that this table is determined by the 1- and 2-tables induced by its subsets of cardinality 1 and 2 plus finitely many atomic formulae that have at least three mutually distinct arguments. Furthermore, these assignments are mutually independent and with each of them, with non-zero probability the same value as in $\text{ext}(\mathfrak{T})$ is assigned. The latter assertion is obvious for atomic formulae with at least three arguments, while for the induced 1- and 2-tables it suffices to observe that all 1- and 2-tables that are realized in $\text{ext}(\mathfrak{T})$ are also realized in \mathfrak{A} and are hence in T_1 or T_2 .

In $\widehat{\mathfrak{B}}_n$, we can pick $\lfloor (n-2)/l \rfloor$ mutually disjoint subsets $\{i_3, \dots, i_{l+2}\}$ as above and the corresponding events that $\text{ext}(\mathfrak{T})$ is realized are mutually independent. So for almost all n , the probability that $\text{ext}(\mathfrak{T})$ is not realized for any of these

subsets can be bounded from above by

$$(1 - \varepsilon(\mathfrak{T}))^{\lfloor \frac{n-2}{l} \rfloor} \leq (1 - \varepsilon(\mathfrak{T}))^{\frac{n}{2l}}. \quad (18)$$

Then the term in (18) bounds the conditional probability that $\psi[i_1, \dots, i_{l+2}]$ is false in $\widehat{\mathfrak{B}}_n$ for any choice of i_3 through i_{l+2} under the assumption that the r -table \mathfrak{T} is assigned to x_1 and x_2 . The corresponding unconditional probability can be bounded by taking the minimum over the finitely many conditional probabilities that correspond to the possible choices of an r -table for $\{x_1, x_2\}$ in T_1 or T_2 . Hence the probability that $\psi[i_1, \dots, i_{l+2}]$ is false in $\widehat{\mathfrak{B}}_n$ for any choice of i_3 through i_{l+2} is at most δ^n where

$$\delta = (1 - \varepsilon)^{\frac{1}{2l}} < 1 \quad \text{and} \quad \varepsilon = \min_{\mathfrak{T} \in T_1 \cup T_2} \varepsilon(\mathfrak{T}).$$

This concludes the proof of Lemma 54 . □

By elaborating on the proof of Theorem 47, one can derive the stronger result that the set of relational sentences without equality and a quantifier prefix of the form $\exists^* \forall^2 \exists^*$ has a decidable satisfiability problem. On the other hand, by a result of Goldfarb, the satisfiability problem for relational $\forall^2 \exists^*$ -sentences with equality is undecidable [6, Sections 6.2.3 and 4.3]

References

- [1] Martin Aigner and Günter M. Ziegler. *Proofs from THE BOOK. 2nd ed.* Springer, Berlin, 2001.
- [2] Noga Alon, Manuel Blum, Amos Fiat, Sampath Kannan, Moni Naor, and Rafail Ostrovsky. Matching nuts and bolts. In *SODA 1994*, pages 690–696, 1994.
- [3] Noga Alon and Joel H. Spencer. *The probabilistic method. 2nd ed.* John Wiley & Sons, New York, 2000.
- [4] M. Blum. Coin flipping by telephone. In *CRYPTO 81*, pages 11–15, 1982.
- [5] Manuel Blum and Hal Wasserman. Reflections on the pentium division bug. *IEEE Transactions on Computers*, 45(4):385–393, 1996.
- [6] Egon Börger, Erich Grädel, and Yuri Gurevich. *The classical decision problem.* Springer, Berlin, 1997.
- [7] Phillip G. Bradford and Rudolf Fleischer. Matching nuts and bolts faster. In *ISAAC 1995*, volume 1004 of *Lecture Notes in Computer Science*, pages 402–408, 1995.
- [8] Gilles Brassard, David Chaum, and Claude Crépeau. Minimum disclosure proofs of knowledge. *Journal of Computer and System Sciences*, 37:156–189, 1988.
- [9] Kai Lai Chung. *Elementary probability theory with stochastic processes. 3rd ed.* Springer, New York, 1979.
- [10] Kevin J. Compton. A logical approach to asymptotic combinatorics I. First order properties. *Adv. Math.*, 65:65–96, 1987.
- [11] Kevin J. Compton. The computational complexity of asymptotic problems. I: Partial orders. *Inf. Comput.*, 78(2):108–123, 1988.
- [12] Kevin J. Compton. A logical approach to asymptotic combinatorics. II: Monadic second-order properties. *J. Comb. Theory, Ser. A*, 50(1):110–131, 1989.
- [13] Thomas H. Cormen, Charles Leiserson, and Ronald Rivest. *Introduction to algorithms. 2nd ed.* McGraw-Hill, 2001.
- [14] Heinz-Dieter Ebbinghaus and Jörg Flum. *Finite model theory. 2nd rev. and enlarged ed.* Springer, Berlin, 1999.
- [15] Heinz-Dieter Ebbinghaus, Jörg Flum, and Wolfgang Thomas. *Mathematical logic.* Springer, Berlin, 1994.

- [16] Herbert B. Enderton. *A mathematical introduction to logic. 2nd ed.* Harcourt/Academic Press, Burlington, MA, 2001.
- [17] William Feller. *An introduction to probability theory and its applications. 3rd ed.* John Wiley & Sons, New York, 1968.
- [18] William Feller. *An introduction to probability theory and its applications. Vol II. 2nd ed.* John Wiley & Sons, New York, 1971.
- [19] Michael Habib, Colin McDiarmid, Jorge Ramirez-Alfonsin, and Bruce Reed, editors. *Probabilistic methods for algorithmic discrete mathematics.* Springer, Berlin, 1998.
- [20] Shai Halevi and Silvio Micali. Practical and provably-secure commitment schemes from collision-free hashing. In *CRYPTO 96*, volume 1109 of *Lecture Notes in Computer Science*, pages 201–215, 1996.
- [21] János Komlós, Yuan Ma, and Endre Szemerédi. Matching nuts and bolts in $O(n \log n)$ time. *SIAM Journal on Discrete Mathematics*, 11(3):347–372, 1998.
- [22] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of applied cryptography.* CRC Press, Boca Raton, 1997.
- [23] Peter Bro Miltersen. Derandomizing complexity classes. To appear in *Handbook on randomization*, preliminary version available from <http://www.daimi.aau.dk/~bromille/Papers/index.html>.
- [24] Rajeev Motwani and Prabhakar Raghavan. *Randomized algorithms.* Cambridge Univ. Press, Cambridge, 1995.
- [25] Walter Rudin. *Principles of mathematical analysis. 3rd ed.* McGraw-Hill Book Company, 1976.
- [26] Bruce Schneier. *Applied cryptography. Protocols, algorithms, and source code in C.* Wiley, New York, 1993.
- [27] Douglas R. Stinson. *Cryptography. Theory and practice.* CRC Press, Boca Raton, 1995.