On Randomization in Sequential and Distributed Algorithms

Rajiv Gupta
GE Corporate R&D
KW-C313, P.O. Box 8
Schenectady, NY 12301
gupta@crd.ge.com

Scott A. Smolka[1]
Dept. of Computer Science
SUNY at Stony Brook
Stony Brook, NY 11794
sas@cs.sunysb.edu

Shaji Bhaskar
Bell Northern Research
35 Davis Drive
Res. Triangle Pk, NC 27709
bhaskar@bnr.ca

September 8, 1993

I returned, and saw under the sun, that the race is not to the swift, nor the battle to the strong, neither yet bread to the wise, nor yet riches to men of understanding, nor yet favor to men of skill; but time and chance happeneth to them all.

Ecclesiastes (King James Version)

Chaos umpire sits,
And by decision more embroils the fray
By which he reigns: next him high arbiter
Chance governs all.

Paradise Lost, John Milton

[1]Research supported by the National Science Foundation under grants CCR–8704309, CCR–9120995, and CCR–9208585; and by the Air Force Office of Special Research under grant AFOSR F49620-93-1-0250DEF.

Contents

1 Introduction **4**

1.1 Probabilistic Techniques 7

1.2 Tradeoffs 14

2 Sequential Randomized Algorithms **15**

2.1 The Sock Selection Problem 16

2.2 Primality Testing 18

2.3 Networks without Large Hierarchies 26

2.4 Probabilistic Hashing 30

2.4.1 Perfect Hashing 31

2.4.2 Universal Hashing 34

2.4.3 Some Recent Results 36

2.5 The Nearest Neighbors Problem 38

2.6 Interactive Probabilistic Proofs 41

3 Distributed Randomized Algorithms **48**

3.1 The Dining Philosophers Problem 49

3.2 Communication Guard Scheduling 52

3.3 Leader Election 55

3.4 Message Routing 58

3.5 Byzantine Agreement 68

4 Additional Topics of Interest and Conclusions **74**

Abstract

Probabilistic, or randomized, algorithms are fast becoming as commonplace as conventional deterministic algorithms. This survey presents five techniques that have been widely used in the design of randomized algorithms. These techniques are illustrated using 12 randomized algorithms — both sequential and distributed — that span a wide range of applications, including: *primality testing* (a classical problem in number theory), *universal hashing* (choosing the hash function dynamically and at random), *interactive probabilistic proof systems* (a new method of program testing), *dining philosophers* (a classical problem in distributed computing), and *Byzantine agreement* (reaching agreement in the presence of malicious processors). Included with each algorithm is a discussion of its correctness and its computational complexity. Several related topics of interest are also addressed, including the theory of probabilistic automata, probabilistic analysis of conventional algorithms, deterministic amplification, and derandomization of randomized algorithms. Finally, a comprehensive annotated bibliography is given.

Categories and Subject Descriptors: I.1.2 [**Computing Methodologies**]: Algorithms; F.1.2 [**Computation by Abstract Devices**]: Modes of Computation - Probabilistic Computation; D.1 [**Software**]: Programming Techniques.

General Terms: Randomized Algorithms; Probabilistic Techniques; Analysis of Algorithms.

Additional Keywords and Phrases: Probabilistic Algorithms; Sequential and Distributed Algorithms; Computational Complexity; Randomized Quicksort; Primality Testing; Transitive Tournaments; Hashing; Perfect Hashing; Universal Hashing; Nearest Neighbors Problem; Interactive Probabilistic Proof Systems; Graph Isomorphism; Dining Philosophers Problem; CSP; Leader Election; Message Routing; Byzantine Agreement.

1 Introduction

We examine the field of *probabilistic algorithms*, that is, algorithms containing statements of the form:

$$x := \textit{outcome of tossing a fair coin}$$

Probabilistic algorithms typically toss coins in order to make multi-way decisions so, in general, the coins in question are n-sided. One of the goals of this survey is to illustrate the interesting and powerful effects coin tossing can have on the behavior of algorithms.

The action of tossing a coin is often implicit in a probabilistic algorithm and may take on various guises. Actions such as "randomly select an item x from a set S", or "randomly choose a process with which to communicate" are typical examples. Computationally, tossing a coin can be viewed as generating a random number between 1 and n. As such, the term *randomized algorithm* is often used in the literature as a synonym for probabilistic algorithm, and so it shall be here. An algorithm not having any coin tossing statements is said to be *deterministic*.

Randomized algorithms entered the computer science spotlight with the publication of Michael Rabin's seminal paper "Probabilistic Algorithms" [Rab76], although their existence can be traced back much further [Sha92a]. Rabin's paper presented surprisingly efficient randomized algorithms for two well-known problems, Nearest Neighbors—a problem in computational geometry, and Primality Testing—the problem of determining whether a given integer is divisible by any number other than itself and one. The probabilistic algorithm of Solovay and Strassen [SS77, SS78], also for primality testing, is another celebrated result in the field. A resurgence of interest in randomized algorithms occurred in the early 1980's with the discovery of the important role randomization can play in distributed computing, e.g., [FR80, LR81, BO83].

More recently, randomized algorithms have been the subject of an ACM Turing Award Lecture [Kar86], an ACM Distinguished Dissertation [Kil90], and of a number of surveys including [Wei78, Hop81, Wel83, Kro85, MSV85, Har87, Val87, BB88, Rag90, Kar90]. Our survey is closest in spirit to [Har87, Val87, BB88, Kar90] in its extensive coverage of both sequential and distributed randomized algorithms.

A distinguishing aspect of our survey is the classification we present in Section 1.1 of general techniques used in the design of randomized algorithms.[2] In Section 1.2, we then identify certain tradeoffs one may encounter in using these techniques. For example, the Primality Testing algorithm of [Rab76], which uses a technique we call "random search", outperforms all known deterministic algorithms for the problem, yet cannot, in general, guarantee absolutely that the answer produced is correct. We next present 12 randomized algorithms which we believe to be representative of the field; in the least, they collectively make use of the techniques that we have presented. Seven of these algorithms are sequential (Section 2) and five are distributed (Section 3). Finally, in Section 4, we spotlight several remaining issues in the field of randomized algorithms. A comprehensive annotated bibliography is included.

The intended audience is one with a basic background in algorithm design and analysis, but not necessarily familiar with the use of probabilistic techniques in algorithm construction. Familiarity with an imperative, sequential programming language such as Pascal is assumed, as the algorithms are presented in pseudo code with a distinctive Pascal flavor. The pseudo code makes use of control constructs such as `REPEAT UNTIL`, `FOR`, `WHILE`, and `IF THEN ELSE` for the sequential algorithms. For the distributed case, message passing constructs `SEND` and `RECEIVE`, as well as constructs for shared memory access, are added to the language. Their semantics are discussed in the introduction to Section 3.

As previously mentioned, we survey both sequential and distributed randomized algorithms. In the sequential case, we examine:

1. Sock Selection (*SockSel*)

2. Primality Testing (*PrimeTest*)

3. Networks without Large Hierarchies (*NetHierarchy*)

4. Perfect Hashing (*PerfHash*)

5. Universal Hashing (*UnivHash*)

6. Nearest Neighbors (*NearNeb*)

7. Graph Isomorphism Program Testing (*GI-Verify*)

[2]Karp's recent and excellent survey [Kar90] contains a slightly different classification.

The distributed randomized algorithms we consider are:

1. Dining Philosophers (*DinPhil*)
2. Communication Guard Scheduling (*CommGuard*)
3. Leader Election in a Ring (*LeadElect*)
4. Message Routing in a Network (*MsgRoute*)
5. Byzantine Agreement (*ByzAgree*)

For each algorithm we briefly define the basic problem and, when appropriate, the model of computation. We then explain why each algorithm is correct, and examine its computational complexity. Only a limited amount of probability theory is required to understand the correctness and complexity analyses, as our emphasis is on illustrating the techniques involved rather than on providing formal proofs.

To be able to cogently discuss the computational complexity of randomized algorithms, it is useful to first introduce several criteria for evaluating the performance of algorithms. Let $\mathcal{A}$ be a sequential algorithm with input I and output O. If $\mathcal{A}$ is deterministic, than an oft-used yardstick of $\mathcal{A}$'s performance is its *average running time*: the average time taken by $\mathcal{A}$ when, for input I of a given size, each possible instance of I is considered equally likely. That is, a uniform distribution on inputs is assumed.

For $\mathcal{A}$ a randomized algorithm, its running time on a *fixed* instance i of I may vary from execution to execution. Therefore, a more natural measure of performance is the *expected running time of* $\mathcal{A}$ *on a fixed instance* i *of* I: the mean time taken by $\mathcal{A}$ to solve instance i over and over.

In the randomized case, it is also useful to talk about the running time of $\mathcal{A}$ with high probability or the running time of $\mathcal{A}$ that occurs almost surely. Let $T(n)$ be a bound on the running time of $\mathcal{A}$ on inputs of size n. The running time of $\mathcal{A}$ is said to be $T(n)$ *with high probability* if $\mathcal{A}$ terminates in time $T(n)$ with probability at least $1 - 1/n$. The running time of $\mathcal{A}$ is said to be *almost surely* $T(n)$ if the algorithm terminates in time $T(n)$ with probability at least $1 - 1/2^{n^c}$, for some constant $c > 0$. In this survey, we have opted, whenever possible, to give the exact expression for the termination probability of a randomized algorithm instead of using qualitative terms such as "with high probability" or "almost surely."

These performance criteria can be applied to distributed algorithms as well. In this case, the quantities of interest include *communication complexity*, the total number and size of messages transmitted during the execution of a distributed algorithm; *queueing delay*, the total time spent by messages in message queues waiting to traverse in-use communication links; and the total number of accesses to shared variables/resources.

1.1 Probabilistic Techniques

We now discuss a number of fundamental techniques used by designers of randomized algorithms. This list is not meant to be exhaustive, and the techniques considered overlap in the sense that more than one may apply to a given randomized algorithm.

Input Randomization—Consider an algorithm $\mathcal{A}$ with input I and output O. As discussed above, if we fix the size of I, then the *average* running time of $\mathcal{A}$ refers to the average time taken by the algorithm when each possible instance of I is considered equally likely. That is, a uniform distribution on inputs is assumed. However, this may not be the actual input distribution to which the algorithm is exposed, making the average time complexity misleading. On the other hand, the *expected* running time of $\mathcal{A}$ on instance i of I refers to the mean time that the algorithm would take to solve instance i over and over.

Input randomization, i.e., rearranging or permuting the input to rid it of any existing patterns, ensures that for *all* inputs, the expected running time matches the average running time. This technique can be effective on problems that have algorithms with good average running time but poor worst-case running time due to some unfavorable input patterns.

A well-known example of this technique is *randomized quicksort* [Knu73]. Quicksort performs very well if the list of numbers to be sorted has a random order to it. However, quicksort degenerates to a comparison of every number with every other number if the input is already nearly sorted. One can think of randomized quicksort as a two step procedure. In the first step, the input sequence to be sorted is randomly permuted. The usual quicksort algorithm is then applied to the resulting sequence. Although the input randomization step can be performed in linear time, in practice, it is usually more efficient to simply pick the pivot element randomly. Our sock selection problem (*SockSel*) is another illustration of the power of input randomization.

An interesting application of input randomization is seen in some *probabilistic interactive proof-systems.* Here a *prover*, which supposedly can solve a hard problem, tries to convince a skeptical *verifier* of its prowess. For some problems, the verifier's task essentially consists of randomizing the input to the prover. This constitutes an attempt by the verifier to *confuse* the prover about the specific problem instance it is being asked to work on. In Section 2.6, we will see this use of input randomization in action for verifying the correctness of any program that purportedly solves the graph isomorphism problem. The proof system will have the additional feature that the prover can convince the verifier of its isomorphism-checking prowess without the verifier having to solve the graph isomorphism problem in any sense.

Input randomization is not restricted to sequential algorithms. Some randomized message routing algorithms, e.g., Valiant's algorithm for hypercubes [Val82] and Aleluinas's algorithm for b-way shuffle networks [Ale82], exhibit what may be termed *distributed input randomization.* In the message routing problem, a set of messages must be routed from source nodes to destination nodes in a network of computers. Moreover, the routing must be done in a distributed manner, i.e., without the help of a central arbiter. In the algorithms of [Val82, Ale82], each message is first sent to a randomly chosen intermediate node before being transmitted to its final destination. This randomization step eliminates "hot points" by distributing the traffic uniformly over the network. That is, it rids the input of any patterns that may exist between source nodes and destination nodes. In Section 3.4, we describe the message routing algorithms of Valiant and Aleluinas as well as a technique for multi-butterfly networks based on randomizing the interconnections between nodes.

Random Search—Random search is one of the most widely used probabilistic techniques. Many problems naturally involve searching a large space for an element having a desired property. If the property in question is easily verified and the elements possessing it are abundant, random search can be very effective.

Consider, for example, the problem of verifying the polynomial identity

$$f(X_1, X_2, \ldots, X_n) \;=\; 0.$$

If f is identically zero, then for all assignments of the X_i's it will evaluate to zero. However, if f is non-zero, then it can be shown that for any suitably constructed set of inputs, f will possess only a bounded number of zeros. In particular, if S is a set with more than $c \cdot deg(f)$ elements from the field generated by the coefficients of f, then f can have at most

$\frac{|S|^n}{c}$ zeros in S^n, for some constant c [Sch79]. Thus every trial evaluation of f on a randomly picked element of S^n will either prove the falsity of the identity, or yield credence to it with $1/c$ as the probability of being wrong. In k trials, therefore, one can either disprove the identity or come to believe it to be true with error probability less than $1/c^k$, a number that can be easily made smaller than the probability of a stray α-particle disrupting the computation. Randomized algorithms for testing polynomial identities and properties of systems of polynomials are discussed in detail in [Sch79, Zip79].

The probabilistic test for polynomial identities can also be used for determining whether a given undirected graph $G(V, E)$ has a *perfect matching*, i.e., a set of edges that covers each vertex exactly once. To see this, let $V = \{1, 2, \ldots n\}$ be the vertex set and associate variable x_{ij} with edge $e_{ij} \in E$. Define the $n \times n$ matrix $B = [b_{ij}]$ as follows. If there is no edge between vertex i and vertex j them $b_{ij} = 0$. Otherwise, $b_{ij} = x_{ij}$ if $i > j$ and $b_{ij} = -x_{ij}$ if $i < j$. Tutte [Tut47] proved that G has a perfect matching if and only if $\det(B)$ is not identically equal to zero. It was first observed by Lóvász [Lov79] that since $\det(B)$ is a polynomial in the x_{ij}'s, one can test for the validity of the polynomial identity $\det(B) = 0$ using the probabilistic technique described above. Lóvász, in the same paper, also describes a probabilistic method for determining the actual perfect matching, if one exists.

More efficient sequential methods for computing the perfect matching, though considerably more complicated, have been described in the literature. The beauty of the above scheme is its simplicity. In addition, it can be efficiently parallelized: the parallel implementation has the same resource requirements as those for evaluating a determinant, viz., $O(\log^2 n)$ time using $O(n^{3.5})$ processors [KUW86, MVV87]. This is significant as perfect matching is a fundamental problem that is not known to be in *NC*, the class of problems having parallel algorithms that run in polylog time while using a polynomially bounded number of processors. The randomized parallel algorithms of [KUW86, MVV87] do, however, place perfect matching in *Random NC*. One can also determine the actual perfect matching in parallel; see [KUW86, MVV87] for details.

Random search has also been used in algorithms on finite fields [Rab80b, Ber70]. It can be shown (e.g., see [Ber70]) that one in about every n polynomials in $Z_p[x]$ (the field of residues (mod p), where p is prime) is an irreducible monic polynomial of degree n. This result has been reproved, using a different technique, in [Rab80b]. Thus a plausible algorithm for finding an irreducible polynomial is to repeatedly pick one at random and test it for irreducibility. Since it takes $O(n^2(\log n)^2 \log\log n \log p)$ steps to test for irreducibility, one can find an irreducible polynomial in a reasonable amount of time. Algorithms for finding roots and irreducible factors based on random search are also given in [Rab80b].

There is a long history in number theory of using random search. For example, the result that 1 out of n polynomials of degree n over a finite field is irreducible, used above to derive a randomized algorithm for finding an irreducible polynomial, was published in 1856 by Richard Dedekind [J. Reine Angew. Math.]. Evidence exists that Gauss knew this result for the integers (mod p). Even earlier, Galois noted that a good way to select an irreducible polynomial over a finite field was by trial. Similarly, a paper by Pocklington [Proc. Cambridge Phil. Soc., 1917] on computing square roots mod p gives an estimate of the probability that a random search will succeed and take no more than cubic time.

In this survey, the algorithms we present for primality testing (*PrimeTest*) and perfect hashing (*PerfHash*) also use random search.

An implicit prerequisite for effective random search is the ability to randomly pick an element, more or less uniformly, from the space under consideration; e.g., the space of "witnesses" having a certain property, the space of spanning trees of a graph, or the space of degree-n polynomials. Determining the spaces for which this is possible is in itself an interesting problem. For example, it is not immediately clear how one would pick one spanning tree, uniformly at random, from the space of all possible spanning trees of a connected, undirected graph. This particular problem was solved by Broder [Bro89] who presented a randomized algorithm with an expected running time of $O(n \log n)$ per generated tree for almost all graphs. In the worst case, the algorithm requires $O(n^3)$ time per generated tree. Babai [Bab91] presents a randomized algorithm that constructs an efficient nearly uniform random generator for finite groups in a very general setting. Other interesting work on the random generation of combinatorial structures and sample spaces can be found in [JVV86, AGHP90].

Not all algorithms based on random search contain a verification step. If the search space is teeming with elements possessing the desired property, one can even dispense with checking the property. This is particularly useful if the property in question is not easily checked. For example, the problem *NetHierarchy* calls for constructing a network (a complete directed graph) on n nodes that does not contain a hierarchy on any subset of m nodes. A *hierarchy*, also known as a *transitive tournament* [ES74], is a graph in which for all nodes x, y and z, if the directed edges (x, y) and (y, z) exist then the edge (x, z) also exists. We will see that with high probability, *any* randomly selected network on n nodes will be devoid of large hierarchies as long as m is sufficiently "large".

Control Randomization—Consider a problem for which many algorithms exist, such as sorting. If each of these algorithms has good expected performance for some problem instances but poor worst-case performance, it is very risky to use any single one of them. This is especially true if the input probability distribution is not known. It may happen that the input is biased in such a way that it favors the bad cases. In such a situation, good average performance, which is typically computed assuming uniform input distribution, does not guarantee much. A way around this problem is to randomly pick one of the algorithms for each input instance. This strategy assumes, of course, that there is no significant correlation among the algorithms on what constitutes the bad inputs.

The randomized string matching algorithm of Karp and Rabin [KR87] exemplifies the use of control randomization. Here the problem is to determine if a given pattern of m symbols occurs in a text of length n. A naive algorithm would compare the pattern to the substrings at all possible text locations resulting in $O(nm)$ time complexity. Karp and Rabin do better by using a *fingerprinting function* that associates an integer with a text string using arithmetic calculations modulo a given prime number. They need only compare the fingerprint of the pattern to the fingerprints of all possible text locations. Control randomization comes into play as the fingerprinting function, actually the prime number underlying the fingerprinting function, is chosen at random.

Although the worst case running time of their algorithm is $O((n-m+1)m)$, like the naive algorithm, in practice one can expect it to run in time $O(n+m)$.[3] There is, however, a small probability ($\frac{1}{q}$, where q is the prime number used in the fingerprinting function) that the algorithm detects a false or *spurious* match. As a result, the algorithm incurs the additional overhead needed to check that detected matches are actually valid.

It is worth noting that a competitive alternative to the Karp-Rabin algorithm is the *deterministic* Knuth-Morris-Pratt algorithm [KMP77] which runs in time $O(n+m)$. The main novel idea behind this algorithm is the calculation of the *prefix function*, which for a given pattern encapsulates knowledge about how the pattern matches against shifts of itself.

As we will see, the problem of universal hashing (*UnivHash*) also admits a solution based on control randomization.

[3]The worst case behavior manifests itself in the presence of $O(n)$ occurrences of the pattern in the text. A more realistic, constant number of occurrences of the pattern within the text leads to the $O(n+m)$ running time cited above.

Random Sampling—Sometimes it is possible to ascertain, with high probability, certain properties of a set S from a randomly chosen subset of S. This technique is usually called "random sampling." As a simple example, consider a set S of n real numbers, and a randomly chosen subset R of S of size r [CS89]. R contains a lot of information about S. For example, if we let $S_>$ be the subset of numbers in S that are greater than the maximum value in R, then the expected size of $S_>$ is $O(n/r)$. Thus the size of $S_>$ diminishes as more and more values from S are sampled. Similarly, the expected size of the corresponding set $S_<$ is $O(n/r)$.

As another example of random sampling, consider the problem of numerically computing the integral

$$I = \int_a^b f(x)dx,$$

using *Monte Carlo integration* (not to be confused with Monte Carlo algorithms discussed in Section 1.2). Assuming that $f(x)$ is bounded by c, for $a \leq x \leq b$, this is accomplished by first randomly choosing a set of points that lie within the rectangle Ω given by

$$\Omega = \{(x, y) \mid a \leq x \leq b, 0 \leq y \leq c\},$$

Next, assuming that our random sample contains N points, determine the number N_H of these points (the "hit points") that lie beneath the curve. Then the desired integral I, which is equal to the area under the curve within the bounding rectangle Ω, is approximated by

$$I \approx c(b-a)\frac{N_H}{N},$$

i.e., the fraction of hit points in our random sample multiplied by the area of Ω (see Figure 1). The error in the computation depends on the number of points chosen. The larger the random sample, the less likely it is that the computed area differs significantly from the correct answer.

Note that for the computation of ordinary integrals with "well behaved" integrands, one is better off efficiency-wise and accuracy-wise using traditional numerical techniques such as the trapezoidal and Simpson's rules. Monte Carlo integration becomes attractive if the function fails to be regular which is often the case for multidimensional integrals [Rub81].

A more involved use of random sampling will be seen in Rabin's [Rab76] algorithm for the nearest neighbors problem (*NearNeb*). Here the distance δ separating the closest pair of points in a given set S, is deduced from a random subset of S containing $n^{\frac{2}{3}}$ of the points.

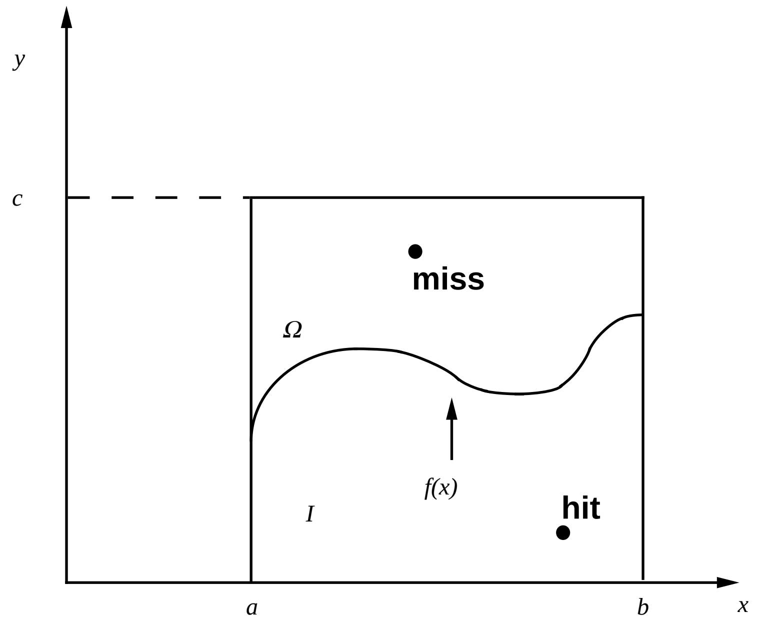

Figure 1: Graphical depiction of Monte Carlo integration from [Rub81]: Ω is the bounding rectangle; I, the desired integral, is the area under the curve; sample points above the curve are misses and those below are hits.

The expected running time of this algorithm is better than any known deterministic algorithm, under certain reasonable assumptions.

Symmetry Breaking—There are certain problems in distributed computing, in particular, problems in which processes must reach some sort of agreement, that do *not* have deterministic solutions. This dilemma surfaces when processes behave in a deterministic and identical fashion, without making any concessions toward the goal of reaching agreement. By introducing randomization into the behavior of the processes themselves, these patterns of identical or "symmetric" behavior can be broken, thereby leading to agreement.

For example consider the "narrow door" problem in which two people are trying to exit a room through a door that at most one person can squeeze through at a time. If both persons react to a collision at the door by backing up two feet and retrying after five seconds, then an initial collision could conceivably result in a never-ending succession of collisions, with neither party ever succeeding in leaving the room. A distributed algorithm that guarantees with probability 1 that someone will eventually be able to leave the room would require each participant to wait a randomly distributed amount of time after each collision before trying again. This essentially describes the hardware protocol for the Ethernet. Other examples of symmetry breaking include the dining philosophers problem (*DinPhil*), communication guard scheduling (*CommGuard*), and leader election (*LeadElect*).

1.2 Tradeoffs

Tradeoffs are often involved in the use of randomized algorithms. Benefits to be reaped by introducing randomization into algorithms include, in the sequential case, reductions in time complexity (e.g., *PrimeTest*, *SockSel*, and *NearNeb*) and in space complexity (e.g., *PerfHash*).

In the distributed case, reductions in communication complexity (e.g., *ByzAgree*) and queueing delay (e.g., *MsgRoute*) can be obtained, and an algorithm's resiliency to faults can be improved (e.g., *MsgRoute*). Perhaps an even more fundamental benefit of randomization in the distributed setting is the ability to solve problems that have no deterministic solutions (e.g., *DinPhil*, *CommGuard*, and *LeadElect*).

In addition to these gains, a randomized algorithm is almost always simpler to understand and easier to implement than its deterministic counterpart. This is perhaps best illustrated by Lóvász's probabilistic algorithm for perfect matching discussed earlier. As we will see, conceptual elegance and simplicity are a hallmark of all the randomized algorithms treated in this survey. In an age of rising software complexity and cost, the simplicity of randomized algorithms will be a key determining factor in their acceptance by the software community.

To profit from the use of randomization, one must often sacrifice the traditional notion of absolute program correctness for a notion of "correct with probability $1 - \epsilon$." For the distributed algorithms *DinPhil*, *CommGuard*, and *ByzAgree* the ϵ is zero, so we have eventual agreement with probability 1. In other cases, such as *PrimeTest*, the ϵ can be made exponentially small in the length of the input by iterating the algorithm some number of times. The beauty of these algorithms is that usually only a small number of iterations are required to establish a very high degree of confidence in their output.

Another potential problem with randomized algorithms is that sometimes there is a small probability of taking an inordinate amount of time to execute (e.g., *NearNeb*) or of even failing to halt (e.g., *LeadElect*).

Analogous to the space-time tradeoff inherent to deterministic sequential algorithms, with randomized algorithms, there is a tradeoff involving resource requirements and absolute correctness. In fact, this tradeoff has led to the distinction of two types of randomized algorithms: *Monte Carlo* algorithms are always fast and probably correct, whereas *Las Vegas* algorithms are probably fast and, upon termination, always correct. Las Vegas algorithms, however, may fail to terminate for some inputs. For example, the algorithm for primality testing (*PrimeTest*) is of the Monte Carlo variety, while the algorithm for nearest neighbors (*NearNeb*) is of the Las Vegas variety.

If a purported solution to a problem is easily verifiable then a Monte Carlo algorithm *MC* for it can be converted into a Las Vegas algorithm by simply repeating *MC* till a correct solution is found. Similarly, any Las Vegas algorithm *LV* can be trivially converted into a Monte Carlo algorithm: one can always return a wrong answer (efficiently!) if *LV* seems to be taking too long. Since *LV* is fast with high probability, the modified algorithm will be correct with high probability.

The Karp-Rabin string matching algorithm described above is a good example of how to convert a Monte Carlo algorithm into a Las Vegas algorithm: the kernel of the Karp-Rabin algorithm will, from time to time, report spurious matches. By first checking if a purported match is a valid match, the Karp-Rabin algorithm always gives a correct answer. Muthukrishnan [Mut93] gives an efficient parallel algorithm for exactly this problem.

In [BB88], Las Vegas algorithms possessing bounded time requirements are called *Sherwood* algorithms. Randomized quicksort is an example of a Sherwood algorithm. It takes at most $O(n^2)$ time on any problem instance. Note that a Las Vegas algorithm that may possibly not terminate (e.g., *LeadElect*), cannot be a Sherwood algorithm.

2 Sequential Randomized Algorithms

In the first part of this survey, we present seven sequential randomized algorithms. The first algorithm (*SockSel*) is a simple illustration of the input randomization technique. The next three algorithms (*PrimeTest*, *NetHierarchy*, and *PerfHash*) illustrate the power of random search. We then give an example of control strategy randomization (*UnivHash*). We conclude this section with a randomized algorithm that uses random sampling (*NearNeb*).

2.1 The Sock Selection Problem

In this section, we provide a randomized solution to the Sock Selection problem (*SockSel*). This problem, although somewhat contrived, illustrates the technique of input randomization in a simple manner. It also bears connections with certain resource allocation problems.

Consider a dresser drawer of $2n$ socks, half of which are red and half of which are blue. Person X has just awoken and is in dire need of a matching pair of socks; a matching pair of either color will do. In his elusive search for this holy grail, person X randomly extracts a sock at a time from the drawer, and may also throw socks away (one at a time) if he believes he has no use for them. He is not allowed to put a sock back in the drawer. The question is, then: How many socks need person X remove from the drawer before a matching pair is obtained?

If there is no limit to the number of socks person X can have in his possession at any one time, then the problem is trivial. He simply removes three socks from the drawer and discards the sock that is not needed. Since two socks out of three must be the same color, this procedure will terminate in constant time.

The problem becomes more interesting if person X can have in his possession at most two socks at any one time, and this is the sock selection problem we study. The simplest deterministic solution, which is basically a sequential search through the sequence of socks extracted from the drawer, is as follows.

```
SockSel1 { (* First Try at Sock Selection *)
    s1 := get-sock()
    s2 := get-sock()
    WHILE color-of(s1) <> color-of(s2) DO {
        discard-sock(s2)
        s2 := get-sock()
    } (* end while *)
}
```

It is not difficult to see that in the worst case this algorithm will take $O(n)$ time. The worst case behavior is manifest when the sequence of socks returned by **get-sock()** is either *red*, *blue*, *blue*, ..., *blue*, *red* or *blue*, *red*, *red*, ..., *red*, *blue*, where the number of intervening socks of opposite color is $O(n)$. In fact we can make a stronger statement: *any* deterministic algorithm will have $O(n)$ worst case running time.

The above "worst case" sequences of socks returned by **`get-sock()`** capture the drawer in an adversarial role with respect to person X. For most of the sequences returned by **`get-sock()`**, however, the while-loop will terminate before n steps. Thus it is reasonable to anticipate that the average running time of *SockSel1* is much less than $O(n)$. This suggests the randomized algorithm *SockSel2*, an improved version of *SockSel1*.

```
SockSel2 { (* Revised Sock Selection Algorithm *)
    s1 := get-sock()
    s2 := get-sock()
    WHILE color-of(s1) <> color-of(s2) DO {
        toss a perfect two-sided coin
        IF heads THEN {
            discard-sock(s1)
            s1 := get-sock()}
        ELSE {
            discard-sock(s2)
            s2 := get-sock()}
    } (* end while *)
}
```

Here we assume that the drawer does not know the random choices made by *SockSel2*, i.e., the coin tosses are *private*.[4] This assumption is critical for, without it, the drawer can force *SockSel2* into long $O(n)$-step executions. Even worse, if the coin tosses are public, an adversarial drawer can force person X to end up with a mismatching pair of socks after the drawer has been emptied.

The way *SockSel2* is formulated above, the latter problem does not completely go away even when the coin tosses are hidden from the drawer: with probability that is exponentially

[4]For a discussion of private vs. public coin tosses, see the last paragraph of Section 2.6 and [GS89]. A related concept called *shared randomness*, which is weaker than both private and public coin tosses, is discussed in [BDMP91].

small in n, *SockSel2* can return a mismatched pair of socks. *SockSel2* can be made foolproof by employing two counters, one for the number of red socks left in the drawer and one for the number of blue socks left in the drawer. If it finds that it possesses the last sock of a particular color, then it should immediately discard that sock. The next call to `get-sock()` will return a matching sock.

Assuming *SockSel2*'s coin tosses are private, a viable strategy for the drawer is to have `get-sock()` return socks of different colors on the first two calls and thereafter flip a perfect two-sided coin to determine the color of the next sock to return. In this case, the probability that the while-loop will be executed i times is $(1/2)^i$, $i \geq 1$, and, thus, the probability that `get-sock()` is called exactly $(i+2)$ times is $(1/2)^i$. The expected running time, for large n, is given by

$$\sum_{i=1}^{i=n}(i+2)(1/2)^i \sim 4. \qquad (1)$$

Notice that the running time of *SockSel1* averaged over all sequences returned by `get-sock()` is 4, the same as the expected running time of *SockSel2* for *any* input sequence. The following properties can thus be ascribed to problems amenable to solution by input randomization:

1. The problem should have a deterministic algorithm with good average running time.
2. The random transformation applied to the input for achieving uniform running time for all the inputs should take less time than the algorithm itself.

The problem of primality testing considered next illustrates another technique for randomized algorithms: random search.

2.2 Primality Testing

The problem of *primality testing* is, Given a positive integer n expressed in binary notation, is n a prime number? Recall that a number n is prime if the only numbers by which it is divisible are 1 and itself; otherwise, n is said to be *composite*.

Since the dawn of number theory, prime numbers have enjoyed considerable attention. Despite all the progress in the field, to date there is no formula (similar to, say, Fibonacci numbers) to enumerate all the prime numbers. Fermat's primes, some of which are actually not prime, and the ancient Chinese assertion that n is prime if and only if n divides $2^n - 2$, are wrong results which exemplify the mysteries enshrined in prime numbers. (For the latter, consider, for example, $n = 341$.)

Of late extremely large prime numbers are in great demand because of their use in defining trap-door functions for public key cryptography systems [RSA78, Sch84, GM84, Smi83]. For example, in the Rivest-Shamir-Adleman (or RSA) cryptosystem [RSA78] the keys are 200-digit numbers. An encryption key is the product of two secret primes, having approximately 100 digits each, which are known only to the creator of the key. The corresponding decryption key is computed from the same two prime numbers using a publicly known algorithm. Difficulty in factoring large numbers is at heart of this cryptosystem: it ensures that one cannot easily deduce, in any reasonable amount of time, the prime numbers that went into forming the publicly advertized encryption key. Clearly, large primes are essential to this scheme. Using randomized search for testing whether a given number is prime — such a test can be used for generating large prime numbers — is the subject of this section.

In the absence of a formula, a plausible strategy for generating large prime numbers might be:

```
GenPrime{
    REPEAT{
        Pick a large number at random;
        Test whether it is prime;}
    UNTIL a prime number of desired size is found
}
```

The mean distance between primes in the neighborhood of a number n is $O(\log n)$ (see, e.g., [Sch84]). Thus we do not have to test very many numbers before finding one in the desired range. For example, in order to find a prime number about 10^{20} in size, we only have to test about 48 numbers. The catch, however, is to test such large numbers for primality in a moderate amount of time.

One might contemplate using trial division, or even Wilson's theorem — which states that a number n is prime if and only if n divides $(n-1)!+1$ without remainder — in order to check for primality. Repeated trial divisions are clearly very inefficient because even if one were to try divisions with only the prime numbers between 1 and n — notwithstanding the fact that there is no formula for generating them — one still has to conduct $O(n/\log n)$

divisions. Since n is encoded in $\lceil \log(n+1) \rceil$ bits, repeated divisions will take exponentially long. Furthermore, the sight of the factorial should dispel any hope for success in using Wilson's theorem as a practical test for primality.

Another fundamental result from number theory also appears promising. Pierre de Fermat, a French mathematician, showed that if a number n is prime then, for all x, n does not divide x implies n divides $x^{n-1} - 1$ [Sch84]. This result has become known as Fermat's theorem, not to be confused with his *last* theorem. The condition n divides $x^{n-1} - 1$ can be restated as $x^{n-1} \equiv 1 \pmod{n}$, which we refer to as Fermat's congruence.

The contrapositive of Fermat's theorem yields a technique for showing the compositeness of a number n. That is, n can be proven composite if we can find an x such that n does not divide x or $x^{n-1} - 1$ (elementary properties of modular arithmetic allow the latter condition to be verified without ever computing the number $x^{n-1} - 1$). Let us call such x *witnesses to the compositeness of* n. Note that a reasonable search space for x are the integers between 1 and $n - 1$, inclusively, as these are guaranteed not to be divisible by n.

The problem with using Fermat's theorem, however, is that the converse of the theorem does not hold and there therefore exist composite n bearing no witnesses to their compositeness. Such n are known as the *Carmichael numbers*, the first three of which are 561, 1105, and 1729. Interestingly, as pointed out in [CLR90], Carmichael numbers are extremely rare; there are, for example, only 255 of them less than 100,000,000. Furthermore, even if a composite n possesses a witness x, i.e., it is not a Carmichael number, there is no obvious way to locate x.

One can also obtain a positive identification of composite numbers using the Lucas-Lehmer heuristic [Leh27]: n is prime if and only if $x^{n-1} \equiv 1 \pmod{n}$ *and* $x^{\frac{n-1}{p}} \not\equiv 1 \pmod{n}$, for each prime factor p of $n - 1$. In general, the prime factors of $n - 1$ may not be known. However, this test can be used effectively if $n = 2^m + 1$ for some positive integer m, a rather restricted subset of the integers.

Let $n = \Pi_{i=1}^{i=m} p_i^{\nu_i}$ be the unique prime factorization of n. Define $\lambda(n) = \mathrm{lcm}\{p_1^{\nu_1 - 1}(p_1 - 1), \ldots, p_m^{\nu_m - 1}(p_m - 1)\}$. It was shown by Carmichael [Car12], of the Carmichael numbers fame, that n satisfies Fermat's congruence if and only if $\lambda(n)$ divides $(n-1)$. The reader can verify that $\lambda(561)$ divides 560.

In light of above theorem, a plausible approach to testing primality — actually compositeness, but for a deterministic algorithm that always terminates with the correct answer, it does not matter — is as follows. Divide composite numbers into two categories according to whether $\lambda(n)$ divides, or does not divide, $(n-1)$. If $\lambda(n)$ does not divide $(n-1)$, then by virtue of Carmichael's result, one can use Fermat's test. On the other hand, if $\lambda(n)$ does divide $(n-1)$ a new test is necessary. If an attempt to place a number in either category fails, it must be prime.

A variation of the above strategy was employed by G. Miller in a paper that has proven to be very useful in primality testing [Mil76]. This paper defined the basic concepts that were later used by Rabin to derive a probabilistic algorithm for primality testing. To arrive at his algorithm for primality testing, Miller divided the composite numbers as suggested above. However, instead of using Carmichael's λ-function, he used $\lambda'(n) = \text{lcm}\{(p_1-1), \ldots, (p_m-1)\}$ to pare down the set of composite numbers that satisfy Fermat's congruence. The following is a simplified version of Miller's algorithm. In this algorithm, f is a computable function.

```
PrimeTest (Miller) { (* a deterministic algorithm for primality testing *)
    Input $n$
    If $n$ is a perfect power, say $m^s$, output 'composite' and HALT
    REPEAT FOR EACH $x \leq f(n)$ {
        (1) if $x$ divides $n$, output 'composite' and HALT
        (2) if $x^{n-1} \not\equiv 1 \pmod{n}$, output 'composite' and HALT
        (3) if there is an $i$ such that $\frac{n-1}{2^i} = m$ is integral,
            and $1 < \gcd(x^m - 1, n) < n$, output 'composite' and HALT
        }
    output 'prime' and HALT
}
```

Miller used the λ' function to characterize the class of composite numbers that satisfy Fermat's congruence. He proved that a function f can be defined such that, if n is composite, then by testing conditions (1) through (3) repeatedly, for all $x \leq f(n)$, the algorithm will indeed identify n as composite. Furthermore, $f(n)$ can be defined so that the above algorithm terminates in $O(n^{\frac{1}{7}})$ steps. Since n is given in $\lceil \log(n+1) \rceil$ bits, $O(n^{\frac{1}{7}})$ is still exponentially long. Using the Extended Riemann Hypothesis (ERH), however, Miller proved that f can be defined so that a slightly more complex version of the above algorithm terminates in

$O(\eta^4 \log\log\eta)$ steps, where $\eta = \lceil \log(n+1) \rceil$ denotes the length of the binary representation of n. Thus, the primality of a number can be determined deterministically in polynomial time assuming ERH.

Like before, let us call any number x between 1 and n for which at least one of conditions (2) and (3) in the main body of the above algorithm is true a witness to the compositeness of n. A key observation which makes randomized testing for primality feasible is that there is an abundance of witnesses for compositeness. The probability that a number is composite, and conditions (2) and (3) are not satisfied is very small. In fact, Rabin [Rab76] has shown that more than half the values of $x \in \{1, 2, ..., n-1\}$ satisfy (2) or (3) if n is indeed composite (see, also, [CLR90], Theorem 33.38). Monier [Mon80] has subsequently strengthened this result by showing that at least 3/4 of the x are witnesses. Even though Miller's polynomial time algorithm for testing primality requires the ERH, these results about the density of witnesses holds in general and can be proved without recourse to this hypothesis.

Figure 2 illustrates the high density of witnesses to compositeness. The figure shows, for each integer n in the range 10,000 to 12,000, the percentage of integers between 1 and n that are witnesses to the compositeness of n. As can be seen, if the number is composite, then the percentage of witnesses in this range of numbers is almost always more than 98%; for only about 18 numbers out of 2000, the percentage of witnesses lies in the 85 to 98% range. As is to be expected, for primes there are no witnesses, resulting in a sparse set of points along $y = 0$.

Miller witnesses, in conjunction with Rabin's result about their density, gives a rather powerful primality testing algorithm:

```
PrimeTest (Rabin) { (* a probabilistic algorithm for primality testing*)
    Input $n$
    REPEAT $r$ times{
        (1) randomly pick an $x$ between 1 and $n$
        (2) if $x^{n-1} \not\equiv 1 \pmod{n}$, output 'composite' and HALT
        (3) if there is an $i$ such that $\frac{n-1}{2^i} = m$ is integral,
            and $1 < \gcd(x^m - 1, n) < n$, output 'composite' and HALT
        }
    output 'prime' and HALT
```

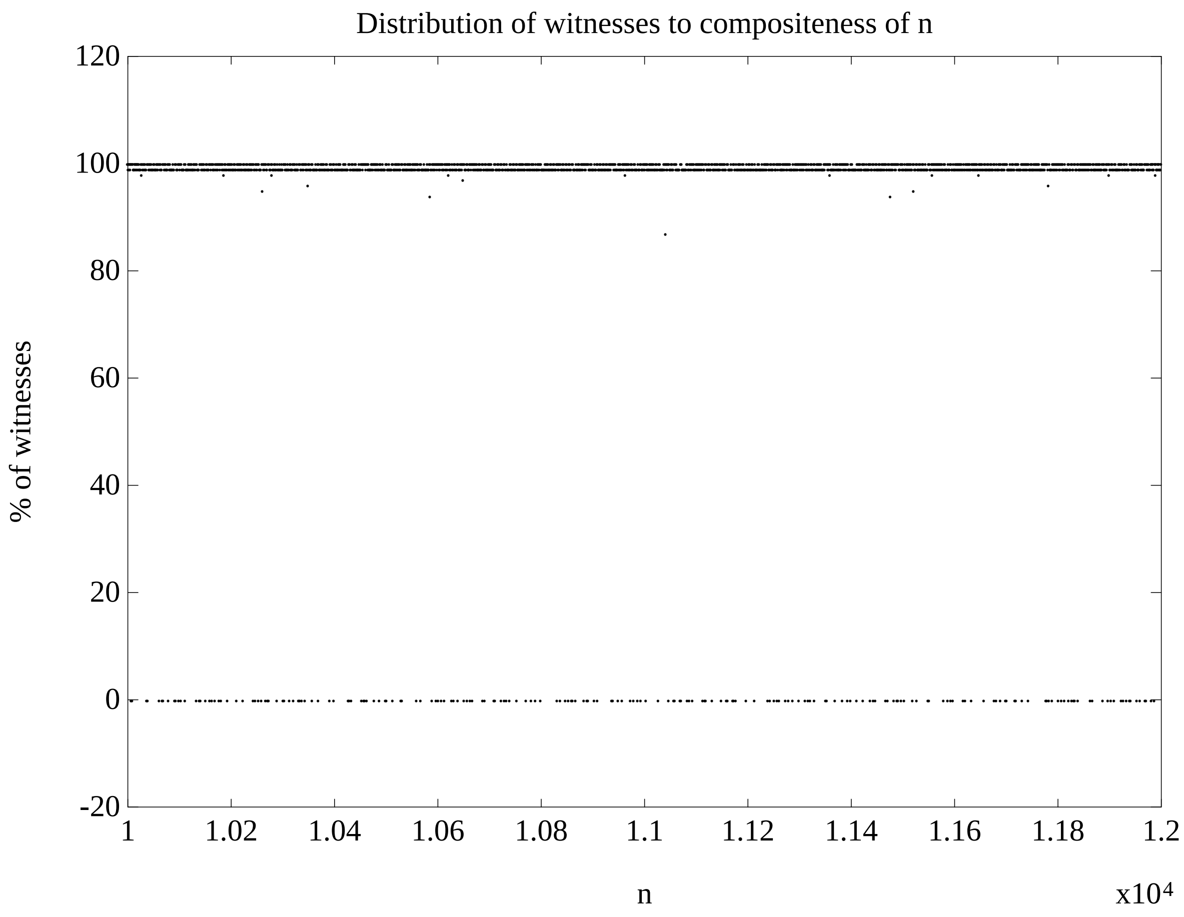

Figure 2: Percentage of witnesses to the compositeness of n in the range 10,000 to 12,000. The points at $y = 0$ represent prime numbers.

```
}
```

In the above algorithm, if either condition (2) or (3) is satisfied then n is composite. On the other hand, if (2) and (3) are not satisfied by x then n may or may not be composite and the procedure must be repeated. If r trials are used, the probability that n is composite and not detected is less than $1/2^r$. Therefore, with very few trials, one can either prove that a number is composite or gain a high degree of confidence that it is prime. See also [BBC$^+$88] for some intriguing observations about the performance of Rabin's primality test and about its reliability when used to generate a random integer that is probably prime.

In the mid-seventies, another probabilistic primality testing algorithm was discovered by Solovay and Strassen [SS77]. Some basic results in number theory are needed to describe their algorithm. For any prime number n, one can define $Z_n^* = \{1, \ldots, n-1\}$, a cyclic group under multiplication mod n. The *Legendre Symbol* for any element $x \in Z_n^*$, denoted by $(\frac{x}{n})$, is defined to be 1 or -1 depending on whether or not x is a perfect square (i.e., a quadratic residue modulo n) of some other element in Z_n^*. More precisely, $(\frac{x}{n}) = 1$ if $x \equiv y^2 \pmod{n}$ for some $y \in Z_n^*$, -1 otherwise.

If x is a perfect square, say $x \equiv y^2 \pmod{n}$, then it is not difficult to see that $x^{\frac{n-1}{2}} \equiv y^{(n-1)} \equiv 1 \pmod{n}$. This leads to a fast way of computing the Legendre symbol. One can extend these concepts to a general n which may or may not be prime. In this case, for any number n, one can define $Z_n^* = \{x | x \in \{1, \ldots, n-1\}, \text{ and } \gcd(x, n) = 1\}$. Once again, Z_n^* is a group under multiplication mod n. The Legendre symbol is generalized to the *Jacobi symbol*: if n is prime, the Jacobi symbol equals the Legendre symbol; when n is composite, the Jacobi symbol is defined to be the product of all the Legendre symbols corresponding to the prime factors of n, i.e., if $n = \Pi p_i$, then $(\frac{x}{n}) = \Pi(\frac{x}{p_i})$.

In the algorithm by Solovay and Strassen, for $x \in \{1, \ldots, n-1\}$ to be a witness to compositeness of n, either $\gcd(x, n) > 1$ or $x^{\frac{n-1}{2}} \pmod{n} \neq (\frac{x}{n})$. Their algorithm can be stated as follows.

```
PrimeTest (Solovay-Strassen) { (* another algorithm for primality testing*)
     Input $n$
     REPEAT $r$ times{
          (1) randomly pick an $x$ between 1 and $n$
```

```
            (2) if $\gcd(x, n) > 1$, output 'composite' and HALT
            (3) if $x^{\frac{n-1}{2}} \pmod n \neq (\frac{x}{n})$, output 'composite' and HALT
            }
      output 'prime' and HALT
}
```

Determining if x and n are relatively prime (e.g. by Euclid's algorithm), computing $x^{\frac{n-1}{2}} \pmod n$, and the Jacobi symbol $(\frac{x}{n})$, can all be accomplished in logarithmic time. If n is prime, then it follows from the fact that Z_n^* is cyclic, that $x^{\frac{(n-1)}{2}} \equiv (\frac{x}{n}) \pmod n$. Thus when n is indeed prime, no x will qualify as a witness. When n is composite, Solovay and Strassen showed that the set of false witnesses — the numbers in $\{1, \ldots, n-1\}$ that violate conditions (1) and (2), i.e., $\gcd(x, n) = 1$ and $x^{\frac{(n-1)}{2}} \equiv (\frac{x}{n}) \pmod n$ — forms a proper subgroup of Z_n^*. Hence the cardinality of this set can be at most $(n-1)/2$. Once again, using the properties of quadratic residues modulo n, the witnesses for compositeness are defined in such a way that they are both easily checkable and abundant.

An interesting comparison of the Miller-Rabin and Solovay-Strassen primality testing algorithms is given in [Mon80], where it is shown that the former is always more efficient than the latter. These two algorithms are of the Monte Carlo variety because when n is prime they can report so only with a certain probabilistic measure of confidence; in particular, no proof is provided that this is the case. Convincing somebody that a number is composite is an easy task: one simply has to exhibit that it is a product of other two numbers. How can one demonstrate that a number n is prime? Certainly it can be done by showing all possible trial divisions, but that is not an efficient proof as it is exponentially long in the length of n. It was shown by Pratt [Pra75], using the Lucas-Lehmer heuristic for primality testing, that one can give a succinct proof for primeness of a number n in $O(\log n)$ lines. While it is easy to verify such a proof, unfortunately, there is no known method for coming up with the proof, or demonstrating the absence thereof, in polynomial time.

Other algorithms utilizing different number theoretic properties for defining witnesses for compositeness *and* primality have also been discovered [Rab80a, Leh82, AH87, GK86, AH88]. For example, Adleman and Huang [AH88] have devised a new algorithm that, instead of deciding primality by the inability to demonstrate witnesses to compositeness, employs a separate Monte Carlo test for primality. Thus, just like composite numbers, there exists a random polynomial time algorithm for the set of prime numbers. The algorithm flip-flops between searching for witnesses to compositeness and witnesses to primality, eventually finding one in polynomially bounded expected time. This algorithm, which is of the Las Vegas variety, will never declare a composite number to be prime or vice versa. However, it may not terminate in polynomial time for some inputs.

The next problem we consider, which concerns the notion of *transitive tournament* due to Erdős and Spencer [ES74], again illustrates random search. In this case, however, the sample space is so abundant with good points that the "checking" step inherent to primality testing can be dispensed with.

2.3 Networks without Large Hierarchies

Long ago, in a place called Confusion Land, there reigned an incompetent king called Nadir. Nadir had appointed 1000 ministers, generals, and other high-ranking officials to various portfolios in his kingdom. As usual, Nadir was afraid that some of his appointees would organize, revolt, and finally usurp the throne. His remedy was simple: keep them confused. He did this by not allowing a clear-cut line of command—a hierarchy—to be formed among these officials. His long experience in politics had convinced him that even if as few as 25 officials got organized they would overthrow him.

Nadir's definition of "being organized" is as follows: k officials are said to be *organized in a hierarchy* if for every three of them, the "is-a-boss-of" relation is transitive. That is, if for all triples of the form (A, B, C), if A is a boss of B and B is a boss of C implies A is a boss of C, then the k officials are organized.

Having made appointments to the 1000 positions, Nadir is stuck with the following task. He must define the is-a-boss-of relation between *every* pair of appointees such that no group of 25 or more officials is organized. At the micro-level (groups of size less than 25), there may be organized groups; at the macro-level, however, confusion should prevail. How will Nadir assign ranks to these thousand appointees in order to achieve his crooked objective?

In this section we consider Nadir's problem in detail and provide a general solution, the key to which is a theorem of Erdős and Spencer (Chapter 1 of [ES74]). To make this section self-contained, their result is proved here as Theorem 1. It turns out that Nadir's problem falls in the category of problems for which the solution space is abundant with candidates possessing a given property and random search can be used to derive the solution.

Nadir's problem can be described as that of constructing a network of nodes, where each node represents an official. Informally, a network represents an assignment of precedence between all possible pairs of nodes. It can be represented by a complete directed graph where an edge from x to y represents the relation "x is a boss of y."

Formally, a *network* T *on a set* V is a directed graph (V, T) where $T \subset V \times V$ such that for all $x, y \in V$, $x \neq y$, either $(x, y) \in T$ or $(y, x) \in T$, but not both. A network T is a *hierarchy* if $(x, y), (y, z) \in T$ implies $(x, z) \in T$, $\forall x, y, z \in V$. Networks and hierarchies are called *tournaments* and *transitive tournaments*, respectively, in [ES74].

Nadir's problem then, which we refer to as the *NetHierarchy* problem, is to construct a network that does not have "large" hierarchies. In particular, he wants a network T_n on n nodes such that every subnetwork of T_n containing m or more nodes is not a hierarchy. (In the case at hand, $n = 1000$ and $m = 25$.) A possible approach to constructing such a network would be to choose a network at random and check that all the $\binom{n}{m}$ subnetworks are not hierarchies. If a large hierarchy is found, another T_n can be picked randomly and checked. This process can be continued until a network with the required property is found. As we will see below, for appropriate values of m, one can even dispense with the check as any random T_n would suffice with a very high degree of confidence.

In a hierarchy it is possible to assign a unique rank to each node. The top-ranked node is a boss of all others, and in general, the ith-ranked node is a boss of all but those with a better rank. Hence a hierarchy is equivalent to a permutation of the n nodes. Figure 3 shows a six-node network that contains a hierarchy on five nodes. The permutation corresponding to the hierarchy on nodes $\{1, \ldots, 5\}$ is $\pi : \{1, 2, 3, 4, 5\} \rightarrow \{2, 3, 1, 4, 5\}$ as 2 is a boss of all other nodes, 3 is a boss of 1, 4, and 5, and so on. Also, note that the full network is not a hierarchy because of the cycles among nodes $\{6, 3, 1\}$, $\{6, 3, 4\}$, and $\{6, 3, 5\}$.

Erdős and Spencer [ES74] have proved an important property concerning the size of hierarchies in arbitrary networks, which we now present. Define $\chi(n)$ to be the largest integer such that *every* network on n nodes contains a hierarchy of $\chi(n)$ nodes. Unless stated otherwise log denotes logarithms to the base 2.

Theorem 1 **([ES74])** $\chi(n) \; < \; 1 + \lfloor 2 \log n \rfloor$.

The theorem is proved by showing that there exist networks that do not have any hierarchy on $1 + \lfloor 2 \log n \rfloor$ nodes. The proof is non-constructive. Let $?_n$ be the class of all networks

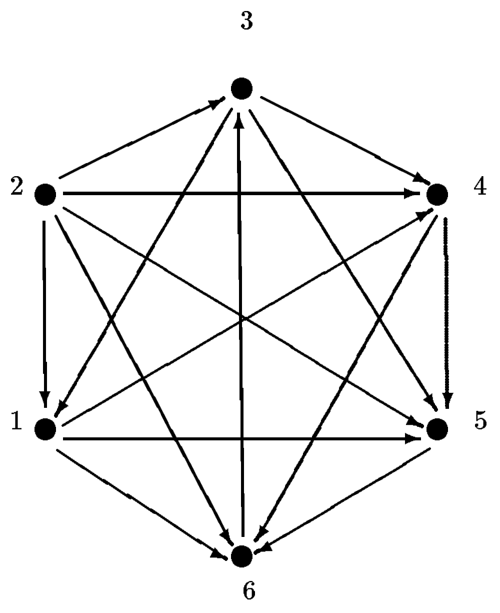

Figure 3: A network with a hierarchy on five Players with $\pi : \{1,2,3,4,5\} \rightarrow \{2,3,1,4,5\}$.

on n nodes and let $?'_n$ be the class of all networks that have a hierarchy on $1 + \lfloor 2\log n \rfloor$ nodes. We show that there are more networks in $?_n$ than in $?'_n$.

We first count the number of networks in $?_n$. Each network in $?_n$ consists of n vertices and $\binom{n}{2}$ edges, each of which can take two possible directions. Thus,

$$|?_n| \; = \; 2^{\binom{n}{2}} \tag{2}$$

Counting the number of networks in $?'_n$ is a bit more involved. Since each network in $?'_n$ has a hierarchy on $\varphi = 1 + \lfloor 2\log n \rfloor$ nodes, we first select the φ nodes and assign them a permutation, which will uniquely determine a hierarchy on these nodes. The remaining edges in the graph consisting of $(n - \varphi)$ nodes can be assigned arbitrarily. We count the number of networks for all the $\binom{n}{\varphi}$ possible choices of φ nodes and all the $\varphi!$ ways of assigning them a permutation. Formally,

$$?'_n \; = \; \bigcup_A \bigcup_\pi T_{A,\pi} \tag{3}$$

where A is a subset of n nodes such that $|A| = \varphi$, π is a permutation of the φ members of A, and $T_{A,\pi}$ is the set of networks on n nodes consistent with the hierarchy on A determined by π. That is, each network in $T_{A,\pi}$ will contain a hierarchy on A uniquely determined by π. The structure of the network on the remaining $n - \varphi$ nodes, however, is unspecified. In particular, the direction of $\binom{n}{2} - \binom{\varphi}{2}$ edges between these $n - \varphi$ nodes is unspecified. Hence,

$$|T_{A,\pi}| \; = \; 2^{\binom{n}{2}-\binom{\varphi}{2}} \tag{4}$$

Therefore, the total number of networks in $?'_n$ is bounded by

$$|?'_n| \; < \; \sum_A \sum_\pi |T_{A,\pi}| \; = \; \begin{pmatrix} n \\ \varphi \end{pmatrix} \varphi!\, 2^{\binom{n}{2}-\binom{\varphi}{2}} \; < \; 2^{\binom{n}{2}} \; = \; |?_n| \tag{5}$$

This implies that $?_n - ?'_n$ is non-empty and there exists $T \in ?_n - ?'_n$ containing no hierarchy on $\varphi = 1 + \lfloor 2 \log n \rfloor$ nodes. □

The above theorem establishes an upper bound on the largest integer χ such that every network on n nodes contains a hierarchy on $\chi(n)$ nodes. It can also be proved, by induction on n, that $\chi(n) \geq 1 + \lfloor \log n \rfloor$. Clearly, if it were the case in Nadir's politics that no hierarchies be formed on $m < 1 + \lfloor \log n \rfloor$ nodes, then every assignment of the is-a-boss-of relation would violate Nadir's requirement and he should make arrangements for a hasty departure. On the other hand, for values of m slightly greater than the upper bound of Theorem 1, the probability that a randomly selected graph contains a large hierarchy is minuscule. For $m \geq 1 + 2\lfloor \log n \rfloor$ this probability is bounded by

$$\frac{|?'_n|}{|?_n|} \; < \; \begin{pmatrix} n \\ m \end{pmatrix} m!\, 2^{-\binom{m}{2}} \tag{6}$$

Therefore, if Nadir were to construct a random network on 1000 nodes, the probability that it will have a hierarchy on any subset of 25 nodes is less than 0.0000000000000004. Thus a very promising strategy for Nadir is to toss a coin to determine the direction of each edge in the network; the odds are less than 4 in 10^{16} that he will construct a bad network.

The preceding discussion, unfortunately, leaves a "gray area" in the solution space: it is not clear how to solve the *NetHierarchy* problem for values of m between $1 + \lfloor \log n \rfloor$ and $1 + 2\lfloor \log n \rfloor$. For values of m less than the lower bound on χ, the solution is immediate; for values slightly greater than the upper bound, Theorem 1 immediately yields a trivial probabilistic algorithm as basic counting procedures reveal that there is an abundance of solutions in this region. However, for the gray area in between the upper and lower bounds on χ — which can possibly be shrunk by making the bounds tighter — exhaustive search seems to be the only way for solving this problem. The latter is prohibitively expensive even for moderate values of n and m. For example, if Nadir required that there be no hierarchies on 18 nodes, $\binom{1000}{18}$ subnetworks must be tested.

2.4 Probabilistic Hashing

Many problems require maintaining a table of values, or keys, and performing insert, search, and delete operations on them. Typically, the set of possible keys is very large, though at any one time only a small fraction of the keys will actually be in the table. In this section, we study a very popular and potentially constant-time solution to table management called *hashing*.

Throughout this section, $T[0 \ldots m-1]$ will denote the hash table and $U[0 \ldots N-1]$ will denote the universe of keys. In general, given a key $x \in U$, we will be interested in inserting x into T, searching for x in T, or deleting x from T. The total number of keys in the table will be limited to n, $n < m \ll N$, and S, $|S| = n$, will denote the set of keys that are to be inserted into the table.

Let $h : U \rightarrow [0 \ldots m-1]$, be a function that can be evaluated in constant time. The basic scheme underlying hashing is as follows. To insert a key x into the table, simply store it at $T[h(x)]$, if possible. To search for or delete x, just check location $h(x)$ in table T. All these operations take constant time, fulfilling the promise made earlier. However, there is a serious problem with this scheme. If there is another key, say y, such that $h(x) = h(y)$, then x and y will try to occupy the same place in the table. This phenomenon is called a *collision*. Much research has been conducted on finding hash functions that result in a minimum number of collisions and on data structures for storing keys that hash to the same table location.

For hashing to perform well the following two requirements are essential: the hash function distributes input keys uniformly over the table, and all the keys are equally likely. While the first requirement can be met by appropriately choosing the function $h(x)$, the second requirement is hard to fulfill as it postulates certain behavior on the input distribution. In practice, this requirement is not only beyond the algorithm designer's control, it is often violated. For example, a typical application of hashing is maintaining symbol tables for compilers. For most programs, variable names such as I, J, K are more common then, say, XQP. Thus it is unreasonable to expect a uniform probability distribution from the input to a symbol table. However, if it is known that the input is biased, it may be possible to tune the hash function. *Perfect hashing* represents the ultimate form of tuning, i.e., total collision avoidance. Another way of minimizing the risk due to biases in the input is to choose the hash function dynamically and at random. These two schemes are explored in the following sections.

2.4.1 Perfect Hashing

Heuristic methods for perfect hashing were first introduced in [Spr77]. A recent overview of perfect hashing can be found in [GBY91]. Several seminal results that make perfect hashing possible were proved in [FKS82, Meh82]. The discussion in this section is based on Section 2.3 of [Meh84a].

A function $h : U \rightarrow [0 \ldots m-1]$ is called a *perfect hash function for* $S \subseteq U$ if $\forall x, y \in S, \;\; h(x) \neq h(y)$ if $x \neq y$. For any given set S of input keys such that $|S| = n \leq m$, clearly there exists a perfect hash function: take any one-to-one mapping from S to any n distinct elements in T, and map all other elements of U so that they do not collide with the elements of S. Such a brute force approach to constructing a perfect hash function, however, is not very beneficial as it involves a table look up that may take $O(n)$ time. For perfect hashing to be of practical use, the following criteria should be met:

- The program to compute a perfect hash function should be small in size.
- For a given S, m and N, it should be easy to find a perfect hash function.
- One should be able to evaluate a perfect hash function in $O(1)$ time.

In this section we consider the problem of finding a perfect hash function given the values of S, m and N. The use of random search, in a suitably constructed family of functions, will be the principal probabilistic technique used in the construction of such a function.

Mehlhorn [Meh84a] has shown that there exists a program of length $O(n^2/m + \log\log N)$ that computes a perfect hash function for a given set $S \subseteq U$. This result, however, only demonstrates the existence of such a function. To find an actual perfect hash function, consider the following family H of hash functions:

$$H \;=\; \{h_k | h_k(x) = (kx \bmod N) \bmod m \;;\; 1 \leq k < N\}. \tag{7}$$

Without loss of generality, let $U = [0 \ldots N-1]$ be the universe of keys with N prime. Primality of N can be achieved by adding non-existent keys to U. The resulting universe will not be substantially larger than the original U as prime numbers are sufficiently dense (see Section 2.2). For a given set S, let

$$B(i,k) = \{x | x \in S \text{ and } (kx \bmod N) \bmod m = i\} \tag{8}$$

be the set of all the keys in S that collide at table location i when h_k is used as the hash function. Each such set $B(i,k)$ is called a *bucket*. Also, let $b(i,k) = |B(i,k)|$, $0 \leq i < m$. Clearly, $b(i,k)$ is one more than the number of collisions at $T(i)$ when the hash function used is h_k. Using elementary counting principles and properties of modulo arithmetic one can verify the following inequality [Meh84a]:

$$\sum_{k=1}^{N-1} \left[\left(\sum_{i=0}^{m-1} b(i,k)^2 \right) - n \right] \leq \frac{2n(n-1)(N-2)}{m}. \tag{9}$$

The quantity $\sum_{i=0}^{m-1} b(i,k)^2 - n$, for any particular value of k (and thus for any particular $h_k(x)$), is a measure of the number of collisions. Let us define $M_S(k)$ to be this measure. Equation (9) puts a bound on the sum of $M_S(k)$ for all possible values of k. Since $M_S(k)$ is always positive, more than half of them cannot exceed twice the upper-bound on the summation in Equation (9). Therefore, at least half of all the possible k's must satisfy the relation $M_S(k) \leq 4n(n-1)/m$, since otherwise equation (9) would be invalidated. In other words, for a randomly picked $k \in [1 \ldots N-1]$,

$$Prob\left[M_S(k) \leq \frac{4n(n-1)}{m} \right] > \frac{1}{2}, \tag{10}$$

and the class H is rich in functions for which $M_S(k)$ is bounded by $O(n^2/m)$.

Equation (10) provides a way of finding, in $O(n)$ expected time, an h_k such that $M_S(k)$ is bounded by $4n(n-1)/m$. Select a random k and compute $M_S(k)$. If it satisfies the bound we are done; else select another k and do the same thing. The computation of $M_S(k)$ will take $O(n)$ time. Equation (10) guarantees that the expected number of tries will be no more than two. Thus, there exists a function h_k such that

$$\sum_{i=0}^{m-1} b(i,k)^2 \leq n + \frac{4n(n-1)}{m}, \tag{11}$$

which can be found in $O(n)$ expected time. One can also show that this procedure will terminate in $O(n \log n)$ time with high probability.

The above procedure forms the basis for finding a perfect hash function for a specific table size. In particular, we consider the two table sizes $m = n$ and $m = O(n^2)$, and prove the following results:

1. If $m = n$ then an h_k satisfying $\sum_{i=0}^{m-1} b(i,k)^2 < 5n$ can be found probabilistically in expected time $O(n)$.

2. If $m = 2n(n-1)+1$ then h_k, such that $h_k(x) = ((kx) \bmod N) \bmod m$, is a perfect hash function for S and can be determined in $O(n)$ expected time.

The first result follows by substituting $m = n$ in equation (11). For the second result, substituting $m = 2n(n-1)+1$ in equation (11) yields:

$$\sum_{i=0}^{m-1} b(i,k)^2 \quad < \quad n+2. \tag{12}$$

Since $\sum_{i=0}^{m-1} b(i,k) = n$, equation (12) implies that $b_i \leq 1$ for all i (the only solution for X_i in the set of equations $\sum X_i = n$ and $\sum X_i^2 \leq n+2$ is $X_i \leq 1$). As $b(i,k)$ is the number of elements in S that will occupy position i in the table, there will not be any collisions for this value of k. Hence h_k in equation (11) with $m = O(n^2)$ is a perfect hash function if an appropriate value of k is used.

Thus the class H of functions has a perfect hash function for any S, $|S| = n$, if the size of the table is $O(n^2)$. Furthermore, such a function can be found in $O(n)$ expected time. The only problem with this scheme is that the size of the table is much larger than $|S|$. Our first result suggests a way out. We can partition S so that the square of the sum of all bucket sizes is no more than $5n$. This can be done with one hash function, which obviously is not perfect. A second hash function, which is perfect for the smaller partition, can be used for each partition. The following theorem gives a more precise statement.

Theorem 2 *Let N be prime and $S \subseteq [0 \ldots N-1]$, $|S| = n$. A perfect hash function $h : S \rightarrow [0 \ldots m-1]$, $m = 9n$, with $O(1)$ evaluation time and $O(n \log n)$ program size can be found in $O(n)$ expected time.*

Proof: The perfect hashing function is constructed in two steps. In the first step we find a k such that $(kx \bmod N) \bmod m$ partitions S into subsets $B(i,k)$, where

$$B(i,k) \quad = \quad \{x | x \in S \text{ } and \text{ } h_k(x) = i\} \tag{13}$$

such that $\sum_{i=0}^{m-1} |B(i,k)|^2 \quad \leq \quad 5n$. Such a k exists and can be found in $O(n)$ expected time. Let c_i denote $2b(i,k)(b(i,k)-1)+1$. In the second step, we find k_i, for all i, such that $(k_i x \bmod N) \bmod c_i$ is a perfect hash function for a table of size c_i and the set of keys $B(i,k)$. By the second result proved earlier, this will take $O(b(i,k))$ expected time. The program *PerfHash* computes the perfect hash function for a table of size $5n$.

```
PerfHash { (* Computes perfect hash function h(x) *)
    i := (kx mod N) mod n
    j := (k_i x mod N) mod c_i
    h := $\sum_{l=0}^{i-1} c_l$ + j
}
```

If the starting index for each sub-table ($\sum_{l=0}^{i-1} c_l$) is stored, $h(x)$ can be evaluated in $O(1)$ time. Also, it is easily seen that the total size of the hash table in the above program is $9n$ based on the fact that one can find a hash function h_k, such that $\sum b(i,k)^2 = 5n$. In the second step each bucket is mapped into a space of size $2b(i,k)(b(i,k)-1)+1$. Hence the total space necessary is

$$\begin{aligned}\sum_{1\le i\le n} \{2b(i,k)(b(i,k)-1)+1\} &= 2\sum_{1\le i\le n} b(i,k)^2 - 2\sum_{1\le i\le n} b(i,k) + n \\ &= 2\times 5n - 2n + n \\ &= 9n.\end{aligned}$$

As for the total space occupied by *PerfHash* itself, each $\sum_{l=0}^{i-1} c_l$ used by the program can be at most $\log n$ bits long as it is an index into an array of size $9n$. Since we have to store n such numbers, the size of the program *PerfHash* is $O(n \log n)$.

The time needed to construct *PerfHash* is the time required to find k and all the k_i's. Thus it will take $O(n) + \sum_{i=0}^{m-1} O(b(i,k)) = O(n)$ units of expected time. The fact that this function is perfect is guaranteed by the two results proved earlier. □

We close this section by pointing out why the technique of random search works for perfect hashing. The class H of function is particularly rich in functions that are "nearly perfect." Thus, a randomly selected function from H will, with high probability, partition the set S evenly. A perfect hash function can then be used for each of these partitions, which are sufficiently small. The key here is the richness of the solution space. Had the perfect hash functions been rare in H, our random selection and testing procedure would require a long search through the m^N possible functions from U to T.

2.4.2 Universal Hashing

As seen earlier, for most *fixed* hash functions, hashing provides us with an $O(1)$ expected time and $O(n)$ worst case time procedure for table maintenance. *Universal hashing* deals with the possibility of biases in the input, which may result in the $O(n)$ complexity, by randomizing over hashing functions. In universal hashing, first discussed in [CW79], one works with an entire class, H, of hashing functions instead of picking any one single hashing function *a priori* and using it for every run. At the beginning of each run a function is randomly chosen from H and used for that run. Since it is unlikely that a "bad" function would be picked in most runs, for H properly defined, the running time averaged over many runs is expected to be small.

For any randomly selected element of H to possess a small expected access time for each set of keys, almost all hashing functions in H should distribute the set of input keys fairly uniformly over the hash table. We define a class H of functions to be c-universal if only a fraction c/m of functions in H produce a collision on any pair x, y in the universe of input keys. Formally, $H \subseteq \{h \,|\, h : [0 \dots N-1] \rightarrow [0 \dots m-1]\}$ is *c-universal* if $\forall x, y \in [0 \dots N-1]$ such that $x \neq y$,

$$|\{h|h \in H \text{ and } h(x) = h(y)\}| \leq \frac{c|H|}{m}. \tag{14}$$

For N prime, consider the particular class H_1 defined as follows:

$$H_1 = \{h_{a,b}|h_{a,b}(x) = [(ax+b) \bmod N] \bmod m, a, b \in [0 \dots N-1]\}. \tag{15}$$

It can be shown that the class H_1 is c-universal for $c = [\frac{\lceil N/m \rceil}{(N/m)}]^2$. Since each function in H_1 is fully specified by a and b, there are N^2 functions in this class and $O(\log N)$ bits are required to pin-point any one function. Also, a random function can be chosen by randomly picking a and b from $[0 \dots N-1]$.

Let us assume that each hash function in H_1 has the same probability of being picked in any run, and *hashing with chaining*[5] is used. Under these assumptions it can be shown that the time taken by universal hashing to perform access, insert and delete operations, or any sequence of such operations, is the same as the expected time taken by hashing with chaining when all inputs are assumed to be equally-likely [Meh84a]. In fact this result holds for any c-universal class of functions. Thus, universal hashing, with no assumptions on the input distribution, should perform as well as hashing with chaining when the best possible input distribution (i.e., completely unbiased input) is assumed. Note that even though the end-result, as far as the performance is concerned, is the same for these two hashing paradigms, there is a considerable difference between the assumptions underlying

[5]In hashing with chaining, all keys that collide at a given index i in the hash table T are stored as a linked list at $T[i]$.

them. In universal hashing the algorithm controls the dice and not the user, and therefore the expected complexity is $O(1)$ even for maliciously designed inputs.

Universal hashing is an example of the control randomization technique we described in Section 1.1. Control randomization is useful for other problems for which many efficient algorithms exist, such as sorting. If each one of these algorithms has good average performance but poor worst case performance, randomization over the space of available algorithms is a way to eliminate the risk involved in using any single one of them.

2.4.3 Some Recent Results

The FKS perfect hashing algorithm discussed in Section 2.4.1 results in a hash table size that is larger than the total number of keys. An algorithm is said to be *order preserving* if it puts entries into the hash table in a prespecified order, and *minimal* if it generates hash functions where the table size is the same as the total number of keys. Recently there has been a flurry of research activity in the areas of minimal and order preserving perfect hash functions [Cic80, Jae81, Cha84, LC88, CHM92, MWHC93].

Czech, Havas and Majewski [CHM92] present a probabilistic algorithm for generating order preserving, minimal perfect hash functions. This algorithm, which runs very fast in practice, uses expected linear time and requires a linear number of words to represent the hash function. The results of [CHM92] are further extended in [MWHC93] to a family of elegant probabilistic algorithms that generate minimal perfect hash functions allowing arbitrary arrangements of keys in the hash table. The idea used is the following. Certain integer congruences that correspond to acyclic $r-$graphs can be solved in linear time. This uses a result in [ER60], which states that the majority of random sparse $2-$graphs are acyclic. It is extended in [MWHC93] to $r-$graphs, with $r > 2$. Perfect hash functions are obtained by randomly mapping a set of keys into an acyclic $r-$graph. The mapping is achieved via universal hashing. Once completed the constructed set of linearly independent congruences, corresponding to the created $r-$graph, is solved, and the solution is a minimal perfect hash function. For this type of set of congruences any integer solution is legal, so the method offers total freedom of choice of the address for each key.

A dictionary is a data structure that allows the storage of a set S of distinct elements such that membership queries of the form "Is x in S?" as well as updates (i.e. "Add x to S" and "Delete x from S") can be performed efficiently. The FKS scheme considers only static sets where no updates to S are allowed. Another line of investigation by Dietzfelbinger et al. [DKM+88, DMadH92, DGMP92] attempts to use perfect hashing for maintaining dictionaries in real-time situations. By using certain classes of universal hash functions they show that the FKS probabilistic method can construct a perfect hash function in $\Theta(n)$ time, with the probability $1-O\left(\frac{1}{n^{\epsilon}}\right)$ [DGMP92]. The perfect hash function can be used to support a real-time dictionary (i.e., a dictionary which allows insertions and deletions of keys, with no knowledge about subsequent events) in expected constant time.

For other related developments in order preserving minimal perfect hash functions, which are practical for very large databases, see [FCDH91, FHCD92]. A considerable body of literature exists on minimal and order preserving hash functions and a complete discussion is beyond the scope of this survey. An overview of some of the results outlined above can be found in [MadH90].

Majewski, Wormald, Havas and Czech [MWHC93] have classified numerous algorithms for perfect hashing into four different broad categories. The first category is comprised of algorithms that rely on number theoretic methods to determine a small number of numeric parameters. The very first discussion of perfect hashing, by Sprugnoli [Spr77], falls into this category. Jaeschke's reciprocal hashing is another example from this category [Jae81].

The second category consists of perfect hash functions that use segmentation of keys. In these algorithms, the keys are first distributed into buckets by a non-perfect hash function. Perfect hash functions are then computed and used for keys in each bucket. The FKS scheme described earlier falls in this category.

The third category of perfect hashing schemes uses some kind of backtracking procedures to search through the space of all possible functions — typically an ordering heuristic is used to cut down the search space — in order to find a perfect hash function [FHCD92]. Finally, the fourth category consists of algorithms that map the given n keys into a $n \times n$ matrix and use matrix packing algorithms to compress the 2-D array into linear space [Meh84a].

All four categories of perfect hashing algorithms are rich in probabilistic methods. For examples of algorithms from each category, we refer the reader to [MWHC93], an excellent guide to a whole panoply of perfect hashing schemes that have appeared in the literature.

Perfect hashing has recently found application in the area of hardware design. In [RP91], perfect hash functions are used to construct a simple associative memory. Gupta [Gup93] uses it for response checking in digital circuit test. In both cases, random search is used to compute a perfect hash function for a given set of keys. This hash function is then implemented in hardware and its constant time, collision-free indexing property is used to access a pre-arranged table of values.

The Nearest Neighbors problem considered next illustrates the technique of random sampling, which is at the heart of many randomized algorithms in computational geometry.

2.5 The Nearest Neighbors Problem

We describe Rabin's probabilistic algorithm for the Nearest Neighbors problem, one of two probabilistic algorithms Rabin presented in his seminal paper [Rab76]. The other, a probabilistic algorithm for primality testing, was the topic of Section 2.2.

Consider a finite set $S = \{x_1, ..., x_n\}$ of points in l-dimensional space, i.e., $S \subset \Re^l$, where $\Re$ denotes the reals. The *Nearest Neighbors* problem is to find a pair of points x_i, x_j such that

$$d(x_i, x_j) = \min\{d(x_p, x_q), 1 \leq p < q \leq n\}, \tag{16}$$

where $d(x_i, x_j)$ is the usual distance function on $\Re^l$. Notice that x_i cannot equal x_j and that there may be more than one such pair of nearest neighbors in S. We refer to the distance separating nearest neighbors in a set S as $\delta_{min}(S)$.

A brute-force algorithm for Nearest Neighbors computes all the $n(n-1)/2$ relevant mutual distances and their minimum. A recursive algorithm in [Ben80] requires $O(n \log n)$ distance computations in both the average and worst case. Rabin's probabilistic algorithm, under a certain reasonable assumption about the problem input (discussed below), has an expected running time of $O(n)$ and thus outperforms any known sequential algorithm. This algorithm, unlike his algorithm for primality testing, is guaranteed to produce the correct answer.

The basic idea behind Rabin's algorithm is one of divide-and-conquer: decompose the set of points S into clusters, and look for nearest neighbors within each cluster. Let

$$S = S_1 \cup S_2 \cup ... \cup S_k \tag{17}$$

be a *decomposition* D of S, and n_i the cardinality of S_i. Let $N(D)$ be a *measure* of D, defined as

$$N(D) = \sum_{i=1}^{k} \frac{n_i(n_i - 1)}{2}. \tag{18}$$

If it is known that a nearest neighbor pair falls within one of the S_i, then $N(D)$ represents the number of distance computations needed to find the nearest neighbors of S: simply use

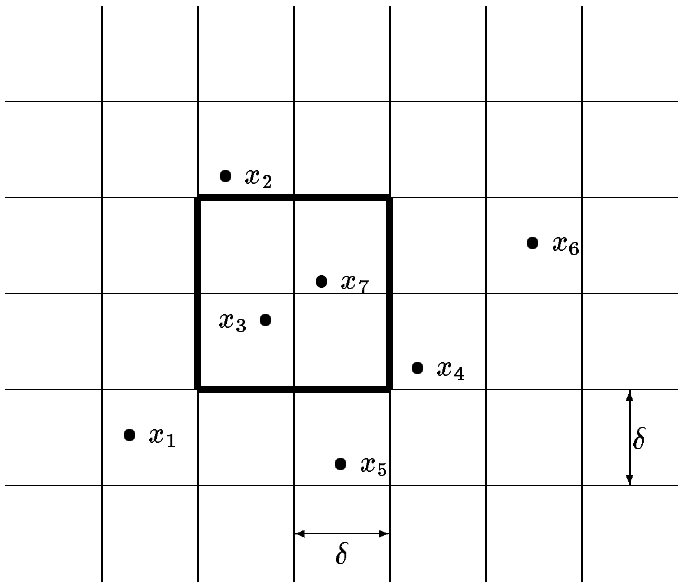

Figure 4: Pictorial explanation of Rabin's Lemma 1.

brute force within a cluster and then compare the nearest neighbors of each cluster. Central to the algorithm then is how to compute, in $O(n)$ time, a "desirable decomposition" D of S, such that a nearest neighbor pair belongs to the same cluster of D and $N(D) = O(n)$. As clarified below, the use of randomization is key to solving this problem.

In the two-dimensional case, a desirable decomposition can be obtained by first enclosing the points of S in a square lattice ? of mesh-size δ. It is not difficult to see that by choosing $\delta \geq \delta_{min}(S)$ we are guaranteed that, at worst, nearest neighbors x_i, x_j lie on squares of ? with a common corner. By doubling the mesh-size, we can hope to obtain a lattice in which these points will certainly lie within a *single* square. But to ensure that *all* adjacent squares of ? are tiled by a single 2δ-by-2δ square, we need to construct four lattices of mesh-size 2δ. Assuming, without loss of generality, that S is a subset of the non-negative quadrant, then the lower, left-hand corners of these lattices should be placed at locations $(0, 0)$, $(0, \delta)$, $(\delta, 0)$, and (δ, δ).

The proof that this lattice-based technique for decomposing S works as advertised, is given in Lemma 1 of [Rab76]. An example of this proof, also from [Rab76], is shown in Figure 4. Here x_3 and x_7 are nearest neighbors, and δ is greater than or equal to the distance between them. Doubling δ encloses the pair in a single square. This argument generalizes to any dimensional space.

We now know that δ, the initial mesh size, should be chosen large enough so that nearest neighbors at worst fall in adjacent squares. On the other hand, we still need to choose δ small enough so that $N(D)$ is $O(n)$, to obtain an efficient algorithm. Rabin used random sampling to arrive at such a δ. In particular, he showed that if δ is chosen to be $\delta_{min}(S_1)$, where S_1 is a randomly chosen subset of S such that $|S_1| = n^{2/3}$, then with a very high probability[6] the measure of the decomposition induced by the lattice of mesh-size δ will be $O(n)$ (Theorems 6 and 7 of [Rab76]). Intuitively, this random sample S_1 of S is large enough in size so that a grid of mesh-size δ will contain a small number of points within any lattice square. Thus, we have algorithm *NearNeb* for the Nearest Neighbors problem:

```
NearNeb {
    $S_1$ := randomly chosen subset of $S$ such that $|S| = n^{2/3}$
    $\delta$ := $\delta_{min}(S_1)$ (* how to do this is described below *)
    $\Gamma$ := square lattice of mesh size $\delta$ and origin (lower left-hand corner) at
          $(0,0)$, enclosing the points of $S$
    $\Gamma_1,\ldots,\Gamma_4$ := four lattices with origins $(0,0)$, $(0,\delta)$, $(\delta,0)$ and $(\delta,\delta)$,
                  respectively, derived from $\Gamma$ by doubling mesh size to $2\delta$
    FOR i := 1 TO 4 {
          find the decomposition $S = S_1^{(i)} \cup \cdots \cup S_{k_i}^{(i)}$ induced by $\Gamma_i$
          FOR j := 1 TO $k_i$
                $(x_j^{(i)}, y_j^{(i)}) :=$ nearest neighbor pair within lattice square $S_j^{(i)}$
    }
    $(x,y)$ := nearest pair in $\{(x_j^{(i)}, y_j^{(i)}) | 1 \le i \le 4, 1 \le j \le k_i\}$
}
```

To show the expected running time of $O(n)$, we first observe that $\delta_{min}(S_1)$ can be computed by invoking the algorithm recursively for a second time. A subset S_2 of S_1 is randomly chosen so that $|S_2| = |S_1|^{2/3} = n^{4/9}$. The brute-force technique can now be used to compute $\delta_{min}(S_2)$ in time $O(n)$ without resorting to any further recursion.

[6] To be precise, Rabin proved that this probability is at least $1 - 2e^{-cn^{1/6}}$, where $c = \sqrt{2\lambda}$ for $\lambda > 0$ a constant.

Next, consider the cost of finding the decompositions induced by the $?_i$. Rabin showed that if n and the x_i, normalized to integers with respect to 2δ, are within "appropriate ranges," then hashing can be used to find the decompositions in expected time $O(n)$. Otherwise, sorting is needed and takes $O(n \log n)$ time. Rabin argued that, in practice, hashing is almost always applicable.

We have previously argued that the expected value of $N(?_i)$, $1 \le i \le 4$, is $O(n)$, and hence the total number of distance computations required is $O(n)$. This gives us the desired running time of $O(n)$ for the algorithm.

There is a small probability that the remaining $n - n^{2/3}$ points not in the sample S_1 will cause the algorithm to behave inefficiently. In the worst case, S_1 will contain widely spaced points, resulting in a δ that is so large that all $n - n^{2/3}$ points not in S_1 fall into the same square of the grid. As a result, the partition of S will consist of set S_1 with $n^{2/3}$ points and the set S_2 with the remaining $n - n^{2/3}$ points. Using brute-force distance computation on the set S_2 will require $O(n - n^{2/3})^2$ or $O(n^2)$ time.

The Nearest Neighbors problem has illustrated the power of random sampling: an algorithm was found that almost always outperforms all known conventional algorithms for the problem. The next problem we consider — interactively checking the correctness of any program that purportedly solves the graph isomorphism problem — provides another example of the input randomization technique.

2.6 Interactive Probabilistic Proofs

Two important requirements of any proof system — a collection of axioms and inference rules used for proving statements about some domain of discourse — are completeness and soundness. *Completeness* refers to the ability to prove all theorems (i.e., all true statements) while *soundness* requires that the negation of a theorem is never a theorem. Thus, the ability to generate proofs and to verify them can be seen as complementary tasks. Typically, verification is simpler.

Traditionally, *P* has been considered the class of problems that can be *solved* efficiently, i.e., in polynomial time, and *NP* has been considered the class of problems that can be *verified* efficiently, i.e., in nondeterministic polynomial time. Recent discoveries, however, of efficient polynomial-time *randomized* algorithms for a large number of problems (such as the ones discussed in this survey) have led to a new notion of efficient computation, viz., the class *RP* of problems that can be solved in randomized polynomial time. Likewise, a new notion of efficient verification has emerged, viz., the class *IP* of problems that can be verified through the use of an *interactive probabilistic proof system*. We will have more to say about *RP* in Section 4. This section examines the concept of interactive probabilistic proof system and its applications.

In an interactive probabilistic proof system (interactive proof system, for short), an all-powerful *prover* tries to convince a skeptical *verifier* that it has a solution to a difficult problem. The prover's unlimited computational power allows it to solve such problems "with ease." For example, a prover can potentially find a Hamiltonian path in a graph, or determine if two graphs are isomorphic. The verifier, on the other hand, is required to be a polynomial-time randomized algorithm.

The prover and the verifier engage in a dialogue in which the verifier can toss coins and ask the prover to solve specific instances of the problem in question. The prover is only expected to provide solutions to these instances and nothing else. It is required that the total length of the messages sent back and forth between the prover and the verifier be bounded by a polynomial in the length of the input. The objective of the verifier is to convince itself that the prover does in fact have a solution to the problem.

Independent formalizations of interactive proof systems by Goldwasser, Micali and Rackoff [GMR89], and Babai and Moran [BM88, Bab85], which have been shown to be equivalent [GS89], allow a polynomial-time verifier to toss coins and arbitrarily interact with the prover. In [GMR89], the outcomes of the coin tosses made by the verifier are hidden from the prover. In [BM88], the proof system is considered as a game played between two players called Arthur and Merlin. Once again, Arthur and Merlin (the verifier and the prover, respectively) can toss coins and can talk back and forth. However, in this proof-system, unlike that in [GMR89], all coin tosses made by the verifier are seen by the prover. These formalizations have led to the emergence of a hierarchy of probabilistic complexity classes that generalizes *NP* [BM88].

One can also view an interactive proof system in complexity theoretic terms where the prover tries to convince a probabilistic verifier that a string w is in a language L. Such a proof system yields probabilistic proofs since the verifier may accept or reject w based on overwhelming statistical evidence rather than on certainties. Recent years have witnessed a multitude of such complexity theoretic results. For example, Ben-Or et al. in [BOGKW88] proposed a *multi-prover* interactive proof model. Using this model, Babai et al. [BFL90] proved that the class of languages that has a two-prover interactive proof system is nondeterministic exponential time. In his paper entitled "$IP = PSACE$," Shamir [Sha92b] showed that the set of problems for which interactive protocols exist is precisely the set of problems which are solvable within polynomial space on a Turing machine. A key result for proving $IP = PSPACE$ (and also, $MIP = NEXP$ [BFL90]) is by Lund et al. [LFKN90] who presented a new algebraic technique for constructing interactive proof systems and proved that every language in the polynomial time hierarchy has an interactive proof system.

An interactive proof system must satisfy *probabilistic* notions of soundness and completeness:

Completeness. if $w \in L$ then, with very high probability the interaction between the prover and the verifier must result in the verifier concluding that w is indeed in L;

Soundness. if $w \notin L$ then, with very high probability, at the end of the protocol the verifier must conclude that w is not in L.

The proof must be sound even if the prover acts maliciously and deliberately tries to fool the verifier. Several properties of interactive proof systems concerning completeness and soundness, and methods for constructing them are investigated in [FGM$^+$89]. Clearly, ruling by probabilistic evidence means relaxing the completeness and correctness criteria. However, it does lead to interesting applications such as program testing [BR88, BK89, BLR90].

For an example of how of an interactive proof system — in particular, the verifier component of the proof — can be used to test the correctness of a program, consider the problem of *graph isomorphism*. The reader should recall that the exact complexity of graph isomorphism is not known: while, to date, no polynomial-time algorithm for this problem has been discovered, a proof that it is NP-complete has been equally elusive. The following efficient procedure for checking the validity of a graph isomorphism program is due to Blum, Raghavan, and Kannan [BR88, BK89]. It is based on an interactive proof system for graph non-isomorphism by Goldreich, Micali and Wigderson [GMW91].

Given a program `P` that purportedly solves the graph isomorphism problem and two graphs `G` and `H`, the verifier wishes to determine whether `P` invoked on `G` and `H` (denoted `P(G,H)`) gives the correct result. The verifier *GI-Verify*, whose pseudocode is now given, operates in a randomized and interactive manner.

```
GI-Verify {(* Inputs:  a program P and graphs G and H *)
IF P(G,H) = true THEN{
      attempt to establish the isomorphism
      IF successful THEN RETURN "P is correct"
      ELSE RETURN "P is buggy"}
ELSE{
      REPEAT $k$ times{
            toss a fair coin
            IF coin = head THEN{
                  generate a random permutation G$'$ of G
                  IF P(G, G$'$) = false THEN RETURN "P is buggy"}
            ELSE{
                  generate a random permutation H$'$ of H
                  IF P(G, H$'$) = true THEN RETURN "P is buggy"}
      } (* end REPEAT *)
      RETURN "P is correct"}
}
```

GI-Verify starts by invoking `P(G,H)`. If `P` pronounces `G` and `H` to be isomorphic (i.e., `P(G,H) = true`), the verifier's task is simple. It attempts to determine the correspondence between the vertices of `G` and `H` (how this is done will be described shortly), and returns `correct` or `buggy` accordingly. If, on the other hand, `P` pronounces `G` and `H` to be non-isomorphic (i.e., `P(G,H) = false`), `V` will put `P` through a series of tests. Should `P` fail any one of these tests, `V` can conclude that `P` is `buggy`. Otherwise, `V` can conclude, with a high degree of confidence, that `P` is `correct`.

Consider the case `P(G,H) = true` first. The verifier can establish a 1-to-1 correspondence between the vertices of `G` and `H`, assuming that `P` is correct in pronouncing `G` and `H` to be isomorphic, as follows. Starting with `G`, arbitrarily number the vertices of `G` and `H` from 1 to n. Attach a clique of $n+1$ vertices to node number 1 of `G` to obtain the graph `G1`. Successively, attach a similar clique to each node i in `H` to obtain `H`i, and test if `P(G1,H`i`) = true`. Clearly, if `G` and `H` are isomorphic, and if node 1 in `G` can be mapped to node i in `H`, then `P(G1,H`i`)` will return `true`. Thus, if `P` returns `false` for all i, `P` is buggy. On the other hand, if `P(G1,Hi) = true` for some i, map node 1 of `G` to node i of `H`. Repeat this procedure for each node $j \in [1 \dots n]$ of `G`. At any point, the inability to find a corresponding node in `H` reflects an error in program `P`. On the other hand, if all the vertices in `G` can be mapped to those in `H` then the verifier can easily test if the mapping is an isomorphism and determine if the original answer `P(G, H) = true` was correct.

Consider the case `P(G,H) = false` next, i.e., `P` declares that `G` and `H` are not isomorphic. In this case, the verifier relies on simple random choice and input randomization as follows. It puts `P` through a series of tests or *rounds*. In each round, `V` tosses a fair two-sided coin to randomly choose between `G` and `H`; randomly permutes the names of the vertices in the selected graph to obtain a graph `K` that is isomorphic to the selected graph; and then invokes `P(G,K)`. We will refer to `K` as `G′`, if the selected graph is `G`, and as `H′` is the selected graph is `H`.

There are two cases depending on whether or not `P` is correct. If it is, i.e., `G` and `H` are actually non-isomorphic, then in each round we should have `P(G, G′) = true` when `G` is selected, and `P(G, H′) = false` when `H` is selected. Thus, in just a very small number of rounds, the verifier can gain a high degree of confidence in the correctness of `P` should it respond correctly in each round.

On the other hand, if `P` is buggy, i.e., `G` and `H` are isomorphic, it has no way of distinguishing between `G′` and `H′`. This is because `G′` and `H′` are isomorphic and are both drawn from the same distribution (essentially they are random permutations of the same graph). Since `P` does not know whether `G′` or `H′` is being passed as the second argument, the only way it can distinguish them is by chance. The probability therefore of `P` responding correctly (i.e., "yes" to `P(G, G′)` and "no" to `P(G, H′)`) k straight times is only 2^{-k}. Therefore, the verifier should only need a few rounds to determine that `P` is buggy.

The verifier makes use of randomization to its advantage at two crucial junctures in the above algorithm. First, it generates random permutations `G′` and `H′`. If `G` and `H` are isomorphic there is no way of telling `G′` and `H′` apart. In addition, it randomly passes `G′` or `H′` as the second argument in each iteration thereby taxing the claimed ability of `P` that it can test for graph isomorphism. The trick is so effective that it will catch `P` even if it is maliciously coded and is designed specifically to fool the verifier.

The above example illustrates the power of input randomization in program testing and interactive proof systems. The reader is referred to [BR88, BK89] for more probabilistic checkers for problems such as matrix multiplication, sorting and several problems in group theory.

It is interesting to note that in the above example, *GI-Verify* was able to do its task without having to solve the graph isomorphism problem in any sense. Also, if the graphs are isomorphic, then the verifier can construct the 1-1 map between the vertices of the two graphs (i.e., it gains more information than a simple yes/no answer about the isomorphism question). However, if they are non-isomorphic, the verifier gains no additional knowledge, other than the fact that they are non-isomorphic, about how this conclusion was reached. This latter property is crucial to the notion of *zero-knowledge proofs* described next.

Zero-Knowledge Proofs

Sometimes, an additional requirement is imposed on the prover, viz., that it completely hide the details of its solution from the verifier. In this case, the proof is referred to as a *zero-knowledge proof* [GMR89, BM88, Bab85, KMO89, GMW91] because, even though the verifier has an efficient means of verifying responses provided by the prover, at the end it has learned nothing except that the prover is right or wrong.

The concept of zero-knowledge proof has turned out to be especially useful in complexity theory [For87, BHZ87] and cryptography [GMW87, CCD88, BOGW88, BC86]. Various notion of zero-knowledge, a classification of these notions, and several related topics appear in [Ore87, FLS90, KMO89]. Some complexity theoretic implications of systems that admit zero-knowledge proofs are discussed in [AH91, For87, GMW91].

Truly Zero-Knowledge and Multi-Prover Interactive Proofs

Zero-knowledge proofs, in the traditional sense, reveal one bit of information to the verifier, viz. $w \in L$ or $w \notin L$. In [FFS87], a notion of *truly zero-knowledge proof* is proposed where the prover convinces the verifier that it knows whether w is or is not in L, without revealing any other information. Thus, at the end of interaction, the verifier only gains knowledge about the state of prover's knowledge and no information about the original membership problem.

Ben-Or et al. [BOGKW88] propose a multi-prover interactive-proof model. In their model, two provers jointly agree on a strategy and then try to convince the verifier, in a polynomially bounded number of interactions, that a certain statement is true. Communication between the provers is disallowed while they interact with the verifier. The authors are able to prove several interesting results without making any intractability assumptions.

Noninteractive Zero-Knowledge Proofs

A zero-knowledge interactive proof system typically has three key features that distinguish it from a traditional proof. The first is the ability of the prover and the verifier to interact with each other. Secondly, the verifier can toss coins that are hidden from the prover, which means there is an element of "hidden randomization". Finally, the prover has the ability to solve a hard problem that the verifier cannot solve directly. Thus, the prover embeds in its proof the computational difficulty of some other problem. As noted by Blum et al. in [BDMP91], this requires a rather rich set of conditions to be present before a zero-knowledge interactive proof can be devised for a problem.

Another notion that is gaining popularity is that of *noninteractive zero-knowledge proofs* first proposed by Blum, Feldman, and Micali [BFM88]. A notion of non-interactive zero-knowledge proofs based on a weaker complexity assumption than that used in [BFM88] is presented in [DSMP87]. Most of the work to date is summarized in [BDMP91].

In interactive zero-knowledge proof-systems, the prover P interactively proves to the verifier V that a certain theorem is true without giving away the details of the proof. In non-interactive zero-knowledge systems, as the name implies, interaction is forbidden: P writes down the proofs and mails it to V for verification under the assertion of zero-knowledge. Instead of interaction, P and V are allowed to share a short random string. While such a concept of "shared randomness" has been used by others (see, for example, [GS89]), shared random strings represent a much weaker requirement than most others (e.g., public coin tosses) used in the literature. As observed in [BDMP91], proofs using shared randomness do not rely on foiling the adversary by the unpredictability of the coin tosses, as has been the case so far, but rather on the "well mixedness" of the bits of the shared random string.

This concludes our survey of sequential randomized algorithms. The next section will consider distributed randomized algorithms.

3 Distributed Randomized Algorithms

In the second half of our survey we look at several randomized algorithms for *distributed computing*, viz., the Dining Philosophers' problem (*DinPhil*), the Communication Guard Scheduling Problem of CSP (*CommGuard*), the Leader Election problem (*LeadElect*), the Permutation Message Routing problem (*MsgRoute*), and the Byzantine Generals' problem (*ByzAgree*). We saw in the sequential case that randomization was used to obtain faster algorithms (sometimes at the expense of absolute accuracy), or to guarantee that the worst-case performance of an algorithm is no worse than the algorithm's expected performance. Similar motivations are also present in the distributed case, as demonstrated in this section. However an important additional concern is present: there are certain problems in distributed computing that have no deterministic algorithm—we have no choice but to toss coins. The probabilistic algorithm for the Dining Philosophers problem typifies this situation.

To obtain a notation for distributed algorithms, we augment the imperative language used in Section 2 with constructs for shared memory access and message passing. For the former we introduce the instruction `TEST&UPDATE`, which is used as follows:

```
result := TEST&UPDATE(flag, true_value, false_value)
```

The effect of this command is, in one instruction cycle, to assign to the variable `result` the old value of the shared boolean variable `flag`, and to assign to `flag` the value `true_value` if its old value was true and `false_value` otherwise. For example, besides returning the old value of variable `flag`, the statement `result := TEST&UPDATE(flag, FALSE, TRUE)` inverts the value of `flag`.

Because everything happens in one instruction cycle, the `TEST&UPDATE` operation cannot be interrupted, and access to shared variables is therefore atomic. `TEST&UPDATE` is also assumed to behave fairly in the sense that no process is ever indefinitely denied access to a shared variable in favor of other processes. As such, the phenomenon of "process starvation" is avoided.

Unconditional updates to shared variables will be expressed using the standard assignment operator. Such assignment is also assumed to be atomic and fair.

For message passing, we introduce constructs of the form

$\texttt{SEND}(\texttt{expr}_1, \ldots, \texttt{expr}_k)$ TO P

$\texttt{RECEIVE}(\texttt{x}_1, \ldots, \texttt{x}_l)$ FROM P

The send command executes asynchronously and results in the transmission of the values of the expressions $\texttt{expr}_1, \ldots, \texttt{expr}_k$ to the named process P. The receive command inputs values for the variables $\texttt{x}_1, \ldots, \texttt{x}_l$ which have previously been transmitted by process P. The underlying communication medium is assumed to be faultless in that messages are received intact and in the order of transmission.

3.1 The Dining Philosophers Problem

We describe the randomized algorithm of Lehmann and Rabin [LR81] for the well-known Dining Philosophers problem. The problem, posed originally in [Dij71], is an anthropomorphized resource allocation problem, and is described in [Hoa85] essentially as follows:

> There once were n philosophers P_0, P_1, ..., P_{n-1} seated around a circular table in a clockwise fashion. To the left of each philosopher laid a golden fork, and in the center stood a large bowl of spaghetti, which was constantly replenished.
>
> A philosopher was expected to spend most of his time thinking; but when he felt hungry, he picked up his own fork on his left, and plunged it into the spaghetti. But such is the tangled nature of spaghetti that a second fork is required to carry it to the mouth. The philosopher therefore had also to pick up the fork on his right. When he was finished he would put down both his forks, and continue thinking. Of course, a fork can be used by only one philosopher at a time. If the other philosopher wants it, he just has to wait until the fork is available again.

Additionally, any algorithm that coordinates the philosophers in the above-described manner must be *deadlock free*—if at any time there is a hungry philosopher, then eventually *some* philosopher will eat; and *lockout free*—every hungry philosopher eventually gets to eat.

Many deterministic solutions based both on shared memory [Hoa74] and message-passing communication [Hoa85] have been proposed. However, none of these algorithms are both: (1) *fully distributed*, i.e., devoid of central memory or a central process with which every other process can communicate; and (2) *symmetric*, i.e., all processes execute the same code and all variables, local and shared, are initialized identically. Moreover, processes in a symmetric algorithm are unaware of their identities, and therefore cannot compare their process id with the id of another process.

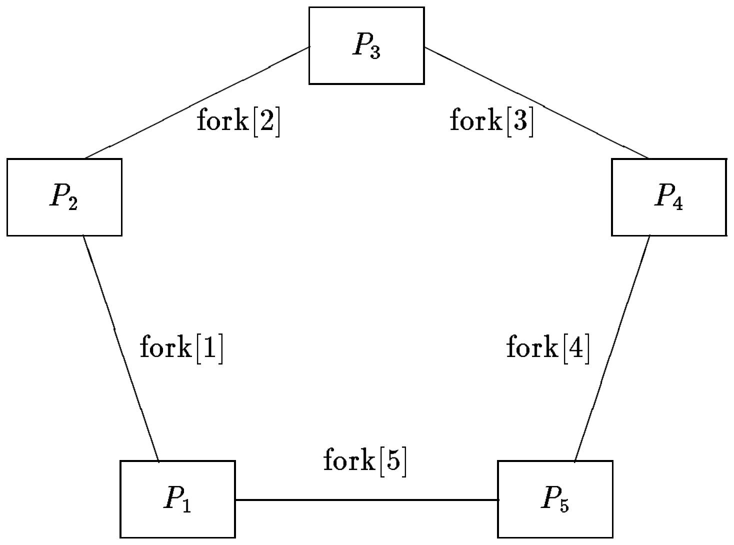

Figure 5: Arrangement of philosophers and forks in the Dining Philosophers Problem.

In fact, it is shown in [LR81] that no fully distributed and symmetric deterministic algorithm for Dining Philosophers is possible. Intuitively, this is due to the existence of an adversary scheduler that can continually thwart the philosophers in their attempts to reach agreement on who is to eat next, thereby leading to deadlock. For example, under the influence of an adversary scheduler, the philosophers could behave as follows: (1) all n philosophers become hungry simultaneously; (2) they each pick up their right fork, again in synchrony; (3) because of the symmetry and the fact that each philosopher's behavior is strictly deterministic, they have no choice but to put down their forks and try again. Furthermore, the clever adversary scheduler can cause this scenario to reoccur without end, resulting in a deadlock. The problem then is one of "breaking symmetry" and this is precisely the reason for introducing randomness into the behavior of the philosophers.

In Lehmann and Rabin's algorithm, presented below as algorithm *DinPhil*, the simple yet key use of randomization is in whether a philosopher waits to first obtain the left fork or the right fork. Communication among philosophers is done strictly in a ring fashion and uses one shared variable, `fork-available[i]`, for each $P_i - P_{i+1}$ pair. All additions and subtractions are to be interpreted modulo n, where n is the number of philosophers. Moreover, `fork-available[i]` is accessed only via the `TEST&UPDATE` instruction or via the unconditional assignment operation for shared variables. The configuration of philosophers and forks for the case $n = 5$ is illustrated in Figure 5.

The algorithm can be shown to be deadlock-free in the following sense: if at any time there is a hungry philosopher, then, *with probability 1*, some philosopher will eventually eat. The proof of this result rests on the fact that the coin tosses made by philosophers are independent random events. Thus, even if the adversary scheduler tries to bring on deadlock, with probability 1, a combination of tosses will eventually arise that enables some philosopher to obtain two forks. Note that the algorithm is indeed symmetric as the index attached to a philosopher is for external naming only; philosophers themselves are not aware of their own names.

```
DinPhil { (* algorithm for $P_i$ *)
    WHILE TRUE DO{
        (* thinking section *)
        trying := true
        WHILE trying DO{
            choose $s$ randomly and uniformly from {0,1}
            wait until TEST&UPDATE(fork-available[$i-s$], FALSE, FALSE)
            IF TEST&UPDATE(fork-available[$i-\overline{s}$], FALSE, FALSE) THEN
                trying := FALSE (* $\overline{s}$ = complement of s *)
            ELSE fork-available[$i-s$] := TRUE
        }
        (* eating section *)
        fork-available[$i-1$], fork-available[i] := TRUE
    }
}
```

Algorithm *DinPhil* is not lockout-free; intuitively, a greedy philosopher P_i can prevent neighbor P_{i+1} from ever eating by continually beating P_{i+1} in their race to pick up their shared fork. The algorithm can be made lockout-free by adding, for each pair of adjacent philosophers P_i, P_{i+1}, two pairs of variables. One pair allows P_i to inform P_{i+1} of its desire to eat (and vice versa), and the other pair is used to indicate which of P_i and P_{i+1} ate last. Details can be found in [LR81].

Lehmann and Rabin's randomized algorithm was one of the first for distributed computing, and clearly illustrated the importance of tossing coins in a new setting—without this capability, fully distributed and symmetric algorithms may not even exist for certain problems. The next algorithm we consider, *CommGuard*, also illustrates the power of symmetry breaking through randomization.

3.2 Communication Guard Scheduling

In this section we present the randomized algorithm of Francez and Rodeh [FR80] for scheduling communication guards in a CSP-like language. In CSP [Hoa78], processes execute asynchronously and exchange data by a "handshaking" style of communication. There are two types of communication statements or commands (to use CSP terminology) in the language: *input* statements of the form $P\,?\,x$ and *output* statements of the form $Q\,!\,e$. An input statement inputs a value from the named process (P) into a local variable (x), while an output statement outputs the value of an expression (e) to the named process (Q). Thus, for example, the simultaneous execution of the statement $P_2\,?\,x$ by process P_1 and the statement $P_1\,!\,e$ by process P_2 results in the value of expression e being assigned to variable x (i.e., $x := e$). The phenomenon is sometimes referred to as "distributed assignment." Input and output statements, such as those in the example, that name each other are said to be *complementary*.

Statements within a process, e.g., assignment, iteration, and communication, can be executed nondeterministically through the use of a construct called the *guarded command*, having the following syntax:

$$[G_1 \Longrightarrow S_1 \square \cdots \square G_n \Longrightarrow S_n]$$

Each statement S_i has an associated communication statement G_i, called its *communication guard*, such that S_i is eligible for selection only if the process named in its communication guard is likewise willing to communicate.

The problem of *communication guard scheduling* can now be stated as follows: Given a set T of processes each currently waiting to execute a guarded command, construct a set of one or more pairs of processes (P, Q) from T such that P and Q have complementary communication guards, and no process appears in more than one pair.[7]

[7] A more general statement of the problem would allow processes in T to be waiting to execute an unguarded communication statement, but such a statement can always be placed in a guarded command

For example, consider the system of processes

$$\begin{aligned} P_1 &= [P_2?x \Longrightarrow skip \,\Box\, P_3!v \Longrightarrow skip] \\ P_2 &= [P_3?x \Longrightarrow skip \,\Box\, P_1!v \Longrightarrow skip] \\ P_3 &= [P_1?x \Longrightarrow skip \,\Box\, P_2!v \Longrightarrow skip] \end{aligned}$$

where *skip* is the CSP notation for the no-op statement. Each process P_i is willing to receive a message from process P_{i+1}, or send a message to process P_{i-1}, where the addition and subtraction are performed modulo 3. There are three possible solutions to the guard scheduling problem in this case: the single pair of processes (P_i, P_{i+1}) is chosen such that P_i is receiving and P_{i+1} is sending, $1 \leq i \leq 3$. An unsatisfactory situation would arise if each process were allowed to decide to send, or if each process were allowed to decide to receive; this is tantamount to cyclic wait or deadlock.

As in the Dining Philosophers problem, an algorithm for guard scheduling must satisfy two correctness criteria. The algorithm must be *deadlock free*, i.e., if two processes P and Q wish to communicate with each other, then either P or Q will eventually participate in a communication (although not necessarily with each other); and *starvation free*, i.e., if a process P tries to communicate and infinitely often some process Q_i is willing to reciprocate, then P will eventually participate in a communication (the process Q_i need not be the same each time).

Several distributed implementations of guard scheduling have been proposed including [Sch78, Ber80, vdS81, Sch82, BS83]. Each of these algorithms must resort to some symmetry breaking technique such as priority ordering of processes [Sch78, Ber80, BS83], or timestamps [Sch78]. In fact, like the Dining Philosophers problem, the existence of a fully distributed and symmetric *deterministic* algorithm for guard scheduling can be shown to be an impossibility [FR80]. In the presence of symmetry, a fully distributed deterministic algorithm is susceptible to the scenario in which a solution exists but is never found. For example, processes may in a cyclic fashion issue communication requests to one another; due to symmetry, this same circular wait may reappear with every future attempt by the processes to establish communication. The lack of a fully distributed and symmetric deterministic algorithm for guard scheduling is indeed one of the reasons the designers of Ada [DoD83] chose an asymmetric rendezvous construct—nondeterministic choice in Ada exists only among the **accept** alternatives of a **select** statement.

We now describe the fully distributed and symmetric randomized algorithm of Francez

having one alternative.

and Rodeh [FR80]. (Other probabilistic algorithms for guard scheduling, which have "real time response", appear in [RS84].) The algorithm is given here as the iterative procedure *CommGuard*, which a process P invokes upon reaching a guarded command in order to schedule itself in a communication. Upon return, a communication link between P and one of the processes designated by P's current guarded command will have been established, and actual data transfer can then occur.

In order to simplify the presentation of the algorithm, we will assume that communication between processes is non-directional. That is, a process specifies only the name of a process in a communication guard and not the direction (i.e., input or output). Under this assumption, *CommGuard* can be implemented by providing each pair of processes a single shared boolean variable `flag`; thus, the algorithm is fully distributed.[8] All such `flag` have initial value `FALSE`. Access to shared variables is through the `TEST&UPDATE` instruction, the semantics of which was described in the introduction to Section 3.

```
CommGuard { (* To schedule communications *)
      trying := TRUE
      WHILE trying DO{
            randomly choose a partner with which to attempt a communication
            let flag be the shared variable between these two processes
            IF TEST&UPDATE(flag, FALSE, TRUE) THEN
                  trying := FALSE (* communication established *)
            ELSE{
                  wait t seconds
                  IF NOT(TEST&UPDATE(flag, FALSE, FALSE)) THEN
                        trying := FALSE (* communication established *)
                  ELSE {} (* try another partner *) }
      }
}
```

To gain some insight into the functioning of the protocol, consider two processes P and

[8]Without the simplifying assumption, two shared variables, $\texttt{flag}_{\texttt{ij}}$ and $\texttt{flag}_{\texttt{ji}}$, are needed for each pair (P_i, P_j) of processes. Variable $\texttt{flag}_{\texttt{ij}}$ is used to establish communication between P_i and P_j by matching an output guard of P_i with an input guard of P_j; $\texttt{flag}_{\texttt{ji}}$ is used in a symmetric fashion.

Q having complementary guards. Intuitively, P sets `flag` to true to inform Q of its desire to communicate. P will wait `t` seconds for Q to respond, which Q does by resetting `flag` back to false. If Q does not respond within this time interval, P will try to establish communication with another process. The "timeout interval" `t` is a predefined constant to the algorithm.

Randomization enters into the protocol in the choice of prospective communication partner. If a request to communicate with a process is not reciprocated within `t` seconds, the `WHILE` loop is iterated once again, at which point another partner is chosen *randomly*. This act of giving up on a potential partner and trying another is called the "retraction phase". `WHILE` loop iterations of this nature persist until, if possible, a communication channel has been successfully established.

There are two points in *CommGuard* where the variable `flag` needs to be tested and then immediately reset. These actions must be performed atomically within a process for the algorithm to function correctly. The `TEST&UPDATE` instruction is used for this purpose. Starvation is avoided as this instruction is also fair.

Algorithm *CommGuard* is deadlock and lockout free. The proofs are similar to those of the Dining Philosophers problem. The main point is that a combination of coin tosses that eventually enables two processes to establish communication can be shown to occur with probability 1. As described above, the coin tosses take place in the retraction phase of the algorithm and constitute a symmetry breaking technique. Symmetry breaking is also behind the algorithm for leader election presented next.

3.3 Leader Election

The coordination of the computers, or nodes, in a network is often the responsibility of a single, distinguished node. This node, called the *leader* of the network, may enforce mutual exclusion in accessing a shared resource, provide services required by other nodes, or serve other similar functions. If the leader fails, a new leader must be selected from among the surviving nodes of the network using an *election* algorithm. In this section we examine the randomized distributed algorithm of Itai and Rodeh [IR81] for leader election.

The problem of electing a leader can be stated as follows. Given a set of n identical processes $\{P_0, P_1, \ldots, P_{n-1}\}$ connected in a ring fashion (i.e., P_i talks to P_{i+1}, where subscript arithmetic is performed modulo n), elect one of these processes as the leader of the ring. At the end of the election, *all* processes must agree upon the identity of the leader. Additionally, an election algorithm must guarantee termination.

Most published leader election algorithms assume that asymmetry exists in the ring to the extent that individual processes have unique *names*, often chosen from some totally ordered set of names. The problem of leader election is then reduced to the problem of picking the process with the smallest, or largest, name. See, for example, [CR79, Pet82].

Several authors [Ang80, IR81] have investigated the consequences of the absence of such totally ordered names on election algorithms. Angluin [Ang80] has shown that there exists no deterministic algorithm to carry out elections in a ring of identical processes. Angluin's argument is based on the observation that, in a deterministic framework, it is possible for an adversary scheduler to force all processes to be in identical states at all times. For example, the adversary scheduler can dictate that every message is in transit for exactly the same amount of time, and that processes proceed in lock-step. Since processes are identical, they start out in the same state, and, by induction, end up in identical states after any k computation steps. Thus any potential progress toward the completion of an election is thwarted by the symmetry of the ring.

Thus, we once again need to toss coins to solve the problem. In the randomized algorithm *LeadElect* of Itai and Rodeh [IR81], the pseudocode of which is given below, each process is equipped with an independent random number generator. Additionally, all processes know n, the size of the ring. The ring is presumed to preserve message order in that two messages sent from a process to its neighbor are received in the same order in which they were sent.

The algorithm is easier to understand if one assumes that the processes operate synchronously in lock-step, and that each transmitted message reaches its destination before the processes execute their next computation step. Each process P_i begins by picking a random name, an integer in $\{1, \ldots, K\}$ for some constant $K > 1$. P_i then propagates its name around the ring, copying and forwarding names of other nodes that it receives. P_i determines the names chosen by all other processes by the time it receives n messages. The nth message received by a process is the one it sent out initially.

Each process determines from its list of names whether at least one process has chosen a unique name, i.e., one that was not chosen by any other process. The process with the largest unique name is elected the leader. If no process picked a unique name, the processes repeat their election attempt. Each attempt is called a *round*.

```
LeadElect { (* algorithm used by process $P_i$ in a ring *)
```

```
      (* s :  a list of names *)
      REPEAT {
            set s to empty
            name := a random number between 1 and K
            REPEAT $n$ times{
                  add name to s
                  SEND(name) TO $P_{i+1}$
                  RECEIVE(name) FROM $P_{i-1}$
            }
      }
      UNTIL at least one name in s is unique
(* the process that picked the largest unique name is the leader *)
}
```

Every time the processes pick random names for themselves, there is a non-zero probability p that at least one node picks a name that is chosen by no other node. (The exact value of p depends on the value of K and on the probability distribution of the random number generators.) The probability that the algorithm fails to terminate in i rounds is $(1-p)^i$, and the probability that the algorithm executes forever is

$$\lim_{k\to\infty}(1-p)^k = 0. \quad (19)$$

In other words, the algorithm will terminate with probability 1. The expected number of rounds for the algorithm to terminate is clearly $1/p$.

This algorithm can be improved in several ways. One way to improve the expected running time is to change the termination condition to examine the pattern of names in the entire ring to determine if an election is possible. For instance, if in a ring where $n = 5$, and processes P_0 and P_2 chose 1, while P_1, P_3 and P_4 chose 2, then the algorithm described above would procced to another round, since no single node chose a unique name. However, closer examination shows that leader election is possible in this situation: P_0 can be elected because it is the only process, whose immediate neighbors in the ring chose 2, that chose a 1. Itai and Rodeh provide a mathematical basis for the use of such techniques.

Leader election in a symmetric ring is one of a variety of problems where reasonably efficient probabilistic solutions can be found, even though a deterministic, symmetric solution is impossible. It is interesting to note that symmetric leader election in a ring with an *unknown* number of processes has no deterministic *nor* probabilistic solution that guarantees both termination and a non-zero probability of correctness. The reader is referred to Itai and Rabin [IR81] for a proof of this claim.

The next problem we consider, message routing in a network, shows how randomization can be used to reduce queueing delay and to improve resiliency to faults.

3.4 Message Routing

An important measure of the performance of any message routing algorithm is how well it solves the *permutation routing problem*. In permutation routing, each node in a network is the origin of a single message destined for another node in the network, subject to the constraint that no two messages have the same destination. The problem is to devise a distributed algorithm to route the messages to their destinations with the minimum possible delay, with at most one message being transmitted over an edge at any time. Each instance of the problem can be viewed as a permutation π on the set of nodes, where $\pi(v) = w$ means that the message originating at v has to be delivered to destination w. This part of the survey is devoted to randomized algorithms for permutation routing.

In message routing algorithms, the normally accepted unit of delay is the time needed to transmit a single message from a node to its neighbor. The assumption is that the time taken by the nodes themselves to process individual messages and decide how they are to be routed is negligible when compared to message transmission delays. This is especially true if the nodes can do parallel processing.

The overall delay incurred by a permutation routing algorithm is obviously related to the underlying topology. For instance, the minimum delay in sending a message from one node to another depends on the length of the shortest path between them. Another type of delay can occur when implementing permutations: the routing algorithm may determine that a message needs to be transmitted over an edge that is already in use for transmitting another message. In this case, the message is often queued up for transmission at a later time. Such *queuing delays* should also be included in any measure of the total delay that a message suffers in transmission from its origin to its destination.

Deterministic permutation routing algorithms have the common drawback that they have poor worst-case performance. In other words, they behave badly on some specific permuta-

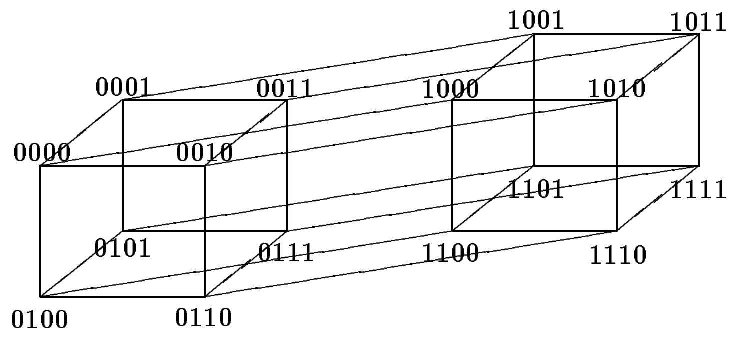

Figure 6: A 4-dimensional binary cube.

tions. In this section, we consider two algorithms that use randomization to break up such input dependencies: Valiant's [Val82] algorithm for the *n-cube*, and Aleluinas's [Ale82] algorithm for *shuffle networks*. A radically different approach, that of randomizing the interconnections between nodes, is also presented. This technique, when applied to multi-butterfly networks, has been shown to outperform conventional butterfly networks, particularly with respect to tolerance to node faults [Upf89, LM89, LLM90].

Message Routing on an n-Cube

Valiant [Val82] proposed the first permutation routing algorithm for an n-cube. His algorithm implemented *any* permutation, with high probability, in $O(\log N)$ time. An n-cube is a network architecture shaped like an n-dimensional cube having $N = 2^n$ nodes, and is often referred to as a (*n-dimensional*) *hypercube*.

We assume that each node of an n-cube is identified by an n-bit binary number v from 0 to $2^n - 1$. A 16-node 4-cube is shown in Figure 6. Two nodes can communicate with each other if their numbers differ in only one bit position or *dimension*.

To implement every permutation in $O(\log N)$ time with high probability, Valiant's algorithm requires each message to carry $O(\log N)$ bits of additional book-keeping information. The algorithm can implement both complete as well as partial permutations. No global synchronization is required (i.e., no help from a central arbiter is needed).

For convenience in describing the algorithm, we shall assume that the message originating at node v is labeled v. The algorithm operates in two phases. In the first phase, a message u is moved from its origin to a random intermediate destination v without regard for its ultimate destination w. The intermediate node v is chosen randomly: a fair coin with sides 0 and 1 is tossed for each of the n dimensions, and the message is moved along the edge in that dimension if a 1 shows up. Clearly, at the end of this procedure, a message may be in any node of the n-cube with equal probability.

The movement of messages to their actual destinations occurs in the second phase. In this phase each node that holds a message chooses at random a dimension in which the message needs to be moved in order to reach its destination, and transmits the message along that dimension.

The pseudocode of Valiant's algorithm appears below. In this algorithm, each message u has an associated set of book-keeping information $T_u \subseteq \{1, \ldots, n\}$. In the first phase, T_u consists of the set of dimensions along which possible transmissions have not been considered. In the second phase, T_u consists of the set of dimensions along which transmissions remain to be made in order for u to reach its destination. Also, each node v maintains a set of queues $Q_v(i), 1 \leq i \leq n$, containing messages to be transmitted from v to its neighbor in the ith dimension. This neighbor, denoted by $v||i$, is the node whose number is obtained by toggling the ith bit of the binary representation of v. The ith bit of the binary representation of number v is denoted by v^i.

In both phases, each node v maintains a set $Loose_v$ of messages that have been received by v but have not been assigned to any queue. A message u in $Loose_v$ with $T_u = \emptyset$ has v as its destination. The notation "Transmit v" means that for each non-empty $Q_v(i)$, v transmits the message u at the head of $Q_v(i)$ to node $v||i$ and causes u to be added to $Loose_{v||i}$. A phase is *finished* when for all messages u, $T_u = \emptyset$. Valiant's algorithm is said to finish successfully if both phases of the algorithm finish.

```
*MessageRoute Phase 1* { (* algorithm used by node $v$ *)
    $Loose_v$ := $\{v\}$;
    $T_v$ := $\{1, \ldots, n\}$;
    FOR $f$ := 1 to $F$ DO {
        FOREACH $u$ IN $Loose_v$ WITH $T_u \neq \emptyset$ DO {
            Pick $i \in T_u$
```

```
                $T_u$ := $T_u - \{i\}$;
                Pick $\alpha \in \{0, 1\}$;
                IF $(\alpha = 1)\{$
                    add $u$ to $Q_v(i)$;
                    $Loose_v$ := $Loose_v - \{u\}$;
                } (* end IF *)
            } (* end FOREACH *)
            Transmit $v$
        } (* end FOR *)
}
```

```
MessageRoute Phase 2 { (* algorithm used by node $v$ *)
        FOREACH message $u$ with destination $w$ at $v$ DO
            $T_u$ := $\{i | v^i \neq w^i\}$
        FOR $g$ := 1 to $G$ DO {
            FOREACH $u$ IN $Loose_v$ WITH $T_u \neq \emptyset$ DO {
                Pick $i \in T_u$
                $T_u$ := $T_u - \{i\}$;
                add $u$ to $Q_v(i)$;
                $Loose_v$ := $Loose_v - \{u\}$;
            } (* end FOREACH *)
            Transmit $v$
        } (* end FOR *)
}
```

The algorithm is synchronous in the sense that for each iteration of both phases, all nodes transmit concurrently, and that all transmitted messages are added to the *Loose* sets of the recipients before the recipients begin the next iteration. This restriction, however, can be relaxed [VB81].

Also, note that the two phases run for F and G iterations, respectively. It is clear that if G is too small, all messages might not reach their final destinations. Valiant shows that for both phases to finish successfully with probability greater than $1 - 2^{-Sn}$, for any constant S, F and G need be no greater than Cn, where C is a constant that depends on S. In other words, both phases of the algorithm terminate correctly in $O(n)$ time with probability $1 - 2^{-Sn}$, for any constant S. The assumption of course is that individual iterations of the algorithm in both phases run in constant time. Formally:

Theorem 3 *For any constant S, there is a constant C such that for $F = G = Cn$, both phases of Valiant's routing algorithm finish with probability greater than $1 - 2^{-Sn}$.*

In both phases, each message takes a *route* from an initial node to another node, where a route is defined as a path in the n-cube where no two edges traverse the same dimension. It is clear that no route is longer than n. Therefore, the theorem is proved once it is established that the queuing delays encountered along the routes are $O(n)$ with probability greater than $1 - 2^{-Sn}$.

Queuing delays can occur for a message u only if the route taken by other messages share common edges with the route taken by u. Analysis shows that for $C > 1$, the probability that any fixed route R shares edges with routes taken by Cn other messages is less than $e^{-Cn/4}$ in either phase of the algorithm. Therefore, queueing delays are also $O(n)$ provided each of the routes that intersect R causes no more than a constant delay with similarly high probability. This part of the proof involves the estimation of the probabilities at the tail end of a binomial distribution, and is one instance of the application of the powerful *Chernoff bounds* analysis technique.

The reader is referred to [Val82] for the detailed probabilistic analysis, but the Chernoff bounds are repeated here for completeness. If X is the number of heads in n independent tosses of a coin where the probability of a head in a single toss is p, then Chernoff's bounds state that

$$Prob\,[X \geq m] \leq \left(\frac{np}{m}\right)^m e^{m-np}$$

$$Prob\,[X \geq (1+\epsilon)np] \leq e^{-\epsilon^2 np/2}$$

$$Prob\,[X \leq (1-\epsilon)np] \leq e^{-\epsilon^2 np/3}$$

for any $0 < \epsilon < 1$, and $m > np$.

It is interesting to note that Valiant's results are obtained by deriving bounds on the probability that two routes intersect and on the probability that two routes share more than a given number of edges. No assumptions are made about how messages from a queue are sent. This means that the implementer is free to use any queuing discipline. The algorithm also has the advantage that each route can be chosen independently of any other route, i.e., no global book-keeping is needed.

Message Routing on Finite Degree Interconnection Networks

Valiant's algorithm is designed for hypercubes, which have the drawback that the degree of each node increases with the number of nodes in the network. Aleluinas [Ale82] extended Valiant's results to the common b-way *shuffle* networks, where each node has a fixed degree b, regardless of the size of the network.

For simplicity of exposition, let us assume b divides N, the number of nodes in the network. Then the network interconnections of a b-way shuffle network are as follows: Assuming the nodes are numbered from 0 to $N-1$, they are divided into N/b blocks, where the ith block consists of nodes $ib, ib+1, \ldots, ib+b-1, 0 \leq i \leq \frac{N}{b}-1$. Each node in block i is allowed to send messages to all nodes whose address modulo $\frac{N}{b}$ is i. Note that the communication paths are directed.

In such a network, there exist paths of length $\lceil \frac{\log N}{\log b} \rceil$ between any pair of nodes. However, the best deterministic routing algorithms known require $O(\log^2 N)$ time [LPV81] in the worst case because an appropriate choice of sources and destinations can cause congestion on individual communication lines.

Aleluinas [Ale82] uses randomization to overcome this input dependency. As in Valiant's algorithm, each node v chooses (with equal probability) an intermediate destination. However, the entire path to the intermediate destination is chosen by v from among the paths of length $\lceil \frac{\log N}{\log b} \rceil$ originating at v. Node v then sends its message along that path to its intermediate destination. This constitutes the first phase of the algorithm. Once a message has arrived at its intermediate destination, the intermediate destination picks, uniformly at random, a path of length $\lceil \frac{\log N}{\log b} \rceil$ leading from itself to the final destination. The message then follows this path. This constitutes the second phase of the algorithm. In both phases, the routing algorithm, unlike Valiant's, must enforce a queuing discipline: there must be only one output queue per node, and priority must be given to nodes that have traveled fewer hops, i.e., those that are late.

The delay of a message is $D_1 + D_2$, where D_i is the delay incurred in the ith phase. Analysis of one of the phases is sufficient, since the two phases mirror each other. There is statistically no difference between the delay of messages proceeding from distinct sources to random destinations, and the delay of messages moving to distinct destinations from sources chosen at random.

Assuming that it takes constant time to send a message, the expected delay of Aleluinas's routing algorithm is no more than μ, where

$$\mu = \frac{b}{b-1}\lfloor \log_b (b-1)N \rfloor$$

Note that μ is $O(\log N)$ when b is a constant. This matches the expected delay of Valiant's algorithm and is accomplished using a *fixed* number of edges per vertex. In addition, the probability that the delay exceeds $c\mu$ for any message is no more than

$$b^{-c\mu(1-O(1))}$$

where $O(1) \to 0$ as $c \to \infty$. Aleluinas has also analyzed the delay for the more general situation where multiple messages originate at each node. The reader is referred to [Ale82] for further details.

Both algorithms discussed above use the technique of distributed input randomization. By sending messages to randomly selected intermediate destinations, any pockets of congestions arising because of certain unfavorable permutations are avoided. This approach at first sight, appears to be unnatural as it may send messages which actually may be very close to their final destination to far away intermediate destinations. However, it is essential. For instance, in Valiant's algorithm, it can be shown that the second phase alone, though adequate for most permutations, does *not* terminate in $O(\log N)$ steps for some permutations.

Randomly Wired Multi-Butterfly Networks

Butterfly networks are used in many parallel computers, such as the BBN Butterfly and Thinking Machine's CM-5, to provide paths of length $\log N$ connecting N inputs to N outputs. For simplicity, N is usually taken to be a power of 2. The path between any input and output is of length $\log N$. These inputs and outputs could be processors, memory, or other resources. An instance of a butterfly network with $N = 8$ is shown in Figure 7. The inputs to the network are on the left, and the outputs of the network are on the right. Each node is a switch that accepts messages from its neighbors to the left and can send them to

Figure 7: An 8-input butterfly network

neighboring switches to the right. The interconnections in this butterfly are straightforward: each node i at level l can send messages to nodes i and j at level $l+1$, where j is the number whose binary representation differs from i in the $l+1$st bit position alone. For instance, in Figure 7, the switch in row 010 at level 0 can communicate with switches in rows 010 and 110 at level 1.

There is a simple greedy algorithm for message routing on a butterfly, best described by an example. In Figure 7, a message to destination 010 (regardless of the source) is routed as follows. The first edge the message traverses takes it to a node in the top four rows, so that the first bit of the row number, in this instance a 0, matches the first bit of the destination row. The second edge takes the message to a node in a row where the first two bits of the row number match the first two bits of the destination row, and the last edge takes it to its correct destination. In general, the ith edge ensures that bit positions 1 through i of the row that the message reaches match bit positions 1 through i of the destination row.

The main disadvantage with butterflies is that they are sensitive to edge or node failures. Another drawback is the possibility of congestion, which occurs at a node when two incoming messages need to be sent over the same outgoing edge. A common scheme that provides some protection against edge failures as well as some reduction in congestion is to make each edge capable of transmitting d messages concurrently, a technique called *dilation*, resulting in a *d-dilated* butterfly. In other words, each outgoing edge of the butterfly is replaced by a bundle of d edges. As in the butterfly, however, the shortest-length path between a given input and a given output still must go through the same sequence of nodes, and an adversary scheduler can take advantage of this fact to thwart routing algorithms. This is where randomization of wiring becomes an advantage. Radomized wiring is exploited in *multi-butterfly networks* [Upf89, LM89, LLM90]. Multi-butterflies are a generalization of both the butterfly and the dilated butterfly.

A butterfly network can be considered to be built from *splitters*, each of which in turn consist of three *blocks* of nodes and the edges interconnecting them. In Figure 7, the different blocks are highlighted using dark shading, and one of the splitters is lightly shaded.

All nodes at level 0 are in the same block. For each block B of M nodes at level l, there are two blocks in level $l+1$, B_{upper} and B_{lower}. B_{upper} consists of the nodes in level $l+1$ that are in the same rows as the upper $M/2$ nodes of B, and B_{lower} consists of the nodes in level $l+1$ that are in the same rows as the lower $M/2$ nodes of B. A splitter consists of the blocks B, B_{upper} and B_{lower}, and the edges interconnecting them. The nodes in B are called the splitter *inputs* and the nodes in B_{lower} and B_{upper} are called the splitter *outputs*. Any edge from B to B_{upper} is said to be an *up*-edge, and any edge from B to B_{lower} is said to be a *down*-edge.

In a butterfly, each splitter input is connected to exactly one node in the upper output block, and one in the lower output block. In a d-dilated butterfly, each node in an input block is connected by d edges to a single node in the upper output block, and by another d edges to a single node in the lower output block.

A *multi-butterfly* of multiplicity d, like a d-dilated butterfly, has d up-edges from each input node of each splitter incident on the upper splitter outputs, and another d down-edges incident on the lower splitter outputs. In a d-dilated butterfly, all d up (down) edges would lead to a single node in the upper (lower) output block. In a multi-butterfly, however, the restriction that all d nodes be connected to the same node is relaxed. Each of the d edges can be connected to any of the inputs of the corresponding outbut block, subject to the restriction that any two splitters with inputs at the same level are isomorphic, and that each node has exactly $2d$ inputs and $2d$ outputs.

A *randomly wired* multi-butterfly network of multiplicity d, on the other hand, is one in which the individual output node to which an edge of a splitter is connected is chosen at random from the output blocks, subject only to the constraint that each input node has exactly d up-edges and d down-edges leading from it, and that each output node is fed by exactly $2d$ inputs. It is not necessary for two splitters at the same level to be isomorphic.

The greedy routing algorithm described earlier for butterfly networks can be extended to multi-butterflies. The edges traversed by a message follow the same logical sequence of up- and down-edges. However, at each node, a choice of d edges is available in a multi-butterfly.

Routing on multi-butterflies is efficient, as shown by Upfal's [Upf89] algorithm that implements P permutations deterministically in $O(\log N + P)$ time. Multi-butterflies also provide protection against failures [LM89], since, unlike the butterfly and dilated butterfly, there are edge-disjoint and node-disjoint paths between inputs and outputs. Also, in a randomized multibutterfly, the exact wiring of the network is unknown, and therefore an adversary scheduler cannot force excessive queuing delays to occur. Simulation results from Leighton, Lisinski and Maggs [LLM90] indicate that multi-butterflies may, in practice, perform better than butterflies and dilated butterflies.

A survey of efficient randomized message routing algorithms for *mesh connected computers*, a network architecture not addressed above, is given in [Raj91b]. In the next subsection, we consider the problem of Byzantine agreement. Besides being another example of how to overcome symmetry via randomization, Byzantine agreement shows how randomization can lead to reduced communication complexity.

3.5 Byzantine Agreement

In this section we examine the Byzantine Generals problem and present Ben-Or's [BO83] randomized distributed solution. The Byzantine Generals problem, known also as "Byzantine agreement," has received considerable attention in the literature, e.g., [PSL80, LSP82, Dol82, Rab83, CC85, Per85, Bra85]. This is due primarily to its fundamental relevance in distributed computation and its surprising complexity given the simplicity of the problem statement.

The problem concerns the ranks of the Byzantine Generals, who need to coordinate their rather limited military strategy; that is, they must decide whether to attack or retreat from encroaching enemy forces. Each general has his or her own opinion on the subject. Since their armies are widely separated, their strategy must be decided by the exchange of messages between the generals. Unfortunately, some of the generals are traitors whose messages cannot be trusted. We may assume, without loss of generality, that the messengers are loyal since a general with a disloyal messenger may be regarded as a traitor.

Let v be a boolean value and $\overline{v} = 1 - v$ its complement. The problem of Byzantine agreement can be stated as follows: Consider a set $\{P_1, P_2, \ldots, P_n\}$ of asynchronously executing processes. Each process P_i has a boolean variable x_i whose initial value is b_i. At most t of the n processes are faulty. A distributed and symmetric algorithm to be followed by the correct processes is required such that the following hold on termination:

Condition 1: All correct processes decide on a common value v, where a process "decides v" by setting a private, write-once register to v. Thus, after deciding, a process can no longer change its decision.

Condition 2: If all correct processes start with the same initial value v for x_i, then their final decision must be v.

Condition 1 is usually referred to as the "Agreement condition", and condition 2 the "Validity condition". The validity condition eliminates the trivial solution where each loyal process simply decides on a prearranged value, say 0.

The Byzantine Generals problem translates to one of consensus-building among a set of n

completely connected processes, some of which may be faulty. In the synchronous case, where messages are delivered to their destinations in one computation step, Pease et al. [PSL80] have shown that there exists an algorithm for Byzantine agreement only if less than one-third of the total number of processes are faulty. (The problem of Byzantine agreement among synchronous processes that are not completely connected has also been studied [LSP82] and constraints on the connectivity required for a solution have been determined.)

For the asynchronous case, Fischer et al. [FLP85] proved that Byzantine agreement is impossible for deterministic processes, even if the processes are *not* symmetric and there is only one faulty process. In particular, deterministic processes are susceptible to nontermination. As evidenced by Ben Or's randomized algorithm, this famous "impossibility result" does not apply to processes that may toss coins; in this case, termination can be guaranteed with probability 1. Thus, as in Dining Philosophers, guard scheduling, and leader election, we must once again resort to randomization to solve this distributed computation problem.

We now describe the behavior of the faulty processes, correct processes, and the communication medium. Faulty processes behave unpredictably, perhaps even sending messages according to some malevolent plan, or at times choosing to send no messages at all. For example, in announcing a decision to the correct processes, a faulty process may send different messages to different processes. However, a faulty process cannot influence communication between correct processes, and cannot influence the behavior of correct processes. In other words, it cannot alter or delete messages sent between correct processes, send messages purporting to originate at a correct process, alter the algorithm used by a correct process, or influence any random choices made by a correct process.

All correct processes are guaranteed to use the same algorithm. The only assumption made regarding the relative speeds of different processes is that no process will be delayed indefinitely between computation steps. The communication medium is such that if a correct process sends a message to another correct process, the message will eventually be delivered unaltered to the intended recipient. Note that faults in the communication medium can be modeled by viewing the sender of a message as faulty if the communication medium does not behave as stipulated.

Ben-Or's randomized algorithm utilizes the fact that if independent random choices are made by each process regarding the consensus value, a sufficient number of them will eventually pick the same value to allow agreement among correct processes. Moreover, agreement is guaranteed if the number of faulty processes, t, is less than one-fifth the total number of processes. This claim is true even in the presence of an adversary scheduler which chooses the next process to make a step, or controls how long a message is in transit, as the scheduler cannot influence the outcome of coin tosses made by the processes.

Each correct process P_i executes algorithm *ByzAgree* given below. Variable x_i, initialized to b_i, contains the process's current choice for the consensus value. The algorithm proceeds in rounds, and the index of the current round is stored in r. Each round has three phases.

In the *notification phase*, P_i outputs the value of x_i to all other processes, and then waits for $n-t$ notification messages. All messages sent in the notification phase are tagged with the enumeration value `N`.

In the *proposal phase*, P_i proposes a consensus value from the set {0, 1, '?'}, based on the notification messages just received. It sends its proposal to all other processes, and then waits for $n-t$ proposals in return. In this phase, messages are tagged with the enumeration value `P`.

P_i proposes 0 if greater than $(n+t)/2$ of the notification messages it has received contain 0. Similarly, it proposes 1 if greater than $(n+t)/2$ of the notification messages contain 1. If neither of these is the case, P_i proposes '?', a recommendation that the consensus value be chosen by each process independently by the toss of a coin. Note that P_i simply terminates after broadcasting its proposal if it has made a decision in the previous round. As will be shown below, if P_i decided on value v in round r, then all correct processes will decide on v in round $r+1$. So it is safe for P_i to stop at this point.

Finally, in the *decision phase*, P_i examines the proposals it just received to determine a new value for x_i, which it uses in the next round. Depending on the proposals, P_i may also output this new value of x_i to a write-once register (the process has decided). The significance of the if-statement conditions in the proposal and decision phases is discussed below.

The round number r is attached to all messages of round r, so the processes can distinguish between messages from different rounds. A process in a particular round discards messages it receives from processes in previous rounds, uses messages it receives from processes in the same round, and saves messages it receives from process in later rounds for use during the correct round. Also, since for any round faulty processes may append incorrect round numbers to their messages, or not send any messages at all, no correct process should wait for more than $n-t$ messages in a single phase as arrival of only $n-t$ messages is guaranteed.

```
ByzAgree { (* algorithm for a correct process $P_i$ *)
    r := 1
    decided := FALSE
    WHILE TRUE DO {
         (* The Notification Phase *)
         SEND (N,r,$x_i$) TO all processes
         wait for (n-t) notification msgs of the form (N,r,*)

         (* The Proposal Phase *)
         IF $>$ (n+t)/2 msgs are of the form (N,r,w) for w=0 or w=1 THEN
               SEND (P,r,w) TO all processes
         ELSE SEND (P,r,?)  TO all processes
         IF decided THEN stop
         ELSE wait for (n-t) msgs of the form (P,r,*)

         (* The Decision Phase *)
         IF $>$ t msgs are of the form (P, r, w) for w=0 or w=1 THEN {
               $x_i$ := w
               IF $>$ 3t messages are of the form (P, r, w) THEN {
                     decide w
                     decided := TRUE}
         } ELSE set $x_i$ to 0 or 1 with equal probability
         r := r + 1
    }
}
```

The following lemmas and theorem, due to Hadzilacos [Had86], provide additional insight into the behavior of the algorithm, and establish its correctness.

Lemma 1 *If a correct process proposes value v in round r, then no other correct process will propose the value $\overline{v}$ within the same round.*

A process sends a message $(\mathtt{P}, r, v)$ if it discovers that more than $(n+t)/2$ processes have chosen the value v. At most t of these processes could be faulty. Therefore, more than $(n+t)/2 - t$ (i.e., $(n-t)/2$) correct processes must have chosen v. Thus, a majority of the correct processes have picked v. For another correct process to propose $\overline{v}$ in the same round, a majority of the correct processes must have picked $\overline{v}$. Since a correct process sends the same message to all processes, this is impossible.

Lemma 2 *If at the beginning of round r all correct processes P_i have the same value v for x_i, then all correct processes will decide v in round r.*

In the beginning of a round, each correct process P_i sends messages notifying the others that it has picked value v for x_i. Each correct process receives $n-t$ messages, at most t of which are from faulty processes. Therefore each process receives at least $n-2t$ messages of the form $(\mathtt{N}, r, v)$. Since $n > 5t$ implies $n - 2t > (n+t)/2$, each correct process will consequently propose v in the proposal phase.

Consider now the proposal phase. In the worst case, a process can receive t proposals for $\overline{v}$ from the faulty processes, and $(n-2t)$ proposals for v from correct processes. Since $(n-2t) > 3t$ if $n > 5t$, each correct process will decide on v.

Lemma 3 *If a correct process decides v in round r, then all correct processes will decide v in round $r+1$.*

If we can now show that whenever a correct process decides v in round r, all correct processes *propose* v at the beginning of round $r+1$, then Lemma 3 follows directly from Lemma 2. For a correct process P_i to decide v in round r, it must receive more than $3t$ proposals for v, and since at most t of these can be from faulty processes, P_i must have received m proposals for v from correct processes, for some $m > 2t$. Let us now look at any other correct process P_j.

Process P_j must, in round r, receive proposals from $n - t$ processes. In other words, P_j receives proposals from all but t processes. Therefore, of the m correct processes that proposed v to P_i, all but t must have had their proposals received by P_j. But $m > 2t$ implies $m - t > t$, and therefore P_j will propose v in the next round. All correct process thus propose v in round $r + 1$. From Lemma 2, it follows that all correct processes will decide v in round $r + 1$.

We now have the following correctness result for Ben-Or's algorithm [Had86].

Theorem 4 *Assuming that $n > 5t$, Ben-Or's algorithm guarantees Agreement, Validity, and, with probability 1, termination.*

Agreement follows from Lemma 3 and validity from Lemma 2, with $r = 1$. Consider now termination. With probability 1, enough correct processes will eventually pick a common value v to permit at least one correct process P_i to decide v in some round r. By Lemma 2, all correct processes will decide v in the next round.

An upper bound on the expected number of rounds is $O(2^n)$, the expected number of tosses of n coins before all n coins yield the same value. Yet if the number of faulty processes is $O(\sqrt{n})$, then the expected number of rounds is constant. This illustrates another advantage of tossing coins, since any deterministic solution to the Byzantine Generals problem cannot reach agreement in less than $t + 1$ rounds [FL82].

As for the per-round message complexity, every process sends a message to every other process in each round. Thus, assuming that faulty processes do not send more than $O(n)$ messages each per round, the total number of messages transmitted per round is $O(n^2)$.

Ben-Or's algorithm, along with Rabin's [Rab83], was one of the first for reaching asynchronous Byzantine agreement, and it remains the simplest. Since then a number of more elaborate, in terms of efficiency or fault-resiliency, randomized algorithms for the problem have been developed, including [CC85, Per85, Bra85] (see also [CD89]).

This concludes our survey of distributed randomized algorithms. The next section addresses some additional important aspects of randomized algorithms, and concludes.

4 Additional Topics of Interest and Conclusions

We close our survey with a brief discussion of some additional important topics in randomized algorithms. It will be seen that most of the topics are more theoretical in nature than the material in the body of the survey.

Complexity Theory of Randomized Algorithms

> A *probabilistic Turing machine* is a Turing machine with distinguished states called "coin-tossing states." For each coin-tossing state, the finite control unit specifies two possible next states. The computation of a probabilistic Turing machine is deterministic except that in coin-tossing states the machine tosses an unbiased coin to decide between the two possible next states [Gil77].

As in the classical setting of deterministic and nondeterministic Turing machines, a theory of computational complexity has been developed for probabilistic Turing machines. For example, consider the class of decision problems solvable in "polynomial" time. This class is called *P* for deterministic Turing machines and *NP* for nondeterministic Turing machines. For probabilistic Turing machines, the analogous class is called *RP* (or simply *R* by some writers), standing for *Random Polynomial* time, and is characterized in [Har87] as follows:

> The class *RP* is defined as the class of decision problems for which there is a polynomial-time probabilistic Turing machine with the following property. If the correct answer for an input X is *no*, the machine says *no* with probability 1, and if the correct answer is *yes*, the machine says *yes* with probability greater than $\frac{1}{2}$. Of course, the interest in *RP* problems stems from the fact that for any given X these possibly erroneous algorithms can be reiterated many times, achieving a diminishing probability of error.

The class *co-RP* is defined similarly except now the probabilistic Turing machine must respond correctly with probability 1 on *yes* answers, and with probability greater than $\frac{1}{2}$ on *no* answers. For example, by virtue of the probabilistic algorithms presented in Section 2.2, the problem of primality testing is in *co-RP* while the complementary problem, compositeness testing, is in *RP*. Interestingly, Adleman and Huang [AH87] showed that primality testing is also in *RP*, thereby putting this problem in the intersection of *RP* and *co-RP*.

Complexity classes for randomized algorithms extend beyond *RP* and include the classes *PP* (*Probabilistic Polynomial* time) and *BPP* (*Bounded Probabilistic Polynomial* time). For a problem in *PP*, the requisite probabilistic Turing machine guarantees the correctness of both *yes* and *no* answers only with probability greater than $\frac{1}{2}$. In *BPP*, however, the probability of error in either a *yes* or *no* answer is bounded from above by some constant $\epsilon < \frac{1}{2}$. It is likely, in fact, that *BPP* is much weaker than *PP*. For example, in *BPP*, the error probability can be made exponentially small in the length of the input at the cost of only a constant factor increase in the number of random bits used by the algorithm [CW89].

It is not difficult to see that we have the following hierarchies of complexity classes: $P \subseteq RP \subseteq NP$ and $RP \cup co\text{-}RP \subseteq BPP \subseteq PP$ (but see, e.g., [Gil77, Joh90] for more in-depth discussions of randomized complexity classes). In words, the former reveals that coin tossing is at least as powerful as deterministic computation, and nondeterminism is at least as powerful as coin tossing. It is conjectured that these inclusions are strict. Empirical evidence includes the fact that, as of now, no one has discovered a polynomial-time randomized algorithm for any *NP*-complete problem.

More recently, the *quantum Turing machine* has been proposed [Deu85] as a quantum physical analogue of the probabilistic Turing machine. A quantum Turing machine, in its most general form, produces a random sample from a probability distribution on any given input. Quantum Turing machines give rise to the new complexity classes *Quantum Polynomial* time (*QP*) and *Bounded Quantum Polynomial* time (*BQP*) [BV93]. There is evidence to suggest that it is impossible to simulate a quantum Turing machine with a probabilistic Turing machine without incurring an exponential slowdown [Fey82].

Theory of Probabilistic Automata

Just as there is a complexity theory of probabilistic algorithms which parallels the complexity theory of deterministic algorithms, there is a theory of *probabilistic automata*, e.g., [Rab63, Sal69, Paz71], which parallels the classical theory of nondeterministic automata. A seminal paper on probabilistic automata is [Rab63], where Rabin explored finite state probabilistic automata. He defined the notion of a language accepted by a probabilistic automaton relative to a *cutpoint* probability λ. One of his key results was that there exists finite state probabilistic automata that define non-regular languages, even if the probabilities involved are all rational. Salomaa [Sal69] has expanded upon the work of Rabin to produce a general theory of *stochastic languages*.

Probabilistic Analysis of Conventional Algorithms

Probabilistic analysis of a conventional, i.e., deterministic, algorithm starts with the assumption that the instances of a problem are drawn from a specified probability distribution. Two major applications are the analysis of average-case behavior of sequential algorithms and data structures (see [VF90] for an excellent survey), and the analysis of approximation algorithms for coping with intractability of combinatorial optimization problems [GJ79]. For such problems, the goal is to prove that some simple and fast algorithm produces "good," near-optimal solutions. A classic example is Karp's divide-and-conquer algorithm for the Traveling Salesman problem in a plane [Kar86]. Bin packing is another problem for which very good approximation algorithms have been discovered.

Randomized Parallel Algorithms

As with sequential and distributed algorithms, the performance of *parallel algorithms* can be improved through the introduction of randomized behavior, i.e., coin tossing. A standard model of computation for parallel algorithms is the PRAM, a multi-processor architecture where each processor has random access to a shared memory. PRAM is actually a family of models including CRCW (memory may be concurrently read and written), CREW (memory may be read concurrently but writes are exclusive), and EREW (all reads and writes of memory are exclusive).

The benefits of randomization in parallel algorithms can perhaps be best illustrated by the results of Vishkin [Vis84] for the following problem: Given a linked list of length n, compute the distance of each element of the linked list from the end of the list. The problem has a trivial linear-time sequential algorithm but Wyllie [Wyl79] conjectured that there is no optimal speed-up parallel algorithm for $n/\log n$ processors. Vishkin showed that such optimal speed-up *can* be obtained via randomization by exhibiting a randomized parallel algorithm for the problem that runs in $O(n/p)$ time using $p \leq n/(\log n \log^* n)$ processors on an EREW PRAM. (Note that for all practical purposes, the poly-logarithmic term $\log^* n$ can be viewed as a constant.)

Other examples of fast randomized parallel algorithms include the sorting algorithm of Reischuk [Rei81], the algorithm for subtree isomorphism by Miller and Reif [MR89], as well as the numerous algorithms described in the annotated bibliography. Miller and Reif's algorithm uses $O(\log n)$ time and $O(n/\log n)$ processors, and was the first polylog parallel algorithm for the subtree isomorphism problem.

Sources of Randomness and their Impact on Randomized Algorithms

Throughout this survey we assumed that a randomized algorithm had the ability to toss unbiased coins. Clearly, this is a key assumption: any bias in the coin tosses can adversely affect the accuracy and performance of the algorithm. In this section we describe research aimed at reducing the number of truly random bits a randomized algorithm requires, and the usefulness of "weak sources of randomness." We also consider means of generating bit strings that have the mathematical properties of truly random strings. Our treatment of these topics is mainly bibliographic in nature and we refer the interested reader to the appropriate references for detailed coverage.

Let $\mathcal{A}$ be a randomized algorithm that when supplied with n truly random bits, produces results with a fixed error probability ϵ. The following two questions naturally arise:

1. Is it possible to reduce the error probability of $\mathcal{A}$ through a small increase in the number of truly random bits that $\mathcal{A}$ has at its disposal?

2. Can $\mathcal{A}$ maintain its error probability when the random bits come from a "weak" or imperfect source of randomness?

These two problems, which are commonly referred to as *deterministic amplification* and *simulating probabilistic algorithms by weak random sources*, have received considerable attention in the recent literature and are discussed next.

Deterministic Amplification

Let $\mathcal{A}$ be a randomized algorithm that uses $q(n)$ random bits on an input of length n. One obvious way of boosting the accuracy of $\mathcal{A}$ is to run it repeatedly with independently chosen $q(n)$-bit patterns. However, this method "wastes randomness" as each random bit is used only once and then discarded. It turns out that $\mathcal{A}$ can be deterministically amplified using fewer random bits if certain types of *expander graphs* can be constructed.

In [KPS85], Karp, Pippenger, and Sipser present the first example of deterministic amplification. Using expander graphs, they show how the error probability of a randomized algorithm can be reduced to n^{-c}, for some constant c. Their technique requires no additional random bits. Let us now look at expander graphs more carefully.

An (l, r, d, k)-*expander* is a bipartite graph from L to R such that

1. $|L| = l$ and $|R| = r$,

2. the degree of all nodes in L is d, and

3. every subset of k nodes in L is connected to more than $\frac{r}{2}$ nodes in R.

In general, given values of l,r,d,k it is easy to prove or disprove the existence of an (l, r, d, k)-expander through probabilistic methods [ES74] or other non-constructive arguments. For example, the reader may enjoy proving, using a probabilistic argument, that there exists $(m^{\log m}, m, 2\log^2 m, m)$-expanders for any m [Sip88]. Replacing m by 2^q certifies the existence of $(2^{q^2}, 2^q, 2q^2, 2^q)$-expanders.

Sipser [Sip88] reduces the deterministic amplification problem to a graph theoretic problem involving expander graphs. Since his reduction requires explicit construction of expanders, let us assume that we have a method for explicitly constructing, for any given q, a $(2^{q^2}, 2^q, 2q^2, 2^q)$-expander. Label the left nodes in this graph with bit strings from Σ^{q^2} and the right nodes with bit strings from Σ^q, where $\Sigma = \{0, 1\}$. Call such an expander graph G_q.

Let $\mathcal{B}$ be the amplifying algorithm for $\mathcal{A}$ that uses $q^2(n)$ random bits and operates as follows. It generates a $q^2(n)$-bit random sequence $\sigma \in \Sigma^{q^2(n)}$ and, using σ, generates a multiset $B(\sigma) \subset \Sigma^{q(n)}$. For each $q(n)$-bit $\alpha \in B(\sigma)$, the algorithm $\mathcal{B}$ runs $\mathcal{A}$ on α internally. The multiset $B(\sigma)$ is generated using the expander graph $G_{q(n)}$ (also called a *disperser* in [CW89]).

The efficiency of algorithm $\mathcal{B}$ depends on the ability to efficiently construct the multiset of neighbors of σ: for a given σ, clearly one should be able to generate, in polynomial time, each edge (σ, α). Hence the earlier assumption about efficiently constructing the expander $G_{q(n)}$.

The accuracy of $\mathcal{B}$ is related to certain "expansion properties" of $G_{q(n)}$ (see Definition 2.2 in [CW89] for an exact formulation of these properties). Under the hypothesis that $G_{q(n)}$ can be explicitly constructed, any randomized algorithm $\mathcal{A}$ utilizing $q(n)$ random bits with error probability $\frac{1}{2}$, can be converted into another algorithm $\mathcal{B}$ that uses $q^2(n)$ bits and has error probability $2^{-(q^2(n)-q(n))}$ [Sip88]. The reduction in the error probability follows from the properties of the expander graph. It can also be shown that random bipartite multigraphs are sufficiently expanding.

While Sipser's reduction assumes the constructability of expander graphs, Ajtai et al. [AKS87] show how to explicitly construct expanders for deterministic amplification. Using these multigraphs, Cohen and Wigderson [CW89] prove that the error probability of any *RP* or *BPP* algorithm can be made exponentially small in the size of the input, with only a constant factor increase in the number of random bits used by the algorithm. They also consider simulations of these algorithms with weak sources of random numbers.

Simulating Probabilistic Algorithms by Weak Random Sources

Since most physical sources of randomness suffer from correlation, it is natural to consider imperfect or weak sources of randomness. Such sources are called *semi-random sources* in [SV86]. In this model, each bit of the output is produced by an adversary by the flip of a coin of variable bias. The adversary can look at the previously output bits, and use these to set the bias in the coin. The bias, which helps model correlation among bits, is constrained to be between two limits, δ and $(1-\delta)$.

It has been shown that if a problem can be solved by a polynomial-time Monte Carlo algorithm that has access to a true source of randomness, then the same problem can be solved using an arbitrarily weak semi-random source [VV85]. In [Vaz87], efficient algorithms for using semi-random sources are presented and a technique for producing a quasi-random sequence at an optimal rate, using two semi-random sources, is described.

In [Zuc90], Zuckerman exhibits a pseudo-random generator that depends only on a weak random source called a δ-source. A δ-source, unlike a semi-random source, is asked only once for R random bits and the source outputs an R-bit string such that no string has a probability more than $2^{-\delta R}$ of being output, for some fixed $\delta > 0$. Zuckerman [Zuc91] also shows how to simulate *BPP* and approximation algorithms in polynomial time using the output from a δ-source. Another notion of an imperfect source of randomness is introduced in [LLS87], where an imperfect source is modeled by a discrete control process.

Pseudo-random Number Generators

In the absence of a true source of randomness, randomized algorithms almost always rely on *pseudo-random number generators* (PRGs) as their source of random bits. The importance and widespread use of PRGs is exemplified by a recent article in the New York Times which declares that:

> Mathematical "models" designed to predict stock prices, atmospheric warming, air-plane skin friction, chemical reactions, epidemics, population growth, the outcome of battles, the location of oil deposits and hundreds of other complex matters increasingly depend on a statistical technique called Monte Carlo Simulation, which in turn depends on a reliable and inexhaustible source of random numbers ["Coin-Tossing Computers Found to Show Subtle Bias," by M.W. Browne, New York Times, Tue., Jan. 12, 1993].

Browne goes on to point out the danger inherent in using PRGs, which was brought to light in a recent paper by Ferrenberg, Landau, and Wong [FLW92]. This paper recounts how the authors were puzzled when a simple mathematical model of the behavior of atoms in a magnetic crystal failed to give expected results. They traced the error to the PRG used in the simulation. Upon further investigation, they demonstrated that five of the most widely used PRGs, all of which passed a sizable battery of tests designed to test their randomness, in fact produce correlated pseudo-random numbers.

PRGs work as follows. They perform a deterministic process on a short, random seed to produce a much larger, pseudo-random string that serves as a substitute for a truly random string of the same size. Thus, a PRG can be thought of as a means to minimize the number of truly random bits used by an algorithm.

Much research has been conducted on conserving the number of random bits used by specific PRG algorithms. An analysis justifying the use of pseudo-random substitutes for true random number generators in a randomized primality tester and a probabilistic algorithm for computing square roots is given in [Bac91]. There Bach shows that an exponentially small error can be obtained for these two problems by increasing the number of random bits by a constant factor. Karloff and Raghavan [KR88] study pseudo-random substitutes that use small seeds for purely random choices in sorting, selection and oblivious message routing.

In their seminal paper, Blum and Micali [BM84] introduced the notion of cryptographically secure pseudo-random number generators. A PRG is *cryptographically secure* if given a small segment of its output, all subsequent output cannot be predicted in polynomial time. Otherwise, a PRG is said to be *predictable*.

A number of PRGs, both predictable and secure, have been studied in the literature. Ajtai and Wigderson [AW89] have demonstrated a family of PRGs that appear random to any polynomial-size logic circuit of constant depth and unbounded fan-in. Such PRGs can be substituted for random number generators in applications such as building simple approximations to complex boolean functions [Val84a].

A strong connection exists between cryptographically secure PRGs and one-way functions. A *one-way function* $F(x)$ is a function that is easily computed, but given $F(x)$, it should not be possible to easily recover x, either with a small circuit or with a fast algorithm. In [ILL89], the existence of one-way functions is shown to be necessary and sufficient for the existence of pseudo-random generators, and algorithms for pseudo-random generators that use one-way functions are provided.

Blum et al. [BBS86] present two pseudo-random sequence generators that from small seeds, generate long well-distributed sequences. The first, the $1/P$ generator, is completely predictable from a small segment of its output. The second, the $x^2(mod N)$ generator, is cryptographically secure as its sequence is polynomial-time unpredictable. The $x^2(mod N)$ generator is based on the hardness of the quadratic residuacity problem.

Babai, Nisan and Szegedy [BNS89] obtain a lower bound for the bit complexity of computing functions of n variables, where the i^{th} variable resides on processor i. The communication mechanism considered is a shared blackboard. Using this bound, they developed algorithms that generate, in polynomial time, pseudo-random sequences of length n from a seed of length $\exp(c\sqrt{\log n})$. These pseudo-random sequences cannot be distinguished from truly random sequences by any logspace Turing machine. Hastad [Has90] has extended the results of [ILL89] to the uniform case.

As noted in [IZ89], cryptographically secure PRGs, though theoretically elegant, have several practical problems: they depend on the unproven assumption about the one-wayness of some function, become useful only asymptotically, and are inefficient when implemented. By contrast, the most commonly used PRGs, which typically are based on linear-congruential generators and are not cryptographically secure, do quite well in practice. Impagliazzo and Zuckerman [IZ89] give a theoretical basis to this empirical finding. They prove that two very simple pseudo-random number generators, which are minor modifications of the linear-congruential generator and the simple shift register generator, are good for amplifying the correctness of probabilistic algorithms. They also introduce a class of PRGs based on universal hashing functions. Some consequences of the existence of PRGs are discussed in [All87].

While most of the work in this area has concentrated on generation of pseudo-random strings, in [GGM86], Goldreich, Goldwasser, and Micali address the issue of generating random functions. They introduce a computational complexity measure of the randomness of functions. Assuming the existence of one-way functions, a pseudo-random function generator is presented.

Sampling From a Distribution

There exists a large class of algorithms that are designed around the concept of a random walk. These algorithms, which borrow heavily from techniques in statistical physics, use random walks to facilitate random sampling for approximating hard counting problems. For example, Jerrum and Sinclair [JS89] give a randomized approximation scheme for approximating the permanent of a matrix by relating the problem to that of uniformly generating perfect matchings in a graph. The matching problem is solved by a Markov chain whose states are matchings in the graph.

In general, the construction of small sample spaces that have some randomness properties is of major theoretical and practical importance. For example, in some applications it may be desirable that in a string selected at random from a sample space, the probability distribution induced on every k bit locations be uniform. This property of random bit strings is known as k-wise independence and its use in the derandomization of probabilistic algorithms is discussed below. In [AGHP90], three simple constructions of small probability spaces on n bits for which any k bits are almost independent are presented.

The general study of random walks — a topic not covered by this survey — has made an impact on several areas of algorithm design such as space-bounded algorithms, on-line algorithms, and amplification of randomness. For a study of this area, and the associated background in Markov chains and techniques for proving rapid mixing — informally, a Markov chain is *rapidly mixing* if it converges to its stationary distribution in a short time — the reader is referred to [KL85, Bro86, DLMV88, JS89, Bro89, KLM89, DFK91, BCD$^+$89].

Derandomization

A flurry of activity has recently emerged around the algorithmic design technique of *derandomization*: the act of taking an efficient randomized algorithm and removing the coin flipping to obtain an deterministic algorithm. The beauty of derandomization is that the resulting deterministic algorithm retains the simplicity inherent to randomized algorithms, often outperforms all previously known deterministic algorithms (e.g., [CF90, Aga90a, Aga90b]), and is always correct. This last point is particularly appealing if the randomized algorithm that gave rise to the deterministic one is of the Monte Carlo variety.

The idea of derandomization can be explained as follows [NN90]. Consider any randomized algorithm $\mathcal{A}$. One can associate a probability space (Ω, P) with $\mathcal{A}$, where Ω is the sample space and P is some probability measure corresponding to the probabilistic choices that $\mathcal{A}$ makes during execution. Let $\mathcal{A}(I, w)$ denote an execution of $\mathcal{A}$ on input instance I in which $\mathcal{A}$ randomly chooses w from Ω. Point w is called a *good* point for input instance I if $\mathcal{A}(I, w)$ computes the correct solution. A derandomization of $\mathcal{A}$ means searching Ω for a good point w with respect to a given input instance I. Upon turning up such a point w, the algorithm $\mathcal{A}(I, w)$ is now deterministic and guaranteed to find the correct solution. The catch is, however, that the sample space is generally exponential in size, rendering exhaustive search infeasible.

Karp and Wigderson [KW85] have devised a technique, based on the concept of k-wise independence, that can potentially avoid searching exponentially large sample sizes. A string of bits is said to be *k-wise independent* if any k of the bits in the sequence are mutually independent. Therefore, if the probabilistic choices of a given randomized algorithm are bit-strings of length n and each choice is only required to exhibit k-wise independence, then a sample space of size $O(n^k)$ suffices. Furthermore, when k is a constant, this sample space can be exhaustively searched for a good point (even in parallel) in polynomial time. Karp and Wigderson, in the same paper, take advantage of k-wise independence to derive a fast parallel algorithm for the maximal independent set problem.

Another approach to derandomization is the method of *conditional probabilities* [Spe88], which was originally introduced with the aim of converting probabilistic proofs of existence of combinatorial structures into deterministic algorithms that can actually construct these structures. Applications of the method of conditional probabilities to derandomization include problems in combinatorial optimization [Rag88] and parallel algorithms [MNN89].

On the Future of Randomized Algorithms

These days, randomized algorithms are appearing in the literature almost as often as conventional algorithms. It is safe to say that there are at least several hundred randomized algorithms that have already been published, and dozens more are being discovered each year. We expect this trend to continue since, as we have tried to demonstrate in this survey, the benefits of coin tossing are many: efficiency, conceptual simplicity of the resulting algorithms, overcoming impossibility, etc. Specifically, we expect to see a steady stream of randomized algorithms in the areas of computational geometry, computational biology, graph and number theory, cryptography, robotics, design automation, operating systems (paging, task scheduling, load balancing, etc.), parallel computing, and distributed computing.

Acknowledgements

We would like to thank the anonymous referees for their critical reading of the manuscript. Their many comments and suggestions, including a number of important pointers to the literature, substantially helped to improve the quality of the survey. We are also grateful for valuable interactions with Donna Albertus, Lauren Cowles, Gudjon Hermannsson, Ker-I Ko, Joe Mitchell, Steve Skiena, and Joel Spencer. Finally, we would like to acknowledge the readers of comp.theory who responded to a call-for-comments on an earlier draft of the paper, including Olivier Devillers, Martin Dietzfelbinger, Philippe Flajolet, Dipankar Gupta, George Havas, Martin Huehne, Danny Krizanc, Bohdan Majewski, Stanley Selkow, and Mark Weiss.

References

[AA88] N. Alon and Y. Azar. The average complexity of deterministic and randomized parallel comparison-sorting algorithms. *SIAM Journal on Computing*, 17:1178–1192, 1988. Even the average-case behavior of randomized parallel comparison-sorting algorithms is shown to be no better than the worst-case behavior of their deterministic counterparts.

[AAG$^+$89] K. Abrahamson, A. Adler, R. Gilbart, L. Higham, and D. Kirkpatrick. The bit complexity of randomized leader election on a ring. *SIAM Journal on Computing*, 18(1):12–29, Feb 1989. Under various assumptions about global knowledge, the bit complexity of leader election on asynchronous unidirectional rings is studied.

[AAK90] A. Aggarwal, R. J. Anderson, and M.-Y. Kao. Parallel depth-first search in general directed graphs. *SIAM Journal on Computing*, 19(2):397–409, 1990. This paper gives the first randomized *NC* algorithm for depth-first search in a general directed graph.

[AASS90] P. K. Agarwal, B. Aronov, M. Sharir, and S. Suri. Selecting distances in the plane. In *Proc. Sixth Ann. ACM Symp. on Computational Geometry*, pages 321–331, Berkeley, CA, June 1990. The authors present a randomized algorithm for computing the kth smallest distance in a set of n points in the plane based on a parametric search technique of Megiddo. The algorithm's expected time is $O(n^{4/3}\log^{8/3} n)$.

[ABI86] N. Alon, L. Babai, and A. Itai. A fast and simple randomized parallel algorithm for the maximal independent set problem. *Journal of Algorithms*, 7:567–583, 1986. An *independent set* in a graph is a set of vertices, no two of which are adjacent. A *maximal independent set* is an independent set that is not properly contained in any other independent set. The authors present a simple randomized (Las Vegas) parallel algorithm for this problem. On an EREW-PRAM, their algorithm uses $|E|$ processors with expected running time $O(\log^2 n)$, for a graph with n nodes and $|E|$ edges. Motivated by [KW85], they also describe a derandomization technique to convert any Monte Carlo parallel algorithm that uses k-wise independent random choices into a deterministic parallel algorithm without loss of time and a polynomial increase in the number of processors for any constant k.

[Adl91] L. M. Adleman. Factoring numbers using singular integers. In *Proc. 23rd Ann. ACM Symp. on Theory of Computing*, pages 64–71, New Orleans, LA, May 1991. Generalizing earlier work of Coopersmith, Odlyzko and Schroeppel, Adleman puts forward an efficient randomized algorithm for factoring the integers.

[AES90] P. K. Agarwal, H. Edelsbrunner, and O. Schwarzkopf. Euclidean minimum spanning trees and bichromatic closest pairs. In *Proc. Sixth Ann. ACM Symp. on Computational Geometry*, pages 203–210, Berkeley, CA, June 1990. The authors present a randomized algorithm to compute a bichromatic closest pair in expected time $O((nm \log n \log m)^{2/3} + m \log^2 n + n \log^2 m)$ in Euclidean three-space, which yields an $O(N^{4/3} \log^{4/3} N)$ expected time algorithm for computing a Euclidean minimum spanning tree of N points in Euclidean three-space.

[AES92] N. Alon, P. Erdős, and J. H. Spencer. *The Probabilistic Method.* John Wiley and Sons, 1992. This paper describes the Probabilistic Method as developed by Paul Erdős and its applications in Discrete Mathematics and Theoretical Computer Science.

[Aga90a] P. K. Agarwal. Partitioning arrangements of lines I: An efficient deterministic algorithm. *Discrete Computational Geometry*, 5:449–483, 1990. Using derandomization techniques due to Chazelle and Friedman [CF90], Agarwal obtains a deterministic algorithm that, given a set $\mathcal{L}$ of n lines and a parameter $1 < r < n$, partitions the plane into $O(r^2)$ triangles, each of which meets at most $O(n/r)$ lines of $\mathcal{L}$. He shows that the algorithm is optimal up to a polylog factor.

[Aga90b] P. K. Agarwal. Partitioning arrangements of lines II: Applications. *Discrete Computational Geometry*, 5:533–574, 1990. Agarwal uses his partitioning algorithm of [Aga90a], which he derived through derandomization, to obtain efficient algorithms for a variety of problems involving line or line segments in the plane (e.g., computing incidence between points and lines, implicit point location, and spanning trees with low stabbing number). These algorithms are deterministic, faster than previously known algorithms, and optimal up to a polylog factor in many cases.

[AGHP90] N. Alon, O. Goldreich, J. Håstad, and R. Peralta. Simple construction of almost k-wise independent random variables. In *Proc. 31st Ann. IEEE Symp. on Foundations of Computer Science*, pages 544–553, 1990. Three simple constructions of small probability spaces on n bits for which any k bits are almost independent are presented in this paper.

[AH87] L. M. Adleman and M. A. Huang. Recognizing primes in polynomial time. In *Proc. 19th Ann. ACM Symp. on Theory of Computing*, pages 462–471, 1987.

The probabilistic algorithms of Rabin [Rab76] and Solovay and Stassen [SS77] placed the problem of compositeness testing in the randomized complexity class *RP*, and thus the problem of primality testing in *co-RP*. Adleman and Huang show that primality testing is also in *RP*, thereby putting this problem in the intersection of *RP* and *co-RP*.

[AH88] L. M. Adleman and M. A. Huang. Recognizing primes in random polynomial time. Technical report, University of Souther California, September 1988. The authors present a Las Vegas algorithm that looks for witnesses to compositeness as well as those for primality.

[AH90] J. Aspnes and M. Herlihy. Fast randomized consensus using shared memory. *Journal of Algorithms*, 11(3), 1990. An expected $O(n^4)$ operations are needed for the solution presented.

[AH91] W. Aiello and J. Hastad. Perfect zero-knowledge languages can be recognized in two rounds. *Journal of Computer and System Sciences*, 42:327–345, 1991. This paper shows that if L has a *perfect* zero-knowledge proof (see [FGM$^+$89] for a definition), then L has a two-round interactive proof if the verifier (of this new IP proof) is permitted a small probability of error in accepting a string w as being in a language L. An earlier version of this paper appeared in *Proc. 28th Ann. IEEE Symp. on Foundations of Computer Science*, 1987.

[AKS87] M. Ajtai, J. Komlós, and E. Szemerédi. Deterministic simulation in LOGSPACE. In *Proc. 19th Ann. ACM Symp. on Theory of Computing*, pages 132–140, 1987. The authors present an explicit construction of multigraphs based on expanders for deterministic amplification. Using these multigraphs, Cohen and Wigderson [CW89] show that the error probability of any *RP* or *BPP* algorithm can be made exponentially small in the size of the input, with only a constant factor increase in the number of random bits used by the algorithm.

[Ale82] R. Aleliunas. Randomized parallel communication (preliminary version). In *Proc. First Ann. ACM Symp. on Principles of Distributed Computing*, pages 60–72, 1982. This paper presents a randomized algorithm for packet delivery that delivers a set of n packets traveling to unique targets from unique sources in $O(\log n)$ expected time on a finite degree interconnection network of n processors.

[All87] E. W. Allender. Some consequences of the existence of pseudorandom generators. In *Proc. 19th Ann. ACM Symp. on Theory of Computing*, pages 151–159, 1987. Connections between pseudorandom generation, Kolmogorov complexity, and immunity properties of complexity classes are described.

[ALM$^+$92] S. Arora, C. Lund, R. Motwani, M. Sundar, and M. Szegedy. Verification and hardness of approximation problems. In *Proc. 33rd Ann. IEEE Symp. on Foundations of Computer Science*, pages 14–23, 1992. This paper extends the results in [AS92] to show that unless $P = NP$, the size of the maximum clique cannot be approximated within a factor of n^ϵ for some $\epsilon > 0$, unless $P = NP$.

[AM93] S. Arya and D. M. Mount. Approximate nearest neighbor queries in fixed dimensions. In *Proc. Fourth Ann. ACM-SIAM Symp. on Discrete Algorithms*, pages 271–280, Austin, TX, January 1993. A randomized algorithm for approximate nearest neighbor searching is given. Consider a set S of n points in d-dimensional Euclidean space, where d is a constant independent of n. The authors produce a data structure, such that given any query point, a point of S will be reported whose distance from the query point is at most a factor of $(1 + \epsilon)$ from that of the true nearest neighbor. Their algorithm runs in $O(\log^3 n)$ expected time and requires $O(n \log n)$ space. The data structure can be built in $O(n^2)$ expected time. The constant factors depend on d and ϵ.

[AN93] N. Alon and M. Naor. Coin-flipping games immune against linear-sized coalitions. *SIAM Journal on Computing*, 22(2):403–417, 1993. The authors consider the problem of distributed coin-flipping and leader-election algorithms where every process has complete information. They show that for every constant $c < 1$ there are protocols involving n processes in which no group of cn processes can influence the outcome with probability greater than Kc, where K is a universal constant.

[Ang80] D. Angluin. Local and global properties in networks of processors. In *Proc. 12th Ann. ACM Symp. on Theory of Computing*, pages 82–93, 1980. The capabilities of networks containing nodes with non-unique names are analyzed. It is shown that there exist networks in which it is *not* possible to elect a leader (For example, in a ring with four nodes). Other computations, such as determining topology, are also considered.

[AS89] C. Aragon and R. Seidel. Randomized search trees. In *Proc. 30th Ann. IEEE Symp. on Foundations of Computer Science*, pages 540–545, 1989. A simple randomized algorithm for maintaining balance in dynamic search trees is presented. The expected time for an update is $O(\log n)$ on a tree with n nodes, and involves fewer than two rotations to re-balance the tree.

[AS91a] P. K. Agarwal and M. Sharir. Counting circular arc intersections. In *Proc. Seventh Ann. ACM Symp. on Computational Geometry*, pages 10–20, North Conway, NH, June 1991. Two randomized algorithms are presented. The first counts intersections in a collection of n circles in expected time $O(n^{3/2+\epsilon})$, for any $\epsilon > 0$. The other counts intersections in a set of n circular arcs in expected time $O(n^{5/3+\epsilon})$, for any $\epsilon > 0$. If all arcs have the same radius, the expected time can be improved to $O(n^{3/2+\epsilon})$.

[AS91b] F. Aurenhammer and O. Schwarzkopf. A simple on-line randomized algorithm for computing higher order Voronoi diagrams. In *Proc. Seventh Ann. ACM Symp. on Computational Geometry*, pages 142–151, North Conway, NH, June 1991. They present a simple on-line randomized algorithm that can compute the order-k Voronoi Diagram for n sites in expected time $O(nk^2 \log n + nk \log^3 n)$ and optimal space $O(k(n-k))$.

[AS92] S. Arora and S. Safra. Probabilistic checking proofs; a new characterization of NP. In *Proc. 33rd Ann. IEEE Symp. on Foundations of Computer Science*, pages 2–11, 1992. The class NP is shown to be the class of languages L for which membership can be verified probabilistically in polynomial time using a logarithmic number of random bits and sub-logarithmic number of queries.

[Auw89] B. Auwerbuch. Randomized distributed shortest path algorithms. In *Proc. 21st Ann. ACM Symp. on Theory of Computing*, pages 490–500, 1989. An algorithm that requires $O(D^{1+\epsilon})$ time and $O(E^{1+\epsilon})$ messages, for any $\epsilon > 0$, is presented, where E is the number of edges in the graph and D is its diameter. The lower bounds are $\Omega(D)$ and $\Omega(E)$ respectively. The algorithm is extended to determine shortest paths when the edges have weights.

[AUY83] A. Aho, J. Ullman, and M. Yannakakis. On notations of information transfer in VLSI circuits. In *Proc. 15th Ann. ACM Symp. on Theory of Computing*, pages 133–139, 1983. This paper presents an interesting result on probabilistic algorithms that admit no error: the communication complexity (measured in bits) of the deterministic solution can be no more than the square of the message complexity of any randomized solution.

[AV79] D. Angluin and L. G. Valiant. Fast probabilistic algorithms for Hamiltonian circuits and matching. *Journal of Computer and System Sciences*, 18(2):82–93, 1979. The authors present two algorithms with $O(n(\log n)^2)$ running time for Hamiltonian circuits and an $O(n \log n)$ algorithm to find perfect matching in random graphs with at least $c\, n \log n$ edges, where c is any positive constant.

[AW89] M. Ajtai and A. Wigderson. Deterministic solution of probabilistic constant depth circuits. In S. Micali, editor, *Advances in Computing Research 5: Randomness and Computation*, Greenwich, CT, 1989. JAI Press. A family of pseudo-random number generators which appear random to any polynomial size logic circuit of constant depth and unbounded fan-in is demonstrated. Such pseudorandom generators can be substituted for random-number generators in applications such as building simple approximations to complex boolean functions [Val84a].

[AW92] J. Aspnes and O. Waarts. Randomized consensus in $O(n \log^2 n)$ operations per processor. In *Proc. 33rd Ann. IEEE Symp. on Foundations of Computer Science*, pages 137–146, 1992. An asynchronous algorithm is presented that achieves randomized consensus using $O(n \log^2 n)$ read and write operations on shared-memory registers. This improves on the $O(n^2 \log n)$ worst-case complexity of the best previously-known algorithm.

[Bab85] L. Babai. Trading group theory for randomness. In *Proc. 17th Ann. ACM Symp. on Theory of Computing*, pages 421–429, 1985. This paper developes interactive proofs to classify certain group-theoretic problems and introduces an alternative notion of interactive proofs for complexity-theoretic analysis.

[Bab91] L. Babai. Local expansion of vertex-transitive graphs and random generation in finite groups. In *Proc. 23rd Ann. ACM Symp. on Theory of Computing*, pages 164–174, New Orleans, LA, May 1991. Babai presents a Monte Carlo algorithm that constructs an efficient nearly uniform random generator for finite groups in a very general setting.

[Bac91] E. Bach. Realistic analysis of some randomized algorithms. *Journal of Computer and System Sciences*, 42:30–53, 1991. Bach's analysis justifies the use of pseudo-random substitutes for true random-number generators in a random primality tester and a probabilistic algorithm for computing square roots.

[BB88] G. Brassard and P. Bratley. *Algorithmics: Theory and Practice*. Prentice-Hall, 1988. This book contains a very nice chapter on probabilistic algorithms for a variety of problems such as numerical integration, sorting, and set equality.

[BBC$^+$88] P. Beauchemin, G. Brassard, C. Crépeau, C. Goutier, and C. Pomerance. The generation of random numbers that are probably prime. *Journal of Cryptology*, 1(1):53–64, 1988. The authors make two intriguing observations on Rabin's probabilistic primality test [Rab76], the subject of Section 2.2 of this survey. The first is a provocative reason why Rabin's test is so good. It turns out that a single iteration of his algorithm has a non-negligible probability of failing only on composite numbers that can actually be split in expected polynomial time. Therefore, factoring would be easy if Rabin's test systematically failed with a 25% probability on each composite integer (which, of course, it does not). The authors also investigate the question of how reliable Rabin's test is when used to generate a random integer that is probably prime, rather than to test a specific integer for primality.

[BBP91] J. Boyar, G. Brassard, and R. Peralta. Subquadratic zero-knowledge. In *Proc. 32nd Ann. IEEE Symp. on Foundations of Computer Science*, pages 69–78, 1991. This work reduces the communication complexity of the boolean Satisfiability problem of size n to $O(n^{1+\epsilon_n} + k\,\sqrt{n}^{1+\epsilon_n})$ bits while providing a probability of undetected cheating not greater than 2^{-k}, where ϵ_n tends to zero as n tends to infinity.

[BBS86] M. Blum, L. Blum, and M. Shub. A simple and secure pseudo-random number generator. *SIAM Journal on Computing*, 15:364–383, 1986. Two pseudo-random sequence generators are presented which, from small seeds, generate long well-distributed sequences. The first, $1/P$ generator, is completely predictable from a small segment of its output. The second, $x^2 \pmod N$ generator, is *cryptographically secure* as its sequence is polynomial-time unpredictable (if quadratic residuacity problem is indeed hard).

[BC86] G. Brassard and C. Crépeau. Zero-knowledge simulation of boolean circuits. In *Advances in Cryptology–CRYPTO 86,* Lecture Notes in Computer Science, Vol. 263, pages 223–233. Springer-Verlag, 1986. An important result by Goldreich, Micali, and Wigderson in the design of cryptographic protocols asserts that if one-way functions exit then every language in *NP* has a minimum-knowledge confirming interactive proof. This paper proves a similar result under the assumption that certain number-theoretic computations are infeasible.

[BCC88] G. Brassard, D. Chaum, and C. Crépeau. Minimum disclosure proofs of knowledge. *Journal of Computer and System Sciences*, 37:156–189, 1988. The authors present a generalized perfect zero-knowledge interactive proof scheme that is valid for any problem in *NP*. Contains protocols that allow "Peggy, the prover," to convince "Vic, the verifier," that she has a certifiable secret without disclosing it. The authors use a notion they call bit-commitment, to accomplish these minimum disclosure proofs.

[BCD+89] A. Borodin, S. A. Cook, P. W. Dymond, W. L. Ruzzo, and M. Tompa. Two applications of inductive counting for complementation problems. *SIAM Journal on Computing*, 18(3):559–578, June 1989. A probabilistic algorithm for s-t connectivity in undirected graphs is presented.

[BCF+91] L. Babai, G. Cooperman, L. Finkelstein, E. Luks, and A. Seress. Fast Monte Carlo algorithms for permutation groups. In *Proc. 23rd Ann. ACM Symp. on Theory of Computing*, pages 90–100, New Orleans, LA, May 1991. Nearly optimal randomized algorithms, of the Monte Carlo variety, are presented for basic permutation group manipulation.

[BCW80] M. Blum, A. Chandra, and M. Wegman. Equivalence of free boolean graphs can be decided probabilistically in polynomial time. *Information Processing Letters*, 10:80–82, 1980. The technique used is reduction to a restricted case of the Straight-Line Program Equivalence Problem [MT85].

[BDBK+90] S. Ben-David, A. Borodin, R. M. Karp, G. Tardos, and A. Wigderson. On the power of randomization in online algorithms. In *Proc. 22nd Ann. ACM Symp. on Theory of Computing*, pages 379–386, Baltimore, MD, May 1990. They prove the existence of an efficient "simulation" of randomized online algorithms by deterministic ones, which is the best possible in the presence of an adaptive adversary.

[BDMP91] M. Blum, A. DeSantis, S. Micali, and G. Persiano. Noninteractive zero-knowledge. *SIAM Journal on Computing*, 20(6):1084–1118, 1991. A key paper that summarizes the previous work on non-interactive zero-knowledge proofs. The concept of shared randomness is introduced, and how that can dispose of interaction between the prover and the verifier is illustrated. The authors show that non-interactive zero-knowledge proofs exist for some number-theoretic languages for which no efficient algorithms are known. They also show that if quadratic residuosity is computationally hard, satisfiability also has a non-interactive zero-knowledge proof.

[BDS+92] J.-D. Boissonnat, O. Devillers, R. Schott, M. Teillaud, and M. Yvinec. Applications of random sampling to on-line algorithms in computational geometry. *Discrete Computational Geometry*, 8:51–71, 1992. This paper treats the same kind of problems as in [CS89], but in a semi-dynamic way: the data can be initially unknown and added one by one. The analysis assumes that the points are inserted in a random order.

[Bec82] M. Becker. A probabilistic algorithm for vertex connectivity of graphs. *Information Processing Letters*, 15(3):135–136, October 1982. A probabilistic algorithm is presented which computes the vertex connectivity of an undirected graph $G = (V, E)$ in expected time $O((-\log \epsilon)|V|^{3/2}|E|))$ with error probability at most ϵ, provided that $|E| \leq \frac{1}{2}d|V|^2$, for some constant $d < 1$.

[Ben80] J. Bentley. Multidimensional divide-and-conquer. *Communications of the ACM*, 23:214–229, 1980. This paper contains an $n \log(n)$ deterministic algorithm for finding nearest neighbors in two-dimensional space.

[Ber70] E. R. Berlekamp. Factoring polynomials over large finite fields. *Math. Comput.*, 24, 1970. This paper presents algorithms for root-finding and factorization, two problems in finite fields. The latter problem is reduced to the root-finding problem, for which a probabilistic algorithm is given. This paper is a precursor of [Rab80b].

[Ber80] A. J. Bernstein. Output guards and nondeterminism in CSP. *ACM Trans. on Programming Languages and Systems*, 2(2):234–238, April 1980. Bernstein presents a distributed algorithm for CSP output guards based on priority ordering of processes.

[BFKV92] Y. Bartal, A. Fiat, H. Karloff, and R. Vohra. New algorithms for an ancient scheduling problem. In *Proc. 24th Ann. ACM Symp. on Theory of Computing*, pages 51–58, Victoria, B.C., Canada, May 1992. They consider the on-line version of the original m-machine scheduling problem: given m machines and n positive real jobs, schedule the n jobs on m machines so as to minimize the makespan, the completion time of the last job. In the on-line version, as soon as job j arrives, it must be assigned immediately to one of the machines. They present a competitive deterministic algorithm for all m and an optimal randomized algorithm for the case $m = 2$.

[BFL90] L. Babai, L. Fortnow, and C. Lund. Non-deterministic exponential time has two-prover interactive protocols. In *Proc. 31st Ann. IEEE Symp. on Foundations of Computer Science*, pages 16–25, 1990. Babai et al. prove, using the two-prover interactive proof systems introduced in [BOGKW88], that the class of languages that have a two-prover interactive proof system is non-deterministic exponential time.

[BFM88] M. Blum, P. Feldman, and S. Micali. Non-interactive zero-knowledge proof systems and applications. In *Proc. 20th Ann. ACM Symp. on Theory of Computing*, pages 103–112, 1988. This paper introduces the notion of non-interactive zero-knowledge proofs where the interaction between the prover and the verifier is replaced by shared, random strings.

[BG81] C. H. Bennett and J. Gill. Relative to a random oracle A, $P^A \neq NP^A \neq$ Co-NP^A with probability 1. *SIAM Journal on Computing*, 10(1):96–113, February 1981. Several relationships are given that hold with probability 1 for language classes relativized to a random oracle A, including the one mentioned in the title.

[BG89a] D. Beaver and S. Goldwasser. Multiparty computation with faulty majority. In *Proc. 30th Ann. IEEE Symp. on Foundations of Computer Science*, pages 468–473, 1989. A variation of zero-knowledge proofs is considered, where slow revealing of knowledge to faulty processors is permitted. An algorithm for distributed boolean function computations in Byzantine networks where more than half the processors are faulty is presented. The constraint is that faulty processors should not be able to compute the function before the non-faulty ones do.

[BG89b] M. Bellare and S. Goldwasser. A new paradigm for digital signatures and message identification based on non-interactive zero-knowledge proofs. In *Advances in Cryptology–CRYPTO 89,* Lecture Notes in Computer Science, Vol. 435, pages 194–211. Springer-Verlag, 1989. This paper shows how non-interactive zero-knowledge can be used to yield a new paradigm for secure digital signature schemes (also see [GMR88]).

[BGG90] M. Bellare, O. Goldreich, and S. Goldwasser. Randomness in interactive proofs. In *Proc. 31st Ann. IEEE Symp. on Foundations of Computer Science*, pages 563–572, 1990. The power of randomness in interactive proof systems, in quantitative terms, is considered. A randomness-efficient error reduction technique for converting one proof system into another one using the same number of rounds is presented.

[BGLR93] M. Bellare, S. Goldwasser, C. Lund, and A. Russell. Efficient probabilistically checkable proofs and applications to approximation. In *Proc. 25th Ann. ACM Symp. on Theory of Computing*, pages 294–304, San Diego, CA, May 1993. Bellare et al. construct multi-prover proof systems for *NP* which use only a constant number of provers to simultaneously achieve low error, low randomness and low answer size. As a consequence, they obtain asymptotic improvements to approximation hardness results for a wide range of optimization problems.

[BHZ87] R. Boppana, J. Hastad, and S. Zachos. Does co-NP have short interactive proofs? *Information Processing Letters*, 25:127–132, 1987. This is important paper, along with [For87], provides a method of gaining high confidence that certain languages are not *NP*-complete.

[BI86] L. Babai and A. Itai. A fast and simple randomized parallel algorithm for the maximal independent set problem. *Journal of Algorithms*, 7(4):567–583, Dec 1986. An Independent Set of a graph is a set of vertices, no two of which are adjacent. A maximal independent set is an independent set that is not a proper subset of any other independent set. A simple algorithm which is always correct and runs in $O(\log n)$ time using $O(|E|\ d_{max})$ processors on a Concurrent Read Concurrent Write parallel machine is shown. Here, d_{max} is the maximum degree of any vertex in the graph. The earlier best was a deterministic algorithm for an Exclusive Read Exclusive Write architecture that ran in $O((\log n)^4)$ time using $O((n/\log n)^3)$ processors.

[BK89] M. Blum and S. Kannan. Designing programs that check their work. In *Proc. 21st Ann. ACM Symp. on Theory of Computing*, pages 86–97, May 1989. A more detailed version of [BR88]. Also see “Designing programs that check their work,” Technical Report, Computer Science Division, University of California, Berkeley, CA 94720, Dec. 1988.

[BKRS92] A. Blum, H. Karloff, Y. Rabani, and M. Saks. A decomposition theoren and bounds for randomized server problems. In *Proc. 33rd Ann. IEEE Symp. on Foundations of Computer Science*, pages 197–207, 1992. In a k-server problem, each server is at some point in a metric space. At each time step, a request arises. Each request is a point in metric space, and must be serviced by moving one of the k servers to the point specified. The cost associated with the request is the distance that the server moves. The competitive ratio of a k-server system is the worst-case ratio of the cost of an interactive algorithm on a sequence of inputs, to the optimal cost that would be incurred if the entire sequence were known in advance. The paper proves a lower bound of $\Omega(\sqrt{\log k / \log\log k})$ for the competitive ratio of a k-server system assuming an oblivious adversary. This improves on the previously known bound of $\Omega(\log\log k)$.

[BL92] P. Beame and J. Lawry. Randomized vs. nondeterministic communication complexity. In *Proc. 24th Ann. ACM Symp. on Theory of Computing*, pages 188–199, Victoria, B.C., Canada, May 1992. The authors show that the two complexities are not always the same.

[BLR90] M. Blum, M. Luby, and R. Rubinfeld. Self-testing/correcting with applications to numerical problems. In *Proc. 22nd Ann. ACM Symp. on Theory of Computing*, pages 73–83, 1990. This paper is a more recent reference on the use of randomization in program testing and adds to the collection of interesting examples contained in [BR88, BK89].

[Blu82] M. Blum. Coin flipping by telephone. In *Proc. 1982 IEEE COMPCON, High Technology in the Information Age*, pages 133–137, 1982. This paper describes how two parties can use encryption and decryption keys in a public key cryptosystem to toss coins and exchange results in a distributed environment.

[BM84] M. Blum and S. Micali. How to generate cryptographically strong sequence of pseudo-random bits. *SIAM Journal on Computing*, 13:850–864, 1984. This paper introduces the notion of cryptographically secure pseudo-random number generator.

[BM88] L. Babai and S. Moran. Arthur-Merlin games: A randomized proof system, and a hierarchy of complexity classes. *Journal of Computer and System Sciences*, 36:254–276, 1988. The proof system is considered as a game played between two players, the verifier and the prover, called Arthur and Merlin, respectively. Arthur and Merlin can toss coins and can talk back and forth. In this type of proof-system, all coin tosses made by the verifier are seen by the prover. A hierarchy of complexity classes "just above *NP*"is derived.

[BM89] M. Bellare and S. Micali. Non-interactive oblivious transfer and applications. In *Advances in Cryptology–CRYPTO 89,* Lecture Notes in Computer Science, Vol. 435, pages 547–559. Springer-Verlag, 1989. Based on a complexity assumption, Bellare and Micali show that it is possible to build public-key cryptosystems in which oblivious transfer is itself implemented without any interaction.

[BMO90] M. Bellare, S. Micali, and R. Ostrovsky. Perfect zero-knowledge in constant rounds. In *Proc. 22nd Ann. ACM Symp. on Theory of Computing*, pages 482–493, 1990. This paper contains the first constant-round solutions with no unproven assumptions for the problems of graph isomorphism and quadratic residuosity.

[BMS86] E. Bach, G. Miller, and J. Shallit. Sums of divisors, perfect numbers and factoring. *SIAM Journal on Computing*, 15(4):1143–1154, November 1986. The authors show that computing the sum of divisors of a number N is as hard as factoring N. They also give three natural sets which are in *BPP* (see [Gil77]) but are not known to be in *RP*.

[BN93] R. B. Boppana and B. O. Narayanan. The biased coin problem. In *Proc. 25th Ann. ACM Symp. on Theory of Computing*, pages 252–257, San Diego, CA, May 1993. A *slightly random source* (with *bias* ϵ) is a sequence $x = (x_1, x_2, \cdots, x_n)$ of random bits such that the conditional probability that $x_i = 1$, given the outcomes of the first $i-1$ bits, is always between $\frac{1}{2} - \epsilon$ and $\frac{1}{2} + \epsilon$. Given a subset of S of $\{0,1\}^n$, its ϵ*-biased probability* is defined to be the minimum of $\Pr[x \in S]$ over all slightly random sources x with bias ϵ. The authors show that for every fixed $\epsilon < \frac{1}{2}$ and almost every subset S of $\{0,1\}^n$, the ϵ-biased probability of S is bounded away from 0. They also show that there exists a perfect-information, collective coin-flipping (leader election) protocol for n players that tolerates ϵn cheaters, for every $\epsilon < (2\sqrt{10} - 5)/3 \approx .44$.

[BNS89] L. Babai, N. Nisan, and M. Szegedy. Multiparty protocols and logspace-hard pseudorandom sequences. In *Proc. 21st Ann. ACM Symp. on Theory of Computing*, pages 1–11, 1989. A lower bound is obtained for the bit complexity of computing functions of n variables, where the i^{th} variable resides on processor i. The communication mechanism considered is a shared blackboard. Using this bound, algorithms are developed that generate, in polynomial time, pseudorandom sequences of length n from a seed of length $\exp(c\sqrt{\log\ n})$. These pseudorandom sequences cannot be distinguished from truly random sequences by any logspace Turing machine.

[BO83] M. Ben-Or. Another advantage of free choice: Completely asynchronous agreement protocols. In *Proc. Second Ann. ACM Symp. on Principles of Distributed Computing*, pages 27–30, 1983. Ben-Or's probabilistic algorithm for asynchronous Byzantine agreement, discussed in Section 3.5, was one of the first published solution to the problem, and remains the simplest. Processes toss coins independently to reach consensus on a value. His algorithm requires that less than one-fifth of the processes are faulty for correctness to be guaranteed. The expected number of rounds is exponential in the number of processes n, but becomes a constant when the number of faulty processes is $O(\sqrt{n})$.

[BO85] M. Ben-Or. Fast asynchronous Byzantine agreement (extended abstract). In *Proc. Fourth Ann. ACM Symp. on Principles of Distributed Computing*, pages 149–151, 1985. This work extends Bracha's [Bra85] algorithm to asynchronous networks, initially obtaining a polynomial expected-time protocol. This protocol is refined with the recursive use of Bracha's techniques to get an $O(\log^k n)$ algorithm, where k is a large constant.

[BOGKW88] M. Ben-Or, S. Goldwasser, J. Kilian, and A. Wigderson. Multi-prover interactive proofs: How to remove the intractability assumptions. In *Proc. 20th Ann. ACM Symp. on Theory of Computing*, pages 113–131, 1988. A multi-prover interactive proof model is proposed and its properties examined.

[BOGW88] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proc. 20th Ann.*

ACM Symp. on Theory of Computing, pages 1–10, 1988. The problem is the same as that in [CCD88] and the results obtained are similar.

[BOL89] M. Ben-Or and N. Linial. Collective coin flipping. In S. Micali, editor, *Advances in Computing Research 5: Randomness and Computation*, Greenwich, CT, 1989. JAI Press. Ben-Or and Linial consider the problem of obtaining a distributed coin toss, where each node is initially assigned either a head or a tail. The outcome of the distributed coin toss should not be affected by bias at individual nodes. To exclude the obvious trivial solution where each non-faulty node picks a predetermined value, it is required that if every node changes its initial value, the result of the distributed coin toss should also change. An efficient solution is obtained under the assumption that unfair (faulty) nodes have complete knowledge of actions taken by all nodes.

[Bop89] R. B. Boppana. Amplification of probabilistic boolean formulas. In S. Micali, editor, *Advances in Computing Research 5: Randomness and Computation*, pages 27–45, Greenwich, CT, 1989. JAI Press. Valiant's [Val84a] algorithm is shown to be the best possible. Also, an $O(k^{4.3} n \log n)$ algorithm for computing the kth threshold function of n variables is given.

[BP92] M. Bellare and E. Petrank. Making zero-knowledge provers efficient. In *Proc. 24th Ann. ACM Symp. on Theory of Computing*, pages 711–722, Victoria, B.C., Canada, May 1992. They prove that if a language possesses a statistical zero-knowledge proof then it also possesses a statistical zero-knowledge proof in which the prover runs in probabilistic polynomial time with an *NP* oracle. Previously, this was only known given the existence of one-way permutations.

[BR88] M. Blum and P. Raghavan. Program correctness: Can one test for it? Technical Report RC 14038 (#62902), IBM T.J. Watson Research Center, September 1988. They present "program checkers" for a number of interesting problems based on interactive proofs.

[BR89a] L. Babai and L. Rónyai. Computing irreducible representations of finite groups. In *Proc. 30th Ann. IEEE Symp. on Foundations of Computer Science*, pages 93–98, Research Triangle Park, NC, October 1989. IEEE Computer Society Press. In this paper, the authors give a randomized (Las Vegas) polynomial time algorithm for decomposing a given representation of a finite group over an algebraic number field into absolutely irreducible constituents.

[BR89b] B. Berger and J. Rompel. Simulating $(\log^c n)$-wise independence in NC. In *Proc. 30th Ann. IEEE Symp. on Foundations of Computer Science*, Research Triangle Park, NC, Oct 1989. IEEE Computer Science Press. A general framework for the derandomization of randomized *NC* algorithms whose analysis uses only polylogarithmic independence is presented. This framework allows the derivation of *NC* algorithms for many problems that were not previously known to be in *NC*.

[Bra85] G. Bracha. An $O(\log n)$ expected rounds randomized Byzantine generals protocol. In *Proc. 17th Ann. ACM Symp. on Theory of Computing*, pages 316–326, 1985. Bracha shows how to partition a set of n synchronous processes (of which at most a third are faulty) into overlapping *groups* of processes such that the number of faulty groups is at most the square root the total number of groups. Ben-Or's algorithm for Byzantine agreement (see Section 3.5) is then used to obtain an $O(\log n)$ protocol.

[Bro85] A. Z. Broder. A provably secure polynomial approximation scheme for the distributed lottery problem (extended abstract). In *Proc. Fourth Ann. ACM Symp. on Principles of Distributed Computing*, pages 136–148, 1985. Rabin's classic Byzantine agreement algorithm [Rab83] uses a coin-toss whose outcome is available to all processes, but which cannot be predicted a priori, to reach Byzantine agreement in constant time. Broder demonstrates a polynomial-time distributed mechanism to implement such a coin toss in a Byzantine environment.

[Bro86] A. Z. Broder. How hard is it to marry at random? (On the approximation of the permanent). In *Proc. 18th Ann. ACM Symp. on Theory of Computing*, pages 50–58, 1986. This paper provides a *full-polynomial randomized approximation scheme (fpras)* for approximating the permanent. Evaluating the permanent of a $n \times n$ matrix is equivalent to counting perfect matchings in an associated bipartite graph. The problem of approximately counting the perfect matchings in a graph is reduced to that of generating them uniformly. See [JS89] for the definition of fpras and other related material. An erratum can be found in *Proc. 20th Ann. ACM Symp. on Theory of Computing*, 1988.).

[Bro89] A. Z. Broder. Generating random spanning trees. In *Proc. 30th Ann. IEEE Symp. on Foundations of Computer Science*, pages 442–453, Oct 1989. This paper solves the problem of generating a spanning tree of a connected, undirected graph G which as the following special property: it is chosen uniformly at random from all possible spanning trees of G. The expected running time of the probabilistic algorithm is $O(n \log n)$ per generated tree for almost all graphs. It can be $O(n^3)$ per generated tree in the worst case.

[BRS91a] R. Beigel, N. Reingold, and D. Spielman. PP is closed under intersection. In *Proc. 23rd Ann. ACM Symp. on Theory of Computing*, pages 1–9, New Orleans, LA, May 1991. The randomized complexity class *PP* is shown to be closed under intersection and union.

[BRS91b] A. Blum, P. Raghavan, and B. Schieber. Navigating in unfamiliar geometric terrain. In *Proc. 23rd Ann. ACM Symp. on Theory of Computing*, pages 494–504, New Orleans, LA, May 1991. They consider the problem of a robot that has to travel from a start location to a target in an environment with opaque obstacles that lie in its way. The robot always knows its current absolute position and that of the target. It does not, however, know the positions and extents of the obstacles in advance; it finds out about obstacles as it encounters them. They present an optimal randomized algorithm for scenes containing arbitrary polygonal obstacles.

[BS83] G. N. Buckley and A. Silberschatz. An effective implementation for the generalized input-output construct of CSP. *ACM Trans. on Programming Languages and Systems*, 5(2), 1983. They present a distributed algorithm for CSP output guards based on priority ordering of processes. Their algorithm has the property that two processes that can communicate and do not establish communication with a third process will communicate within a bounded time.

[BT93] J.-D. Boissonnat and M. Teillaud. On the randomized construction of the Delaunay tree. *Theoretical Computer Science*, 112:339–354, 1993. An on-line randomized algorithm which computes Delaunay triangulation and Voronoi diagrams of points in any number of dimensions is given. The complexity of the algorithm is optimal provided that the points are inserted in a random order.

[BV93] E. Bernstein, , and U. Vazirani. Quantum complexity theory. In *Proc. 25th Ann. ACM Symp. on Theory of Computing*, pages 11–20, San Diego, CA, May 1993. A *quantum Turing Machine*, as originally formulated by

Deutsch [Deu85], may be thought of as a quantum physical analogue of a probabilistic Turing Machine: it has an infinite tape, a finite state control, and, in its most general form, produces a random sample from a probability distribution on any given input. Bernstein and Vazirani prove the existence of a *universal* quantum Turing Machine, whose simulation overhead is polynomially bounded. They also present the first evidence that quantum TMs might be more powerful than classical probabilistic TMs. Specifically, they prove that there is an oracle relative to which there is a language that can be accepted in polynomial time by a quantum TM but cannot be accepted in $n^{o(\log n)}$ time by a bounded-error probabilistic TM.

[Car12] R. D. Carmichael. On composite numbers p which satisfy the Fermat congruence $a^{p-1} \equiv p$. *American Mathematical Monthly*, 19:22–27, 1912. Let $n = \Pi_{i=1}^{i=m} p_i^{\nu_i}$ be the unique prime factorization of n, and let $\lambda(n) = \text{lcm}\{p_1^{\nu_1-1}(p_1-1), \ldots, p_m^{\nu_m-1}(p_m-1)\}$. Carmichael shows that n satisfies Fermat's congruence if and only if $\lambda(n)$ divides $(n-1)$.

[CC85] B. Chor and B. Coan. A simple and efficient randomized Byzantine agreement algorithm. *IEEE Trans. on Software Engineering*, SE-11(6):531–539, June 1985. Chor and Coan present a randomized algorithm for synchronous Byzantine agreement when $n \geq 3t+1$, where n is the total number of processors and t is the number of faulty processors. Their algorithm reaches agreement in $O(t/\log n)$ expected rounds and $O(n^2 t/\log n)$ expected message bits, independently of the distribution of processor failures.

[CCD88] D. Chaum, C. Crépeau, and I. Damgärd. Multiparty unconditionally secure protocols. In *Proc. 20th Ann. ACM Symp. on Theory of Computing*, pages 11–19, 1988. Assuming the existence of authenticated secrecy channels between each pair of participants (P_is), this paper shows that if at least $2n/3$ P_is are honest then a function $f(x_1, x_2, \ldots x_n)$, where x_i is known only to P_i for each i, can be computed without any P_i revealing its information.

[CCT91] K. L. Clarkson, R. Cole, and R. E. Tarjan. Randomized parallel algorithms for trapezoidal diagrams. In *Proc. Seventh Ann. ACM Symp. on Computational Geometry*, pages 152–161, North Conway, NH, June 1991. Describes randomized parallel CREW PRAM algorithms for building trapezoidal diagrams of line segments in the plane. For general segments, they give an algorithm requiring optimal $O(A + n \log n)$ expected work and optimal $O(\log n)$ time, where A is the number of intersecting pairs of segments.

[CD89] B. Chor and C. Dwork. Randomization in Byzantine agreement. In *Advances in Computing Research 5: Randomness and Computation*, pages 443–497. JAI Press, 1989. A useful survey of the myriad of randomized distributed algorithms for Byzantine agreement.

[CDRS90] D. Coppersmith, P. Doyle, P. Raghavan, and M. Snir. Random walks on weighted graphs and applications to on-line algorithms (preliminary version). In *Proc. 22nd Ann. ACM Symp. on Theory of Computing*, pages 369–378, Baltimore, MD, May 1990. They show that the problem of designing and analyzing randomized on-line algorithms is closely related to the synthesis of random walks on graphs with positive real costs on their edges.

[CF86] J. D. Cohen and M. J. Fischer. A robust and verifiable cryptographically secure election scheme. In *Proc. 27th Ann. IEEE Symp. on Foundations of Computer Science*, pages 372–381, 1986. A cryptographic election scheme and an IP proof for convincing participants of the correctness of the election procedure.

[CF90] B. Chazelle and J. Friedman. A deterministic view of random sampling and its use in geometry. *Combinatorica*, 10(3):229–249, 1990. Using techniques due to Lovász and Spencer, the authors present a unified framework for derandomizing probabilistic algorithms that resort to repeated random sampling over a fixed domain. In the process, they establish results of independent interest concerning the covering of hypergraphs. Specifically, via a modification of Lovász's *greedy cover algorithm*, they give an algorithm that, given a hypergraph with n vertices and m edges, each of size $\geq \alpha n$, computes an r-sample that intersects every edge e of the hypergraph in $\Omega(|e|r/n)$ vertices, where $r = O((\log n + \log m)/\alpha)$. This improves upon Lovász's algorithm in terms of the number of covered vertices. The tools they use for computing covers "are powerful enough to derandomize just about every probabilistic algorithm proposed in computational geometry".

[CFLS93] A. Condon, J. Feigenbaum, C. Lund, and P. Shor. Probabilistically checkable debate systems and approximation algorithms for PSPACE-hard functions. In *Proc. 25th Ann. ACM Symp. on Theory of Computing*, pages 305–314, San

Diego, CA, May 1993. A *probabilistically checkable debate system* (PCDS) for a language L consists of a probabilistic polynomial-time verifier V and a debate between player 1, who claims that the input x is in L, and player 0, who claims that the input x is not in L. The authors show that there is a PCDS for L in which V flips $O(\log n)$ random coins and reads $O(1)$ bits of debate if and only if L is in *PSPACE*. This characterization of *PSPACE* is used to show that certain *PSPACE*-hard functions are as hard to approximate as they are to compute exactly.

[CG88] B. Chor and O. Goldreich. Unbiased bits from sources of weak randomness and probabilistic communication complexity. *SIAM Journal on Computing*, 17:230–261, 1988. Given sources of stings in which no string is “too probable”, a method of extracting almost unbiased random bits is presented.

[CG90] R. Canetti and O. Goldreich. Bounds on tradeoffs between randomness and communication complexity. In *Proc. 31st Ann. IEEE Symp. on Foundations of Computer Science*, pages 766–775, 1990. Instead of considering the qualitative question, Is an algorithm deterministic or randomized?, the authors try to determine, quantitatively, how much randomization does an algorithm use. Tight lower bounds on the length of the random input of parties computing a function f — depending on the number of bits communicated and the deterministic complexity of f — are derived.

[CGMA85] B. Chor, S. Goldwasser, S. Micali, and B. Auwerbuch. Verifiable secret sharing and achieving simultaneity in the presence of faults. In *Proc. 26th Ann. IEEE Symp. on Foundations of Computer Science*, pages 383–395, 1985. The problems of verifiable secret-sharing and simultaneous broadcast are introduced. Many problems such as distributed coin flipping can be reduced to these problems.

[CH89] J. Cheriyan and T. Hagerup. A randomized maximum-flow algorithm. In *Proc. 30th Ann. IEEE Symp. on Foundations of Computer Science*, pages 118–123, Research Triangle Park, NC, October 1989. IEEE Computer Society Press. An efficient randomized algorithm for computing the maximum flow in a network is presented. For a network with n vertices and m directed edges, the algorithm runs in expected time $O(nm+n^2(\log n)^3)$. The running time is actually $O(nm)$ for all except relatively sparse networks. This improves upon the best known deterministic solution which requires $O(mn \log(n^2/m))$ time. The algorithm, of the Las Vegas variety, is always correct and requires $O(nm \log n)$ time in the worst case.

[Cha84] C.C. Chang. The study of an ordered minimal perfect hashing scheme. *Communications of the ACM*, 27(4):384–387, Apr 1984. Chang uses hash functions of the form $h(x) = (C \bmod p(x))$ where C is an integer constant and $p(x)$ generates a different prime for each integer x. No general method for finding $p(x)$ is given.

[Che93] J. Cheriyan. Random weighted Laplacians, Lovász minimum digraphs and finding minimum separators. In *Proc. Fourth Ann. ACM-SIAM Symp. on Discrete Algorithms*, pages 31–40, Austin, TX, January 1993. Cheriyan gives an $O(n^{2.38})$-time randomized algorithm for the problem of finding a minimum X-Y separator in a digraph, and of finding a minimum vertex cover in a bipartite graph, thereby improving on the previous best bound of $O(n^{2.5}/\log n)$.

[CHM92] Z.J. Czech, G. Havas, and B.S. Majewski. An optimal algorithm for generating minimal perfect hash functions. *Information Processing Letters*, 43(5):257–264, Oct 1992. The authors describe a randomized algorithm for generating perfect hash functions that are space optimal and allow an arbitrary arrangement of keys in the hash table. The algorithm is based on the result of P. Erdős and A. Rényi [ER60], which states that the majority of random sparse 2−graphs are acyclic. The authors present a method of mapping a set of keys, using universal hash functions, into a random graph. Once the mapping is computed it is refined to a perfect hash function in linear deterministic time. The method strongly improves on the space requirements of the other probabilistic methods for generating minimal perfect hash functions.

[Cic80] R. Cichelli. Minimal perfect hash functions made simple. *Communications of the ACM*, 23(1):17–19, Jan 1980. A heuristic for computing a simple, fast, and machine-independent hash function is presented. Because of these properties, several attempts have been made to extend this paper since its publication.

[CL89] A. Condon and R. Lipton. On the complexity of space bound interactive proofs. In *Proc. 30th Ann. IEEE Symp. on Foundations of Computer Science*, pages 462–467, 1989. Interactive proof systems that use two-way probabilistic finite-state verifiers can accept any recursively enumerable language if they are not required to halt with high probability on rejected inputs. An upper bound on the power of IP systems that halt with high probability on all inputs is also derived; such systems accept only a more restricted set of languages. It is shown that any language accepted by such a system is in $ATIME(2^{2^{O(N)}})$.

[CLR90] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms.* The MIT Press, 1990. This well-written encyclopedic introduction to algorithms covers a number of randomized algorithms including those for boolean matrix multiplication, binary search trees, primality testing, partitioning, universal hashing, and parallel prefix.

[CM91] E. Cohen and N. Megiddo. Improved algorithms for linear inequalities with two variables per inequality. In *Proc. 23rd Ann. ACM Symp. on Theory of Computing*, pages 145–155, New Orleans, LA, May 1991. A randomized polynomial time algorithm is given for solving a system of linear inequalities wherein every inequality the two nonzero coefficients have opposite signs.

[CPV91] P. Caspi, J. Piotrowski, and R. Velzaco. An *a priori* approach to the evaluation of signature analysis efficiency. *IEEE Trans. on Computers*, 40(9):1068–1071, Sept 1991. This paper presents an interesting application of control randomization for compressing the results from a digital circuit under test. Instead of imposing any distribution on the input sequence, the linear feedback shift register used for compression is chosen at random.

[CR79] E. Chang and R. Roberts. An improved algorithm for decentralized extrema-finding in circular configurations of processors. *Communications of the ACM*, 22(5):281–283, May 1979. They present a deterministic distributed algorithm for finding the largest of a set of n uniquely numbered processes in a ring. The algorithm uses $O(n \log n)$ messages on the average and $O(n^2)$ messages in the worst case, and does not assume that n is known a priori.

[CR93] R. Canetti and T. Rabin. Fast asynchronous Byzantine agreement with optimal resilience. In *Proc. 25th Ann. ACM Symp. on Theory of Computing*, pages 42–51, San Diego, CA, May 1993. The *resilience* of a protocol is the maximum number of faults in the presence of which the protocol meets its specification. It is known that no Byzantine agreement (BA) protocol for n players (either synchronous or asynchronous) can be $\lceil \frac{n}{3} \rceil$-resilient, and the only known $(\lceil \frac{n}{3} \rceil - 1)$-resilient BA protocol runs in expected exponential time. The authors show that there exists a *fast* $(\lceil \frac{n}{3} \rceil - 1)$-resilient BA protocol by presenting a randomized protocol such that, with overwhelming probability, all the non-faulty players complete execution of the protocol in constant expected time.

[CRS93] S. Chari, P. Rohatgi, and A. Srinivasan. Randomness-optimal unique element isolation, with applications to perfect matching and related problems. In *Proc. 25th Ann. ACM Symp. on Theory of Computing*, pages 458–467, San Diego, CA, May 1993. The authors give a randomness-efficient RNC^2 algorithm for perfect matching that uses $O(\log Z + \log n)$ random bits, where Z is any given upper bound on the number of perfect matchings in the given graph.

[CS89] K. L. Clarkson and P. W. Shor. Applications of random sampling in computational geometry, II. *Discrete Computational Geometry*, 4:387–421, 1989. Efficient probabilistic algorithms are presented for the problems of line segment intersection, convex hull, polygon triangulation, and halfspace partitions of point sets. Each algorithm is of the Las Vegas variety and uses the technique of random sampling. An earlier version of this paper appeared in *Proc. Fourth ACM Symp. on Computational Geometry*, 1988.

[CW79] J. L. Carter and M. N. Wegman. Universal classes of hash functions. *Journal of Computer and System Sciences*, 18:143–154, 1979. This paper contains the first discussion on universal hashing. An earlier version appeared in *Proc. Ninth Ann. ACM Symp. on Theory of Computing*, 1977, pp. 106–112.

[CW89] A. Cohen and A. Wigderson. Dispensers, deterministic amplification, and weak random sources (extended abstract). In *Proc. 30th Ann. IEEE Symp. on Foundations of Computer Science*, pages 14–25, Research Triangle Park, NC, October 1989. IEEE Computer Society Press. The authors use highly expanding bipartite multigraphs (dispensers) to show that the error probability of any *RP* or *BPP* algorithm can be made exponentially small in the size of the input at the cost of only a constant factor increase in the number of random bits used by the algorithm. The simulation of these algorithms with weak sources of random numbers is also considered.

[Deu85] D. Deutsch. Quantum theory, the Church-Turing principle and the universal quantum computer. *Proc. Royal Society of London*, A400:97–117, 1985. Deutsch introduces the *quantum physical computer*, later referred to as the "quantum Turing Machine" in [BV93], which can be thought of as a quantum physical analogue of a probabilistic Turing Machine: it has an infinite tape, a finite state control, and, in its most general form, produces a random sample from a probability distribution on any given input.

[Dev92] O. Devillers. Randomization yields simple $O(n \log^* n)$ algorithms for difficult $\Omega(n)$ problems. *International Journal of Computational Geometry and Applications*, 2(1):97–111, 1992. This papers provides two $O(n \log^* n)$ randomized algorithms. One computes the skeleton of a simple polygon and the other the Delaunay triangulation of a set of points knowing the euclidean minimum spanning tree. The existence of deterministic $O(n \log n)$ algorithms for both problems is an open problem.

[DFK91] M. Dyer, A. Frieze, and R. Kannan. A random polynomial time algorithm for approximating the volume of a convex body. *Journal of the ACM*, 38:1–17, 1991. A constant time oracle is assumed for determining if a point in space is inside or outside a convex body in n-dimensional Euclidean space. The algorithm runs in time bounded by a polynomial in n, the dimension of the body, and $1/\epsilon$, where ϵ is the relative error bound. With probability 3/4, it finds an approximation satisfying the error bound.

[DGMP92] M. Dietzfelbinger, J. Gil, Y. Matias, and N. Pippenger. Polynomial hash functions are reliable. In *Proc. 19th Int'l. Colloq. on Automata, Languages and Programming,* Lecture Notes in Computer Science, Vol. 623, pages 235–246, Vienna, Austria, July 1992. Springer-Verlag. This paper, along with [DMadH92], shows how to construct a perfect hash function in $\Theta(n)$ time, which is suitable for real-time applications (Theorems 6.1 and 7.1).

[Dij71] E. W. Dijkstra. Hierarchical ordering of sequential processes. *Acta Informatica*, 1(2):115–138, 1971. Reprinted in *Operating Systems Techniques*, C.A.R. Hoare and R.H. Perrot, Eds., Academic Press, 1972, pp. 72–93. This paper introduces the classical synchronization problem of Dining Philosophers.

[DKM$^+$88] M. Dietzfelbinger, A. Karlin, K. Mehlhorn, F. Meyer auf der Heide, H. Rohnert, and R. E. Tarjan. Dynamic perfect hashing: Upper and lower bounds. In *Proc. 29th Ann. IEEE Symp. on Foundations of Computer Science*, pages 524–531, White Plains, NY, Oct 1988. A randomized algorithm for the dictionary problem based on perfect hashing is presented.

[DKS88] C. Dwork, P. C. Kanellakis, and L. J. Stockmeyer. Parallel algorithms for term matching. *SIAM Journal on Computing*, 17(4):711–731, 1988. In the context of a parallel algorithm for the term matching problem, this paper shows how randomization can be used to reduce the initial processor complexity from $O(n^5)$ to $O(M(n))$, where $M(n)$ is the processor complexity of multiplying two $n \times n$ matrices.

[DLMV88] P. Dagum, M. Luby, M. Mihail, and U.V. Vazirani. Polytopes, permanents and graphs with large factors. In *Proc. 29th Ann. IEEE Symp. on Foundations of Computer Science*, pages 412–421, 1988. Randomized algorithms for approximating the number of perfect matchings in a graph based on a geometric reasoning are presented.

[dlVKS93] F. de la Vega, S. Kannan, and M. Santha. Two probabilistic results on merging. *SIAM Journal on Computing*, 22(2):261–271, 1993. Two probabilistic algorithms for merging two sorted lists are presented. When $m < n$, the first algorithm has a worst-case time better than any deterministic algorithm for $1.618 < n/m < 3$. The algorithm is extended to perform well for any value of n/m.

[DMadH90] M. Dietzfelbinger and F. Meyer auf der Heide. How to distribute a dictionary in a complete network. In *Proc. 22nd Ann. ACM Symp. on Theory of Computing*, pages 117–127, Baltimore, MD, May 1990. A randomized algorithm is given for implementing a distributed dictionary on a complete network of p processors. The algorithm is based on hashing and uses $O(n/p)$ expected time to execute n arbitrary instructions (insert, delete, lookup). The response time for each lookup is expected constant.

[DMadH92] M. Dietzfelbinger and F. Meyer auf der Heide. Dynamic hashing in real time. In *Informatik · Festschrift zum 60. Geburtstag von Günter Holtz,* Teubner-Texte zur Informatik, Band 1, pages 95–119. B. G. Teubner, Stuttgart, Germany, 1992. The FKS probabilistic procedure is extended to real-time. See Theorems 6.1 and 7.1 in [DGMP92]. A preliminary version of this paper appeared as "A new universal class of hash functions and, dynamic hashing in real time," *Proc. 17th Int'l. Colloq. on Automata, Languages and Programming*, 1990, pp. 6–19.

[DMT92] O. Devillers, S. Meiser, and M. Teillaud. Fully dynamic Delaunay triangulation in logarithmic expected time per operation. *Computational Geometry: Theory and Applications*, 2(2):55–80, 1992. This paper extends the results of [BT93] by considering the deletion of points. The Delaunay triangulation of n points is updated in $O(\log n)$ expected time per insertion and $O(\log\log n)$ expected time per deletion. The insertion sequence is assumed to be in a random order, and deletions are assumed to concern any currently present point with the same probability.

[DoD83] DoD (United States Dept. of Defense). *Reference Manual for the Ada Programming Language,* MIL-STD 1815A, February 1983. Section 3.2 of our survey discusses a randomized distributed algorithm for the scheduling of input and output guards. The designers of Ada chose only to allow nondeterministic choice among the **accept** alternatives of a **select** statement. This design decision makes the guard scheduling problem in Ada much easier and, in particular, obliviates the need for randomization.

[Dol82] D. Dolev. The Byzantine generals strike again. *Journal of Algorithms*, 3(1):14–30, 1982. This is the introductory paper on Byzantine Generals. Dolev proves that Byzantine agreement is achievable in *any* distributed system if and only if the number of faulty processors in the system is (1) less than one-third of the total number of processors; and (2) less than one-half the connectivity of the system's network. In cases where agreement is achievable, deterministic algorithms for obtaining it are given.

[DS90] C. Dwork and L. Stockmeyer. The time complexity gap for 2-way probabilistic finite-state automata. *SIAM Journal on Computing*, 19(6):1011–1023, 1990. Among other results, this paper shows that any 2-way probabilistic finite automaton recognizing a non-regular language must use exponential expected time infinitely often. Since any regular language can be recognized in linear time, a time-complexity gap is established. Similar results were published in the paper entitled "On the Power of 2-Way Probabilistic Finite Automata," which appeared in *Proc. 30th Ann. IEEE Symp. on Foundations of Computer Science*, 1989.

[DSMP87] A. De Santis, S. Micali, and G. Persiano. Non-interactive zero-knowledge proof-systems. In *Advances in Cryptology–CRYPTO 87,* Lecture Notes in

Computer Science, Vol. 293, pages 52–72. Springer-Verlag, 1987. This paper introduces the notion of non-interactive zero-knowledge proofs based on a weaker complexity assumption than that used in [BFM88].

[DSMP88] A. De Santis, S. Micali, and G. Persiano. Non-interactive zero-knowledge proof-systems with preprocessing. In *Advances in Cryptology–CRYPTO 88,* Lecture Notes in Computer Science, Vol. 403, pages 269–283. Springer-Verlag, 1988. The authors show that if any one-way function exists after an interactive preprocessing stage then any sufficiently short theorem can be proven *non-interactively* in zero-knowledge.

[DSS90] C. Dwork, D. Shmoys, and L. Stockmeyer. Flipping persuasively in constant time. *SIAM Journal on Computing*, 19(2):472–499, 1990. An efficient randomized protocol is presented that tolerates up to $n/(\log n)$ malicious processors that requires constant expected number of rounds to achieve a distributed coin toss. Also given is a Byzantine Generals algorithm that tolerates $n/(\log n)$ failures and runs in constant expected number of rounds. A preliminary version of this paper appeared in *Proc. 27th Ann. IEEE Symp. on Foundations of Computer Science*, 1986.

[DSY90] A. De Santis and M. Yung. Cryptographic applications of non-interactive metaproofs and many-prover systems. In *Advances in Cryptology–CRYPTO 90,* Lecture Notes in Computer Science, Vol. 537. Springer-Verlag, 1990. The authors show how many provers can share the same random string in proving multiple theorems non-interactively in zero-knowledge.

[ER60] P. Erdős and A. Rényi. On the evolution of random graphs. *Publ. Math. Inst. Hung. Acad. Sci.*, 5:17–61, 1960. A seminal paper on random graphs. Reprinted in *Paul Erdős: The Art of Counting. Selected Writings*, J.H. Spencer, Ed., Vol. 5 of the series *Mathematicians of Our Time*, MIT Press, 1973, pp. 574–617.

[ES74] P. Erdős and J. Spencer. *Probabilistic Methods in Combinatorics.* Academic Press, New York and London, 1974. Recognized experts in the field present a small, power packed monograph on non-constructive probabilistic methods in combinatorics. Our algorithm for networks without large hierarchies is based on the discussion in Chapter 1 of this book. Other highlights include, Ramsey's theorems and evolution of random graphs.

[FCDH91] E. Fox, Q.F. Chen, A. Daoud, and L.S. Heath. Order preserving minimal perfect hash functions and information retrieval. *ACM Trans. on Information Systems*, 9(2):281–308, July 1991. This algorithm combines the techniques of embedding the keys into an $r-$graph and two-level hashing to design hash functions that are optimal in terms of hashing time and space utilization. The algorithm to generate the hash functions uses near-optimal space and time. Any desired order can be maintained.

[Fey82] R. P. Feynman. Simulating physics with computers. *International Journal of Theoretical Physics*, 21(6/7):467–488, 1982. Feynman points out the curious problem that it appears to be impossible to simulate a general quantum physical system on a probabilistic Turing Machine without an exponential slowdown, even if the quantum physical system to be simulated is discrete (like some kind of quantum cellular automaton).

[FFS87] U. Feige, A. Fiat, and A. Shamir. Zero-knowledge proofs of identity. In *Proc. 19th Ann. ACM Symp. on Theory of Computing*, pages 210–217, 1987. Zero-knowledge proofs, in the traditional sense, reveal 1 bit of information to the verifier, viz. $w \in L$ or $w \notin L$. This paper proposes the notion of "truly zero knowledge" proofs where the prover convinces the verifier that he/she knows whether w is or is not in L, without revealing any other information. An RSA-like scheme based on the difficulty of factoring, which is much more efficient than RSA, is also presented.

[FGM$^+$89] M. Fürer, O. Goldreich, Y. Mansour, M. Sipser, and S. Zachos. On completeness and soundness in interactive proof systems. In S. Micali, editor, *Advances in Computing Research 5: Randomness and Computation*, pages 429–442. JAI Press, Greenwich, CT, 1989. An interactive proof system for a language L is said to have *perfect completeness* if the verifier *always* accepts w if $w \in L$. This paper proves that any language having an interactive, possibly unbounded, proof has one with perfect completeness. Only languages in *NP* have interactive proofs with perfect soundness. This paper first appeared under the title "Interactive proof system: provers that never fail and random selection," authored by O. Goldreich, Y. Mansour and M. Sipser, in *Proc. 28th Ann. IEEE Symp. on Foundations of Computer Science*, 1987, pp. 449–461.

[FH84] V. A. Feldman and D. Harel. A probabilistic dynamic logic. *Journal of Computer and System Sciences*, 28(2):193–215, 1984. This paper defines a formal logic **Pr**DL to reason about probabilistic programs. It extends the semantics of Kozen [Koz81] formulas involving probabilistic programs.

[FHCD92] E. Fox, L.S. Heath, Q.F. Chen, and A. Daoud. Practical minimal perfect hash functions for large databases. *Communications of the ACM*, 35(1):105–121, January 1992. This paper presents two randomized algorithm for minimal perfect hashing functions that are designed for use with data bases with as many as a million keys. The algorithms have been experimentally evaluated. The first algorithm generates hash functions that are less than $O(n)$ computer words long, and the second generates functions that approach the theoretical lower bound of $\Omega(n/\log n)$ words. This work is a predecessor of [FCDH91].

[FKS82] M. L. Fredman, J. Komlós, and E. Szemeredi. Sorting a sparse table with $O(1)$ worst case access time. In *Proc. 23rd Ann. IEEE Symp. on Foundations of Computer Science*, pages 165–169, 1982. This paper proves many fundamental results that are essential for constructing a perfect hashing function for a given set of keys.

[FL82] M. J. Fischer and N. Lynch. A lower bound for the time to assure interactive consistency. *Information Processing Letters*, 14(4):182–186, 1982. They prove that no deterministic solution to the Byzantine Generals problem can reach agreement in less than $t+1$ rounds, where t is the number of faulty processes.

[Fla85] P. Flajolet. Approximate counting: A detailed analysis. *BIT*, 25:113–134, 1985. In 1978, R. Morris published an article in *Communications of the ACM* entitled "Counting large numbers of events in small registers." This paper presented a randomized algorithm, known as *Approximate Counting*, that allows one to approximately maintain a counter whose values may range in the interval 1 to M using only about $\log\log M$ bits, rather than the $\log M$ bits required by a standard binary counter. The algorithm has proven useful in the areas of statistics and data compression. Flajolet provides a complete analysis of approximate counting which shows (among other things) that, using suitable corrections, one can count up to M keeping only $\log\log M + \delta$ bits with an accuracy of order $O(2^{-\delta/2})$.

[Fla90] P. Flajolet. On adaptive sampling. *Computing*, 34:391–400, 1990. *Adaptive Sampling* is a probabilistic technique due to Wegman that allows one to estimate the cardinality (number of distinct elements) of a large file typically stored on disk. This problem naturally arises in query optimization of database systems. Flajolet shows that using m words of in-core memory, adaptive sampling achieves an expected relative accuracy close to $1.20/\sqrt{m}$. This compares well with the *probabilistic counting* technique of Flajolet and Martin [FM85b]: adaptive sampling appears to be about 50% less accurate than probabilistic counting for comparable values of m. Adaptive sampling, however, is completely free of non-linearities for smaller values of cardinalities (probabilistic counting is only *asymptotically* unbiased).

[FLP85] M. J. Fischer, N. Lynch, and M. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2), April 1985. This paper proves that every completely asynchronous, deterministic algorithm for Byzantine agreement has the possibility of nontermination, even with only one faulty processor. This impossibility result does not hold in the synchronous case. For completely asynchronous *probabilistic* algorithms, the problem is avoided since termination is only required with probability 1. See Section 3.5 for an example of such a probabilistic algorithm for asynchronous Byzantine agreement.

[FLS90] U. Feige, D. Lapidot, and A. Shamir. Multiple non-interactive, zero-knowledge proofs based on a single random string. In *Proc. 31st Ann. IEEE Symp. on Foundations of Computer Science*, pages 308–317, 1990. The following two problems posed in [DSMP88], associated with non-interactive zero-knowledge proof systems, are solved: (1) how to construct NIZK proofs under general complexity assumptions rather than number-theoretic assumptions, and (2) how to enable multiple provers to prove, in writing, polynomially many theorems based on a single random string. The authors show that any number of provers can share the same random string and that any trap-door permutation can be used instead of quadratic residuosity. Also, if the prover is allowed to have exponential computing power, then one-way permutations are sufficient for bounded non-interactive zero-knowledge proofs.

[FLW92] A. M. Ferrenberg, D. F. Landau, and Y. J. Wong. Monte Carlo simulations: Hidden errors from "good" random number generators. *Physical Review Letters*, 69(23):3382–3388, December 1992. The authors unveil subtle correlations in five widely used pseudo-random number generators. They undertook this investigation when a simple mathematical model of the behavior of atoms in a magnetic crystal failed to give the expected results. They traced the error to the pseudo-random number generator used in the simulation.

[FM85a] P. Flajolet and G. N. Martin. Probabilistic counting algorithms for data base applications. *Journal of Computer and System Sciences*, 25(31):182–209, 1985. This paper presents a probabilistic counting technique for determining the number of distinct records in a file. The technique requires $O(1)$ storage and a single pass over the file. Also appeared as "Probabilistic counting," *Proc. 24th Ann. IEEE Symp. on Foundations of Computer Science*, 1983, pp. 76–84.

[FM85b] P. Flajolet and G. N. Martin. Probabilistic counting algorithms for data base applications. *Journal of Computer and System Sciences*, 31:182–209, 1985. *Probabilistic Counting* is a technique for estimating the cardinality (number of distinct elements) of a large file typically stored on disk. This problem naturally arises in query optimization of database systems. Using m words of in-core memory, probabilistic counting achieves an expected relative accuracy close to $0.78/\sqrt{m}$. Moreover, it performs only a constant number of operations per element of the file.

[FM88] P. Feldman and S. Micali. Optimal algorithms for Byzantine agreement. In *Proc. 20th Ann. ACM Symp. on Theory of Computing*, pages 162–172, 1988. The expected running time of this algorithm is constant in a synchronous network of n nodes if the number of faults is less than $n/3$, and in an asynchronous network of n nodes if the number of faults is less than $n/4$.

[For87] L. Fortnow. The complexity of perfect zero-knowledge. In *Proc. 19th Ann. ACM Symp. on Theory of Computing*, pages 204–209, May 1987. The notion of perfect zero-knowledge requires that the verifier, no matter how powerful it is, not learn any additional information. Fortnow proves that for any language which has a perfect zero-knowledge protocol, its complement has a single round interactive protocol. This result implies that for NP-complete languages, there are no perfect zero-knowledge protocols (unless the polynomial time hierarchy collapses).

[FR80] N. Francez and M. Rodeh. A distributed abstract data type implemented by a probabilistic communication scheme. In *Proc. 21st Ann. IEEE Symp.*

on Foundations of Computer Science, pages 373–379, 1980. They also give a deadlock-free, truly distributed and symmetric solution to the dining philosophers problem based on a probabilistic implementation of CSP. In particular, they present a randomized algorithm for the scheduling of input/output guards in CSP, which we discuss in Section 3.2. This was one of the first papers on probabilistic distributed algorithms. A revised version appears as TR 80, IBM Scientific Center, Haifa, Israel, April 1980 (same title).

[FS89] L. Fortnow and M. Sipser. Probabilistic computation in linear time. In *Proc. 21st Ann. ACM Symp. on Theory of Computing*, pages 148–156, 1989. An oracle is specified, under which all problems solvable in random polynomial time are solvable in random linear time, thus collapsing a number of randomized complexity classes into one. Analogous results in deterministic computations are demonstratably false.

[FS92] U. Feige and A. Shamir. Multiple oracle interactive proofs with constant space verifiers. *Journal of Computer and System Sciences*, 44:259–271, 1992. The authors show that the expected payoff of reasonable games of incomplete information are undecidable. The Turing-machine simulation uses polynomial cost and stops with probability 1.

[Fur87] M. Furer. The power of randomness for computational complexity. In *Proc. 19th Ann. ACM Symp. on Theory of Computing*, pages 178–181, 1987. This paper improves on the VLSI algorithm by Mehlhorn and Schmidt [MS82]. An $O(n)$ average bit complexity algorithm with no probability of error is demonstrated.

[Gaz91] H. Gazit. An optimal randomized parallel algorithm for finding the connected components of a graph. *SIAM Journal on Computing*, 20(6):1046–1067, 1991. The expected running time of this algorithm is $O(\log n)$ with $O((m+n)/\log n)$ processors, where n is the number of vertices and m is the number of edges. It uses $O(m+n)$ space. The algorithm is optimal in the time-processor product sense, as well as in space complexity.

[GBY91] G.H. Gonnet and R. Baeza-Yates. *Handbook of Algorithms and Data Structures.* Addison-Wesley, Reading, Mass., 1991. Section 3.3.16 gives an overview of perfect hashing.

[GGK92] M. Gereb-Graus and D. Krizanc. The average complexity of parallel comparison merging. *SIAM Journal on Computing*, 21:43–47, 1992. The authors establish a lower bound on the time complexity of randomized merging of two sorted lists in a parallel computation tree model. An earlier version of this paper, entitled "The Complexity of Parallel Comparison Merging," appeared in *Proc. 28th Symp. on Foundations of Computer Science*, 1987.

[GGM86] O. Goldreich, S. Goldwasser, and S. Micali. How to construct random functions. *Journal of the ACM*, 33:792–807, 1986. A computational complexity measure of the randomness of functions is introduced, and, assuming the existence of one-way functions, a pseudo-random function generator is presented.

[GHY89] Z. Galil, S. Haber, and M. Yung. Minimum-knowledge interactive proofs for decision problems. *SIAM Journal on Computing*, 18(4):711–739, Aug 1989. This paper extends the work of [GMR89], the concept of minimum knowledge is defined and a minimum-knowledge protocol for transferring the results of any fixed computation from one party to another (e.g. prover to verifier) is described.

[Gil77] J. T. Gill. Computational complexity of probabilistic Turing machines. *SIAM Journal on Computing*, 6(4):675–695, December 1977. This paper defines the basic notion of a probabilistic Turing machine (PTM). A PTM computes a partial function that assigns to each input the output which occurs with a probability greater than half. It is shown that a NDTM can be simulated by a PTM in the same space but with a small error probability. Gill also considers the complexity classes RP, PP, and BPP for polynomial-time probabilistic Turing machines (see Section 4.1). He shows that $P \subseteq RP \subseteq BPP \subseteq PP \subseteq P$SPACE and that $RP \subseteq NP \subseteq PP$.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness.* W.H. Freeman and Company, 1979. This well-known book on the theory of NP-completeness contains a section on the probabilistic analysis of approximation algorithms for NP-complete combinatorial optimization problems.

[GK86] S. Goldwasser and J. Kilian. Almost all primes can be quickly certified. In *Proc. 18th Ann. ACM Symp. on Theory of Computing*, pages 316–329, 1986. The authors show that if Cramér's conjecture about the spacing of prime numbers is true than there exists a random polynomial time algorithm for primality testing.

[GKS92] L. J. Guibas, D. E. Knuth, and M. Sharir. Randomized incremental construction of Delaunay and Voronoi diagrams. *Algorithmica*, 7(4):381–413, 1992. They give a new randomized incremental algorithm for the construction of planar Voronoi diagrams and Delaunay triangulations. Their algorithm takes expected time $O(n/\log n)$ and space $O(n)$, is very practical to implement, and along with the algorithm of [BT93], is more "on-line" than earlier similar methods.

[GKS93] W. Goddard, V. King, and L. Schulman. Optimal randomized algorithms for local sorting and set-maxima. *SIAM Journal on Computing*, 22(2):272–283, April 1993. Nearly optimal randomized algorithms are presented for the local sorting problem (i.e., determining the relative order in every pair of adjacent vertices in a graph in which each vertex is assigned an element of a total order) and the set-maxima problem (i.e., determining the maximum element of each set in a collection of sets whose elements are drawn from a total order).

[GL89] R. I. Greenberg and C. E. Leiserson. Randomized routing on fat-trees. In S. Micali, editor, *Advances in Computing Research 5: Randomness and Computation*, pages 345–374. JAI Press, Greenwich, CT, 1989. Fat-Trees are a class of routing networks in parallel computation. Given a set of messages to send, the choice is made at random of which message is to be sent at what time. This approach is different from that of [Val82]. See also *Proc. 17th Ann. ACM Symp. on Theory of Computing*, 1985, pp. 241–249.

[GM84] S. Goldwasser and S. Macali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299, 1984. This paper introduces a new probabilistic encryption technique. It also contains an excellent introduction to other public key cryptosystems with discussion on objections to cryptosystems based on trapdoor functions.

[GMR88] S. Goldwasser, S. Micali, and R. Rivest. A digital signature scheme secure against adaptive chosen-message attack. *SIAM Journal on Computing*, 17:281–308, 1988. This is a companion paper of [KPU88].

[GMR89] S. Goldwasser, S. Macali, and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, pages 186–208, 1989. This paper first appeared in *Proc. 17th Ann. ACM Symp. on Theory of Computing*, 1985, pp. 291–304. It introduces the important notion of zero-knowledge interactive proofs. The authors show that it is possible to prove that certain theorems are true without divulging why this is so.

[GMV91] J. Gil, Y. Matias, and U. Vishkin. Towards a theory of nearly constant-time parallel algorithms. In *Proc. 32nd Ann. IEEE Symp. on Foundations of Computer Science*, pages 698–710, 1991. This paper presents a paradigm for obtaining $O(\log^* n)$ running time for problems such as directory maintenance, load balancing and hashing using $n/\log^* n$ processors.

[GMW87] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In *Proc. 19th Ann. ACM Symp. on Theory of Computing*, pages 218–229, 1987. Goldreich et al. demonstrate the use of zero-knowledge proofs on proving the completeness theorem for protocols with honest majority.

[GMW91] O. Goldreich, S. Micali, and A. Wigderson. Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems. *Journal of the ACM*, 38(1):691–729, 1991. They show that for a language L in NP and a string w in L, there exists a probabilistic interactive proof that efficiently demonstrates membership of x in L without conveying additional information. Previously, zero-knowledge proofs were known only for some problems that were in both *NP* and *co-NP*. A preliminary version of this paper appeared in *Proc. 27th Ann. IEEE Symp. on Foundations of Computer Science*, 1986, under the title "Proofs that yield nothing but their validity and a methodology of cryptographic protocol design.".

[Gol92] M. Goldwurm. Probabilistic estimation of the number of prefixes of a trace. *Theoretical Computer Science*, 92:249–268, 1992. The author uses the result to determine the behavior of several algorithms relating to trace languages.

[Gon84] G.H. Gonnet. Determining the equivalence of expressions in random polynomial time. In *Proc. 16th Ann. ACM Symp. on Theory of Computing*, pages 334–341, 1984. Hashing functions are used to determine algebraic expression equivalence with a small probability of error. The probability of error can be made arbitrarily small, depending on the number of iterations of the algorithm. See [Gon86] for some related work.

[Gon86] G.H. Gonnet. New results for random determination of equivalence of expressions. In B.W. Char, editor, *ISSAC '86: Proc. Int'l. Symp. on Symbolic and Algebraic Computation*, pages 127–131. ACM Press, 1986. Some open problems in the same general area as that covered by [Gon84] are solved in this paper.

[GRSS93] M. Golin, R. Raman, C. Schwarz, and M. Smid. Randomized data structures for the dynamic closest-pair problem. In *Proc. Fourth Ann. ACM-SIAM Symp. on Discrete Algorithms*, pages 301–310, Austin, TX, January 1993. The authors describe a new randomized data structure, the *sparse partition*, for solving the dynamic closest-pair problem. Using this data structure, the closest pair of a set of n points in k-dimensional space, for any fixed k, can be found in constant time. If the points are chosen from a finite universe, and if the floor function is available at unit-cost, then the data structure supports insertions into and deletions from the set in expected $O(\log n)$ time and requires expected $O(n)$ space. Here, it is assumed that the updates are chosen by an adversary who does not know the random choices made by the data structure.

[GS89] S. Goldwasser and M. Sipser. Private coins versus public coins in interactive proof systems. *Advances in Computing Research 5: Randomness and Computation*, 1989. This work establishes equivalence between the notions of interactive proofs introduced in [GMR89] and [BM88]. (A preliminary version appeared in *Proc. 18th Ann. ACM Symp. on Theory of Computing*, 1986, pp. 59–68).

[Gup93] R. Gupta. Φ-test: Perfect hashed index test for response validation. In *Proc. 1993 IEEE Int'l. Conf. on Computer Design*, Cambridge, MA, Oct 1993. A scheme for checking the fidelity of test responses generated by a specially tailored sequence of test inputs is described. Randomized search is used to compute a special perfect hashing function $h(x)$ that map the expected test outcomes to the sequence $[1 \ldots m]$. This sequence is checked by a hardware implementation of $h(x)$ and an up-counter.

[GW86] A. G. Greenberg and A. Weiss. A lower bound for probabilistic algorithms for finite state machines. *Journal of Computer and System Sciences*, 33(1):88, August 1986. A proof that the running time cannot be better than $\Omega(2^{\frac{n}{2}}n)$ is presented.

[GY89] R. Graham and A. Yao. On the improbability of reaching Byzantine consensus. In *Proc. 21st Ann. ACM Symp. on Theory of Computing*, pages 467–478, 1989. The maximum probability $\beta_{n,t}$ of obtaining consensus is attacked for $t \geq n/3$ (For smaller values, deterministic algorithms are available, so $\beta_{n,t} = 1$.) The smallest non-trivial case, $\beta_{3,1}$, is shown to be $(\sqrt{(5)} - 1)/2$, the reciprocal of the golden ratio. In a restricted model, it is shown that for all ϵ, $0 < \epsilon < 1$, if $t/n > 1 - \frac{1-\log 1-\epsilon^{1/2}}{\log(1-(1-\epsilon)^{1/2})}$, then $\beta_{n,t} < \epsilon$.

[Had86] V. Hadzilacos. Ben-Or's randomized protocol for consensus in asynchronous systems. Course notes: Computer Science 2221F, Department of Computer Science, University of Toronto, October 1986. An elegant proof of the correctness of Ben-Or's [BO83] probabilistic algorithm for Byzantine agreement is presented.

[Hag91] T. Hagerup. Constant-time parallel integer sorting. In *Proc. 23rd Ann. ACM Symp. on Theory of Computing*, pages 299–306, New Orleans, LA, May 1991. Standard sorting algorithms return the elements of an array in nondecreasing order. In the *chain sorting problem*, the elements of a linked list are returned in nondecreasing order. This problem can be viewed as more primitive than the standard sorting problem as it does not involve list ranking computation, which is implicit in the standard problem. Hagerup presents several efficient randomized parallel algorithms for the chain sorting problem, some of which require only constant expected time.

[Har87] D. Harel. *Algorithmics: The Spirit of Computing.* Addison-Wesley, 1987. This book contains a well-written chapter on probabilistic algorithms and their complexity theory.

[Has90] J. Hastad. Pseudo-random generators under uniform assumptions. In *Proc. 22nd Ann. ACM Symp. on Theory of Computing*, pages 395–404, Baltimore, MD, May 1990. Hastad proves that given a function f that is one-way in the uniform model (i.e., cannot be inverted except on a vanishing fraction of the inputs by a probabilistic polynomial time Turing machine), it is possible to construct a pseudo random bit-generator that passes all probabilistic polynomial time statistical tests.

[Her92] T. Herman. Self-stabilization: randomness to reduce space. *Distributed Computing*, 6(2):95–98, 1992. Herman uses randomization to convert Dijkstra's k-state mutual exclusion protocol for unidirectional rings to a 3-state protocol.

[HM87] A. Hajnal and W. Maass. Threshold circuits of bounded depth. In *Proc. 28th Ann. IEEE Symp. on Foundations of Computer Science*, pages 99–109, 1987. Polynomial size threshold circuits of bounded depth are viewed as mechanisms for parallel computations, where elements of the circuit are threshold gates (output high if the weighted sum of inputs exceeds a set threshold). Probabilistic, deterministic, imprecise and unreliable threshold circuits are considered.

[Hoa74] C. A. R. Hoare. Monitors: An operating system structuring concept. *Communications of the ACM*, 17(2):549–557, October 1974. Erratum in *Communications of the ACM*, Vol. 18, No. 2, 1975. This paper contains one of the first solutions to the Dining Philosophers problem. A probabilistic algorithm for this problem is the subject of Section 3.1.

[Hoa78] C. A. R. Hoare. Communicating Sequential Processes. *Communications of the ACM*, 21:666–677, August 1978. Hoare's novel language CSP combined nondeterminism and synchronized message passing. Since its inception, various schemes have been proposed to add output guards to the language. In Section 3.2, we discuss a probabilistic algorithm for output guards.

[Hoa85] C. A. R. Hoare. *Communicating Sequential Processes.* Prentice-Hall International, U.K., 1985. Hoare's book contains an elegant message-passing solution to the Dining Philosophers problem. A probabilistic algorithm for this problem is the subject of Section 3.1.

[Hop81] J. E. Hopcroft. Recent directions in algorithmic research. In P. Deussen, editor, *Proc. Fifth Conf. on Theoretical Computer Science*, pages 123–134. Springer-Verlag, 1981. This work is an early survey of probabilistic algorithms.

[HS85] S. Hart and M. Sharir. Concurrent probabilistic programs, or: How to schedule if you must. *SIAM Journal on Computing*, 14(4):991–1012, November 1985. The authors analyze the worst-case probability of termination of a set of concurrently running processes. Each process may use randomization, and fair interleaving is assumed.

[HT82] J. H. Halton and R. Terada. A fast algorithm for the Euclidean Traveling Salesman problem, optimal with probability one. *SIAM Journal on Computing*, 11(1), Feb. 1982. Halton and Terada present an algorithm for the Travelling Salesman Problem over n points, which, for appropriate choice of a function σ takes less than $n\sigma(n)$ time and asymptotically converges to the minimum length tour, with probability one, as $n \to \infty$.

[ILL89] R. Impagliazzo, L. Levin, and M. Luby. Pseudorandom generation from one-way functions. In *Proc. 21st Ann. ACM Symp. on Theory of Computing*, pages 12–24, 1989. The existence of one-way functions is shown to be necessary and sufficient for the existence of pseudorandom generators. A one-way function $F(x)$ is one that is easily computed, but given $F(x)$, it should not be possible to easily recover x, either with a small circuit or with a fast algorithm. Algorithms for pseudorandom generators are provided that use one-way functions whose inverses are difficult to obtain using small circuits or fast algorithms. See also [Has90].

[IM83] O. H. Ibarra and S. Moran. Probabilistic algorithms for deciding the equivalence of straight-line programs. *Journal of the ACM*, 30(1):217–228, January 1983. They study the complexity of deciding the equivalence of straight-line programs, i.e., those in which there are no loops, and only statements of the form x := y, x := y + z, x := y - z, and x := y * z are permitted. Given two such programs P and Q, Ibarra and Moran ask the question: Is P = Q? If the domain of the variables is an infinite field such as the integers, then they show that there exists a polynomial-time probabilistic algorithm to solve this problem. If the domain is a finite field, the problem is shown to be *NP*-hard.

[IR81] A. Itai and M. Rodeh. The lord of the ring or probabilistic methods for breaking symmetry in distributed networks. Technical Report RJ 3110, IBM, San Jose, 1981. Itai and Rodeh consider the problems of choosing a leader and determining the size of a ring of indistinguishable processors. If the size of the ring is known, efficient probabilistic algorithms exit for choosing a leader. However, there exists no probabilistic solution to the problem of determining the size of a ring that can guarantee both termination and a non-zero probability of correctness.

[IRM81] O. H. Ibarra, L. Rosier, and S. Moran. Probabilistic algorithms and straight-line programs for some rank decision problems. In *Information Processing Letters*, volume 12, pages 227–232, 1981. Given a positive integer r and a matrix A with polynomial entries (where the polynomials are represented by arbitrarily parenthesized arithmetic expressions using +, -, *, and exponentiation to a positive constant), the problem of deciding whether A has rank r is reduced in polynomial time to the zero-equivalence problem (i.e., the problem of determining whether a program always outputs 0) of straight-line programs [MT85].

[IZ89] E. Impagliazzo and D. Zuckerman. How to recycle random bits. In *Proc. 30th Ann. IEEE Symp. on Foundations of Computer Science*, pages 248–253, 1989. This paper proves that two very simple pseudo-random number generators, which are minor modifications of linear congruential generator and the simple shift register generator, are good for amplifying the correctness of probabilistic algorithms.

[Jae81] G. Jaeschke. Reciprocal hashing: A method for generating minimal perfect hashing functions. *Communications of the ACM*, 24(12):829–823, Dec 1981. Hash functions, for a key x in a set S of positive integers, of the form $h(x) = (C/(Dx+E)) \bmod N$ are considered. Though the existence of h is guaranteed, the scheme suffers from many practical problems because of exhaustive nature of the search for h.

[JKS84] J. Ja'Ja', V. K. Prasanna Kumar, and J. Simon. Information transfer under different sets of protocols. *SIAM Journal on Computing*, 13(4):840–849, November 1984. This paper is a study of the communication complexity of information transfer in deterministic, random, non-deterministic and probabilistic computation models. It is widely conjectured that $P \subseteq R \subseteq NP \subseteq PP$ for polynomial time complexity classes. The authors prove that exponential gaps exist among the corresponding communication complexity classes.

[Joh90] D. S. Johnson. A catalog of complexity classes. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science,* Volume A: Algorithms and Complexity, chapter 9, pages 67–161. Elsevier and The MIT Press (co-publishers),

1990. Johnson presents an extensive survey of computational complexity classes. Of particular interest here is his discussion of randomized, probabilistic, and stochastic complexity classes.

[JS89] M. R. Jerrum and A. Sinclair. Approximating the permanent. *SIAM Journal on Computing*, 18(6):1149–1178, 1989. Broder [Bro86] related the task of approximating the permanent of a matrix to that of uniformly generating perfect matchings in a graph. This paper gives a randomized approximation scheme for the latter problem by simulating it as a Markov chain whose states are matchings in the graph. For this scheme to be efficient the Markov chain must be rapidly mixing, i.e. converge to its stationary distribution in a short time.

[JVV86] M. R. Jerrum, L. G. Valiant, and V. V. Vazirani. Random generation of combinatorial structures from a uniform distribution. *Theoretical Computer Science*, 43:169–188, 1986. This paper considers the class of problems involving the random generation of combinatorial structures from a uniform distribution. It is shown that *exactly* uniform generation of 'efficiently verifiable' combinatorial structures is reducible to approximate counting.

[Kal92] G. Kalai. A subexponential randomized simplex algorithm. In *Proc. 24th Ann. ACM Symp. on Theory of Computing*, pages 475–482, Victoria, B.C., Canada, May 1992. A randomized variant of the simplex algorithm is presented that, given a linear program with d variables and n constraints, uses an expected subexponential number of arithmetic operations.

[Kam89] M. Kaminski. A note on probabilistically verifying integer and polynomial products. *Journal of the ACM*, 36(1):845–876, 1989. The author describes probabilistic algorithms for verifying the product of two n-bit integers in $O(n)$ bit operations, and for verifying the product of two polynomials of degree n over integral domains in $4n + o(n)$ algebraic operations. The error probability is is $o(\frac{1}{n^{1-\epsilon}})$ for any $\epsilon > 0$.

[Kar86] R. M. Karp. Combinatorics, complexity and randomness. *Communications of the ACM*, 29(2):98–109, February 1986. This is the 1985 Turing Award Lecture. It traces the development of combinatorial optimization and computational complexity theory. It discusses probabilistic algorithms and probabilistic analysis of approximation algorithms for *NP*-complete optimization problems.

[Kar90] R. M. Karp. An introduction to randomized algorithms. Technical Report TR-90-024, Computer Science Division, University of California, Berkeley, CA 94704, 1990. A recent, comprehensive survey of randomized algorithms.

[Kar91] R. M. Karp. Probabilistic recurrence relations. In *Proc. 23rd Ann. ACM Symp. on Theory of Computing*, pages 190–197, New Orleans, LA, May 1991. In order to solve a problem instance of size x, a divide-and-conquer algorithm invests an amount of work $a(x)$ to break the problem into subproblems of sizes $h_1(x), h_2(x), \cdots, h_k(x)$, and then proceeds to solve the subproblems. When the h_i are random variables — because of randomization within the algorithm or because the instances to be solved are assumed to be drawn from a probability distribution — the running time of the algorithm on instances of size x is also a random variable $T(x)$. Karp gives several easy-to-apply methods for obtaining fairly tight bounds on the upper tails of the probability distribution of $T(x)$, and presents a number of typical applications of these bounds to the analysis of algorithms. The proofs of the bounds are based on an interesting analysis of optimal strategies in certain gambling games.

[Kar93] D. R. Karger. Global min-cuts in RNC, and other ramifications of a simple min-cut algorithm. In *Proc. Fourth Ann. ACM-SIAM Symp. on Discrete Algorithms*, pages 21–30, Austin, TX, January 1993. Given a graph with n vertices and m (possibly weighted) edges, the *min-cut* problem is to partition the vertices into two non-empty sets S and T so as to minimize the number of edges crossing from S to T (if the graph is weighted, the problem is to minimize the total weight of crossing edges). Karger gives an *RNC* algorithm for the min-cut problem which runs in time $O(\log^2 n)$ on a CRCW PRAM with $mn^2 \log n$ processors.

[Kel92] P. Kelsen. On the parallel complexity of computing a maximal independent set in a hypergraph. In *Proc. 24th Ann. ACM Symp. on Theory of Computing*, pages 339–350, Victoria, B.C., Canada, May 1992. A maximal independent set in a hypergraph is a subset of vertices that is maximal with respect to the property of not containing any edge of the hypergraph. Kelsen derandomizes the randomized algorithm of Beame and Luby to obtain the first sublinear time deterministic algorithm for hypergraphs with edges of size $O(1)$.

[KGY89] M. Kharitonov, A. V. Goldberg, and M. Yung. Lower bounds for pseudorandom number generators. In *Proc. 30th Ann. IEEE Symp. on Foundations of Computer Science*, pages 242–247, Research Triangle Park, NC, October 1989. IEEE Computer Society Press. A *pseudorandom generator* is a deterministic algorithm that expands a truly random seed into a longer *pseudorandom* string. Such generators play an important role in applications like cryptography. The authors provide lower bounds on the computational resources needed for the generation of pseudorandom strings.

[Kil88] J. Kilian. Zero-knowledge with log-space verifiers. In *Proc. 29th Ann. IEEE Symp. on Foundations of Computer Science*, pages 25–34, 1988. Interactive proof systems where the verifiers are assumed to be log-space probabilistic automata are considered. The class of languages that are amenable to zero-knowledge proofs with such verifiers is described.

[Kil90] J. Kilian. *Uses of Randomness in Algorithms and Protocols.* MIT Press, 1990. Kilian's Ph.D. dissertation, which was selected as an ACM Distinguished Dissertation for the year 1989, is in three parts. The first part describes a randomized algorithm to generate large prime numbers which have short, easily verified certificates of primality. The algorithm provides short, deterministically verifiable proofs of primality for all but a vanishing fraction of prime numbers. The second part considers the *secure circuit evaluation* problem in which two parties wish to securely compute some function on their private information. Kilian reduces this problem to an *oblivious transfer protocol*. The third part of the dissertation generalizes probabilistic interactive proof systems to multiple provers. He shows that any language that has a multi-prover interactive proof system has a zero-knowledge multi-prover interactive proof system.

[Kil92] J. Kilian. A note on efficient zero-knowledge proofs and arguments. In *Proc. 24th Ann. ACM Symp. on Theory of Computing*, pages 723–732, Victoria, B.C., Canada, May 1992. The standard definition of an interactive proof requires that the verifier accept a correct proof and reject an incorrect assertion with probability at least $\frac{2}{3}$. This paper shows how to efficiently reduce the error probability to less than 2^{-k}, where k is some easily adjustable security parameter.

[KL85] R. M. Karp and M. Luby. Monte-Carlo algorithms for planar multiterminal reliability problems. *Journal of Complexity*, 1:45–64, 1985. They present a general Monte-Carlo technique for obtaining approximate solutions of several enumeration and reliability problems including counting the number of satisfying assignments of a propositional formula given in disjunctive normal form (a #P-complete problem) and estimating the failure probability of a system. An earlier version appeared in *Proc. 24th Ann. IEEE Symp. on Foundations of Computer Science*, 1983, pp. 56–64. See also [KLM89].

[KL93] R. Klein and A. Lingas. A linear-time randomized algorithm for the bounded Voronoi diagram of a simple polygon. In *Proc. Ninth Ann. ACM Symp. on Computational Geometry*, pages 124–132, San Diego, CA, May 1993. For a polygon P, the *bounded Voronoi diagram* of P is a partition of P into regions assigned to the vertices of P. Klein and Lingas present a randomized algorithm that builds the bounded Voronoi diagram of a simple polygon in linear expected time.

[KLM89] R. M. Karp, M. Luby, and N. Madras. Monte-Carlo approximation algorithms for enumeration problems. *Journal of Algorithms*, 10:429–448, 1989. A companion paper of [KL85]; an earlier version appeared in *Proc. 24th Ann. IEEE Symp. on Foundations of Computer Science*, 1983, pp. 56–64.

[KLMadH92] R. M. Karp, M. Luby, and F. Meyer auf der Heide. Efficient PRAM simulation on a distributed memory machine. In *Proc. 24th Ann. ACM Symp. on Theory of Computing*, pages 318–326, Victoria, B.C., Canada, May 1992. They present a randomized simulation of an $n \log\log(n)\log^*(n)$-processor shared memory machine (PRAM) on an n-processor distributed memory machine (DMM) with optimal expected delay $O(\log\log(n)\log^*(n))$ per step of simulation.

[KM93] D. Koller and N. Megiddo. Constructing small sample spaces satisfying given constraints. In *Proc. 25th Ann. ACM Symp. on Theory of Computing*, pages 268–277, San Diego, CA, May 1993. The authors prove *NP*-completeness for the problem of finding small sample spaces for joint distributions of n discrete random variables satisfying a given set of constraints of the form $\Pr(Event) = \pi$. For the case where the constraints have a certain form and are consistent with a joint distribution of independent random variables, a small sample space can be constructed in polynomial time; a result that can be used to derandomize algorithms.

[KMO89] J. Kilian, S. Micali, and R. Ostrovsky. Minimum resource zero-knowledge proof. In *Proc. 30th Ann. IEEE Symp. on Foundations of Computer Science*, pages 474–479, Oct 1989. The various resources such as number of envelopes, number of oblivious transfers, and total amount of communication required by zero-knowledge protocols are considered. The paper presents a technique of executing k rounds of a protocol, which guarantees that any polynomial number of NP-theorems can be proved non-interactively in zero-knowledge, with the probability of accepting a false theorem below $1/2^k$. The main result in this paper assumes the existence of trap-door permutations in order to implement Oblivious Transfer Protocol.

[KMP77] D. E. Knuth, J. H. Morris, and V. R. Pratt. Fast pattern matching in strings. *SIAM Journal on Computing*, 6:323–350, 1977. This paper presents a fast deterministic algorithm for the problem of determining if a given pattern of m symbols occurs in a text of length n. Their well-known algorithm runs in time $O(n + m)$, making judicious use of a *prefix function*, which for a given pattern encapsulates knowledge about how the pattern matches against shifts of itself.

[KMRZ93] E. Kushilevitz, Y. Mansour, M. O. Rabin, and D. Zuckerman. Lower bounds for randomized mutual exclusion (extended abstract). In *Proc. 25th Ann. ACM Symp. on Theory of Computing*, pages 154–163, San Diego, CA, May 1993. The authors establish a lower bound of $\Omega(\log\log n)$ bits on the size of the shared variable required by randomized mutual exclusion algorithms ensuring strong fairness. Slightly weakening the fairness condition results in an exponential reduction in the size of the required shared variable.

[Knu73] D. E. Knuth. *The Art of Computer Programming, Vol. 3: Sorting and Searching*. Addison-Wesley, 1973. This volume is a repository of sorting and searching algorithms and their analysis. It contains a detailed and thorough treatment of hashing.

[Ko82] K. Ko. Some observations on probabilistic algorithms and NP-Hard problems. *Information Processing Letters*, 14(1):39–43, March 1982. Ko shows that if there is a probabilistic algorithm for an *NP*-hard problem with a small "two-sided error", then there is a probabilistic algorithm for any NP-complete problem with a small "one-sided error".

[Koz81] D. Kozen. Semantics of probabilistic programs. *Journal of Computer and System Sciences*, 22(3):328–350, 1981. A novel attempt at defining the semantics of probabilistic programs. Two equivalent semantics are presented.

[Koz85] D. Kozen. A probabilistic PDL. *Journal of Computer and System Sciences*, 30(2):162–178, 1985. Kozen defines a formalism for reasoning about probabilistic programs at the propositional level. Probabilistic Propositional Dynamic Logic (PPDL), which has an arithmetic extension for each logical construct in PDL, is presented along with some decision procedure formulas and a deductive calculus.

[KPRR92] Z. M. Kedem, K. V. Palem, M. O. Rabin, and A. Raghunathan. Efficient program transformations for resilient parallel computation via randomization. In *Proc. 24th Ann. ACM Symp. on Theory of Computing*, pages 306–317, Victoria, B.C., Canada, May 1992. The authors show how randomization can be used to automatically transform an arbitrary program written for an ideal parallel machine to run on a completely asynchronous machine, such that the resulting program is work and space efficient relative to the ideal program from which it was derived.

[KPS85] R. M. Karp, N. Pippenger, and M. Sipser. A time randomness tradeoff. In *AMS Conf. on Probabilistic Computational Complexity*, Durham, New Hampshire, 1985. This paper gives the first example of deterministic amplification using expander graphs.

[KPU88] D. Krizanc, D. Peleg, and E. Upfal. A time-randomness tradeoffs for oblivious message routing. In *Proc. 20th Ann. ACM Symp. on Theory of Computing*, pages 93–102, 1988. Given the probability Q that an algorithm fails to complete its task in T steps, a lower bound on the entropy of the random source used in the algorithm is obtained. Near-optimal algorithms for oblivious packet-routing in a bounded-degree network are included (see also [PU90]).

[KR87] R. M. Karp and M. O. Rabin. Efficient randomized pattern-matching algorithms. *IBM Journal of Research and Development*, 31(2):249–260, March

1987. An elegant randomized algorithm for the string matching problem is presented. Mismatches reported by the algorithm are always correct, while a claimed match may be erroneous with small probability. The algorithm uses a fingerprinting function (on the finite field of $\bmod p$ residues, where p is chosen at random) to efficiently check for occurrences of the pattern string in the text string. The running time of the algorithm is $O((n-m+1)m)$ in the worst case, where the text is of length n and the pattern is of length m, but can be expected to run in time $O(n+m)$ in practice. The probability that the algorithm reports a false match is $1/n$. Two-dimensional patterns are also considered. An earlier version of this paper appeared as Technical Report TR-31-81, Aiken Computation Lab, Harvard University, 1981.

[KR88] H. Karloff and P. Raghavan. Randomized algorithms and pseudorandom numbers. In *Proc. 20th Ann. ACM Symp. on Theory of Computing*, pages 310–321, 1988. Following up on Bach's work [Bac91], this paper studies pseudo-random substitutes (with small seeds) for purely random choices in sorting, selection and oblivious message routing. An interesting result is that the linear congruence pseudo-random number generator proposed by Knuth [Knu73] can interact with some quicksort algorithms.

[Kro85] L. Kronsjo. *Computational Complexity of Sequential and Parallel Algorithms.* John Wiley and Sons, New York, 1985. Chapter 5, Section 5.3, addresses probabilistic algorithms. Rabin's algorithms for primality and the Nearest Neighbors problem are described.

[KRR91] H. Karloff, Y. Rabani, and Y. Ravid. Lower bounds for randomized k-server and motion planning. In *Proc. 23rd Ann. ACM Symp. on Theory of Computing*, pages 278–288, New Orleans, LA, May 1991. Lower bounds are proved on the competitive ratio of randomized algorithms for the on-line k-server problem and an on-line motion-planning problem.

[KRT93] M.-Y. Kao, J. H. Reif, and S. R. Tate. Searching in an unknown environment: An optimal randomized algorithm for the cow-path problem. In *Proc. Fourth Ann. ACM-SIAM Symp. on Discrete Algorithms*, pages 441–447, Austin, TX, January 1993. The first randomized algorithm for the w-lane cow-path problem, a problem of searching in an unknown environment, is given. The algorithm is optimal for $w = 2$ and evidence is supplied that it is optimal for larger values of w.

[KS92] P. N. Klein and S. Sairam. A parallel randomized approximation scheme for shortest paths. In *Proc. 24th Ann. ACM Symp. on Theory of Computing*, pages 750–758, Victoria, B.C., Canada, May 1992. A randomized algorithm is given for approximate shortest path computation in an undirected weighted graph.

[KS93] D. R. Karger and C. Stein. An $\tilde{O}(n^2)$ algorithm for minimum cuts. In *Proc. 25th Ann. ACM Symp. on Theory of Computing*, pages 757–765, San Diego, CA, May 1993. A *minimum cut* is a set of edges of minimum weight whose removal disconnects a given graph. Karger and Stein give a strongly polynomial randomized algorithm which finds a minimum cut with high probability in $O(n^2 \log^3 n)$ time. Their algorithm can be implemented in *RNC* using only n^2 processors, and is thus the first efficient *RNC* algorithm for the min-cut problem.

[KST90] P. Klein, C. Stein, and E. Tardos. Leighton-Rao might be practical: Faster approximation algorithms for concurrent flow with uniform capacities. In *Proc. 22nd Ann. ACM Symp. on Theory of Computing*, pages 310–321, Baltimore, MD, May 1990. They give an $O(m^2 \log m)$ expected-time randomized algorithm for approximately solving the concurrent multicommodity flow problem with uniform capacities.

[Kur87] S. A. Kurtz. A note on random polynomial time. *SIAM Journal on Computing*, 16(5):852–853, October 1987. Shows that $P^A \cap P^B = BPP$ with probability 1 for independent random sets A and B. Here, A and B are sets consisting of strings chosen at random, and P^A and P^B are relativized to A and B respectively. See [Gil77] for additional notation.

[KUW86] R. M. Karp, E. Upfal, and A. Wigderson. Constructing a perfect matching in Random NC. *Combinatorica*, 6:35–48, 1986. Perfect matching is a fundamental problem that is not known to be solvable by an *NC* algorithm, i.e., a parallel algorithm running in time polynomial in $\log n$ and using a number of processors polynomial in n. This paper proves that perfect matching is in random *NC* and gives a fast, parallel, randomized algorithm for finding a perfect matching in a simple graph.

[KVV90] R. M. Karp, U. V. Vazirani, and V. V. Vazirani. An optimal algorithm for on-line bipartite matching. In *Proc. 22nd Ann. ACM Symp. on Theory of Computing*, pages 352–358, Baltimore, MD, May 1990. An on-line algorithm receives a sequence of requests and must respond to each request as soon as it is received. In contrast, an off-line algorithm may wait until all requests have been received before determining its responses. The authors give a simple, randomized, optimal, on-line algorithm for bipartite matching.

[KW85] R. M. Karp and A. Wigderson. A fast parallel algorithm for the maximal independent set problem. *Journal of the ACM*, 32(4):762–773, 1985. This important paper showed that the maximal independent set problem for graphs can be solved in polylogarithmic time using a polynomial number of processes on a PRAM in which concurrent reads and writes are disallowed. They derive their algorithm from a randomized one using a technique that has become known as derandomization via k-wise independence.

[KZ88] R. M. Karp and Y. Zhang. A randomized parallel branch and bound procedure. In *Proc. 20th Ann. ACM Symp. on Theory of Computing*, pages 290–300, 1988. A general technique assuming no special communication capabilities is presented.

[Lak90] Y. N. Lakshman. On the complexity of computing a Gröbner basis for the radical of a zero dimensional ideal. In *Proc. 22nd Ann. ACM Symp. on Theory of Computing*, pages 555–562, Baltimore, MD, May 1990. Lakshmanan shows that if a system of polynomials $f_1, f_2, \ldots, f_r$ in n variables with $deg(f_i) \leq d$ over the rational numbers has only finitely many affine zeros, then all the affine zeros can be determined in time polynomial in d^n by a Las Vegas type randomized algorithm.

[LC88] T.G. Lewis and C.R. Cook. Hashing for dynamic and static internal tables. *Computer*, 21:45–56, 1988. The authors survey the classical hashing function approach to information retrieval and show how general hashing techniques exchange speed for memory. It is a tutorial paper that covers, among other topics, dynamic and static hash tables, perfect hashing, and minimal perfect hashing.

[Leh27] D. H. Lehmer. *Bulletin of the American Mathematical Society*, 33:327–340, 1927. This paper presents the Lucas-Lehmer heuristic for primality testing.

[Leh82] D. Lehmann. On primality tests. *SIAM Journal on Computing*, 11(2), May 1982. Lehmann presents two algorithms for testing primality based on the extended Riemann hypothesis. The second algorithm is faster than that proposed by [SS77] as it does not involve computing the Jacobi symbol.

[Lei92] T. Leighton. Methods for message routing on parallel machines. In *Proc. 24th Ann. ACM Symp. on Theory of Computing*, pages 77–96, Victoria, B.C., Canada, May 1992. This survey includes the topic of randomized wiring.

[LFKN90] C. Lund, L. Fortnow, H. Karloff, and N. Nisan. Algebraic methods for interactive proof systems. In *Proc. 31st Ann. IEEE Symp. on Foundations of Computer Science*, pages 2–10, 1990. The authors present a new algebraic technique for constructing IP systems and prove that every language in the polynomial time hierarchy has an interactive proof system. This is a key paper essential for proving $IP = PSPACE$ [Sha92b] and $MIP = NEXP$ [BFL90].

[LLM90] F. T. Leighton, D. Lisinski, and B. M. Maggs. Empirical evaluation of randomly-wired multistage networks. In *Proc. 1990 IEEE Int'l. Conf. on Computer Design*, pages 380–385, 1990. This paper presents simulation results comparing the fault-tolerance, delay and other characteristics of butterflies, dilated butterflies and randomly-wired multibutterflies. Randomly-wired multibutterflies perform better by many yardsticks.

[LLS87] D. Lichtenstein, N. Linial, and M. Saks. Imperfect random sources and discrete controlled processes. In *Proc. 19th Ann. ACM Symp. on Theory of Computing*, pages 169–177, 1987. Imperfect sources are modeled by discrete control processes where the output string of zeros and ones has been tampered with by a controller who can specify certain bits. Several questions concerning the membership of such a string in a prespecified set L are answered.

[LLW88] N. Linial, L. Lovász, and A. Wigderson. Rubber bands, convex embeddings, and graph connectivity. *Combinatorica*, 8:91–102, 1988. Several probabilistic algorithms for connectivity computation, both of the Monte Carlo and Las Vegas variety, are given, as is a formalization of the connectivity problem in terms of embedded graphs. Efficient parallel implementations are included. (First appeared under the title "A physical interpretation of graph connectivity and its algorithmic applications" in *Proc. 27th Ann. IEEE Symp. on Foundations of Computer Science*, 1986, pp. 39–53.).

[LM89] F. T. Leighton and B. M. Maggs. Expanders might be practical: fast algorithms for routing around faults in multibutterflies. In *Proc. 30th Ann. IEEE Symp. on Foundations of Computer Science*, pages 384–389, 1989. This paper contains a simpler version of Upfal's results [Upf89] and algorithms for routing on randomized multibutterflies in the presence of faults.

[LM92a] F. T. Leighton and B. M. Maggs. Fast algorithms for routing around faults in multibutterflies and randomly-wired splitter networks. *IEEE Trans. on Computers*, 41(5):578–587, May 1992. This paper describes simple deterministic $O(\log N)$-step algorithms for routing permutations of packets in multibutterflies and randomly-wired splitter networks. The algorithms are robust against faults (even in the worst case) and are efficient from a practical point of view.

[LM92b] F. T. Leighton and B. M. Maggs. The role of randomness in the design of interconnection networks. *Information Processing*, I:291–305, 1992. A survey of recent research on randomly wired interconnection networks, which have been found to be exceptionally fault-tolerant and well-suited for both packet-routing and circuit-switching applications.

[LMP$^+$91] F. T. Leighton, F. Makedon, S. Plotkin, C. Stein, E. Tardos, and S. Tragoudas. Fast approximation algorithms for multicommodity flow problems. In *Proc. 23rd Ann. ACM Symp. on Theory of Computing*, pages 101–111, New Orleans, LA, May 1991. The paper presents randomized algorithms for approximately solving the multicommodity flow problem. The algorithms run in polynomial time with high probability.

[Lov79] L. Lovasz. On determinants, matchings and random algorithms. In L. Budach, editor, *Fundamentals of Computing Theory*. Akademia-Verlag, Berlin, 1979. Lovasz describes a probabilistic method for determining the perfect matching in a simple graph, if one exists, using Tutte's theorem.

[LP90] F. T. Leighton and C. G. Plaxton. A (fairly) simple circuit that (usually) sorts. In *Proc. 31st Ann. IEEE Symp. on Foundations of Computer Science*, pages 264–274, 1990. A k-round tournament over $n = 2^k$ payers which has very good sorting properties is introduced. There properties are then exploited in a sorting network and two randomized algorithms.

[LPV81] G. Lev, N. Pippenger, and L. Valiant. A fast parallel algorithm for routing in permutation networks. *IEEE Trans. on Computers*, C-30(2):93–100, February 1981. This paper presents deterministic algorithms for routing in permutation networks. The fastest algorithms require global knowledge and $\Omega(\log^2 N)$ parallel time.

[LR81] D. Lehmann and M. O. Rabin. On the advantage of free choice: A symmetric and fully distributed solution to the Dining Philosophers problem (extended abstract). In *Proc. Eighth Ann. ACM Symp. on Principles of Programming Languages*, pages 133–138, 1981. A classic paper in the area of randomized distributed algorithms. They show there is no deterministic, deadlock-free, truly distributed and symmetric solution to the Dining Philosophers problem, and describe a simple probabilistic alternative.

[LS91] D. Lapidot and A. Shamir. Fully parallelized multi-prover protocols for NEXP-time. In *Proc. 32nd Ann. IEEE Symp. on Foundations of Computer Science*, pages 13–18, 1991. This paper presents a one-round, zero-knowledge protocol (without cryptographic assumptions) for every language in *NEXP-time*. In a multi-prover protocol, several provers try to convince a polynomial-time verifier that a string X belongs in language L. Provers cannot communicate among themselves or observe communications between the verifier and other provers. The protocol ensures that if X is not in L, the probability that the verifier accepts the string as belonging to L is exponentially small.

[LS92] L. Lovasz and M. Simonovits. On the randomized complexity of volume and diameter. In *Proc. 33rd Ann. IEEE Symp. on Foundations of Computer Science*, pages 482–492, 1992. The authors present an $O(n^7 \log^2 n)$ algorithm to approximate the volume of a convex body, and an $O(n^6 \log n)$ algorithm to sample a point from the uniform distribution over a convex body.

[LS93] J. Lutz and W. Schmidt. Circuit size relative to pseudo-random oracles. *Theoretical Computer Science*, 107:95–120, 1993. Assuming pseudo-random oracles, circuit-size complexity is compared with deterministic and non-deterministic complexity. The paper also shows that for every p-space random oracle A and almost every oracle A in $EPSPACE$, NP^A is not contained in $SIZE^A(2^{\alpha n})$ for any real $\alpha < 1/3$, and E^A is not contained in $SIZE^A(2^n/n)$.

[LSP82] L. Lamport, R. Shostak, and M. Pease. The Byzantine Generals problem. *ACM Trans. on Programming Languages and Systems*, 4(3):382–401, July 1982. They proved that Byzantine agreement (the subject of Section 3.5) cannot be reached unless fewer than one-third of the processes are faulty. This result assumes that authentication, i.e., the crypting of messages to make them unforgeable, is not used. With unforgeable messages, they show that the problem is solvable for any $n \geq t > 0$, where n is the total number of processes and t is the number of faulty processes.

[Lut92] J. Lutz. On independent random oracles. *Theoretical Computer Science*, 92:301–307, 1992. This paper shows that for every random language $A \oplus B$, $P(A) \cap P(B) = BPP$, where $P(A)$ and $P(B)$ are the class of languages in polynomial time relativized to A and B. This improves on the results of [Kur87].

[LV92] J.-H. Lin and J. S. Vitter. ϵ-approximations with minimum packing constraint violation. In *Proc. 24th Ann. ACM Symp. on Theory of Computing*, pages 771–782, Victoria, B.C., Canada, May 1992. Efficient randomized and deterministic algorithms are presented for transforming optimal solutions for a type of relaxed integer linear program into provably good approximate solutions for the corresponding *NP*-hard discrete optimization problem.

[MadH85] F. Meyer auf der Heide. Simulating probabilistic by determining algebraic computation trees. *Theoretical Computer Science*, 41:325–330, 1985. This paper overlaps with the paper "Nondeterministic Versus Probabilistic Linear Search Algorithms," *Proc. 26th Ann. IEEE Symp. on Foundations of Computer Science*, 1985, pp. 65–73. It is shown that nondeterministic algorithms are less complex than their probabilistic counterparts even when the probabilistic choices are assigned zero cost and error is allowed in all computations. The specific algorithms considered are linear search algorithms.

[MadH90] F. Meyer auf der Heide. Dynamic hashing strategies. In *Proc. 15th Symp. on Mathematical Foundations of Computer Science,* Lecture Notes in Computer Science, Vol. 452, pages 76–87, Banska Bystrica, Czechoslovakia, August 1990. Springer-Verlag. This paper contains a survey of dynamic hashing techniques. It evaluates hashing algorithms with respect to probability of collisions, bucket sizes, evaluation time, and the time needed to construct a hash function. Parallel, distributed and sequential algorithms are considered.

[MC87] D. Mitra and R. A. Cieslak. Randomized parallel communication on an extension of the Omega network. *Journal of the ACM*, 34(4):802–824, 1987. This is an extension of Valiant and Aleliunas' algorithm to eliminate the need for scheduling. This algorithm also works on networks of fixed degree nodes.

[Meh82] K. Mehlhorn. On the program size of perfect and universal has functions. In *Proc. 23rd Ann. IEEE Symp. on Foundations of Computer Science*, pages 170–175, 1982. A must for readers interested in perfect hashing. It proves that for n distinct keys from $[0 \ldots N-1]$, there exists a prime number $p = O(n^2 ln(N))$ such that for any two keys x_i and x_j, $x_i(modp) \neq x_j(modp)$. Further, a good deterministic algorithm exists for finding p; it can be determined even more efficiently using a randomized algorithm. Several other results concerning the construction and length of perfect and universal hashing functions are proved.

[Meh84a] K. Mehlhorn. *Data Structures and Algorithms 1: Sorting and Searching*, volume 1 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, 1984. Volume 1 of this three-volume series is an excellent source for searching and sorting algorithms. It contains sections on quicksort (Section II.1.3), perfect hashing (Section III.2.3)and universal hashing (Sections III.2.3).

[Meh84b] K. Mehlhorn. *Graph Algorithms and NP-Completeness*, volume 2 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, 1984. Section IV.9.2 gives a probabilistic algorithm for graph connectivity and Section VI.8 deals, in part, with primality testing.

[Meh84c] K. Mehlhorn. *Multi-dimensional searching and computational geometry*, volume 3 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, 1984. This book is the last of three volmes. Chapter 7 is devoted to multi-dimensional data structures and Chapter 8 to problems in computational geometry.

[Mig80] M. Mignotte. Tests de primalite. *Theoretical Computer Science*, 12:109–117, 1980. Surveys the field of primality testing from a computational complexity perspective. In French.

[Mil76] G. L. Miller. Reimann's Hypothesis and test for primality. *Journal of Computer and System Sciences*, 13:300–317, 1976. A seminal paper in the development of primality testing algorithms. This paper presents two algorithms for primality testing. The first one runs in $O(n^{\frac{1}{7}})$ time. The second one, which is actually a polynomial time algorithm ($O(\log^4 n)$), assumes the Extended Reimann Hypothesis. This paper also proves a certain class of functions is computationally equivalent to factoring integers. (This paper first appeared in *Proc. Seventh Ann. ACM Symp. on Theory of Computing*, 1975, pp. 234–239.).

[MMN93] J. Matoušek, D. M. Mount, and N. S. Netanyahu. Efficient randomized algorithms for the repeated median line estimator. In *Proc. Fourth Ann. ACM-SIAM Symp. on Discrete Algorithms*, pages 74–82, Austin, TX, January 1993. Computing a statistical estimator can be viewed as the problem of fitting a straight line to a collection of n points in the plane. The *breakdown point* of an estimator is the fraction of outlying data points (up to 50%) that may cause the estimator to take on an arbitrarily large aberrant value. The authors present a (not-so simple) $O(n \log n)$ randomized expected time algorithm for the problem of computing a 50%-breakdown-point estimator, namely, the Siegel, or repeated median, estimator. A simpler $O(n \log^2 n)$ randomized algorithm for the problem is also given, which the authors contend actually has $O(n \log n)$ expected time for "many realistic input distributions.".

[MNN89] R. Motwani, J. Naor, and M. Naor. The probabilistic method yields deterministic parallel algorithms. In *Proc. 30th Ann. IEEE Symp. on Foundations of Computer Science*, pages 8–13, Research Triangle Park, NC, Oct 1989. This paper presents a method of converting randomized parallel algorithms into deterministic parallel (NC) algorithms. Their approach is based on a parallel implementation of the method of conditional probabilities due to Joel Spencer [Spe88], which was originally introduced with the aim of converting probabilistic proofs of existence of combinatorial structures into deterministic algorithms that can actually construct these structures. Restrictions on the technique to a certain class of randomized NC algorithms are discussed.

[MNT93] Y. Mansour, N. Nisan, and P. Tiwari. The computational complexity of universal hashing. *Theoretical Computer Science*, 107:121–133, 1993. They prove that any implementation of universal hashing from n-bit strings to m-bit strings requires a time-space tradeoff of $TS = \Omega(nm)$.

[Mon80] L. Monier. Evaluation and comparison of two efficient probabilistic primality testing algorithms. *Theoretical Computer Science*, 12:97–108, 1980. Monier presents an interesting comparison of the Miller-Rabin [Rab76] and Solovay-Strassen [SS77] primality testing algorithms, showing that the former is always more efficient than the latter. In the process, he proves that at least 3/4 of the numbers in the set $\{1, 2, ..., n-1\}$ are witnesses to the compositeness of n, for n composite. This strengthens the bound given in [Rab76].

[MOOY92] A. Mayer, Y. Ofek, R. Ostrovsky, and M. Yung. Self-stabilizing symmetry breaking in constant-space. In *Proc. 24th Ann. ACM Symp. on Theory of Computing*, pages 667–678, Victoria, B.C., Canada, May 1992. A randomized protocol is presented for the problem of self-stabilizing round-robin token management scheme on an anonymous bidirectional ring of identical processors.

[Mor82] S. Moran. On accepting density hierarchy in NP. *SIAM Journal on Computing*, 11(2), May 1982. Moran investigates a characterization of sets in *NP* based on accepting density of a polynomial time nondeterministic algorithm. The accepting density is defined as the ratio between the accepting computations and the total number of computations.

[MR89] G. L. Miller and J. H. Reif. Parallel tree contraction, Part 1: Fundamentals. In S. Micali, editor, *Advances in Computing Research 5: Randomness and Computation*. JAI Press, Greenwich, CT, 1989. They exhibit a randomized parallel algorithm for subtree isomorphism that uses $O(\log n)$ time and $O(n/\log n)$ processors. This was the first polylog parallel algorithm for the problem. See also the related paper "Parallel tree contraction and its applications," in *Proc. 26th Ann. IEEE Symp. on Foundations of Computer Science*, 1985, pp. 478–489; and the companion paper [MR91].

[MR91] G. L. Miller and J. H. Reif. Parallel tree contraction, Part 2: Further Applications. *SIAM Journal on Computing*, 20(6):1128–1147, December 1991. In this follow-up of [MR89], the authors present many applications of their "parallel tree contraction technique," including algorithms for subexpression evaluation, tree and graph isomorphism, and building cononical forms of trees and planar graphs.

[MS82] K. Mehlhorn and E. Schmidt. Las Vegas is better than determinism in VLSI and distributed computing. In *Proc. 14th Ann. ACM Symp. on Theory of*

Computing, pages 330–337, 1982. This paper demonstrates a problem where the theoretical lower bounds for distributed deterministic solutions can be improved using randomness. Let $X = (x_1, x_2, \ldots x_n)$, $Y = (y_1, y_2, \ldots y_n)$, where x_i and y_i are integers between 0 and $2^n - 1$, be stored on two different sites. The function $f(X, Y)$—which is defined to be 1 if there exists an i such that $x_i = y_i$, and 0 otherwise—is to be computed with minimum communication. This problem requires n^2 message bits in the deterministic case, but an $O(n \log n \log n)$ average running-time probabilistic algorithm is demonstrated.

[MS88] S. Micali and A. Shamir. An improvement of the Fiat-Shamir identification and signature scheme. In *Advances in Cryptology–CRYPTO 88,* Lecture Notes in Computer Science, Vol. 403. Springer-Verlag, 1988. They speed up zero-knowledge based identification and digital signature schemes of Fiat and Shamir, which require only 10 to 30 modular multiplications per party. Their improved scheme reduces the verifier's complexity to less than 2 modular multiplications and leaves the prover's complexity unchanged.

[MS92] B. M. Maggs and R. K. Sitaraman. Simple algorithms for routing on butterfly networks with bounded queues. In *Proc. 24th Ann. ACM Symp. on Theory of Computing*, pages 150–161, Victoria, B.C., Canada, May 1992. The authors present a simple, but non-pure, algorithm for routing a random problem on a fully loaded N-input butterfly with bounded-size queues in $O(\log N)$ steps, with high-probability.

[MSV85] F. Maffioli, M. G. Speranza, and C. Vercellis. Randomized algorithms. *Combinatorial Optimization—Annotated Bibliographies*, pages 89–105, 1985. This is a useful annotated bibliography on randomized algorithms.

[MSW92] J. Matousek, M. Sharir, and E. Welzl. A subexponential bound for linear programming. In *Proc. Eighth Ann. ACM Symp. on Computational Geometry*, pages 1–8, Berlin, Germany, June 1992. They present a simple randomized algorithm which solves linear programs with n constants and d variables in expected $O(nde^{4\sqrt{d \ln(n+1)}})$ time in the unit cost model.

[MT85] U. Manber and M. Tompa. Probabilistic, nondeterministic and alternating decision trees. *Journal of the ACM*, 32(3):720–732, July 1985. This paper compares lower bounds on the running times of algorithms that allow probabilistic, non-deterministic and alternating control on decision trees. Decision trees that allow internal randomization at the expense of a small probability of error are shown to run no faster asymptotically than ordinary decision trees for a collection of problems. An earlier version of this publication appeared in *Proc. 14th Ann. ACM Symp. on Theory of Computing*, 1982, pp. 234–244.

[Mul] K. Mulmuley. Computational geometry: An introduction through randomized algorithms. This book, due out in Fall 1993, presents a number of randomized algorithms for problems in computational geometry. The book is meant to serve as an introduction to computational geometry; the author chooses randomized algorithms to do the job as they are usually simpler to understand than their deterministic counterparts. The book is divided into two parts, basics and applications. Application areas considered include arrangements of hyperplanes, convex polytopes, range search, and computer graphics. A chapter on derandomization is also given.

[Mul89] K. Mulmuley. On obstructions in relation to a fixed view point. In *Proc. 30th Ann. IEEE Symp. on Foundations of Computer Science*, pages 592–597, Oct 1989. Randomized algorithms for the following computational geometry problems are given: (1) construction of levels of order 1 to k in an arrangement of hyperplanes; (2) construction of Voronoi diagrams of order 1 to k, and (3) hidden surface removal for a general scene. Both (1) and (2) are solved in any dimension, and (3) allows intersection of curved surfaces.

[Mul91a] K. Mulmuley. Randomized multidimensional search trees: Dynamic sampling. In *Proc. Seventh Ann. ACM Symp. on Computational Geometry*, pages 121–131, North Conway, NH, June 1991. This paper develops a general technique, called dynamic sampling, that can be used to "dynamize" randomized incremental algorithms, so to allow additions as well as deletions of objects from multidimensional search trees.

[Mul91b] K. Mulmuley. Randomized multidimensional search trees: Further results in dynamic sampling. In *Proc. 32nd Ann. IEEE Symp. on Foundations of Computer Science*, pages 216–227, 1991. This paper extends the approach presented in [Mul91c] to Nearest Neighbors and other problems.

[Mul91c] K. Mulmuley. Randomized multidimensional search trees: Lazy balancing and dynamic shuffling. In *Proc. 32nd Ann. IEEE Symp. on Foundations of Computer Science*, pages 180–196, 1991. This paper presents a general randomized algorithm for problems such as the construction and management of Convex Hulls and Voronoi Diagrams.

[Mul92] K. Mulmuley. Randomized geometric algorithms and pseudo-random generators. In *Proc. 33rd Ann. IEEE Symp. on Foundations of Computer Science*, pages 90–100, 1992. This paper shows that a generalization of the familiar linear congruential pseudo-random generator that uses $O(\log n)$ bits can be substituted for the random source in many randomized incremental algorithms used in computational geometry without affecting the order of complexity of the expected running time, thereby reducing the number of truly random bits needed.

[Mut93] S. Muthukrishnan. Detecting false matches in string matching algorithms. In *Proc. Fourth Int'l. Conf. on Combinatorial Pattern Matching,* Lecture Notes in Computer Science, Vol. 684, pages 164–178, Padova, Italy, 1993. Springer-Verlag. The Karp and Rabin randomized string matching algorithm [KR87] may report, with a small probability, a false match. Muthukrishnan presents a parallel algorithm to detect the existence of such a false match. His algorithm runs in $O(1)$ time and uses $O(n)$ CRCW PRAM processors, where n is the length of the input text, and can be used to efficiently convert the Monte Carlo Type string matching algorithm of Karp and Rabin into a Las Vegas type algorithm. Muthukrishnan also considers the problem of detecting *all* false matches.

[MV91] Y. Matias and U. Vishkin. Converting high probability into nearly constant time – with applications to parallel hashing. In *Proc. 23rd Ann. ACM Symp. on Theory of Computing*, pages 307–316, New Orleans, LA, May 1991. Randomized parallel algorithms are given for constructing a perfect hash function in expected polylogarithmic time and for generating a random permutation in polylogarithmic time.

[MVN93] Y. Matias, J. S. Vitter, and W.-C. Ni. Dynamic generation of discrete random variables. In *Proc. Fourth Ann. ACM-SIAM Symp. on Discrete Algorithms*, pages 361–370, Austin, TX, January 1993. Efficient randomized algorithms are given to generate a random variate distributed according to a dynamically set of weights. The base version of each algorithm generates the discrete random variate in $O(\log^* N)$ expected time and updates a weight in $O(2^{\log^* N})$ expected time in the worst case. It is shown how to reduce the update time to $O(\log^* N)$ amortized expected time.

[MVO91] A. Menezes, S. Vanstone, and T. Okamoto. Reducing elliptic curve logarithms to logarithms in a finite field. In *Proc. 23rd Ann. ACM Symp. on Theory of Computing*, pages 80–89, New Orleans, LA, May 1991. They present a probabilistic polynomial-time algorithm for the elliptic curve logarithm problem, the first subexponential-time, general-purpose algorithm for the problem.

[MVV87] K. Mulmuley, U. V. Vazirani, and V. V. Vazirani. Matching is as easy as matrix inversion. *Combinatorica*, 7:105–113, 1987. An elegant parallel, randomized algorithm for finding a perfect matching in a simple graph based on Tutte's matrix is presented. The algorithm, which is made possible by a probabilistic lemma called the isolation lemma, requires inversion of a single integer matrix which can be parallelized.

[MW90] B. McKay and N. Wormald. Uniform generation of random graphs of moderate degree. *Journal of Algorithms*, 11:52–67, 1990. A randomized algorithm is given for generating k-regular graphs on n vertices, uniformly at random. The expected running time of the algorithm is $O(nk^3)$ for $k = O(n^{\frac{1}{3}})$. Special cases, such as bipartite graphs with given degree sequences, are considered.

[MWHC93] B.S. Majewski, N.C. Wormald, G. Havas, and Z.J. Czech. Graphs, hypergraphs and hashing. In *Proc. 19th Int'l. Workshop on Graph-Theoretic Concepts in Computer Science (WG'93)*, Utrecht, The Netherlands, June 1993. The authors generalize the method presented in [CHM92] by mapping the input set into a hypergraph rather than a graph. This modification allows a reduction in the size of the program, while maintaining all other features of the method. Also, the hash function generation time is reduced.

[MZ86] N. Megiddo and E. Zemel. An $O(n \log n)$ randomizing algorithm for the weighted Euclidean 1-center problem. *Journal of Algorithms*, 7(3):358–368, Sep 1986. A set of points $p_i = (x_i, y_i)$ and their weights w_i, $1 \leq i \leq n$ are given. It is required to find a point p that minimizes the maximum first moment of the weights of the p_i s, i.e., the p that minimizes $H(p) = MAX_{1 \leq i \leq n} w_i \, d(p, p_i)$ where $d(p, p_i)$ is the magnitude of the distance between p and p_i. A randomized algorithm that does this with a small probability of error is presented.

[Nat92] B. K. Natarajan. Probably approximate learning over classes of distributions. *SIAM Journal on Computing*, 21(3):438–449, June 1992. Natarajan generalizes the model of probably approximate learning proposed by Valiant [Val84b].

[Nis90] N. Nisan. Pseudorandom generators for space-bounded computations. In *Proc. 22nd Ann. ACM Symp. on Theory of Computing*, pages 204–212, Baltimore, MD, may 1990. Pseudorandom generators are constructed that convert $O(S \log R)$ truly random bits to R bits that appear random to any algorithm that runs in *SPACE(S)*. In particular, any randomized polynomial time algorithm that runs in space S can be simulated using only $O(S \log n)$ random bits. Applications are given for "deterministic amplification," the problem of reducing the probability of error of randomized algorithms.

[Nis93] N. Nisan. On read-once vs. multiple access to randomness in logspace. *Theoretical Computer Science*, 107:135–144, 1993. This paper shows that every language accepted with bounded two-sided error by a read-once randomized logspace machine can be accepted with zero error by a randomized logspace machine with multiple access to the random bits. Also, the class of languages accepted with two-sided error by a randomized logspace machine with multiple access to the random bits is shown to be the class of languages that are in logspace relative to almost every oracle.

[NN90] J. Naor and M. Naor. Small-bias probability spaces: Efficient constructions and applications. In *Proc. 22nd Ann. ACM Symp. on Theory of Computing*, pages 213–223, Baltimore, MD, May 1990. This paper shows an efficient construction of a small probability space on n binary random variables such that for every subset, its parity is either zero or one with "almost" equal probability. Applications are shown in problems such as the derandomization of algorithms and reducing the number of random bits required by certain randomized algorithms.

[NS93] M. Naor and L. Stockmeyer. What can be computed locally? In *Proc. 25th Ann. ACM Symp. on Theory of Computing*, pages 184–193, San Diego, CA, May 1993. In the context of a distributed network, Naor and Stockmeyer investigate *Locally Checkable Labeling* (*LCL*) problems, where the legality of a labeling (e.g., coloring) can be checked locally; i.e., within time (or distance) independent of the size of the network. Among their results they show that randomization cannot make an LCL problem local; i.e., if a problem has a local randomized algorithm then it has a local deterministic algorithm.

[NY90] M. Naor and M. Yung. Public-key cryptosystems provably secure against chosen cypher-text attack. In *Proc. 22nd Ann. ACM Symp. on Theory of Computing*, pages 427–437, 1990. The authors show how to construct a public-key cryptosystem secure against *chosen ciphertest attacks*, given a publi-key cryptosystem secure against passive eavesdropping and a noninteractive zero-knowledge proof system in the shared string model.

[NZ93] N. Nisan and D. Zuckerman. More deterministic simulation in logspace. In *Proc. 25th Ann. ACM Symp. on Theory of Computing*, pages 235–244, San Diego, CA, May 1993. It is shown that any randomized *space(S)* algorithm that uses only *poly(S)* random bits can be simulated deterministically in *space(S)*, for $S(n) \geq \log n$.

[Ore87] Y. Oren. On the cunning power of cheating verifiers: Some observations about zero knowledge proofs. In *Proc. 28th Ann. IEEE Symp. on Foundations of Computer Science*, pages 462–471, 1987. Oren differentiates between *auxiliary-input* zero-knowledge and *blackbox-simulation* zero-knowledge. He shows that all known zero-knowledge proofs are in the latter category. In addition, it is proved that *blackbox-simulation* zero-knowledge implies *auxiliary-input* knowledge, and that the latter corresponds to the original definition given in [GMR89].

[Pac87] J. Pachl. A lower bound for probabilistic distributed algorithms. *Journal of Algorithms*, 8(1):53–65, 1987. The minimum number of messages required to find the extremal value of node ids in an asynchronous network deterministically is $\Theta(n \log n)$. This paper shows that this bound holds even for probabilistic algorithms.

[Paz71] A. Paz. *Introduction to Probabilistic Automata.* Academic Press, 1971. Paz develops a theory of equivalence among probabilistic automata.

[Pel90] M. Pellegrini. Stabbing and ray shooting in 3 dimensional space. In *Proc. Sixth Ann. ACM Symp. on Computational Geometry*, pages 177–186, Berkeley, CA, June 1990. The author presents a number of results about line stabbing and ray shooting including the following two: (1) One can determine the the first triangles hit by m rays in a set of n disjoint triangles using a randomized algorithm whose expected running time is $O(m^{5/6-\delta}n^{5/6+5\delta}\log^2 n + m\log^2 n + n\log n\log m)$; and (2) One can determine the first box hit by m rays in a set of disjoint axis-oriented boxes using a randomized algorithm whose expected running time is $O(m^{3/4-\delta}n^{3/4+3\delta}\log^4 n + m\log^4 n + n\log n\log m)$. Here δ is any constant greater than zero.

[Pel92] M. Pellegrini. Incidence and nearest neighbor problems for lines in 3-space. In *Proc. Eighth Ann. ACM Symp. on Computational Geometry*, pages 130–137, Berlin, Germany, June 1992. Given a set of n lines in 3-space, this paper demonstrates a randomized algorithm that finds the shortest vertical segment between any pair of lines in randomized expected time $O(n^{8/5+\epsilon})$ for every $\epsilon > 0$.

[Pel93] M. Pellegrini. On line missing polyhedral sets in 3-space (extended abstract). In *Proc. Ninth Ann. ACM Symp. on Computational Geometry*, pages 19–28, San Diego, CA, May 1993. Pellegrini gives an $O(n^{1.5+\epsilon})$ randomized expected time algorithm that tests the *separation* property: does there exist a direction v along which a set of n red lines can be translated away from a set of n blue lines without collisions?

[Per85] K. Perry. Randomized Byzantine agreement. *IEEE Trans. on Software Engineering*, SE-11(6):539–546, June 1985. Perry presents randomized algorithms for Byzantine agreement that, like the algorithm of Rabin [Rab83], terminate in an expected number of rounds which is a small constant independent of n and t. As usual, n is the total number of processes and t is the number of faulty processes. However, Perry's algorithm can tolerate a greater number of faulty processes. He requires only that $n \geq 6t + 1$ in the asynchronous case and $n \geq 3t + 1$ in the synchronous case.

[Pet82] G. L. Peterson. An $O(n\log n)$ unidirectional algorithm for the circular extrema problem. *ACM Trans. on Programming Languages and Systems*, 4(4):758–762, October 1982. Peterson presents a deterministic distributed algorithm for finding the largest of a set of n uniquely numbered processes in a ring. The algorithm requires $O(n\log n)$ messages in the worst case, and is unidirectional. The number of processes is not initially known.

[Pit89] L. Pitt. Probabilistic inductive inference. *Journal of the ACM*, 36(2):383–433, 1989. Inductive inference machines construct total recursive functions $\phi(x)$ given examples of the input and output of ϕ. Probabilistic inductive inference machines are permitted coin tosses while constructing ϕ, and are only required to construct ϕ with probability p, $0 < p < 1$. This paper shows a discrete hierarchy of inferability parameterized by p, for $p \leq 1/2$. Any machine that can be constructed by probabilistic inference with $p > 1/2$ can also be constructed deterministically.

[Pra75] V. R. Pratt. Every prime has a succinct certificate. *SIAM Journal on Computing*, 4(3):214–220, 1975. This paper proves, using the Lucas-Lehmer heuristic for testing primeness, that just like composite numbers, the primeness of a prime number n can be demonstrated by an $O(\log n)$ long proof.

[PS83] R. Paturi and J. Simon. Lower bounds on the time of probabilistic on-line simulations. In *Proc. 24th Ann. IEEE Symp. on Foundations of Computer Science*, pages 343–350, 1983. They show that coin tossing cannot compensate for inadequate memory access.

[PSL80] M. Pease, R. Shostak, and L. Lamport. Reaching agreement in the presence of faults. *Journal of the ACM*, 27(2):228–234, 1980. This paper is similar to their 1982 publication [LSP82], but contains a rigorous proof of the impossibility of Byzantine agreement for the case $n = 3$, $t = 1$. As usual, n is the total number of processes and t is the number of faulty processes.

[PU90] D. Peleg and E. Upfal. A time-randomness tradeoffs for oblivious routing. *SIAM Journal on Computing*, 19:256–266, 1990. This is a companion paper of [KPU88].

[Pug90] W. Pugh. Skip lists: A probabilistic alternative to balanced trees. *Communications of the ACM*, 33(6):668–676, June 1990. This paper presents skip lists, a list in which a node may have a pointer to a node some number of places ahead of it on the list. Such pointers, called "forward pointers", therefore "skip" over intermediate nodes. A node with k forward pointers is said to be a *level* k node. Skip lists are probabilistic in that the level of a node is chosen randomly with the property that a node's ith forward pointer points to the next node of level i or higher. It is shown that skips lists can efficiently implement abstract data types such as dictionaries and ordered lists in that the expected time to search for an item is $O(\log n)$.

[PZ86] A. Pnueli and L. Zuck. Verification of multiprocess probabilistic protocols. *Distributed Computing*, 1:53–72, 1986. They present a temporal logic for proving liveness properties of probabilistic concurrent programs based on the notion of "extreme fairness".

[Rab63] M. O. Rabin. Probabilistic automata. *Information and Control*, 6:230–245, 1963. This is a seminal paper on the theory of probabilistic automata. Rabin defined the notion of a language being accepted by a probabilistic automaton relative to a cutpoint lambda. One of his key results was to show that there exist finite state probabilistic automata that define non-regular languages.

[Rab76] M. O. Rabin. Probabilistic algorithms. In J.F. Traub, editor, *Algorithms and Complexity: New Directions and Recent Results*, pages 21–39. Academic Press, 1976. This classic paper on probabilistic algorithms features algorithms for primality testing and nearest neighbors.

[Rab80a] M. O. Rabin. A probabilistic algorithm for testing primality. *Journal of Number Theory*, 12, 1980. Rabin's paper introduces another celebrated algorithm for fast, randomized primality testing. This paper is based on a different number theoretic property than that used by Solovay and Strassen [SS77].

[Rab80b] M. O. Rabin. Probabilistic algorithms in finite fields. *SIAM Journal on Computing*, 9(2):273–280, May 1980. Rabin presents probabilistic algorithms for finding an irreducible polynomial of degree n over a finite field, the roots of a polynomial, and the irreducible factors of a polynomial.

[Rab83] M. O. Rabin. Randomized Byzantine Generals. In *Proc. 24th Ann. IEEE Symp. on Foundations of Computer Science*, pages 403–409, 1983. Rabin presents a randomized algorithm for asynchronous Byzantine agreement that terminates in a constant expected number of rounds. Cryptography is used to simulate a trusted dealer that distributes random coin tosses before the start of the algorithm. Rabin's algorithm works only if less than one-tenth of all processes are faulty.

[Rac82] C. Rackoff. Relativized questions involving probabilistic algorithms. *Journal of the ACM*, 29(1):261–266, January 1982. Rackoff attempts to prove $R \neq P$ by assuming $P \neq NP$ and relativization (i.e., for a class of languages C, C^A is the same as C except that one can answer questions concerning membership in A in constant time). Interestingly, he proves that for some oracle A, $P^A \neq NP^A$ and $R^A \neq P^A$, and at the same time, for some other oracle B, $P^B \neq NP^B$ and $R^B \neq P^B$. An earlier version of this paper appeared in *Proc. 10th Ann. ACM Symp. on Theory of Computing*, 1978, pp. 338–342.

[Rag88] P. Raghavan. Probabilistic construction of deterministic algorithms: Approximating packing integer problems. *Journal of Computer and System Sciences*, 37:130–143, 1988. Based on the derandomization technique of conditional probabilities, Raghavan develops a methodology for converting the probabilistic existence proof of a near-optimum integer solution to an integer program into a deterministic approximation algorithm.

[Rag90] P. Raghavan. Lecture notes on randomized algorithms. Research Report RC 15340 (#68237), IBM T.J. Watson Research Center, January 1990. This Research Report consists of lecture notes from a course taught by the author. These notes give a thorough introduction to many randomized algorithms in computational geometry, graph theory, VLSI, and networks. The basic mathematical background essential for understanding these algorithms is presented in detail.

[Raj91a] S. Rajasekaran. $k-k$ routing, $k-k$ sorting, and cut through routing on the mesh. Technical Report MS-CIS-91-93, Dept. of Computer and Information Sciences, Univ. of Pennsylvania, Philadelphia, PA, 1991. This paper presents randomized algorithms for $k-k$ routing, $k-k$ sorting, and cut through routing on mesh connected computers. The time bounds of these algorithms improve upon those of the best known algorithms prior to this paper.

[Raj91b] S. Rajasekaran. Randomized algorithms for packet routing on the mesh. Technical Report MS-CIS-91-92, Dept. of Computer and Information Sciences, Univ. of Pennsylvania, Philadelphia, PA, 1991. Efficient randomized algorithms for sore and forward, multipacket, and cut through routing of packets on a mesh connected computer are surveyed. The expected running times and queueing complexity of these algorithms are analyzed.

[Ram93] H. Ramesh. On traversing layered graphs on-line. In *Proc. Fourth Ann. ACM-SIAM Symp. on Discrete Algorithms*, pages 412–421, Austin, TX, January

1993. A *layered graph* is a connected weighted graph whose vertices are partitioned into sets (i.e., layers) L_0, L_1, $L_2, \ldots$, and all edges connect vetices in consecutive layers. Ramesh presents a randomized on-line algorithm for traversing width-w layered graphs with a competitive ratio of $O(w^{15})$. His algorithm represents the first polynomially competitive randomized algorithm for layered graph traversal.

[Rei80] J. H. Reif. Logics for probabilistic programs. In *Proc. 12th Ann. ACM Symp. on Theory of Computing*, 1980. Reif presents yet another attempt at a formal logic, PROB-DL, for probabilistic programs.

[Rei81] R. Reischuk. A fast probabilistic parallel sorting algorithm. In *Proc. 22nd Ann. IEEE Symp. on Foundations of Computer Science*, pages 212–219, 1981. Reischuk considers the problems of selecting k smallest elements out of a set of n keys, and sorting the n elements using n processors in parallel. He shows that the former can be done in *constant* time with probability $1 - 2^{-cn^{\frac{1}{8}}}$ and the later in $O(\log n)$ time. This achieves the information theoretic lower-bound in terms of processor-time product as well as the optimal speed-up attainable using n processors.

[Rei85a] J. H. Reif. Optimal parallel algorithms for integer sorting and graph connectivity. In *Proc. 26th Ann. IEEE Symp. on Foundations of Computer Science*, 1985. This paper contains some results on the use of randomization in parallel algorithms.

[Rei85b] R. Reischuk. Probabilistic parallel algorithms for sorting and selection. *SIAM Journal on Computing*, 14(2):396–409, May 1985. This paper considers the problems of selecting the k smallest elements out of a set of n keys, and sorting the n keys using n processors in parallel. Reischuk showed that the former can be done in constant time with probability $1 - 2^{-cn^{\frac{1}{8}}}$ and the later in $O(\log n)$ time. Both algorithms meet the corresponding information theoretic lower bounds in terms of processor-time product as well as the optimal speed-up attainable using n processors. An earlier version appeared as "A Fast Probabilistic Parallel Sorting Algorithm" in *Proc. 22nd Ann. IEEE Symp. on Foundations of Computer Science*, 1981, pp. 212–219.

[RP91] M. V. Ramakrishna and G. A. Portice. Perfect hashing functions for hardware applications. In *Proc. Seventh Int'l. Conf. on Data Engineering*, April 1991. A hardware scheme for constructing an associative memory using a perfect hash function is described. A simple trail and error scheme is used to find a perfect hash function.

[RR89] S. Rajasekaran and J. H. Reif. Optimal and sublogarithmic time randomized parallel sorting algorithm. *SIAM Journal on Computing*, 18(3):594–607, June 1989. This paper presents an optimal, randomized, parallel algorithm for sorting n numbers in the range $[1 \ldots n]$ on a parallel random access machine that allows both concurrent reads and concurrent writes of a global memory.

[RS82] J. H. Reif and P. G. Spirakis. Real time resource allocation in distributed systems. In *Proc. First Ann. ACM Symp. on Principles of Distributed Computing*, pages 84–94, 1982. This paper considers a resource allocation problem in distributed systems and provides real-time solutions in the form of two probabilistic algorithms.

[RS84] J. H. Reif and P. G. Spirakis. Real time synchronization of interprocess communication. *ACM Trans. on Programming Languages and Systems*, 6:215–238, 1984. They present probabilistic distributed algorithms for the guard-scheduling problem (Section 3.2) that guarantee real-time response. A preliminary version of this paper appeared as "Distributed Algorithms for Synchronizing Interprocess Communication in Real Time," in *Proc. 13th Ann. ACM Symp. on Theory of Computing*, 1981.

[RS89] J. H. Reif and S. Sen. Polling: A new random sampling technique for computational geometry. In *Proc. 21st Ann. ACM Symp. on Theory of Computing*, pages 394–404, 1989. A randomized sampling technique called polling is introduced. For the first time, this technique allows the calculation of 'high likelihood bounds' rather than simply expected running time, in computational geometric randomized algorithms. The technique is illustrated using an algorithm for the intersection of half-spaces in three dimensions.

[RS92] J. H. Reif and S. Sen. Optimal parallel randomized algorithms for three-dimensional convex hulls and related problems. *SIAM Journal on Computing*, 21(3):466–485, June 1992. An optimal parallel randomized algorithm for computing the intersection of half-spaces in 3-D is given. The algorithm provides efficient solution techniques for convex hulls in 3-D and Vornoi diagrams of point sites on a plane. An earlier version of the paper appeared as "Polling:

a new random sampling technique for computational geometry" in *Proc. 21st Ann. ACM Symp. on Theory of Computing*, 1989, pp. 394–404.

[RSA78] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public key cryptosystems. *Communications of the ACM*, 21(2):120, February 1978. The basics of trap-door functions and the famous RSA public key cryptosystem are presented in this paper.

[Rub81] R. Y. Rubinstein. *Simulation and the Monte Carlo Method.* John Wiley & Sons, 1981. This work is an in-depth look at the use of random sampling (the Monte Carlo method) in the context of simulation and numerical integration.

[RV89] M. Rabin and Vazirani V. Maximum matchings in general graphs through randomization. *Journal of Algorithms*, 10:557–567, 1989. This paper presents a conceptually simple algorithm for maximal matching in a graph of n nodes with complexity $O(M(n)n \log \log n)$, where $M(n)$ is the number of operations needed to multiply two $n \times n$ matrices.

[RW89] R. Raz and A. Wigderson. Probabilistic communication complexity of boolean relations. In *Proc. 30th Ann. IEEE Symp. on Foundations of Computer Science*, pages 562–567, 1989. Exponential gaps are demonstrated between deterministic and probabilistic complexity, and between the probabilistic complexity of monotone and non-monotone relations.

[Sal69] A. Salomaa. *Theory of Automata.* Pergamon Press, 1969. Chapter 2 of this book discusses probabilistic automata and develops a general theory of stochastic languages.

[Sch78] J. Schwartz. Distributed synchronization of communicating sequential processes. Technical report, DAI Research Report 56, University of Edinburgh, 1978. Schwartz presents a distributed algorithm for CSP output guards based on priority ordering of processes. A probabilistic algorithm for output guards is described in Section 3.2.

[Sch79] J. T. Schwartz. Probabilistic algorithms for verification of polynomial identities. In *ISSAC '79: Proc. Int'l. Symp. on Symbolic and Algebraic Computation,* Lecture Notes in Computer Science, Vol. 72. Springer-Verlag, 1979. This paper, which also appeared in *Journal of the ACM*, 1980, pp. 701–717, presents probabilistic methods for testing polynomial identities and properties of systems of polynomials.

[Sch82] F. B. Schneider. Synchronization in distributed programs. *ACM Trans. on Programming Languages and Systems*, 4(2):1982, April 1982. Schneider presents a timestamp-based distributed algorithm for CSP output guards. A probabilistic algorithm for output guards is described in Section 3.2.

[Sch84] M. R. Schroeder. *Number Theory in Science and Communication with Applications in Cryptography, Physics, Biology, Digital Information and Computing*. Springer-Verlag, 1984. Schroeder presents intuitive discussions on prime numbers, their distribution, fractions, congruences, etc. Several applications of number theory in such diverse fields as cryptography and Fraunhofer diffraction are discussed. A good source of basic number theory results for algorithm designers.

[Sch88] A. Schonhage. Probabilistic computation of integer polynomial GCDs. *Journal of Algorithms*, 9(3):365–371, September 1988. The GCD of two univariate integer polynomials of degree $\leq$ n, with their l^1 norms bounded by 2^n, is shown to be reducible to GCD computation for long integers. A probabilistic approach yields an expected complexity of $O(n(n+h)^{1+o(1)})$ bit operations.

[Sch91] O. Schwarzkopf. Dynamic maintenance of geometric structures made easy. In *Proc. 32nd Ann. IEEE Symp. on Foundations of Computer Science*, pages 180–196, 1991. Schwarzkopf presents a randomized algorithm for maintaining Convex Hulls with m points that runs in expected time $O(\log m)$ per update for dimensions 2 and 3, $O(m \log m)$ for dimensions 4 and 5, and $O(m^{\lfloor d/2 \rfloor - 1})$ for dimensions greater than 5.

[Sei90] R. Seidel. Linear programming and convex hulls made easy. In *Proc. Sixth Ann. ACM Symp. on Computational Geometry*, pages 211–215, Berkeley, CA, June 1990. Seidel presents two simple randomized algorithms. One solves linear programs involving m constraints in d variables in expected time $O(m)$. The other constructs convex hulls of n points in $\Re^d$, $d > 3$ in expected time $O(n^{\lfloor d/2 \rfloor})$. In both bounds, d is considered to be a constant.

[Sei91] R. Seidel. A simple and fast incremental randomized algorithm for computing trapezoidal decompositions and for triangulating polygons. *Computational*

Geometry: Theory and Applications, 1:51–64, 1991. Seidel's randomized algorithm runs in $O(n \log^* n)$ expected time and is simpler than the deterministic $O(n)$ algorithm due to B. Chazelle.

[Sei92] R. Seidel. On the all-pairs-shortest-path problem. In *Proc. 24th Ann. ACM Symp. on Theory of Computing*, pages 745–749, Victoria, B.C., Canada, May 1992. Given an undirected, unweighted n-vertex graph, a simple randomized algorithm is presented that finds a shortest path between each pair of vertices in expected $O(M(n) \log n)$ time, where $M(n)$ is the time necessary to multiply two $n \times n$ matrices of small integers.

[Sha92a] J. Shallit. Randomized algorithms in "primitive cultures". *SIGACT News*, 23(4):77–80, 1992. Shallit, in a slightly tongue-in-cheek manner, traces back some of the concepts of randomized algorithms to the native American society of the Naskapi and the central African society of the Azande. Roots in the works of Pierre Laplace and Lord Kelvin are also pointed out.

[Sha92b] A. Shamir. $IP = PSPACE$. *Journal of the ACM*, 39(4), 1992. This paper shows that the set of problems for which interactive protocols exist is precisely the set of problems which are solvable within polynomial space on a Turing machine.

[Sho93] V. Shoup. Fast construction of irreducible polynomials over finite fields. In *Proc. Fourth Ann. ACM-SIAM Symp. on Discrete Algorithms*, pages 484–492, Austin, TX, January 1993. A randomized algorithm is presented that constructs an irreducible polynomial of given degree n over a finite field F_q. It uses an expected number of $O^{\sim}(n^2 + n \log q)$ operations in F_q, where the "soft-O" $O^{\sim}$ indicates an implicit factor of $(\log n)^{O(1)}$.

[Sie89] A. Siegel. On universal classes of fast high performance hash functions, their time-space tradeoff, and their applications. In *Proc. 30th Ann. IEEE Symp. on Foundations of Computer Science*, pages 20–25, Oct 1989. An algorithm for constructing $\log n$-wise independent hash functions that can be evaluated in constant time is presented.

[Sip88] M. Sipser. Expanders, randomness, or time versus space. *Journal of Computer and System Sciences*, 36, 1988. Contains a discussion on efficiently reducing the probability of error in randomized algorithms. It also describes a relationship between pseudorandomness, time and space used by certain algorithms if certain types of expander graphs can be explicitly constructed.

[Smi83] J. Smith. Public key cryptography. *Byte*, pages 198–218, January 1983. This is a simple exposition of public key cryptography.

[Spe88] J. Spencer. Ten lectures on the probabilistic method. *SIAM Journal on Computing*, 1988. Spencer presents a method of converting probabilistic proofs of existence of certain combinatorial structures into deterministic algorithms that construct these structures.

[Spi82] P. G. Spirakis. *Probabilistic Algorithms, Algorithms with Random Inputs and Random Combinatorial Structures.* PhD thesis, (UMI Order Number DA 8216206) Harvard University, Cambridge, MA, 1982. This thesis puts forth a new model, 'Random Independence Systems', for the probabilistic analysis of deterministic algorithms with random inputs, i.e., algorithms for which the space of all inputs has a known probability distribution. It also presents two probabilistic algorithms with real time response for the problem of communication guard scheduling.

[Spr77] R. Sprugnoli. Perfect hash functions: A single probe retrieval method for static sets. *Communications of the ACM*, 20:841–850, 1977. This is the first discussion on perfect hashing; describes heuristics for constructing perfect hash functions.

[SS77] R. Solovay and V. Strassen. A fast Monte-Carlo test for primality. *SIAM Journal on Computing*, 6(1):84–85, March 1977. Another test for primality based on the abundance of witnesses to compositeness of n is presented. The test entails picking a random number a $(1 \leq a < n)$ and computing $\varepsilon = a^{(n-1)/2} (\pmod n)$, where $-1 \leq \varepsilon < n-2$. If the Jacobi symbol $\delta = (a/n)$ equals ε then n is prime, else, if either $gcd(a,n) > 1$ or $\delta \neq \varepsilon$, decide n to be composite. The second decision has less than $\frac{1}{2}$ probability of being wrong.

[SS78] R. Solovay and V. Strassen. Erratum: A fast Monte-Carlo test for primality. *SIAM Journal on Computing*, 7(1), Feb. 1978. A minor correction in the analysis presented in [SS77] is reported by the authors. The basic results of [SS77], however, still hold.

[SS90] J. P. Schmidt and A. Siegel. The spatial complexity of oblivious k-probe hash functions. *SIAM Journal on Computing*, 19(5):775–786, 1990. This paper gives, among other results, a lower bound for the average space required by program for oblivious k-probe hash function. A probabilistic construction of a family of oblivious k-probe hash function that nearly match this bound is also given.

[SSS93] J. P. Schmidt, A. Siegel, and A. Srinivasan. Chernoff-Hoeffding bounds for applications with limited independence. In *Proc. Fourth Ann. ACM-SIAM Symp. on Discrete Algorithms*, pages 331–340, Austin, TX, January 1993. Chernoff-Hoeffding bounds are frequently used in the design and analysis of randomized algorithms to bound the tail probabilities of the sums of bounded and independent random variables. The authors give a simple technique which gives slightly better bounds than these and which requires only *limited independence* among the random variables.

[Sto85] L. Stockmeyer. On approximation algorithms for #P. *SIAM Journal on Computing*, 14(4):849–861, 1985. The author explores the effect of approximation and randomization on the complexity of counting problems (Valiant's class #P which has problems such as counting the number of perfect matchings in a graph, the size of backtrack search trees, etc.).

[SV86] M. Santha and U. V. Vazirani. Generating quasi-random sequences from semi-random sources. *Journal of Computer and System Sciences*, 33(1):75–87, April 1986. The authors introduce the notion of semi-random sources where the next bit of the output is produced by an adversary by the flip of a coin of variable bias. The adversary can look at the previously output bits, and use them to set the bias in the coin. The bias, which helps model correlation among bits, is constrained to be between two limits.

[TN91] T. Tokuyama and J. Nakano. Geometric algorithms for a minimum cost assignment problem. In *Proc. Seventh Ann. ACM Symp. on Computational Geometry*, pages 262–271, North Conway, NH, June 1991. An efficient randomized algorithm is given for the minimum cost λ-assignment problem, which is equivalent to the minimum weight one-to-many matching problem in a complete bipartite graph $? = (A, B)$. If A and B have n and k nodes respectively, then the algorithm requires $O(kn + k^{3.5}n^{0.5})$ expected time.

[TO92] S. Toda and M. Ogiwara. Counting classes are at least as hard as the polynomial-time hierarchy. *SIAM Journal on Computing*, 21(2):316–328, April 1992. Many counting classes are shown to be computationally as hard as the polynomial time hierarchy, under a notion of randomized reducibility, unless the polynomial-time hierarchy collapses.

[Tut47] W. T. Tutte. The factorization of linear graphs. *Journal of the London Mathematical Society*, 22:107–111, 1947. Let $G(V, E)$ be a given simple graph where $V = \{1, 2, \ldots n\}$. Associate a variable x_{ij} with each edge $e_{ij} \in E$ and define the $n \times n$ matrix $B = [b_{ij}]$ as follows. If there is no edge between vertex i and vertex j them $b_{ij} = 0$. Otherwise, $b_{ij} = x_{ij}$ if $i > j$ and $b_{ij} = -x_{ij}$ if $i < j$. This paper proves that G has a perfect matching if and only if $\det(B) \neq 0$.

[TW87] M. Tompa and H. Woll. Random self-reducibility and zero-knowledge interactive proofs of possession of information. In *Proc. 28th Ann. IEEE Symp. on Foundations of Computer Science*, pages 472–482, 1987. Tompa and Woll present a general theory, of which IP proofs for graph isomorphism, quadratic residuosity and knowledge of discrete logarithms are special cases.

[Tze89] W. G. Tzeng. The equivalence and learning of probabilistic automata. In *Proc. 30th Ann. IEEE Symp. on Foundations of Computer Science*, pages 268–273, 1989. The equivalence problem of probabilistic automata is solvable in time $O((n_1 + n_2)^4)$, where n_1 and n_2 are the number of states in the two automata. The problem of learning probabilistic automata by a system of queries in polynomial time is also presented.

[Upf89] E. Upfal. An $O(\log N)$ deterministic packet routing scheme. In *Proc. 21st Ann. ACM Symp. on Theory of Computing*, pages 241–250, 1989. This paper presents the first deterministic $O(\log N)$ permutation routing algorithm for a multibutterfly network. A multibutterfly network is a special instance of a delta network. Upfal also shows that P instances of the permutation problem can be routed in $O(\log N + P)$ steps using a pipelining approach.

[UY91] J. D. Ullman and M. Yannakakis. High-probability parallel transitive closure algorithms. *SIAM Journal on Computing*, 20(1):100–125, Feb 1991. Parallel transitive closure algorithms are presented for the case when the graph is sparse or only a single source information is desired. The algorithms presented can converted to the Las Vegas type.

[Val82] L. G. Valiant. A scheme for fast parallel communication. *SIAM Journal on Computing*, 11(2):350–361, May 1982. Valiant gives a distributed randomized algorithm for routing packets from unique sources to unique destinations in an n-dimensional binary cube in $O(\log N)$ time, where $N = 2^n$ is the number of nodes in the network, with high probability.

[Val84a] L. G. Valiant. Short monotone formulae for the majority function. *Journal of Algorithms*, 5:363–366, 1984. A probabilistic approximation of a deterministic boolean function can yield simple circuits having a small proportion of inputs that cause wrong outputs. Independent probabilistic approximations of the same function can be combined to reduce the probability of error. In this paper Valiant uses such a technique to obtain $O(n^{5.3})$ size monotone formulas that compute the majority function of n boolean variables.

[Val84b] L. G. Valiant. A theory of the learnable. *Communications of the ACM*, 27:1134–1142, 1984. Valiant introduces a formal framework for the probabilistic analysis of algorithms that learn sets defined on a predetermined universe.

[Val87] D. Valois. Algorithmes probabilistes: une anthologie. Master's thesis, Département d'informatique et de recherche opérationnelle, Université de Montréal, 1987. In French, this paper covers a number of probabilistic algorithms including matrix multiplication and inversion, manipulation of polynomials, set equality, Byzantine Generals, and cryptography.

[Vaz87] U. V. Vazirani. Efficiency considerations in using semi-random sources. In *Proc. 19th Ann. ACM Symp. on Theory of Computing*, pages 160–168, 1987. Efficient algorithms for using semi-random sources are presented.

[VB81] L. Valiant and G. Brebner. Universal schemes for parallel communication. In *Proc. 13th Ann. ACM Symp. on Theory of Computing*, pages 263–277, 1981. This paper extends Valiant's message routing algorithm [Val82] to asynchronous networks.

[vdS81] J. L. A. van de Snepscheut. Synchronous communication between asynchronous components. *Information Processing Letters*, 13(3):127–130, December 1981. Snepscheut presents a distributed algorithm for CSP output guards in which processes are related by a tree structure. A probabilistic algorithm for output guards is described in Section 3.2.

[VF90] J. S. Vitter and P. Flajolet. Average-case analysis of algorithms and data structures. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science,* Volume A: Algorithms and Complexity, chapter 9, pages 432–524. Elsevier and The MIT Press (co-publishers), 1990. Vitter and Flajolet present analytic methods for average-case analysis of algorithms, with special emphasis on the main algorithms and data structures used for processing nonnumerical data. Problems considered include sorting, searching, pattern matching, register allocation, tree compaction, retrieval of multidimensional data, and efficient access to large files stored on secondary memory. The main mathematical tools used include generating functions (for recursively defined structures), statistics of inversion tables (for sorting algorithms), and valuations on combinatorial structures (for trees and structures with tree-like recursive decomposition, such as plane trees, multidimensional search trees, quicksort, and algorithms for register allocation and tree compaction).

[Vis84] U. Vishkin. Randomized speed-ups in parallel computation. In *Proc. 16th Ann. ACM Symp. on Theory of Computing*, pages 230–239, 1984. Vishkin considers the problem of computing the position of each element of a linked list, given the length n of the list. He presents a probabilistic algorithm for this problem running time $O(n/p + \log n \log {}^*n)$ using p processors.

[Vis90] S. Vishwanathan. Randomized online graph coloring. In *Proc. 31st Ann. IEEE Symp. on Foundations of Computer Science*, pages 464–469, 1990. It shown that randomization helps in coloring a graph in an online manner and the randomized online algorithm is quite competitive with the best-known, deterministic, off-line algorithm.

[VV85] U. V. Vazirani and V. V. Vazirani. Random polynomial time is equal to semi-random polynomial time. In *Proc. 26th Ann. IEEE Symp. on Foundations of Computer Science*, pages 417–428, 1985. This paper analyzes of the behavior of randomized algorithms where perfectly random sources are substituted with sources which have small bias and dependence. It shows that if a problem can be solved by a polynomial-time Monte Carlo algorithm which has access to a true source of randomness, the the same problem can be solved using an arbitrarily weak semi-random source.

[VV89] U. V. Vazirani and V. V. Vazirani. The two-processor scheduling problem is in random NC. *SIAM Journal on Computing*, 18(6):1140–1148, 1989. An efficient, randomized, parallel solution to the well-studied two-processor scheduling problem is presented.

[vzG89] J. von zur Gathen. Testing permutation polynomials. In *Proc. 30th Ann. IEEE Symp. on Foundations of Computer Science*, pages 88–98, Research Triangle Park, NC, October 1989. IEEE Computer Society Press. The author presents a randomized algorithm for testing whether a given polynomial over a finite field with q elements is a permutation polynomial in expected $O(q)$ time.

[vzG91] J. von zur Gathen. Tests for permutation polynomials. *SIAM Journal on Computing*, 20(3):591–602, June 1991. An element of a finite field $F_q[x]$ is called a *permutation polynomial* if the mapping $F_q \rightarrow F_q$ induced by it is bijective. A probabilistic algorithm for testing this property is given.

[vzGS92] J. von zur Gathen and V. Shoup. Computing Frobenius maps and factoring polynomials. In *Proc. 24th Ann. ACM Symp. on Theory of Computing*, pages 97–105, Victoria, B.C., Canada, May 1992. A probabilistic algorithm for factoring univariate polynomials over finite fields is presented whose asymptotic running time improves upon previous results.

[Wei78] B. W. Weide. *Statistical Methods in Algorithmic Design and Analysis.* PhD thesis, Computer Science Department, Carnegie-Mellon University, Pittsburgh, PA, Report CMU-CS-78-142, 1978. An early survey of probabilistic algorithms and analysis.

[Wel83] D. J. A. Welsh. Randomized algorithms. *Discrete Appl. Math.*, 5:133–146, 1983. This is a well-written introduction to randomized algorithms. Welsh discusses probabilistic algorithms for checking polynomial identities, primality, matrix and polynomial multiplication, and deciding whether a graph has a perfect matching. The work also contains a nice discussion on random polynomial time, random log-space, and the probabilistic hierarchy.

[WVZT90] K.-Y. Whang, B. T. Vander-Zanden, and H. M. Taylor. A linear-time probabilistic counting algorithm for database applications. *ACM Trans. on Database Systems*, 15(2):208–229, Sept 1990. A probabilistic technique called linear counting, based on hashing, for counting the number of unique values in the presence of duplicates is presented in this paper.

[Wyl79] J. C. Wyllie. The complexity of parallel computation. Technical Report TR 79-387, Department of Computer Science, Cornell University, Ithaca, NY, 1979. Wyllie conjectures that there is no optimal speed-up parallel algorithm for $n/\log n$ processors for the problem: Given a linked list of length n, compute the distance of each element of the linked list from the end of the list. However, Vishkin showed that such optimal speed-up *can* be obtained via randomization (see Section 4).

[Yao79] A. C. Yao. The complexity of pattern matching for a random string. *SIAM Journal on Computing*, 8(3):368–387, August 1979. Yao proves that the minimum average number of characters which need be examined in a random string of length n for locating patterns of length m, in an alphabet with q symbols, is $\theta(\lceil \log_q(\frac{n-m}{\ln m}+2)\rceil)$ if $m \leq n \leq 2m$ and $\theta(\frac{\lceil \log_q m\rceil}{m}n)$ if $n > 2m$. This confirms Knuth, Morris, and Pratt's conjecture in [KMP77].

[Yao83] A. C. Yao. Lower bounds by probabilistic arguments (extended abstract). In *Proc. 24th Ann. IEEE Symp. on Foundations of Computer Science*, pages 420–428, 1983. Though not a paper on probabilistic algorithms, this paper illustrates the power of probabilistic arguments by proving lower bounds for three important problems.

[Yao91] A. C. Yao. Lower bounds to randomized algorithms for graph properties. *Journal of Computer and System Sciences*, 42:267–287, 1991. Yao shows that $\Omega(n(\log n)^{\frac{1}{12}})$ edges must be examined by any randomized algorithm (as opposed to $\Omega(n^2)$ by any deterministic algorithm) for determining any non-trivial monotone graph property. An earlier version of this paper appeared in *Proc. 28th Ann. IEEE Symp. on Foundations of Computer Science*, 1987.

[YL91] M. Yannakakis and D. Lee. Testing finite state machines (extended abstract). In *Proc. 23rd Ann. ACM Symp. on Theory of Computing*, pages 476–485, New Orleans, LA, May 1991. A *checking sequence* for a finite state machine A having n states is an input sequence that distinguishes A from all other machines with n states. In addition to some other results on testing finite state machines, the authors present a simple randomized polynomial time algorithm that constructs with high probability a checking sequence of length

$O(pn^4 \log n)$, where p is the size of the input alphabet. (There is a lower bound of pn^3 on the length of checking sequences; previous algorithms are exponential in general or work only for special cases.).

[Zac88] S. Zachos. Probabilistic quantifiers and games. *Journal of Computer and System Sciences*, 36:433–451, 1988. This paper attempts to give a uniform picture of the various polynomial time complexity classes.

[Zip79] R. Zippel. Probabilistic algorithms for sparse polynomials. In *ISSAC '79: Proc. Int'l. Symp. on Symbolic and Algebraic Computation,* Lecture Notes in Computer Science, Vol. 72. Springer-Verlag, 1979. Zippel discusses probabilistic methods for testing polynomial identities and properties of systems of polynomials.

[Zuc90] D. Zuckerman. General weak random sources. In *Proc. 31st Ann. IEEE Symp. on Foundations of Computer Science*, pages 534–543, 1990. A pseudo-random generator that depends only on a *weak random source* is exhibited. By a weak random source it is meant that the source is asked only once for R random bits and the source outputs an R-bit string such that no string has a probability more than $2^{-\delta R}$ of being output, for some fixed $\delta > 0$. This paper shows how to simulate *RP* using a string from a δ-source in time $n^{O(\log n)}$, or in polynomial time under the Generalized Paley Graph Conjecture. See [Zuc91] for a correction to a result in this paper.

[Zuc91] D. Zuckerman. Simulating BPP using a general weak random source. In *Proc. 32nd Ann. IEEE Symp. on Foundations of Computer Science*, pages 79–89, 1991. Using the weak random source defined in [Zuc90], this paper shows how to simulate *BPP* and approximation algorithms in polynomial time using the output from a such a source.