A Taste of Randomized Computations*

Oded Goldreich
Department of Computer Science and Applied Mathematics
Weizmann Institute of Science, Rehovot, ISRAEL.

March 6, 2001

Abstract

The purpose of this text is to demonstrate the usage of randomization in a variety of computational settings. Our choice is governed by the desire to focus on the randomization aspect of the solution and avoid any complicated details that are due to other aspects of the computational problem. Thus, we avoid any example that requires substantial problem-specific background. Our examples are grouped in three (subjective) categories:

1. Traditional algorithmic problems. Here we consider *randomized algorithms* for graph theoretic problems such as finding a perfect matching, algebraic problems such as testing polynomial identity, and approximation problems such as approximating the number of satisfying assignments to a DNF formula.
2. Traditional complexity questions. Here we present results such as the *randomized reductions* of Approximate Counting to $\mathcal{NP}$, and of SAT to unique-SAT.
3. Distributed and Parallel Computing. Here we consider *randomized procedures* for *distributed tasks* such as Testing String Equality, Byzantine Agreement, and routing in networks.

We stress that our presentation is merely aimed at *demonstrating* the usage of randomization, and that no attempt was made to present a coherent theory of randomized computation. Furthermore, our presentation tends to be laconic (i.e., it lacks some technical details as well as wider perspective).

*Adapted from Appendix B of *Modern Cryptography, Probabilistic Proofs and Pseudorandomness* [11].

Contents

Comment: The interplay between randomness and computation is one of the most fascinating scientific phenomena uncovered in the last couple of decades. This interplay is at the heart of modern cryptography and plays a fundamental role in complexity theory at large. Specifically, the interplay of randomness and computation is pivotal to several intriguing notions of probabilistic proof systems and is the focal of the computational approach to randomness. In this text, we have avoided the above areas. For an introduction to to these three, somewhat interwoven areas, the interested reader is referred to the text *Modern Cryptography, Probabilistic Proofs and Pseudorandomness* [11]. For a more systematic, detailed and inclusive exposition of Randomized Algorithms, the interested reader is referred to a textbook by Motwani and Raghavan [24].

1 Randomized Algorithms

Conspicuous omissions in this category include some of the most well-known randomized algorithms (e.g., many in the domain of computational number theory), as well as the Markov Chain approach to approximate counting. As stated above, the reason for these omissions is that these algorithms either require specialized (and unrelated to randomness) background or are quite involved to present and/or analyze.

1.1 Approximate Counting of DNF satisfying assignments or, a twist on naive sampling

The problem considered here is to approximate the number of satisfying assignment to a DNF formula up-to a *constant factor*. We note that given ϵ and oracle access to any function $f : \{0,1\}^n \to \{0,1\}$, it is easy to approximate the fraction $|\{x : f(x) = 1\}|/2^n$ up-to an ϵ *additive deviation*. Specifically, a sample of $O(\epsilon^{-2}\log(1/\delta))$ points has average value that, with probability at least $1-\delta$, is at most ϵ-away from the correct value. However, our aim is to provide relative (rather than absolute) approximation of this fraction (i.e., given $\epsilon > 0$ the task is to approximate the above fraction up-to a $1 \pm \epsilon$ factor).

Let $\varphi = \bigvee_{i=1}^m C_i$, where $C_i : \{0,1\}^n \to \{0,1\}$ is a conjunction, be a DNF formula. Actually, we will deal with the more general problem in which we are given (implicitly) m subsets $S_1, ..., S_m \subseteq \{0,1\}^n$ and wish to approximate $|\bigcup_i S_i|$. In our case S_i will be the set of assignments satisfying the conjunction C_i. We make several computational assumptions regarding these sets (letting efficient mean implementable in time polynomial in $n \cdot m$):

1. Given i and x, one can efficiently determine whether or not $x \in S_i$.
2. Given i, one can efficiently determine $|S_i|$.
3. Given i, one can efficiently generate a uniformly distributed element of S_i.

These assumptions are clearly satisfied in the case $S_i = C_i^{-1}(1)$ considered above. The key observation is that

$$\left|\bigcup_{i=1}^m S_i\right| = \sum_{i=1}^m \left|S_i \setminus \bigcup_{j<i} S_j\right| \tag{1}$$

$$= \sum_{i=1}^m |S_i| \cdot \Pr_{s\in S_i}\left[s \notin \bigcup_{j<i} S_j\right] \tag{2}$$

and that the probabilities in Eq. (2) can be approximated up-to ϵ' (with overwhelming success probability) by taking $\text{poly}(n/\epsilon')$ many samples. This leads to the following algorithm

Algorithm: On input parameters ϵ and δ, set $\epsilon' = \epsilon/m$ and $\delta' = \delta/m$. For $i = 1$ to m do

1. Let $p_i \stackrel{\text{def}}{=} \Pr_{s\in S_i}[s \notin \bigcup_{j<i} S_j]$. Using a sample of size $t \stackrel{\text{def}}{=} O((1/\epsilon')^2 \log(1/\delta'))$, approximate p_i by $\widetilde{p}_i$ so that $\Pr[|\widetilde{p}_i - p_i| > \epsilon'] < \delta'$. That is, we uniformly select t samples in S_i, and test for each sample whether or not it resides in $\bigcup_{j<i} S_j$.
2. Compute $|S_i|$, and let $a_i \stackrel{\text{def}}{=} \widetilde{p}_i \cdot |S_i|$.

Output the sum of the a_i's.

Analysis: Let $N_i = p_i \cdot |S_i|$. We are interested in the quality of the approximation to $\sum_i N_i$ provided by $\sum_i a_i$. With probability at least $1 - m \cdot \delta'$, we have $a_i = (p_i \pm \epsilon') \cdot |S_i| = N_i \pm \epsilon' \cdot |S_i|$, for all i's, and so $\sum_i a_i = \sum_i N_i \pm \epsilon' \cdot \sum_i |S_i|$. However, $\max_i(|S_i|) \leq |\bigcup_i S_i| = \sum_i N_i$, and so

$$\begin{aligned}\sum_{i=1}^{m} a_i &= \sum_{i=1}^{m} N_i \ \pm \ m \cdot \epsilon' \cdot \max_{1\leq i\leq m} |S_i| \\ &= (1 \pm m\epsilon') \cdot \sum_{i=1}^{m} N_i \ = \ (1 \pm \epsilon) \cdot \sum_{i=1}^{m} N_i\end{aligned}$$

Note that the above approach does not require exact computation of $|S_i|$, nor exact uniform selection in S_i. Instead, ability to approximate $|S_i|$ up-to a factor of $1 \pm \epsilon'$ within time related to $\mathrm{poly}(n/\epsilon')$ suffices. Likewise, it suffice to generate in time related to $\mathrm{poly}(n/\epsilon')$ a distribution that is at most ϵ'-away from the uniform distribution over S_i.

The algorithm presented above is actually a deterministic reduction of the task of approximating the size of one set (in the relative sense) to the task of providing absolute approximations to some fractions. It utilizes the hypothesis that the first set can be expressed as a union of feasibly many sets for which certain natural operations (e.g., deciding membership, approximating the size) can be performed efficiently. Thus, this approach may be applicable to some sets, but not to their complement. We stress that, in general, relative approximation may be feasible for one quantity, but not for its complement (e.g., it is NP-Hard to approximate the number of UNsatisfying assignment to a DNF formula up-to any factor).

1.2 Finding a perfect matching or, on the loneliness of the extremum

The problem considered here is to find a perfect matching in a graph. The specific goal is to obtain a fast parallel algorithm, which is the reason we do not follow the standard combinatorial approach (of iteratively augmenting the current matching using alternating paths). Instead, we rely on the following Isolation Lemma that asserts that when assigning each edge a random weight, taken from a sufficiently large domain, there is a unique perfect matching of minimum (resp., maximum) weight. The lemma extends to arbitrary set systems.

Lemma 1 (The Isolation Lemma): *Let $S_1, S_2, ..., S_t \subseteq [m] \stackrel{\rm def}{=} \{1, 2, ..., m\}$ be distinct sets, and let $w_1, w_2, ..., w_m$ be independently and uniformly chosen in $[2m]$. Then, with probability at least $1/2$, there exists a unique j so that $\sum_{i\in S_j} w_i$ equals $\min_{k\in[t]}(\sum_{i\in S_k} w_i)$.*

In our application $[m]$ corresponds to the set of edges, and the S_i's to perfect matchings in the graph.

Proof: For $i = 1, ..., m$, consider the event E_i defined as the existence of two sets (i.e., S_j's) with minimum weight so that one set contains i and the other set does not contain i. It suffices to show that the probability that E_i occurs is at most $1/2m$. The latter is proven by considering a random process in which the weight of i (i.e., w_i) is selected last.

Suppose that the values of all other w_j's (with $j \neq i$) have already been determined. Let S^- be a set of minimum weight among all sets not containing i, and w^- be its weight (i.e., $w^- \stackrel{\rm def}{=} \min_{j:i\notin S_j}(\sum_{k\in S_j} w_k)$). Similarly, let S^+ be a set of minimum weight among all sets obtained by omitting i from sets that contain it, and w^+ be its weight (i.e., $w^+ \stackrel{\rm def}{=} \min_{j:i\in S_j}(\sum_{k\in S_j\setminus\{i\}} w_k)$). Then, event E_i occurs if and only if $w^- = w^+ + w_i$, which happens with probability $1/2m$ if $(w^- - w^+) \in [2m]$, and with probability 0 otherwise. ■

Algorithm: On input a bipartite graph $G = (U, V, E)$, do:

1. For each edge $e \in E$, uniformly and independently select a weight $w_e \in [2m]$, where $m \stackrel{\text{def}}{=} |E|$.
2. Try to compute the value of the minimum-weight perfect-matching. This is done by computing the determinant of the matrix, denoted A, obtained by setting the (u, v)-entry to 2^{w_e} if $e = (u, v)$ and to 0 if $(u, v) \notin E$. In case the determinant is 0, halt stating that the graph has no perfect matching. Otherwise, the value of the minimum-weight perfect-matching is set to be the largest i so that the value of the determinant is divisible by 2^i. (The determinant can be computed by a fast parallel algorithm.)
3. For each $e \in E$, try to compute the value of the minimum-weight perfect-matching among those not containing the edge e. This is done (as above) by computing the determinant of the matrix, denoted A_e, obtained from A by resetting the e-entry to 0. All these computations can be conducted in parallel.
4. A candidate perfect matching is retrieved by including all edges e for which the value (of the min-weight perfect matching) found in Step 3 is different than the one found in Step 2.

The algorithm for general graphs is a variation of the above (and is not described here). Note that Steps 1 and 2 (by themselves) provide a randomized algorithm for determining whether a bipartite graph has a perfect matching.

Analysis: The determinant of A sums (possibly with minus sign) the contributions of all perfect matchings in the graph G, where the contribution of a perfect matching M equals $\pm \prod_{e \in M} 2^{w_e} = \pm 2^{\sum_{e \in M} w_e}$. We may assume that the graph has a perfect matching, or else the determinant computed in Step 2 is 0. Assume that the weights (i.e., w_e's) are such that there exists a unique perfect matching of minimum weight. Denote this (minimum-weight perfect) matching by M, and denote its weight by W. In such a case, the determinant of A is of the form $2^W + r \cdot 2^{W+1}$, where r is an integer (possibly zero). This is so because the contribution of the *unique* minimum-weight perfect-matching is $\pm 2^W$, and the contribution of each other perfect-matching is $\pm 2^{W'}$, where $W' > W$ (are both integers). Likewise, for every edge e not in M, the determinant of A_e is of the form $2^W + r \cdot 2^{W+1}$, where again r is an integer. On the other hand, for every edge e in M, the determinant of A_e is either zero or $r \cdot 2^{W+1}$, with r being a non-zero integer.

A Parenthetical Comment[1]

It is tempting to think that when selecting weights as above, the minimum-weight perfect matching may be uniformly distributed among all perfect matchings. As shown below, this is not always the case (which is unfortunate, because otherwise we would have obtained a simple probabilistic polynomial-time algorithm for uniformly generating a perfect matching in a graph).

Consider a graph in which the set of perfect matchings consists of two types of matchings. There are 2^n matchings of the first type, a generic one having the form $\{e_{2i-\sigma_i} : i = 1, ..., n\}$, where $\sigma_1, ..., \sigma_n \in \{0, 1\}$. In addition, there is a single matching of the second type, denoted $\{e_{2n+i} : i = 1, ..., n\}$. We claim that the probability that the minimum-weight perfect matching is a specific matching of the first type is exponentially smaller than the probability that the minimum-weight perfect matching is the matching of the second type. This claim holds for weights distributed

[1] This parenthetical comment is based on discussions with Madhu Sudan (during March 1998).

as above, as well as for several other distributions (e.g., the Normal Distribution). For sake of simplicity, we consider weights uniformly distributed in the interval $[0,1]$. The claim is proven by combining the following two facts.

Fact 1.2.1 *With overwhelmingly high probability, the value of the minimum-weight matching among all 2^n matchings of the first type is* at least *cn, where c is any constant smaller than* $1/3$ (e.g., $c = 0.32$).

Proof Sketch: This follows by observing that

$$\min_{\sigma_1,\ldots,\sigma_n \in \{0,1\}} \left(\sum_{i=1}^{n} w_{2i-\sigma_i} \right) = \sum_{i=1}^{n} \min(w_{2i-1}, w_{2i})$$

and that the expected value of each $\min(w_{2i-1}, w_{2i})$ equals 1/3. $\Box$

Fact 1.2.2 *The probability that any specific perfect matching* (and in particular the one of the second type) *has weight* less than, *say,* $0.31 \cdot n$ *is greater than* $\frac{0.6^n}{2} = \exp(\Omega(n)) \cdot 2^{-n}$.

Proof Sketch: This follows by observing that

$$\begin{aligned} &\Pr\left[\sum_{i=1}^{n} w_i < 0.31 \cdot n\right] \\ &> \Pr[\forall i\,(w_i \le 0.6)] \cdot \left(1 - \Pr\left[\sum_{i=1}^{n} w_i \ge 0.31 \cdot n \;\middle|\; \forall i\,(w_i \le 0.6)\right]\right) \\ &> 0.6^n \cdot \frac{1}{2} \end{aligned}$$

where the last inequality uses $\mathrm{Exp}[w_i \mid w_i \le 0.6] = 0.3$. ■

Combining Facts 1.2.1 and 1.2.2, we conclude that, with probability $\exp(\Omega(n)) \cdot \frac{1}{2^n+1}$, the single (second type) matching has weight *less than* $0.31 \cdot n$ and every perfect matching of the first type has weight *at least* $0.32 \cdot n$. In this case, the single (second type) matching is of minimum-weight among all $2^n + 1$ perfect matchings, and the claim follows.

1.3 Testing whether polynomials are identical or, on the discrete charm of polynomials

The problem considered here is to determine whether two multi-variant polynomials are identical. We assume that one is given an oracle for the evaluation of each of the polynomials. We further assume that the polynomials are defined over a sufficiently large finite field, denoted F. Finally, let n denote the number of variables in these polynomials.

Algorithm: Given n and black-box access to $p, q : \mathrm{F}^n \to \mathrm{F}$, uniformly select $r_1, ..., r_n \in \mathrm{F}$, and accept if and only if $p(r_1, ..., r_n) = q(r_1, ..., r_n)$.

Analysis: Clearly, if $p \equiv q$ then the algorithm always accepts. The following lemma implies that if p and q are different polynomials, each of total degree at most d, then the algorithm accepts with probability at most $d/|\mathrm{F}|$.

Lemma 2 *Let $p : \mathrm{F}^n \to \mathrm{F}$ be a non-zero polynomial of total degree d. Then*

$$\mathrm{Pr}_{r_1,...,r_n}[p(r_1,...,r_n) = 0] \;\leq\; \frac{d}{|\mathrm{F}|}$$

Proof: The lemma is proven by induction on n. The base case of $n = 1$ follows immediately by the Fundamental Theorem of Algebra (i.e., the number of distinct roots of a degree d univariant polynomial is at most d). In the induction step, we write p as a polynomial in its first variable. That is,

$$p(x_1, x_2, ..., x_n) = \sum_{i=0}^{d} p_i(x_2, ..., x_n) \cdot x_1^i$$

where p_i is a polynomial of total degree at most $d - i$. Let t be the biggest integer i for which p_i is not identically zero. (We dismiss the case $t = 0$.) Then, using the induction hypothesis, we have

$$\begin{aligned}\mathrm{Pr}_{r_1,r_2,...,r_n}[p(r_1,r_2,...,r_n) = 0] \;&\leq\; \mathrm{Pr}_{r_2,...,r_n}[p_t(r_2,...,r_n) = 0] \\ &\quad + \mathrm{Pr}_{r_1,r_2,...,r_n}[p(r_1,r_2,...,r_n) = 0 \,|\, p_t(r_2,...,r_n) \neq 0] \\ &\leq\; \frac{d-t}{|\mathrm{F}|} + \frac{t}{|\mathrm{F}|}\end{aligned}$$

where the second term is bounded by fixing any sequence $r_2, ..., r_n$ for which $p_t(r_2, ..., r_n) \neq 0$ and considering the univariant polynomial $p'(x) \stackrel{\text{def}}{=} p(x, r_2, ..., r_n)$, which by hypothesis is a non-zero polynomial of degree t. ■

Comment: The lesson is that whenever the situation is such that *almost* any choice will do, taking a random choice yields an algorithm with a rigorous performance guarantee. In a sense any randomized algorithm is based on this paradigm, except that here the space of choices seems more straightforward than in any other case. That is, most randomized algorithms are based on introducing a sample space that is not obvious from the problem at hand; whereas here the sample space is the obvious one.

1.4 Randomized Rounding applied to MaxSAT or, on being fractionally pregnant

We slightly deviate from the above style of exposition by considering a general methodology for approximating combinatorial optimization problems. The methodology, which relies on the fact that linear programming is solveable in polynomial-time, consists of two steps. First, one presents a linear programming relaxation of an integer program (corresponding to a combinatorial problem). Next, one derives from a solution to the linear program (LP) a solution to the integer program (IP). This is done by using the former (LP) solution in order to determine a probability distribution over integer solutions (i.e., solution to the IP), and picking a solution according to this distribution. We exemplify this methodology by applying it to Max-SAT. Specifically, we consider the task of approximating the maximum number of clauses that can be simultaneously satisfied in a given CNF formula.

Let $\varphi = \bigwedge_{j=1}^m C_j$ be a CNF formula, where $C_j = (\bigvee_{i \in S_j^+} x_i) \vee (\bigvee_{i \in S_j^-} \neg x_i)$ with $S_j^+, S_j^- \subseteq [n] \stackrel{\rm def}{=} \{1, ..., n\}$. Abusing notation, we may express Max-SAT as an integer optimization problem in which the task is to maximize $\sum_{j=1}^m y_j$ subject to

$$x_i, y_j \in \{0,1\} \qquad (\forall i,j) \tag{3}$$

$$\sum_{i \in S_j^+} x_i + \sum_{i \in S_j^-} (1 - x_i) \geq y_j \qquad (\forall j) \tag{4}$$

In the Linear Programming (LP) relaxation, one replaces Eq. (3) by

$$0 \leq x_i, y_j \leq 1 \quad (\forall i,j) \tag{5}$$

Clearly, the value of the LP is lower bounded by the value of the integer program. Given an (optimal) solution, $\hat{x}_i, \hat{y}_j$, to the LP, we randomly derive a solution to the original integer formulation. It will be shown that the expected value of the integer solution is at least $1 - e^{-1}$ times the value of the LP (and hence at least a $1 - e^{-1}$ fraction of the optimum of the integer problem). Specifically, we set $x_i = 1$ with probability $\hat{x}_i$ (and $x_i = 0$ otherwise).

Analysis: Suppose that clause C_j has c_j literals. Below, we will show that the probability that C_j is satisfied by the integer assignment (generated by above randomized rounding of the above LP solution) is at least

$$\left(1 - \left(1 - \frac{1}{c_j}\right)^{c_j}\right) \cdot \hat{y}_j \geq \left(1 - e^{-1}\right) \cdot \hat{y}_j$$

and so the expected number of satisfied clauses is at least $(1 - e^{-1}) \cdot \sum_j \hat{y}_j$ (as stated above). The above is proven by noting that the probability of the complementary event (i.e., C_j is not satisfied) is

$$\left(\prod_{i \in S_j^+} (1 - \hat{x}_i)\right) \cdot \left(\prod_{i \in S_j^-} \hat{x}_i\right) \tag{6}$$

where, by Eq. (4), $\sum_{i \in S_j^+} (1 - \hat{x}_i) + \sum_{i \in S_j^-} \hat{x}_i \leq (c_j - \hat{y}_j)$. Eq. (6) is maximized when $1 - \hat{x}_i = (c_j - \hat{y}_j)/c_j$ for all $i \in S_j^+$, and $\hat{x}_i = (c_j - \hat{y}_j)/c_j$ for all $i \in S_j^-$. Thus, Eq. (6) is bounded above by $\left(1 - \frac{\hat{y}_j}{c_j}\right)^{c_j}$, and the above claim follows.

Comments: Combining the above algorithm with the naive algorithm that uniformly selects a truth assignment, one derives a randomized algorithm of a 3/4-approximation factor. The key observation is that the performance of the LP-based algorithm improves as the clause sizes decrease, whereas the performance of the naive algorithm improves when the sizes increase. In a different vein, we mention that the randomized rounding paradigm has been extended also to *semidefinite* (rather than linear programming) relaxations of combinatorial problems. In fact, improved approximation ratios for various versions of MaxSAT were obtained that way (cf., [10, 16]).

1.5 Primality Testing or, on hiding information from an algorithm

The problem considered here is to decide whether a given number is a prime. The only Number Theoretic facts that we use are:

Fact 1.5.1 *For every prime $p > 2$, each quadratic residue mod p has exactly two square roots mod p (and they sum-up to p).*

Fact 1.5.2 *For every (odd and non-integer-power) composite number N, each quadratic residue mod N has at least four square roots mod N.*

Our algorithm uses as a black-box an algorithm, denoted R, that given a prime p and a quadratic residue mod p, returns the smallest among the two square roots. There is no guarantee as to what is the output in case the input is not of the above form (and in particular in case p is not a prime).

Algorithm: On input a natural number $N > 2$ do

1. If N is either even or an integer-power then `reject`.
2. Uniformly select $r \in \{1, ..., N-1\}$, and set $s \leftarrow r^2 \bmod N$.
3. Let $r' \leftarrow R(N, s)$. If $r' \equiv \pm r \pmod N$ then `accept` else `reject`.

Analysis: By Fact 1.5.1, on input a prime number N, the above algorithm always accepts (since in this case $R(N, r^2 \bmod N) = \pm r$ for any $r \in \{1, ..., N-1\}$). On the other hand, suppose that N is an odd composite that is not an integer-power. Then, by Fact 1.5.2, each quadratic residue s has at least four square roots, and each is equally likely to be chosen at Step 2 (becuase s yields no information on the specific r). Thus, for every such s, the probability that $\pm R(N, s)$ has been chosen in Step 2 is at most $2/4$. It follows that, on input a composite number, the algorithm rejects with probability at least $1/2$.

Comment: The above analysis presupposes that the algorithm R is always correct when fed with a pair (p, s), where p is prime and s a quadratic residue mod p. In case R has error probability $\epsilon < 1/2$, our algorithm still distinguishes primes from composites (since on the former it accepts with probability at least $1 - \epsilon > 1/2$). We note that efficient randomized algorithms for extracting square roots modulo a prime are known (cf., [5, 24]). Thus, the above establishes that primality can be decided in probabilistic polynomial-time (alas, with two-sided error).

1.6 Testing Graph Connectivity via a random walk or, the accidental tourist sees it all

The problem considered here is to decide whether a given graph is connected. The aim is to devise an algorithm that does so while using little space (i.e., essentially, as little as needed for storing the identity of a single vertex). This task can be reduced (in small space) to testing connectivity between any given pair of vertices. Thus, we focus on the task of determining whether or not two given vertices are connected in a given graph.

Algorithm: On input a graph $G = (V, E)$ and two vertices, s and t, we take a *random walk* of length $O(|V| \cdot |E|)$, starting at vertex s, and test at each step whether or not vertex t is encountered. By a random walk we mean that, at each step, we uniformly select one of the edges incident at the current vertex and traverse this edge to the other endpoint.

Analysis: We will show that if s is connected to t in the graph G then, with probability at least $1/2$, vertex t is encountered in a random walk starting at s. In the following, we consider the connected component of vertex s, denoted $G' = (V', E')$. For any edge, (u, v) (in E'), we let $T_{u,v}$ be a random variable representing the number of steps taken in a random walk starting at u until v is first encountered. It can be shown that $\mathrm{E}[T_{u,v}] \leq 2|E'|$.[2] Also, letting cover$(G')$ be the expected number of steps in a random walk starting at s and ending when the last of the vertices of V' is encountered, and C be any directed cycle that visits all vertices in G', we have

$$\begin{aligned} \mathrm{cover}(G') &\leq \sum_{(u,v)\in C} \mathrm{E}[T_{u,v}] \\ &\leq |C| \cdot 2|E'| \end{aligned}$$

Letting C be a traversal of some spanning tree of G', we conclude that $\mathrm{cover}(G') < 4 \cdot |E'| \cdot |V'|$. Thus, with probability at least $1/2$, a random walk of length $8 \cdot |E'| \cdot |V'|$ starting at s visits all vertices of G'.

1.7 Finding minimum cuts in graphs or, random is better than arbitrary

The problem considered here is to find the minimum cut in a graph. The randomized algorithm that follows is simpler than the traditional flow-based algorithms, and lends itself to parallel implementation (omitted here).

Algorithm: On input a graph $G = (V, E)$, with $n = |V|$, the algorithm makes $n - 2$ random edge *contraction* steps: In each step one selects *uniformly* an edge of the current multi-graph and contracts the two endpoints into one vertex, allowing parallel edges but dropping self-loops that may be created. That is, if (u, v) is the contracted edge of the current graph G' then we replace vertices u and v by a new vertex x, and replace edges of the form (w, v) (resp., (w, u)), where $w \notin \{u, v\}$, by a similar number of edges (w, x). When these $n - 2$ contraction steps are completed, we are left with a multi-graph on two vertices, and just output the number of parallel edges.

Analysis: Suppose that G has a minimum cut $C \subset E$. Then, the probability that no edge of C is contracted in the first step is $\frac{|E|-|C|}{|E|} \geq 1 - \frac{2}{n}$ (where $|C| \leq 2|E|/n$ because the minimum cut cannot be bigger than the average degree). The question is what happens in subsequent steps. A key observation is that $|C|$ is a lower bound on the average degree of any multi-graph obtained from G by any sequence of edge contractions. Thus, the probability that the $(n - 2)$-step contraction process leaves C intact is at least

$$\prod_{i=1}^{n-2} \left(1 - \frac{2}{n-(i-1)}\right) = \prod_{i=1}^{n-2} \frac{n-1-i}{n+1-i} = \frac{2}{n \cdot (n-1)}$$

Thus, repeating the above algorithm for a quadratic number of times we obtain the minimum cut, with probability at least, say, $2/3$. We comment that it follows that the number of minimum cuts

[2] A hand-waving argument follows: Consider a very long walk, starting at u, and returning to u many times. Note that each directed edge appears on this walk for about the same number of times. Partition the walks into segments so that each segment ends with a move from vertex v to vertex u. Then, the number of segments is about a $1/2|E'|$ fraction of the length of the walk, and so the average length of a segment is $2|E'|$. Note that each segment constitutes a walk starting at u and passing through v (possiblly several times) before returning to u. Thus, the average length of segments in the big walk upper bounds the expected length of random walks from u to v.

is at most $(n-1)n/2$ (becuase each such cut is generated by the above algorithm with probability at least $2/(n-1)n$).

Comment: Observe that if the random choices in the above algorithm are replaced by arbitrary choices then the output gives little indication towards the minimum cut in G. That is, an algorithm that makes $n-2$ arbitrary edge-contraction steps provides no useful information, whereas the above algorithm that picks these steps at random is useful. In general, making random choices may be better than making arbitrary choices. This lesson is important because many algorithms are presented in a non-fully specified manner, allowing some choices to be made arbitrarily (in which case these choices are typically made in a way most convenient for implementation). It is important to bear in mind that, in some cases, replacing an arbitrary choice by a random one may yield improved performance.

2 Randomness in Complexity Theory

In this section we demonstrate the power of randomized reductions (rather than randomized algorithms discussed in the previous section). Again, we focus on simple examples, and avoid the central role of randomness in the context of proof systems. For a survey of probabilistic proof systems, the interested reader is referred to [11, Chap. 2].

2.1 Reducing (Approximate) Counting to Deciding or, the Random Sieve

We consider the class $\#\mathcal{P}$ of functions that count the number of NP-witnesses (w.r.t an NP-relation). That is, $f \in \#\mathcal{P}$ if for some NP-relation, R, it holds that $f(x) = |\{y : (x,y) \in R\}|$, for every $x \in \{0,1\}^*$. We will show that such f can be approximated in probabilistic polynomial-time given oracle to an NP-complete set. The (randomized Cook) reduction uses any efficient family of Universal_2 Hash functions[3], as well as the following lemma.

Lemma 3 (Leftover Hash Lemma [30, 6, 12]):[4] *Let $H_{m,k}$ be a family of* Universal_2 *Hash functions mapping $\{0,1\}^m$ to $\{0,1\}^k$, and let $\epsilon > 0$. Let $S \subseteq \{0,1\}^m$ be arbitrary provided that $|S| \geq \epsilon^{-3} \cdot 2^k$. Then, for all but at most an ϵ fraction of the h's in $H_{m,k}$, it holds that*

$$|\{e \in S : h(e) = 0^k\}| = (1 \pm \epsilon) \cdot \frac{|S|}{2^k}$$

Proof: For a uniformly selected $h \in H_{m,k}$, the random variables $\{h(e)\}_{e \in S}$ are pairwise independent and uniformly distributed over $\{0,1\}^k$. On top of these $h(e)$'s, we define 0-1 random variables, denoted ζ_e's, so that $\zeta_e = 1$ if $h(e) = 0^k$. Then $\text{Exp}[\zeta_e] = 2^{-k}$ and we need to show that the sum

[3] A family of functions mapping $\{0,1\}^m$ to $\{0,1\}^k$ is called Universal_2 if for a uniformly selected h in the family, the random variables $\{h(e)\}_{e \in \{0,1\}^m}$ are pairwise independent and uniformly distributed over $\{0,1\}^k$. An efficient family is required to have algorithms for selecting and evaluating functions. A popular example is the family of all affine transformations from $\{0,1\}^m$ to $\{0,1\}^k$.

[4] A stronger statement of the lemma, supported by essentially the same proof, refers to an arbitrary random variable X over $\{0,1\}^m$ satisfying $\Pr[X = x] \leq \epsilon^3 \cdot 2^{-k}$, for every x. The lemma was discovered independently in [6, 12], yet it is an extension of the ideas underlying [30]. The lemma's name was coined in [13].

$\sum_{e\in S}\zeta_e$ is concentrated around $|S|/2^k$. Using Chebyshev's Inequality and the fact that the ζ_e's are pairwise independent, we get

$$\begin{aligned}\Pr\left[\left|\sum_{e\in S}\zeta_e - \frac{|S|}{2^k}\right| > \frac{\epsilon\cdot|S|}{2^k}\right] &< \frac{\mathrm{Var}[\sum_{e\in S}\zeta_e]}{(\epsilon|S|/2^k)^2} \\ &< \frac{|S|/2^k}{\epsilon^2\cdot(|S|/2^k)^2} \le \epsilon\end{aligned}$$

(Pairwise independence is used in deriving $\mathrm{Var}[\sum_{e\in S}\zeta_e] = \sum_{e\in S}\mathrm{Var}[\zeta_e] < |S|\cdot 2^{-k}$.) ■

Reduction: On input $x\in\{0,1\}^n$, the probabilistic polynomial-time oracle machine (for approximating f) sets m to be the length of NP-witness w.r.t the guaranteed R. For every $k=0,1,...,m+2$ it performs the following experiment n times.

1. Uniformly select $h\in H_{m,k}$, and construct (via Cook's reduction) a CNF formula φ so that φ is satisfiable if and only if there exists a string $y\in\{0,1\}^m$ so that $(x,y)\in R$ and $h(y)=0^k$.
2. Query the oracle whether φ is satisfiable.

Once all these experiments are completed, the machine determines the smallest non-negative integer k (possibly zero) so that the oracle has answered NO at least $n/2$ times, and outputs 2^k.

Analysis: We analyze the performance of the above machine when it is given oracle access to SAT. Clearly, if $S_x \stackrel{\rm def}{=} \{y:(x,y)\in R\}$ has cardinality N then the probability that the machine outputs a number $k\ge L\stackrel{\rm def}{=}\lceil\log_2(4N)\rceil$ is exponentially vanishing (because the probability that a uniformly selected $h\in H_{m,L}$ maps some element of S_x to 0^L is at most $1/4$, and so in each iteration with value of $k\ge L$, with probability at least $3/4$, the oracle says NO). On the other hand, using the above lemma, if $N\stackrel{\rm def}{=}|S_x|\ge 2^{k+2}$ then for a uniformly selected $h\in H_{m,k}$ with probability at least $3/4$ there exists $y\in S_x$ so that $h(y)=0^k$. Thus, with overwhelmingly high probability, the output of the oracle machine is at least $\log_2(N/4)$. We conclude that approximating f up-to a factor of 4 is reducible in probabilistic polynomial-time to $\mathcal{NP}$. Higher accuracy – that is, approximation factor of $1+\frac{1}{p(n)}$, for any fixed positive polynomial p – can be obtained by considering the "direct product function" $F_p(x)\stackrel{\rm def}{=}(f(x))^{p(|x|)}$ that counts the number of NP-witnesses w.r.t the NP-relation R_p defined by

$$R_p \stackrel{\rm def}{=} \{(x,y_1,...,y_{p(|x|)}) : \forall i\ (x,y_i)\in R\}$$

A related reduction may be used to reduced SAT (or even "approximating $\#\mathcal{P}$") to unique-SAT. By the latter, we mean the promise problem in which the YES-instances are CNF formula having a unique satisfying assignment, and the NO-instances are CNF formula having no satisfying assignment. All that is needed is to notice that in the above reduction, for $k=(\log_2 N)\pm 2$, the reduction produces CNF formula that are typically (i.e., w.p. at least $3/4$) either not satisfiable or have few (say up-to 8) satisfying assignments. Thus, we augment Step 1 as follows. Having produced φ, as above, we produce 8 new formulae, $\psi_1,...,\psi_8$, so that ψ_i asserts that φ has at least i different satisfying assignments (e.g., $\psi_i(y_1,...,y_i)=\bigwedge_j\varphi(y_j)\wedge\bigwedge_{1\le j<j'\le i}(y_j<y_{j'})$). We refer each of these ψ_i to the oracle and use YES as answer if the oracle has answered YES on any of the ψ_i (as this may happen only if φ is indeed satisfiable). Thus, whenever φ has few satisfying assignments, YES will be returned.

2.2 Two-sided error versus one-sided error

We consider the extension of the classes $\mathcal{RP}$ and $\mathcal{BPP}$ to promise problems and show that $\mathcal{BPP} = \mathcal{RP}^{\mathcal{RP}}$ (in the extended sense). It is evident that $\mathcal{RP}^{\mathcal{RP}} \subseteq \mathcal{BPP}^{\mathcal{BPP}} = \mathcal{BPP}$ (where the last equality utilizes standard "error reduction"). So we focus on the other direction, considering an arbitrary BPP-problem with a characteristic function χ (which may be only partially defined over $\{0,1\}^*$). Let R be an NP-relation and p be a polynomial, such that for every x on which χ is defined it holds that

$$|\{y \in \{0,1\}^{p(|x|)} : R(x,y) \neq \chi(x)\}| \;<\; \frac{2^{p(|x|)}}{3p(|x|)}$$

where $R(x,y)=1$ if $(x,y)\in R$ and $R(x,y)=0$ otherwise. We show a randomized one-sided error (Karp) reduction of χ to (the promise problem extension of) co$\mathcal{RP}$.

Reduction: On input $x \in \{0,1\}^n$, the randomized polynomial-time mapping uniformly selects $s_1,...,s_m \in \{0,1\}^m$, and outputs the pair $(x,\overline{s})$, where $m = p(|x|)$ and $\overline{s} = (s_1,...,s_m)$.

We define the following co$\mathcal{RP}$ promise problem, denoted Π. The YES-instances, denoted $\Pi_{\rm yes}$, are pairs $(x,\overline{s})$ so that for every $r \in \{0,1\}^m$ there exists an i so that $R(x, r\oplus s_i) = 1$. The NO-instances, denoted $\Pi_{\rm no}$, are pairs $(x,\overline{s})$ so that for at least half of the possible $r \in \{0,1\}^m$, it holds that $R(x,r\oplus s_i)=0$ for every i. Clearly, Π is indeed a co$\mathcal{RP}$ promise problem (via an algorithm that uniformly selects r, and computes $R(x,r\oplus s_i)$ for all i's).

Analysis: We claim that the above randomized mapping reduces χ to Π. Suppose first that $\chi(x)=0$. Then, for every possible choice of $s_1,...,s_m \in \{0,1\}^m$, the fraction of r's for which $R(x,r\oplus s_i)=1$ holds for some i is at most $m\cdot\frac{1}{3m}=\frac{1}{3}$. Thus, the reduction always maps such an x to a NO-instance (i.e., an element of $\Pi_{\rm no}$). On the other hand, we will show shortly that in case $\chi(x)=1$, with probability at least $1/2$ the reduction maps x to a YES-instance. Thus, the above reduction has one-sided error and indeed reduces χ to Π (which, as observed above, is in co$\mathcal{RP}$). It is left to analyze the probability that the reduction fails in case $\chi(x)=1$. That is,

$$\begin{aligned}
{\rm Pr}_{\overline{s}}[(x,\overline{s}) \notin \Pi_{\rm yes}] &= {\rm Pr}_{s_1,\ldots,s_m}[\exists r \in \{0,1\}^m \text{ s.t. } (\forall i)\ R(x,r\oplus s_i)=0] \\
&\leq \sum_{r\in\{0,1\}^m} {\rm Pr}_{s_1,\ldots,s_m}[(\forall i)\ R(x,r\oplus s_i)=0] \\
&\leq 2^m \cdot \left(\frac{1}{3m}\right)^m \;\ll\; \frac{1}{2}
\end{aligned}$$

Comment: The traditional presentation uses the above reduction to show that $\mathcal{BPP}$ is in the Polynomial-Time Hierarchy. One defines the polynomial-time predicate $\varphi(x,\overline{s},r) \stackrel{\rm def}{=} \bigvee_{i=1}^m (R(x,s_i\oplus r)=1)$, and observes that

$$\begin{aligned}
\chi(x)=1 &\;\Rightarrow\; \exists\overline{s}\,\forall r\ \ \varphi(x,\overline{s},r) \\
\chi(x)=0 &\;\Rightarrow\; \forall\overline{s}\,\exists r\ \ \neg\varphi(x,\overline{s},r)
\end{aligned}$$

2.3 The permanent: Worst-Case versus Average Case or, the self-correction paradigm

We consider the problem of computing the permanent of a matrix.[5] This problem is known to be $\#\mathcal{P}$-complete even in case the matrix has only 0-1 entries. Here we consider the problem of computing the permanent over sufficiently large finite fields (i.e., the field size is larger than the dimension). We show that the (worst-case) problem can be reduced to solving the problem on random (or typical) instances.

Reduction: On input an n-by-n matrix, M, over F (s.t., $|F| > n+1$), the probabilistic polynomial-time oracle machine (i.e., the reduction) proceeds as follows.

1. Uniformly select an n-by-n matrix, R, over F.
2. For $i = 1, ..., n+1$, obtain from the oracle the value, denoted v_i, of the permanent of the matrix $M + iR$.
3. Obtain by interpolation, the value of the degree n univariant polynomial, p, satisfying $p(i) = v_i$ (for $i = 1, ..., n+1$).
4. Output $p(0)$.

The key observation, underlying the above reduction, is that, for fixed M and R, the permanent of $M + iR$ is a degree n polynomial in the variable i.

Analysis: We consider the performance of the above reduction assuming it is given access to an oracle that answers correctly on all but at most an $1/3(n+1)$ fraction of the instances. We will show that in such a case, on any input, the reduction answers correctly with probability at least 2/3. Observe that, for each fixed M and $i \neq 0$, the matrix $M + iR$ is uniformly distributed over the instance space. Thus, the probability that the oracle returns an incorrect answer on any of the $n+1$ queries is at most 1/3. But otherwise, having the permanent of $M + iR$ for every $i = 1, .., n+1$, we obtain the permanent of the formal matrix $M + xR$ (which is a polynomial of degree n in $x \in$ F), and thus the permanent of M (when substituting $x = 0$).

Comments: As seen above, the reduction of a problem to random instances of itself allows to reduce its "worst" instances to its average (or typical) cases, and thus means that the problem does not really have "worst" (or "pathological") instances: The problem's complexity, in case the problem is hard, must stem from typical (or random) instances. Viewed from the other side (i.e., of feasibility), such a reduction allows to *self-correct* a (possibly efficient) procedure that is correct on a large majority of the instances, and obtain a randomized procedure that is correct on every instance. Thus, as any reduction, a reduction to random instances is open to interpretation: For example, Ajtai's reduction of approximating shortest vectors in integer lattices to such random instances [1], is commonly viewed as a demonstration of average-case hardness based on worst-case hardness, but it may be also viewed as a self-corrector for (possibly efficient) programs that find short vectors in a certain class of integer lattices.

[5] The permanent of an n-by-n matrix $A = (a_{i,j})$ is the sum, taken over all permutations π of $[n]$, of the products $\prod_{i=1}^{n} a_{i,\pi(i)}$.

3 Randomness in Distributed Computing

As much as randomness is a powerful tool in the design of algorithms and reductions, its power in the distributed context is even more striking. In particular, randomized distributed protocols are known to beat some impossibility results and lower bound that refer to deterministic protocols. Various examples are given in [8, 23, 4, 19, 24].

As a warm-up consider the problem of electing a leader among a set of n *identical* processes. Clearly, there is no deterministic procedure to elect such a leader (even when all processes are guaranteed to be non-faulty), because there is no way to "*deterministically* break the symmetry" among the processors. However, a simple randomized procedure will do the job: Let each processor toss, independently of all other processors, a coin with bias $1/n$ towards 1, and announce its coin-flip to all processors. If a single processor sends 1 then it is elected leader, otherwise the process is repeated. In general, randomness can be used to "break symmetry" in a variety of distributed settings. Other uses of randomness in such settings include avoiding "pathological" configurations (see Section 3.2), and making the actions of non-faulty processors unpredictable to malicious ones (i.e., Byzantine faults; see Section 3.3). We start with a much simpler problem.

3.1 Testing String Equality or, randomized fingerprints

The problem considered here is to decide whether or not two strings, each held by a different party, are identical. The aim is to devise a protocol for this problem using low communication complexity. We present three such protocols.

Protocol 1: Party A holds $x \in \{0,1\}^n$, whereas party B holds $y \in \{0,1\}^n$. Here we view x and y as non-negative integers in $\{0,1,...,2^n-1\}$. In the protocol, party A uniformly selects $i \in \{1,...,n\}$, finds the i^{th} prime, denoted p_i, and sends the pair $(i, x \bmod p_i)$ to B. Party B recovers p_i and accepts if and only if $y \bmod p_i$ equals the value $x \bmod p_i$ (received from A).

Clearly, if $x=y$ then B always accepts. On the other hand, using the Chinese Reminder Theorem, we know that if $x \neq y$, then $x \not\equiv y \pmod{p_i}$ for at least $n/2$ of the p_i's (since otherwise $x \equiv y \pmod{\prod_{i\in I} p_i}$, for $|I| \geq n/2$, and $x=y$ follows (because $x,y < 2^n < \prod_{i\in I} p_i$)). Thus, B will reject with probability at least 1/2. The number of bits sent is $\log_2 n + \log_2 p_n = O(\log n)$.

Protocol 2: Again, party A holds $x \in \{0,1\}^n$, whereas party B holds $y \in \{0,1\}^n$. Here we use a small-bias probability space $S \subset \{0,1\}^n$, with bias 1/6 and $|S| = \text{poly}(n)$ (see [26]). By definition, for every non-zero string $z \in \{0,1\}^n$, with probability at least 1/3 a uniformly chosen $r \in S$ has inner product mod 2 with z equal to 1. In the protocol, party A uniformly selects $r \in S$, computes the inner product mod 2 of x and r, and sends the result along with the index of r (in S) to B. Party B retrieves r, computes the inner product mod 2 of y and r, and accepts if it matches the bit received.

Clearly, if $x=y$ then B always accepts. On the other hand, by the above, if $x \neq y$ then the inner products of x and y with a uniformly chosen $r \in S$ differ with probability at least 1/3 (hint: consider $z = x \oplus y$). The number of bits sent is $1 + \log_2 |S| = O(\log n)$.

Protocol 3: The inputs are as above, but here we use a different tool: An error-correcting code, denoted $E : \{0,1\}^n \to \{0,1\}^m$, with $m = O(n)$ and distance $\Omega(n)$ (cf., [14]). In the protocol, party A computes the codeword $E(x)$, uniformly selects $i \in \{1,...,m\}$, and sends i along with the

i^{th} bit of $E(x)$ to Party B. The latter computes the codeword $E(y)$ and accepts if its i^{th} bit matches the bit received.

Clearly, if $x = y$ then B always accepts. On the other hand, if $x \neq y$ then $E(x)$ and $E(y)$ differ on a constant fraction of the bit positions, and so B will reject with constant probability. The number of bits sent is $1 + \log_2 m = O(1) + \log_2 n$.

3.2 Routing in networks or, avoiding pathological configurations

The problem considered here is to allow parallel routing of messages in a network in which processors have relatively few immediate neighbors (i.e., processors connected to them by a direct link). In many such networks, *routing to random destinations* can be done quite efficiently (i.e., fast even assuming that each processor can only deliver a single message at a time, and without coordination among the processors). Off course, we are interested in routing messages to "non-random" destinations; that is, to destinations that are imposed upon us by some high-level application. Still the above fact (regarding routing to random destinations) becomes relevant, via the following two phase *randomized routing* strategy: Suppose that processor i wishes to deliver a message to processor d_i, where the d_i's consist of an arbitrary a permutation of the processor names $[n] \stackrel{\text{def}}{=} \{1, ..., n\}$. Then, processor i selects a random intermediate processor, $r_i \in [n]$, and sends its message to processor r_i with a request to forward it to processor d_i. (The r_i's are not likely to be distinct!) Thus, the routing is in two phases:

1. The message of processor i, denoted m_i, is delivered to r_i.
2. Message m_i is delivered from r_i to d_i.

By our hypothesis, Phase 1 can be completed fast with high probability. It is appealing to say that, by symmetry, the same should hold also for Phase 2. This is not known to be generically true, but has been proved to be so for a wide class of networks (cf., [21, Sec. 3.4]). Specifically, if one changes the model a little, allowing and measuring edge congestion, then bounds on congestion in Phase 1 apply also to Phase 2.

3.3 Byzantine Agreement or, take actions the adversary cannot predict

The problem considered here is to allow non-faulty processors to agree on a common value, in presence of Byzantine (malicious) faulty processors. Specifically, it is required that (1) the non-faulty processors must terminate with the same output value, and (2) in case their input values are the same this should also be their output value. We may consider, without loss of generality, the problem of agreeing on a Boolean value. The primary parameters are the total number of processors, denoted n, and a bound on the number of faulty processors, t. We assume a synchronous model of point-to-point communication.

Protocol: We use auxiliary (threshold) parameters L, H, D so that $L > \frac{n}{2} + t$, $H \geq L + t$ and $H + t \leq D \leq n - t$ (which is feasible for $t < n/8$). The protocol utilizes a *global coin* (which may be implemented in various ways). It is postulated that, for each flipping of this coin, each of the two possible outcomes occurs with probability at least $p > 0$ ($p = 0.1$ will do, whereas $p = 0.5$ corresponds to an unbiased coin).

Following is the program to processor $i \in [n] \stackrel{\text{def}}{=} \{1, ..., n\}$. On input $b_i \in \{0,1\}$, the processor sets its (initial) vote, denoted $\texttt{vote}_i$, to b_i. The processor repeats the following steps $r+1$ times, where r is the iteration in which it decides (see below):

1. Send $\texttt{vote}_i$ to each processor.

2. Receive votes from all processors, including itself. (In case no message is received from processor j, use the value last received from it, and if no value was ever received use value 0.) Let $\texttt{cnt}_i$ denote the number of votes in favor of 1. If $\texttt{cnt}_i > n/2$ set $\texttt{maj}_i = 1$ and $\texttt{tally}_i = \texttt{cnt}_i$, otherwise set $\texttt{maj}_i = 0$ and $\texttt{tally}_i = n - \texttt{cnt}_i$.

3. Let $C \in \{L, H\}$ be the value of the global coin, for the current round (in each round the global coin is flipped anew).

4. If $\texttt{tally}_i \geq C$ then set $\texttt{vote}_i = \texttt{maj}_i$ else set $\texttt{vote}_i = 0$.

5. If $\texttt{tally}_i \geq D$ then *decide* $\texttt{vote}_i$, and proceed for a single additional iteration (skipping this step in the next iteration).

   (Actually, as shown below, if the processor were to decide again in the next iteration its decision would have been identical.)

Analysis: Let G denote the set of non-faulty (or good) processors. The following observation regarding members of G is extensively used: In each iteration, $|\texttt{cnt}_i - \texttt{cnt}_j| \leq t$, for every $i, j \in G$. Thus, if $\texttt{tally}_i \geq L > n/2 + t$ for some $i \in G$ then $\texttt{maj}_j = \texttt{maj}_i$ for all $j \in G$. Similarly, if $\texttt{tally}_i \geq D$ (resp., $\texttt{tally}_i \geq H$) for some $i \in G$ then $\texttt{tally}_j \geq H$ (resp., $\texttt{tally}_j \geq L$) for all $j \in G$. Using these facts it follows that

1. *If all good processors enter some round with identical votes then they all decide by the end of the current round, and their decision equals this vote.* This follows since (at this round) this identical vote would have support of at least $|G| \geq n - t \geq D$. (As a special case, we conclude that the second requirement of Byzantine Agreement holds.)

2. *If at some round a good processor decides v then by the end of the next round all good processors decide v.* Suppose that $i \in G$ decides v in the current round. Then, $\texttt{tally}_i \geq D$, and for each $j \in G$ it follows that $\texttt{tally}_j \geq H$ and so at Step 4 $\texttt{vote}_j = \texttt{maj}_j = v$. Using the previous fact, the current one follows. (As a special case, we conclude that the first requirement of Byzantine Agreement holds.)

3. *If at some round $\texttt{tally}_i \geq H$ holds for some $i \in G$ then with constant probability all good processors enter the next round with vote equal to $\texttt{maj}_i$.* This follows since with constant probability the outcome of the global coin is L, in which case for every $j \in G$, $\texttt{tally}_j \geq L = C$ and so at Step 4 $\texttt{vote}_j = \texttt{maj}_j = \texttt{maj}_i$.

4. *If at some round $\texttt{tally}_i < H$ holds for all $i \in G$ then with constant probability all good processors enter the next round with vote 0.* This follows since with constant probability the outcome of the global coin is H.

Thus, the above protocol terminates in *constant expected number of rounds*, and the output always satisfies the agreement requirements. This remain valid even if we use a global coin the outcome of which may be viewed differently by different processors, as long as for each of the two possible values, with probability at least $p > 0$, all non-faulty processors view the outcome as equal to that value. We comment that such a global coin can be easily implemented in case $t = O(\sqrt{n})$, by letting each processor toss a local coin, announce the outcome, and view the outcome of the global coin to be the majority vote it has received (which, with constant probability, will be identical at all good processors). We note that $t+1$ is a lower bound on the number of rounds in any correct *deterministic* protocol. Furthermore, the above protocol can be adapted to the asynchronous model, whereas there exist no correct *deterministic* protocol for the latter model (even for $t = 1$).

4 Bibliographic Notes

Section 1.1 (*approximating the number of DNF satisfying assignments*) is based on [17], Section 1.2 (*finding perfect matching*) is based on [25], and Section 1.3 (*testing polynomial identities*) is based on [29, 35]. The *Randomized Rounding* technique was introduced in [28], and the *MaxSAT application* described in Section 1.4 is due to [9]. The *primality testing* algorithm described in Section 1.5 is folklore attributed to several people; I heard it attributed to M. Blum. Section 1.6 (*random walk algorithm for testing connectivity*) is based on [2], and Section 1.7 (*the randomized min-cut algorithm*) is based on [15].

Section 2.1 (*reduction of approximate counting to deciding* and of *SAT to uniqueSAT*) is based on [30, 31] and [34], but the presentation in these sources is quite different. The reduction of Section 2.2 is based on [20], where it was used to show (independently of [30]) that $\mathcal{BPP} \in \mathcal{PH}$; the current presentation is due to Fortnow (priv. comm. 1997, see [3]). Section 2.3 (*self-corrector for the permanent*) is based on [22].

Protocol 1 for *string equality* (in Section 3.1) is commonly attributed to M. Rabin and A. Yao, Protocol 2 is due to [26, Sec. 9], and Protocol 3 is due to E. Kushilevitz (priv. comm. 1998). Section 3.2 (*randomized routing*) is based on [32, 33], and Section 3.3 (*randomized Byzantine Agreement*) is based on [7, 27].

References

[1] M. Ajtai. Generating Hard Instances of Lattice Problems. In *28th ACM Symposium on the Theory of Computing*, pages 99–108, 1996.

[2] R. Aleliunas, R.M. Karp, R.J. Lipton, L. Lovász and C. Rackoff. Random walks, universal traversal sequences, and the complexity of maze problems. In *20th IEEE Symposium on Foundations of Computer Science*, pages 218–223, 1979.

[3] A.E. Andreev, A.E.F. Clementi, J.D.P. Rolin and L. Trevisan, Weak Random Sources, Hitting Sets, and BPP Simulations. To appear in *SIAM Journal on Computing*. Preliminary version in *38th IEEE Symposium on Foundations of Computer Science*, pages 264–272, 1997.

[4] H. Attiya and J. Welch. *Distributed Computing: Fundamentals, Simulations and Advanced Topics*. McGraw-Hill Publishing Company, London, 1998.

[5] E. Bach and J. Shallit. *Algorithmic Number Theory* (Volume I: Efficient Algorithms). MIT Press, 1996.

[6] C.H. Bennett, G. Brassard and J.M. Robert. Privacy Amplification by Public Discussion. *SIAM Journal on Computing*, Vol. 17, pages 210–229, 1988. Preliminary version in *Crypto85*, Springer-Verlag Lecture Notes in Computer Science (Vol. 218), pages 468–476 (titled "How to Reduce your Enemy's Information").

[7] M. Ben-Or. Another advantage of free choice: Completely Asynchronous Byzantine Agreement. In *2nd ACM Symposium on Principles of Distributed Computing*, pages 27–30, 1983.

[8] B. Chor and C. Dwork. Randomization in Byznatine Agreement. *Advances in Computing Research: A Research Annual*, Vol. 5 (Randomness and Computation, S. Micali, ed.), pages 443–497, 1989.

[9] M. Goemans and D. Williamson. New 3/4-approximation algorithms for the maximum satisfiablity problem. *SIAM Journal on Discrete Mathematics*, Vol. 7, No. 4, pages 656–666, 1994.

[10] M. Goemans and D. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM*, Vol. 42, No. 6, 1995, pages 1115–1145.

[11] O. Goldreich. *Modern Cryptography, Probabilistic Proofs and Pseudorandomness. Algorithms and Combinatorics series* (*Vol. 17*), *Springer*, 1999.

[12] R. Impagliazzo, L.A. Levin and M. Luby. Pseudorandom Generation from One-Way Functions. In *21st ACM Symposium on the Theory of Computing*, pages 12–24, 1989.

[13] R. Impagliazzo and D. Zuckerman. How to Recycle Random Bits. In *30th IEEE Symposium on Foundations of Computer Science*, 1989, pages 248–253.

[14] J. Justesen. A class of constructive asymptotically good alegbraic codes. *IEEE Trans. Inform. Theory*, Vol. 18, pages 652–656, 1972.

[15] D.R. Karger. Global Min-cuts in RNC, and Other Ramifications of a Simple Min-Cut Algorithm. In *4th SODA*, pages 21–30, 1993.

[16] H. Karloff and U. Zwick. A 7/8-approximation algorithm for MAX 3SAT? In *38th IEEE Symposium on Foundations of Computer Science*, 1997, pages 406–415.

[17] R.M. Karp and M. Luby. Monte-Carlo algorithms for enumeration and reliability problems. In *24th IEEE Symposium on Foundations of Computer Science*, pages 56-64, 1983. See [18].

[18] R.M. Karp, M. Luby and N. Madras. Monte-Carlo approximation algorithms for enumeration problems. *Journal of Algorithms*, Vol. 10, pages 429–448, 1989.

[19] E. Kushilevitz and N. Nisan. *Communication Complexity*, Cambridge University Press, 1996.

[20] C. Lautemann. BPP and the Polynomial Hierarchy. *Information Processing Letters*, 17, pages 215–217, 1983.

[21] F.T. Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes.* Morgan Kaufmann Publishers, San Mateo, CA, 1992.

[22] R.J. Lipton. New Directions in Testing. In *Proc. of DIMACS Workshop on Distr. Comp. and Crypto.*, pages 191–202, 1991.

[23] N. Lynch. *Distributed Algorithms.* Morgan Kaufmann Publishers, San Mateo, CA, 1996.

[24] R. Motwani and P. Raghavan. *Randomized Algorithms*, Cambridge University Press, 1995.

[25] K. Mulmuley and U.V. Vazirani and V.V. Vazirani. Matching is as Easy as Matrix inversion. *Combinatorica*, Vol. 7, pages 105–113, 1987.

[26] J. Naor and M. Naor. Small-bias Probability Spaces: Efficient Constructions and Applications. *SIAM J. on Computing*, Vol 22, 1993, pages 838–856.

[27] M.O. Rabin. Randomized Byznatine Agreement. In *24th IEEE Symposium on Foundations of Computer Science*, pages 403–409, 1983.

[28] P. Raghavan and C.D. Thompson. Randomized Rounding. *Combinatorica*, Vol. 7, pages 365–374, 1987.

[29] J. T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *Journal of the ACM*, Vol. 27, pages 701–717, 1980.

[30] M. Sipser. A Complexity Theoretic Approach to Randomness. In *15th ACM Symposium on the Theory of Computing*, pages 330–335, 1983.

[31] L. Stockmeyer. On Approximation Algorithms for #P. *SIAM Journal on Computing*, Vol. 14 (4), pages 849–861, 1985. Preliminary version in *15th ACM Symposium on the Theory of Computing*, pages 118–126, 1983.

[32] L.G. Valiant. A scheme for fast parallel communication. *SIAM Journal on Computing*, Vol. 11 (2), pages 350–361, 1982.

[33] L.G. Valiant and G.J. Brebner. Universal schemes for parallel communication. In *13th ACM Symposium on the Theory of Computing*, pages 263–277, 1981.

[34] L.G. Valiant and V.V. Vazirani. NP Is as Easy as Detecting Unique Solutions. *Theoretical Computer Science*, Vol. 47 (1), pages 85–93, 1986.

[35] R. Zippel. Probabilistic algorithms for sparse polynomials. *Proc. Int'l. Symp. on Symbolic and Algebraic Computation*, Springer-Verlag Lecture Notes in Computer Science (Vol. 72), pages 216–226, 1979.