Computational Complexity Since 1980

Russell Impagliazzo*
Department of Computer Science
University of California, San Diego
La Jolla, CA 92093-0114
russell@cs.ucsd.edu

No Institute Given

1 Introduction

The field of computational complexity is reaching what could be termed middle age, with over forty years having passed since the first papers defining the discipline. With this metaphor in mind, the early nineteeneighties represented the end of adolescence for the area, the time when it stopped wondering what it would be when it grew up. During the childhood period of the sixties, research centered on establishing the extent to which computational complexity, or the inherrent computational resources required to solve a problem, actually existed and was well-defined.

In the early seventies, Cook (with contributions from Edmonds, Karp and Levin) gave the area its central question, whether P equals NP. Much of this decade was spent exploring the ramifications of this question. However, as the decade progressed, it became increasingly clear that $P vs. NP$ was only the linking node of a nexus of more sophisticated questions about complexity. Researchers began to raise computational issues that went beyond the time complexity of well-defined decision problems by classical notions of algorithm.

Some of the new questions that had arisen included:

- Hardness of approximation: To what extent can NP-hardness results be circumvented? More precisely, which optimization problems can be efficiently solved approximately?
- Average-case complexity: Does NP-hardness mean that intractible instances of a problem actually arise? Or can we devise heuristics that solve typical instances? Which problems can be solved on "most" instances?
- Foundations of cryptography: Can computational complexity be used as a foundation for cryptography? What kind of computational hardness is needed for such a cryptography? ([DH], [RSA])
- Power of randomness: What is the power of randomized algorithms? Should randomized algorithms replace deterministic ones to capture the intuitive notion of efficient computation? ([Ber72],[Rab80], [SS79], [Sch80], [Gill]).

* Research supported by NSF Award CCF0515332, but views expressed here are not endorsed by the NSF.

– Circuit complexity: Which problems require many basic operations to compute in non-uniform models such as Boolean or arithmetic circuits? How does the circuit complexity of problems relate to their complexity in uniform models?
– Constructive combinatorics: Many mathematically interesting objects, such as from extremal graph theory, are proved to exist non-constructively, e.g., through the probabilistic method. When can these proofs be made constructive, exhibiting explicit and easily computable graphs or other structures with the desired properties?

This is a very incomplete list of the issues that faced complexity theory. My choice of the above topics is of course biased by my own research interests and by the later history of the area. For example, at the time, any list of important areas for complexity would include time-space tradeoffs and parallel computation, which I will have to omit out of my own concerns for time and space. However, it is fair to say that all of the topics listed were and remain central questions in computational complexity.

It is somewhat sad that we still do not know definitive answers to any of these questions. However, in the last twenty-five years, a number of significant and, at the time, highly counter-intuitive, connections have been made between them. The intervening period has made clear that, far from being six independent issues that will be addressed separately, the questions above are all interwoven to the point where it is impossible to draw clear boundaries between them. This has been established both by direct implications (such as, "If any problem in E requires exponential circuit size, then $P = BPP$") and by the transferance of issues and techniques that originated to address one question but ended up being a key insight into others.

In the following, we will attempt to trace the evolution of some key ideas in complexity theory, and in particular highlight the work linking the above list of questions. Our central thesis is that, while the work of the sixties showed that complexity theory is a well-defined field of mathematics, and that of the seventies showed how important this field is, the work since then has demonstrated the non-obvious point that complexity theory is best tackled as a single, united field, not splintered into specialized subareas. Of course, due to the clarity of hindsight, we may be biased in picking topics that support this thesis, but we feel that the existence of so many interrelated fundamental questions is prima facia evidence in itself.

2 Some key techniques in complexity

A common complaint about complexity theory used to be that complexity theorists took an *ad hoc* approach to their field, rather than developing a set of unifying and deep techniques. We will attempt to show that this complaint, if it were ever true, is obsolete. In particular, we will trace a few fundamental technical approaches that have evolved to become essential tools that span complexity.

1. Arithmetization Complexity studies mainly Boolean functions, with discrete zero-one valued inputs and outputs. In essence, arbitrary Boolean functions are just strings of bits, and hence have no fundamental mathematical properties. The technique of arithmetization is to embed or interpolate a Boolean function within a algebraic function, such as a polynomial. This can be used algorithmically (to perform operations on functions), or conceptually (to reason about the nature of functions computable with low complexity).
   While introduced as a tool for proving lower bounds on circuits, arithmetization has evolved into a fundamental approach to complexity, and is now essential in the study of interactive proofs, hardness of approximation, learning theory, cryptography, and derandomization.
2. Error-correcting codes As mentioned before, a Boolean function can be identified with a string of bits. The process of arithmetization became understood as computing a code on this string, to obtain a version of the function that codes the original function redundantly. Many of the applications of arithmetization are in fact consequences of the good error-correction properties of this coding. This made decoding algorithms of various kinds essential to complexity.
3. Randomness Complexity theorists frequently view the hardness of a problem as a contest between a solver (trying to solve instances of the problem) and an unknown adversary, trying to create intractible instances. Game theory suggests that the best strategies for such a game might have to be randomized. Perhaps this is one reason why randomness plays such a huge role in complexity. In any case, randomized computation has become the default model for reasoning in complexity, even when reasoning about deterministic algorithms or circuits.
4. Pseudo-randomness and computational indistinguishability The flip side of randomness is pseudo-randomness. As mentioned earlier, randomness often comes into the arguements even when we want to reason about deterministic computation. We then want to eliminate the randomness. When does a deterministic computation look "random enough" that it can be safely substituted for the randomness in a computation? More generally, when do two distributions look sufficiently "similar" that conclusions about one can be automatically tranferred to another? The core idea of computational indistinguishablity is that, when it is computationally infeasible to tell two distributions apart, then they will have the same properties as far as efficient computation is concerned. This simple idea originating in cryptography has percolated throughout complexity theory. Different contexts have different answers to what is "random enough", based on the type of computation that will be allowed to distinguish between the two.
5. Constructive extremal graph theory Randomness had a similar role in combinatorics. In particular, randomly constructed graphs and other structures often had desireable or extreme combinatorial properties. This raised the challenge of coming up with specific constructions of structures with similar properties. This can be thought of as particular cases of derandomization, which links it to the previous topic. In fact, there are connections both

ways: "derandomizing" the probabilistic constructions of extremal graphs gives new constructions of such graphs; constructions of such graphs give new ways to derandomize algorithms.

3 The challenges to complexity as of 1980

During the 1970's, it became clear (if it wasn't already) that the field had to go beyond deterministic, worst-case, time complexity, and beyond techniques borrowed from recursion theory. "Traditional" complexity was challenged by the following results and issues:

Circuit complexity Circuit complexity, the number of Boolean operations needed to compute functions, was a major impetus to much foundational work in complexity. Riordan and Shannon ([RS]) had introduced circuit complexity in 1942, and had shown, non-constructively, that most Boolean functions require exponential circuit size. However, there were no natural functions that were known to require large circuits to compute. The first challenge that lay before complexity theory was to find examples of such functions.

A related challenge is the circuit minimization problem. This is circuit complexity viewed algorithmically: given a Boolean function, design the circuit computing the function that uses the minimum number of gates. This seems harder than proving a lower bound for a specific function, since such an algorithm would in some sense characterize what makes a function hard. In his Turing Award lecture ([K86]), Karp reports that it was in attempting to program computers to solve circuit minimization that he became aware of the problem of exponential time. In the Soviet Bloc, Yablonski ([Yab]) was motivated by this problem to introduce the notion of *perebor*, usually translated as "brute-force search". (Unfortunately, he also falsely claimed to have proved that *perebor* was necessary for this problem.)

There was much work in circuit complexity before 1980; see [We87] for references. However, for natural Boolean functions, the hardness results were relatively minor, in the sense of proving only small polynomial bounds on circuit size, and only for restricted models. Even now, no one knows any super-linear bound for a function computable in strictly exponential ($2^{O(n)}$) time. (It is possible to simply define a Boolean function as "The lexicographically first Boolean function that requires exponential circuit size". This function will be in the exponential hierarchy, so functions hard for EH will require exponential size circuits. In his 1974 thesis, Stockmeyer gave a specific function, true sentences of weak monadic second-oder theory of the natural numbers with successor, whose circuit size is almost the maximum possible. However, the proof is by showing that this function is hard for EXPSPACE. To make the lower bound challenge precise, we can formalize a "natural" function as one that has reasonably low uniform complexity.)

Cryptography In [DH], Diffie and Hellman pointed towards basing cryptography on the inherrent computational intractibility of problems. Shortly afterwards, [RSA] gave a suggestion of a public-key cryptosystem based on the intractibility of factoring. This raised the question, how hard are factoring and other number-theoretic problems? Are they NP-complete? More importantly, is it relevant whether they are NP-complete? One of the main challenges of cryptography is that, while intractibility is now desireable, hardness is no longer simply the negation of easiness. In particular, worst-case hardness is not sufficient for cryptography; one needs some notion of *reliable hardness*. While [DH] discusses the difference between worst-case and average-case complexity, a formal notion was not fully described. The challenge presented to complexity-theory circa 1980 was to clarify what kinds of hardness constituted useful hardness for cryptography, and what kinds of evidence could be used to argue that specific problems had such hardness.

Randomized algorithms A phenomenon that arose during the 1970's was the use of random choices in designing efficient algorithms. In 1972, Berlekamp gave a probabilistic algorithm to factor polynomials ([Ber72]). Later, Solovay and Strassen [SS79] and Rabin [Rab80] gave such algorithms for the classical problem of primality. A third example was the Schwartz-Zippel polynomial-identity testing algorithm ([Sch80], [Zip79]). Another interesting randomized algorithm was given in [AKLLR], where it is shown that a sufficiently long random walk in an undirected graph visits all nodes within the connected component of the starting node. This shows reachability in undirected graphs can be computed in logarithmic space by a randomized algorithm.

These algorithms presented a challenge to the then recently adopted standard that (deterministic) polynomial-time captured the intuitive notion of tractible computation, the so-called time-bounded Church's Thesis. Should deterministic or randomized polynomial time be the default standard for efficient computation? This question was formalized by Gill ([Gill]), who defined the now standard complexity classes corresponding to probabilistic algorithms with different error conditions, $P \subseteq ZPP \subseteq RP \subseteq BPP \subseteq PP \subseteq PSPACE$. Of course, it could be that the above notions are identical, that $P = ZPP = RP = BPP$. Unlike for P vs NP, there was no consensus about this. On the one hand, in his 1984 NP-completeness column ([J84]), Johnson refers to the above containments and states "It is conjectured that all of these inclusions are proper." In contrast, Cook in his 1982 Turing Award lecture ([C83]) says, "It is tempting to conjecture yes [that $RP = P$] on the philisophical grounds that random coin tosses should not be of much use when the answer sought is a well-defined yes or no." (It should be noted that neither is willing to actually make a conjecture, only to discuss the possibiltiy of conjectures being made.)

If the two models are in fact different, then it still needs to be decided which is the right model of computation. If one is interested in computation possible within the physical world, this then becomes a question of whether random bits are physically obtainable. Of course, quantum mechanics suggests that the universe is inherrently probabilistic. (We can knowingly realize the genie that this consideration will eventually let out of the bottle.) However, this does not mean that fair random bits are physically obtainable, another question which will grow in importance.

Random-like graphs and structures Meanwhile, in combinatorics, randomness has long played a role in showing that objects exist non-constructively. Erdos's *probabilistic method* is perhaps the most important tool in the area. In particular, there are certain very desireable properties of graphs and other structures which hold almost certainly under some simple probability distribution, but where no specific, constructive example is known. For example, random graphs were known to be good expanders and super-concentrators. Deterministic constructions could come close to the quality of random graphs ([GG]) but didn't match them.

If we use the standard for "constructive" as polynomial-time computable (in the underlying size), these properties of random graphs can be thought of as examples of the power of randomized algorithms. A coin-tossing machine can easily construct graphs with these properties, but no deterministic machine was known to be able to. More subtly, question of whether randomized algorithms can be derandomized can be viewed as a special case of the construction of "quasi-random objects". Let $S = \{x_1, ..x_m\}$ be a (multi)-set of n bit strings. We call S an n, s *hitting set* if for any circuit C with n inputs and at most s gates, if $Prob_x[C(x) = 1] > 1/2$ then $\exists i, C[x_i] = 1$. In other words, a hitting set can produce witnesses of satisfiability for any circuit with ample numbers of such witnesses. Adleman ([Adl78]) proved that RP had small circuits by giving a construction via the probabilistic method of a small ($m = poly(n, s)$) hitting set. If we could deterministically produce such a set, then we could simulate any algorithm in RP by using each member of the hitting set (setting s equal to the time the randomized algorithm takes and n as the number of random bits it uses) as the random choices for the algorithm. We accept if any of the runs accepts. (More subtly, it was much later shown in [ACRT] that such a hitting set could also derandomize algorithms with two-sided error, BPP.)

A similar notion was introduced by [AKLLR]. A n node *universal traversal sequence* is a set of directions to take so that following them causes a walk in any n-node undirected graph to visit the entire connected component of the starting place. They showed that random sequences of polynomial length were universal. Constructing such a sequence would place undirected connectivity in L. Heintz and Schnorr *HS* introduced the notion of *perfect test set*, a set of input sequences to an arithmetic circuit one of which would disprove any invalid polynomial identity. They showed a probabilistic construction (for identities over any field, and whose size is independent of the field size.)

In fact, we can also view the problem of constructing a hard function for circuit complexity as another example of making the probabilistic method constructive. Riordan and Shannon's proof established that almost all functions require exponential circuit complexity. If we construct such a function in polynomial-time (in, say, its truth-table size, 2^n), this would produce a hard function in E.

To close the circle, observe that constructing a polynomial-size hitting set would also produce a hard function. Let $S(n, s) = \{x_1, ..x_m\}$ be a hitting set constructed in poly(s) time. Then let $k = \log m + 1 = O(\log s)$ and define a Boolean function $f(y)$ on k bit inputs by: $f(y) = 1$ if and only if y is not a prefix of any x_i. With the choice of k as above, $f(y) = 1$ with probability at least 1/2, so if C computed f with less than s gates, $C'(y_1..y_n) = C(y_1..y_k)$ would have to be 1 for some x_i in S, which contradicts the definition of f. f is computable in time $poly(s) = 2^{O(k)}$, so $f \in E$ and requires $s = 2^{\Omega(k)}$ circuit complexity. This simple proof does not seem to appear in print until the late nineties. However, the analagous result for aritmetic circuit complexity (a perfect hitting set construction implies an arithmetic circuit lower bound) is in [HS].

4 Meeting the challenges

We can see that even before 1980, there were a number of connections apparrent between these questions. As complexity went forward, it would discover more and deeper connections. In the following sections, we will attempt to highlight a few of the discoveries of complexity that illustrate both their intellectual periods and these connections.

5 Cryptography, the muse of modern complexity

As mentioned earlier, the growth of modern cryptography motivated a new type of complexity theory. In fact, many of the basic ideas and approaches of modern complexity arose in the cryptographic literature. Especially, the early 1980's were a golden age for cryptographic complexity.

Complexity theory and modern cryptography seemed a match made in heaven. Complexity theorists wanted to understand which computational problems were hard; cryptographers wanted to use hard problems to control access to information and other resources. At last, complexity theorists had a reason to root for problems being intractible! However, it soon became clear that cryptography required a new kind of complexity. Some of the issues complexity was forced to deal with were:

Reliable intractibility: Complexity theory had followed algorithm design in taking a conservative approach to definitions of tractibility. A problem being tractible meant that it was reliably solved by an algorithm that always ran quickly and always produced the correct result. This is a good definition of "easy computational problem". But the negation of a conservative definition of "easy" is a much too liberal to be a useful notion of "hard", especially when even occasional easy instances of a problem would compromise security completely. To handle this, complexity theory had to move beyond worst-case complexity to an understanding of distributional or average-case complexity. Once complexity moved to the average-case, it was necessary

Randomized computing: A deterministic algorithm is equally accessible to everyone, intended user and attacker alike. Therefore, cryptographic problems must be generated randomly. This made randomized computation the default model for cryptography. Randomized algorithms moved from being viewed as an exotic alternative to the standard model.

Going beyond completeness: Most of the computational problems used in cryptography fall within classes like $NP \cap Co - NP$ or UP that do not seem to have complete problems. Cryptosystems that were "based" on NP-complete problems were frequently broken. This had to do with the gap between worst-case and average-case complexity. However, even after Levin introduced average-case complete problems ([Lev86]), it was (and is) still unknown whether these can be useful in cryptography. Complexity theory had to have new standards for believable intractibility that were not based on completeness.

Adversaries and fault-tolerant reductions: While the notion of completeness was not terribly helpful, the notion of reduction was essential to the new foundations for cryptography. However, it needed to be dramatically altered. When talking about reductions between average-case problems, one needed to reason about oracles that only solved the problem being reduced to some fraction of the time (while believing that no such oracles actually are feasible). Since we don't know exactly what function this oracle performs the rest of the time, it seems the only safe approach is to view the oracle as being created by an "adversary", who is out to fool the reduction. Thus, what is needed is a fault-tolerant approach to reductions, where the small fraction of true answers can be used despite a large fraction of incorrect answers. This would link cryptography with other notions of fault-tolerant computation, such as error-correcting codes.

Computation within an evolving social context: In traditional algorithm design, and hence traditional complexity, the input arrived and then the problem had to be solved. In cryptography, there had to be communication between the parties that determined what the problem to be solved was. Cryptography was in essence social and interactive, in that there was almost always multiple, communicating parties performing related computations. In particular, this meant that an attacker could partially determine the instance of the problem whose solution would crack the system. Complexity theory had to go beyond reasoning about the difficulty of solving problems to understand the difficulty of breaking protocols, patterns of interleaving communication and computation.

In retrospect, it is astonishing how quickly a complexity-theoretic foundations for cryptography that addressed all of these issues arose. Within a decade of Diffie and Hellman's breakthrough paper, complexity-based cryptography had established strong, robust definitions of security for encryption and electronic signatures, and had given existence proofs that such secure cryptographic functions existed under reasonable assumptions. Moreover, complexity-theoretic cryptography unleashed a wave of creativitiy, that encompassed such avant garde notions as oblivious transer, zero-knowledge interactive proofs, and secure distributed "game" playing, aka, "mental poker". Complexity would never be the same.

We'll look at some landmark papers of 80's cryptography, that were not only important for their role in establishing modern cryptography, but introduced fundamental ideas and tools into general complexity theory.

5.1 Cryptographic pseudo-randomness

In 1982, Blum and Micali ([BM]) introduced the notion of cryptographic pseudorandomness. Shortly thereafter, Yao ([Yao82]) strengthened their results considerably and explored some of the ramifications of this concept. Together, these two papers presented a dramatic rethinking of information theory, probability, and the likely power of randomized algorithms in the face of complexity theory. (In addition, of course, they gave cryptography one of its most important tools.)

Blum and Micali's paper introduced what would become the gold standard for "hard" Boolean function, unpredictability. A function b is computationally unpredictable (given auxilliary information f) if, over choice of a random x, the probability that an adversary, given $f(x)$, can predict $b(x)$ is only negligibly more than a random coin toss. Intuitively, this means that b *looks like* a random coin to any feasible adversary, and this intution can frequently be made formal.

Blum and Micali give an example of such a function, assuming the difficulty of finding discrete logarithms. The bit they show is hard, is, given $g^x modp$, determine whether $x \bmod p-1 \leq (p-1)/2$. Let's call this function $b(x)$. The way they prove that $b(x)$ is unpredictable is also, in hindsight, prescient. In fact, a recent paper by Akavia, Goldwasser, and Safra ([AGS]) gives the modern insight into why this bit is hidden. There argument combines the random self-reducibility of the discrete logarithm and a *list-decoding algorithm* for a simple *error-correcting code*. The original proof of Blum and Micali is close in spirit, but also uses the ability to take square roots mod p, so the corresponding code would be more complex. First, we observe that, given $g, g^x modp$, and z_1, z_2, we can compute $(g^x)^{z_1} g^{z_2} = g^{xz_1+z_2 modp-1}$. This means that a predictor for $b(x)$ given g^x also gives us a predictor for $b(xz_1 + z_2 modp-1)$. Consider the exponentially long code $C(x)$ that maps x to the sequence $b(xz_1 + z_2)$ for each $z_1, z_2 \in Z_{p-1}$. Note that for $x - y$ relatively prime to $p-1$ and random z_1, z_2, $xz_1 + z_2$ and $yz_1 + z_2$ take on all possible pairs of values mod each odd prime factor of $p-1$. It follows that $C(x)$ and $C(y)$ will be almost uncorrelated, so the code has large distance, at least for almost all pairs.

A predicting algorithm $P(g^r)$, which guesses $b(r)$ given g^r with probability $1/2+\epsilon$ determines the flawed code word $\overline{C}$ where $\overline{C}_z = P(g^{xz})$ which has relative hamming distance $1/2-\epsilon$ from C. Is that enough information to recover the message x? Not completely, but it is enough to recover a polynomial number of possible x's, and we can then exponentiate each member of this list and compare to g^x to find x. The final step is to do this *list-decoding* algorithmically. However, since the code-word itself is exponentially long, we can only afford to look at the code in a small fraction of positions. This means that we need a *local list decoding* algorithm for this code, one that produces such a list of possible messages using a polynomial (or poly-log in the size of the code word) number of random access queries to bits of the codeword. [AGS] provide such an algorithm. The original proof of Blum and Micali would provide such an algorithm for a more complex code. Locally-decodeable error-correcting codes of various kinds will arise repeatedly in different guises, before finally being made explicit in the PCP constructions. Intuitively, they arise whenever we need to use an adversarial oracle only correlated with a function to compute a related function reliably. The oracle will correspond to a corrupted code-word, and the message will correspond to the related function.

Yao's sequel paper (citeYao82) is perhaps even more prescient. First, it shows that the unpredictibility criterion of Blum and Micali is equivalent to a *computational indistinguishability* criterion. Two computational objects (strings, functions, or the like) are computationally indistinguishable, informally, if no feasible algorithm, given a random one of the two objects, can determine which object it has been given significantly better than random guessing. Thus, it is a type of computational Turing test: if no feasible algorithm can tell two things apart, for purposes of effective computation, they are the same. This notion, implicit in Yao's paper, and made explicit in the equally revolutionary [GMR], is indespensible to modern complexity. The proof of equivalence also introduced the hybrid method, of showing two objects are indistinguishable by conceptualizing a chain of objects, each indistinguishable from the next, that slowly morph one object into another.

Second, this paper introduces the possibility of *general derandomization* based on a hard problem. This is the use of sufficiently hard problems to transform randomized algorithms into deterministic ones. What makes this counter-intuitive is that intractibility, the *non-existence* of algorithms, is being used to design algorithms! However, once that logical leap is made, derandomization results make sense. Yao proved that a Blum-Micali style pseudo-random generator transforms a relatively small number of random bits into a larger number of bits computationally indistinguishable from random bits. Thus, these pseudo-random bits can replace truly random bits in any algorithm, without changing the results, giving a randomized algorithm that uses far fewer random bits. The algorithm can then be made deterministic at reduced cost by exhaustive search over all possible input sequences of bits to the pseudo-random generator. There was one caveat: to derandomize algorithms in the worst-case, Yao needed the generator to be secure against small circuits, not just feasible algorithms. This was due to the fact that the randomized algorithm using pseudo-random bits might only be incorrect on a vanishingly small fraction of inputs; if there were no way to locate a fallacious input, this would not produce a distinguishing test. However, such inputs could be hard-wired into a circuit.

In fact, the set of possible outputs for a pseudo-random generator in Yao's sense is a *small discrepancy set*, in that, for any small circuit, the average value of the circuit on the set is close to its expected value on a random input. This implies that it is also a hitting set, and we saw that such a set implies a circuit lower bound. So it seems for a Yao-style derandomization, assuming hardness versus circuits is necessary. But this leaves open whether other types of derandomization could be performed without such a hardness assumption.

A third basic innovation in Yao's paper was the xor lemma, the first of a class of *direct product results*. These results make precise the following intuition: that if it is hard to compute a function b on one input, then it is harder to compute b on multiple unrelated inputs. In particular, the *xor* lemma says that, if no small circuit can compute $b(x)$ correctly for more than a $1-\delta$ fraction of x, and $k = \omega(\log n/\delta)$, then no small circuit can predict $b(x_1) \oplus ...b(x_k)$ with probability $1/2 + 1/n^c$ over random sequences $x_1...x_k$. There are a huge number of distinct proofs of this lemma, and each proof seems to have applications and extensions that others don't have. Ironically, the most frequently proved lemma in complexity was stated without proof in Yao's paper, and the first published proof was in [Lev87].

Again invoking hindsight, as suggested in [Tre03] and [I03], we can view the xor lemma as a result about *approximate local list decoding*. Think of the original function b as a message to be transmitted, of length 2^n, whose x'th bit is $b(x)$. Then the code $H_k(b)$ (which we'll call the k-sparse Hadamard code) is the bit sequence whose $(x_1, ..x_k)$th entry is $b(x_1) \oplus b(x_2) ... \oplus b(x_k)$. In the xor lemma, we are given a circuit C that agrees with $H_k(b)$ on $1/2 + \epsilon$ fraction of bit positions, and we wish to reconstruct b, except that we are allowed to err on a δ fraction of bit possitions. This allowable error means that we can reduce the traditional error correction goal of recovering the message to the weaker condition of recovering a string that agrees with the message on all but a δ on bits. In fact, it is easy to see that it will be information-theoretically impossible to avoid a $\Omega(\log 1/\epsilon/k)$ fraction of mistakes, and even then, it will only be possible to produce a list of strings, one of which is this close to the message. Fortunately, this weaker goal of approximately list decoding the message is both possible and sufficient to prove the lemma. Each member of the list of possible strings produced in a proof of the xor lemma can be computed by a relatively small circuit (using C as an oracle), and one such circuit is guaranteed to compute b with at most a δ fraction of errors, a contradiction. This description of what a proof of the xor lemma is also gives an indication of why we need so many different proofs. Each proof gives a slightly different approximate list decoding algorithm, with different trade-offs between the relevant parameters, ϵ, δ, the total time, and the size of the list produced. (The logarithm of the size of the list can be viewed as the non-uniformity, or number of bits of advice needed to describe which circuit to use.)

Finally, Yao's paper contains a far-reaching discussion of how computational complexity's view of randomness differs from the information-theoretic view. For example, information theoretically, computation only decreases randomness, whereas a cryptographic pseudo-random generator in effect increases computational randomness.

An important generalization of cryptographic pseudo-random generator was that of *pseudo-random function generator* introduced by Goldreich, Goldwasser, and Micali ([GGM]), who showed how to construct such a function generator from any secure pseudo-random generator. A pseudo-random function generator can be thought of as producing an exponentially long pseudo-random string. The string is still computable in polynomial-time, in that any particular bit is computable in polynomial time. It is indistinguishable from random by any feasible adversary that also has random access to its bits. Luby and Rackoff ([LR]) showed that a pseudorandom function generator could be used to construct a secure block cipher, the format for conventional private-key systems. It is ironic that while public-key encryption motivated a complexity based approach to cryptography, the complexity of private-key cryptography is well understood (combining the above with [HILL]), but that of public-key cryptography remains a mystery ([IR], [Rud91]).

Skipping fairly far ahead chronologically, there is one more very important paper on pseudo-randomness, by Goldreich and Levin ([GL89]). This paper shows how to construct a unpredictable bit for any one-way function. Like Blum and Micali's, this result is best understood as a list-decoding algorithm. However, without knowing any properties of the one-way function, we cannot use random self-reducibility to relate the hidden bit on one input to a long code-word. Instead, [GL89] use a randomized construction of a hidden bit. Technically, they define $f'(x, r) = f(x), r$ as a padded version of one-way function f, and define $b(x, r)$ as the parity of the bits in x that are 1 in r, or the inner product mod 2 of x and r. Thus, fixing x, a predictor for $b(x, r)$ can be thought of as a corrupted version of the *Hadamard code* of x, $H(x)$, an exponentially long string whose r'th bit is $< x, r >$, the inner product of x and r mod 2. Their main result is a local list-decoding algorithm for this code, thus allowing one to reconstruct x (as one member of a list of strings) from such a predictor. The claimed result then follows, by simply comparing $f(x)$ with $f(y)$ for each y on the list. More generally, this result allows us to convert a secret string x (i.e., hard to compute from other information) to a pseudo-random bit, $< x, r >$, for a randomly chosen r.

5.2 Interactive proofs

A second key idea to arise from cryptography grew out of thinking about computation within the context of protocols. Security of protocols was much more complex than security of encryption functions. One had to reason about what an adversary could learn during one part of a protocol and how that affected the security of later parts. Frequently, an attacker can benefit by counter-intuitive behaviour during one part of a protocol, so intuition about "sensible" adversaries can be misleading. For example, one way for you to convince another person of your identity is to decrypt random challenges with your secret key. But what if instead of random challenges, the challenges were chosen in order to give an attacker (pretending to question your identity) information about your secret key? For example, using such a chosen cyphertext attack, one can easily break Rabin encryption. To answer such questions, Goldwasser, Micali, and Rackoff ([GMR]) introduced the concept of *the knowledge leaked by a protocol*. Their standard for saying that a protocol only leaked certain knowledge was *simuability*: a method should exist so that, with the knowledge supposed to be leaked, for any strategy for the dishonest party, the dishonest party could produce transcripts that were indistinguishable from participating in the protocol *without further interaction with the honest party.* This meant that the knowledge leaked is a cap on what useful information the dishonest party could learn during the protocol.

In addition, [GMR] looked at a general purpose for protocols: for one party to prove facts to another. NP can be characterized as those statements which a prover of unlimited computational power can convince a skeptical polynomial-time verifier. If the verifier is deterministic, there is no reason for the prover to have a conversation with the verifier, since the prover can simulate the verifier's end of the line. But if verifiers are allowed to be randomized, as is necessary for zero-knowledge, then it makes sense to make such a proof interactive, a conversation rather than a monologue. [GMR] also defined the complexity classes IP of properties that could be proved to a probabilistic verifier interactively. Meanwhile, Babai (in work later published with Moran, [BMor]) had introduced a similar but seemingly more limited complexity class, to represent probabilistic versions of NP. Goldwasser and Sipser ([GS]) eventually proved that the two notions, IP and AM, were equivalent.

Once there was a notion of interactive proof, variants began to appear. For example, Ben-Or, Goldwasser, Micali and Wigderson ([BGKW]) introduced multiple prover interactive proofs (MIP), where non-communicating provers convinced a skeptical probabilistic verifier of a claim. Frankly, the motivation for MIP was somewhat weak, until Fortnow, Rompel and Sipser showed that MIP was equivalent to what would later be known as probabilistically checkable proofs. These were languages with exponentially long (and thus too long for a verifier to look at in entirety) proofs, where the validity of the proof could be verified with high probability by a probabilistic polynomial-time machine with random access to the proof. Program checking ([BK], [BLR], [Lip91]) was a motivation for looking at interactive proofs from a different angle. In program checking, the user is given a possibly erroneous program for a function. Whatever the program is, the user should never output an incorrect answer (with high probability), and if the program is correct on all inputs, the user should always give the correct answer, but the user may not output any answer if there is an error in the program. The existence of such a checker for a function is equivalent to the function having an interactive proof system where the provers strategy can be computed in polynomial-time with an oracle for the function. A simple "tester-checker" method for designing such program checkers was described in [BLR], which combined two ingredients: random-self reducibility, to reduce checking whether a particular result was fallacious to checking whether the program was fallacious for a large fraction of inputs; and downward self-reducibility, to reduce checking random inputs of size n to checking particular inputs of size $n-1$.

6 Circuit complexity and Arithmetization:

The history of circuit complexity is one of dramatic successes and tremendous frustrations. The 1980's were the high point of work in this area. The decade began with the breakthrough results of [FSS,Aj83], who proved super-polynomial lower bounds on constant-depth unbounded fan-in circuits. For a while subsequently, there was a sense of optimism, that we would continue to prove lower bounds for broader classes of circuits until we proved $P \neq NP$ via lower bounds for general circuits. This sense of optimism became known as the Sipser program, although we do not believe Sipser ever endorsed it in writing. With Razborov's lower bounds for monotone circuits ([Raz85]), the sense that we were on the verge of being able to separate complexity classes became palpable. Unfortunately, the switching lemma approach seemed to be stuck at proving better lower bounds for constant depth circuits ([Yao85,Has86]) and the monotone restriction seemed essential, since, in fact, similar monotone lower bounds could be proved for functions in P ([Tar88]). While there have been numerous technically nice new results in circuit complexity, it is fair to say that there have been no real breakthroughs since 1987. This lack of progress is still a bitter disappointment and a mystery, although we will later see some technical reasons why further progress may require new techniques.

However, in its death, circuit complexity bequeathed to us a simple technique that has gone on to produce one amazing result after another. This technique is arithmitization, conceptually (or algorithmically) interpolating a Boolean function into a polynomial over a larger field. Actually, it is more correct to say this technique was rediscovered at this time. Minsky and Pappert had used a version of arithmitization to prove lower bounds on the power of perceptrons, majorities of circuits that depended on small numbers of inputs ([MP]); their aim was to understand the computational power of neurons. Also, some papers on using imperfect random sources ([CGHFRS]) used similar ideas. However, it was Razborov ([Raz87]) and Smolensky ([Smol87]) that really showed the power and beauty of this idea. The result they proved seems like a small improvement over what was known. Parity was known not to be computable with small constant-depth circuits; what if we allowed parity as a basic operation? Razborov and Smolensky proved that adding parity gates, or, more generally, counting modulo any fixed prime, would not help compute other fundamental functions, like majority or counting modulo a different prime. This is a technical improvement. The revolution was in the simplicity of the proof. They showed how to approximate unbounded fan-in Boolean operations with low degree polynomials over any finite field. Each such gates then contributed a small amount to the error, and each layer of such gates, a small factor to the degree. Choosing the field of characteristic equal to the modular gates in the circuit, such gates had no cost in error or degree, translating directly to linear functions. An equally elegant argument showed that counting modulo another prime was in some sense "complete" over all functions for having small degree approximations, which led to a contradiction via a counting argument.

A direct use of the ideas in these papers is part of Toda's Theorem, that $PH \subseteq P^{\#P}$ ([2]). The polynomial hierarchy has the same structure as a constant-depth circuit, with existential quantifiers being interpretted as "Or" and universal as "And". A direct translation of the small degree approximation of this circuit gives a probabilistic algorithm that uses modular counting, i.e., an algorithm in $BPP^{\oplus P}$. The final step of Toda's proof, showing $BPP^{\oplus P} \subseteq P^{\#P}$, also uses arithimetization, finding a clever polynomial that puts in extra 0's between the least significant digit and the other digits in the number of solutions of a formula.

The power of arithimetization is its generality. A multilinear polynomial is one where, in each term, there is no variable raised to a power greater than one, i.e., it is a linear combination of products of variables. Take any Boolean function and any field. There is always a unique multilinear polynomial that agrees with the function on $0, 1$ inputs. We can view the subsequent "multi-linearization" of the function as yet another error-correcting code. The message is the Boolean function, given as its truth table, and the code is the sequence of values of the multi-linearization on all tuples of field elements. (This is not a binary code, but we can compose it with a Hadamard code to make it binary.) Beaver and Feigenbaum ([BF90]) observed that every multilinear function is random-self-reducible, by interpolating its values at $n+1$ points along a random line passing through the point in question. This can be viewed as a way to locally error correct the code, when there is less than a $1/(n+1)$ fraction of corruptions in the code word. Lipton ([Lip91]) made the same observation independently, and observed that the permanent function, which is complete for $\#P$, is already multi-linear. Thus, he concluded, if $BPP \neq \#P$, then the permanent is hard in the average-case (with non-negligible probability.) Since the multi-linearization can be computed in logspace given the truth table for the function, $PSPACE$, EXP and similar classes are closed under multi-linearization. Therefore, the same conclusion holds for these classes and their complete problems.

These observations about the complexity of multi-linear functions are relatively straight-forward, but they would soon lead to deeper and deeper results.

7 The power of randomized proof systems

The most stunning success story in recent complexity is the advent of hardness of approximation results using probabilistically checkable proofs. The most surprising element of this story is that it was surprising. Each move in the sequence of papers that produced this result was almost forced to occur by previous papers, in that each addressed the obvious question raised by the previous one. (Of course, much work had to be done in finding the answer.) But no one had much of an idea where this trail of ideas would end until the end was in sight. We will not be able to present many technical details of this series of results, but we want to stress the way one result built on the others. The following is certainly not meant to belittle the great technical achievments involved, but we will have to slur over most of the technical contributions.

The ideas for the first step were almost all in place. As per the above, we all knew the permanent was both random-self-reducible and downward self-reducible. [BLR] had basically shown that these two properties implied a program checker, and hence membership in IP. However, it took [LFKN92] to put the pieces together, with one new idea. The random self-reducibility of the permanent reduces one instance to several random instances, and the downward self-reducibility to several smaller instances. The new idea involved combining those several instances into one, by finding a small degree curve that passed through them all. The prover is asked to provide the polynomial that represents the permanent on the curve, which must be consistent with all of the given values, and is then challenged to prove that the formula given is correct at a random point. This gave the result that $PH \subseteq P^{\#P} \subseteq IP$. What made this result exciting is that few had believed the inclusion would hold. The rule of thumb most of us used was that unproven inclusions don't hold, especially if there are oracles relative to which they fail, as in this case ([AGH]).

The next obvious question, was this the limit of IP's power? The known limit was $PSPACE$, and many of the techniques such as random-self reduction and downwards self-reduction were known to hold for $PSPACE$. So we were all trying to prove $IP = PSPACE$, but Adi Shamir ([Sha92]) succeeded. His proof added one important new element, degree reduction, but it was still similar in concept to [LFKN92]. Once the power of IP was determined, a natural question was to look at its generalizations, such as multi-prover interactive proofs, or equivalently, probabilistically checkable proofs of exponential length. Since such a proof could be checked deterministically in exponential time, $MIP \subseteq NEXP$. [BFL91] showed the opposite containment, by a method that was certainly more sophisticated than, but also clearly a direct descendent of, Shamir's proof.

Now, most of the [BFL91] proof is not about $NEXP$ at all; it's about 3-SAT. The first step they perform is reduce the $NEXP$ problem to a locally defineable 3-SAT formula. Then they arithmetize this formula, getting a low degree polynomial that agrees with it on Boolean inputs. They also arithmetize a presumed satisfying assignment, treated as a Boolean function from variable names to values. Using the degree reduction method of Shamir to define intermediate polynomials, the prover then convinces the verifier that the polynomial, the sum of the unsatisfied clauses, really has value 0. There was only one reason why it couldn't be interpreted as a result about NP rather than $NEXP$: the natural resource bound when you translate everything down an exponential is a verifier running in poly-logarithmic time. Even if such a verifier can be convinced that the given string is a valid proof, how can she be convinced that it is a proof of the given statement (since she does not have time to read the statement)? [BFLS91] solved this conundrum by assuming that the statement itself is arithmetized, and so given in an error-corrected format.

However, later work has a different interpretation. We think of the verification as being done in two stages: in a pre-processing stage, the verifier is allowed to look at the entire input, and perform arbitrary computations. Then the prover gives us a proof, after which the probabilistic verifier is limited in time and access to the proof. When we look at it this way, the first stage is a traditional reduction. We are mapping an instance of 3-SAT to an instance of the problem: given the modified input, produce a proof that the verifier will accept. Since the verifier always accepts or rejects with high probability, we can think of this latter problem as being to distinguish between the existence of proofs for which almost all random tapes of the verifier lead to acceptance, and those where almost all lead to rejection. In other words, we are reducing to an approximation problem: find a proof that is approximately optimal as far as the maximum number of random tapes for which the verifier accepts. So in retrospect, the connection between probabilistically checkable proofs and hardness of approximation is obvious. At the time of [FGLSS], it was startling. However, since [PY88] had pointed to the need of a theory of just this class of approximation problems, it also made it obvious what improvements were needed: make the number of bits read constant, so the resulting problem is $MAX - kSAT$ for some fixed k, and make the number of random bits used $O(\log n)$ so the reduction is polynomial-time, not quasi-polynomial. The papers that did this ([AS98,ALM$^+$98]) are tours de force, and introduced powerful new ways to use arithmetization and error-correction.

They were followed by a series of papers that used similar techniques tuned for various approximation problems. For many of these problems, the exact threshold when approximation became hard was determined. See e.g., [AL] for a survey. This area remains one of the most vital in complexity theory. In addition to more detailed information about hardness of approximation, the PCP theorem is continually revisited, with more and more of it becoming elementary, with the most elegant proof so far the recent work of Dinur ([D05]).

8 The combinatorics of randomness

As mentioned earlier, many problems in constructive extremal combinatorics can be viewed as derandomizing an existence proof via the probabilistic method. However, it became clear that solving such construction problems also would give insight into the power of general randomized algorithms, not just the specific one in question. For example, Karp, Pippenger, and Sipser ([KPS]) showed how to use constructions of expanders to decrease the error of a probabilistic algorithm without increasing the number of random bits it uses. Sipser ([Sip88]) shows how to use a (then hypothetical) construction of a graph with strong expansion properties to either get a derandomization of RP or a non-trivial simulation of time by space. Ajtai, Komlos and Szemeredi ([AKS87]) showed how to use constructive expanders to simulate randomized logspace algorithms that use $O(\log^2 n/\log\log n)$ random bits in deterministic logspace.

Later, results began to flow in the other direction as well. Solutions to problems involving randomized computation lead to new constructions of random-like graphs. A basic question when considering whether randomized algorithms are the more appropriate model of efficient computation is, are these randomized algorithms physically implementable? Although quantum physics indicates that we live in a probabilistic world, nothing in physics guarantees the existence of unbiased, independent coin tosses, as is assumed in designing randomized algorithms. Can we use possibly biased sources of randomness in randomized algorithms? Van Neummann ([vN51]) solved a simple version of this problem, where the coin tosses were independent but had an unknown bias; this was generalized by Blum ([B86]) to sequences generated by known finite state Markov processes with unknown transition probabilities. Santha and Vazirani ([SV]) proved that from more general unknown sources of randomness, it is impossible to obtain unbiased bits. (Their source was further generalized by [Z90] to sources that had a certain *min-entropy* k, meaning no one string is produced with probability greater than 2^{-k}. However, there were some loopholes: if one is willing to posit two independent such sources, or that one can obtain a logarithmic number of truly random bits that are independent from the source, it is then (information-theoretically) possible to extract nearly k almost unbiased bits from such a source. For the purpose of a randomized algorihtm, it is then possible to use the source to simulate the algorithm by trying all possible sequences in place of the "truly random one". The idea of such an extractor was implicit in [Z90,Z91], and then made explicit by Nisan and Zuckerman ([NZ]).

We can view such an extractor as a very uniform graph, as follows. Let n be the number of bits produced by the source, s the number of random, k the min-entropy guarantee, and m the length of the nearly unbiased string produced, so the extractor is a function $Ext : \{0,1\}^n \times \{0,1\}^s \leftarrow \{0,1\}^m$. View this function as a bipartite graph, between n bit strings and m bit strings, where $x \in \{0,1\}^n$ and $y \in \{0,1\}^m$ are adjacent if there is an $r \in \{0,1\}^s$ with $Ext(x,r) = y$. Since the extractor, over random r, must be well-distributed when x is chosen from any set of size at least 2^k, this means that the number of edges in the graph between any set of nodes on the left of size 2^k or greater and any set of nodes on the right, is roughly what we would expect it to be if we picked $D = 2^s$ random neighbors for each node on the left. Thus, between all sufficiently large sets of nodes, the numbers of edges looks like that in a random graph of the same density. Using this characterization, Nisan and Zuckerman used extractors to construct better expanders and superconcentrators.

While we will return to extractors when we discuss hardness vs. randomness results, some of the most recent work has shown connections between randomized algorithms and constructions of random-like graphs in suprising ways. For example, the zig-zag product was introduced by Reingold, Vadhan and Wigderson to give a modular way of producing expander graphs. Starting with a small expander, one can use the zig-zag and other graph products to define larger and larger constant degree expanders. However, because it is modular, products make sense starting with an arbitrary graph. Reingold showed, in effect, that graph products can be used to increase the expansion of graphs without changing their connected components. He was able to use this idea to derandomize [AKLLR] and give the first logspace algorithm for undirected graph reachability.

A different connection is found in [BKSSW], where a new construction of bipartite Ramsey graphs (which have no large complete or empty bipartite subgraphs) used techniques from [BIW], where a method was given for combining multiple independent sources into a single almost unbiased random output. That method was itself based on results in combinatorics, namely results in additive number theory due to Bourgain, Katz, and Tao ([BKT], and Konyagin ([Kon]).

9 Converting Hardness to Pseudorandomness

As we saw earlier, Yao showed that a sufficiently hard cryptographic function sufficed to derandomize arbitrary algorithms. Starting with Nisan and Wigderson ([NW94]), similar results have been obtained with a weaker hardness condition, namely, a Boolean function f in E that has no small circuits computing it. What makes this weaker is that cryptographic problems need to be sampleable with some form of solution, so that the legitimate user creating the problem is distinguished from the attacker. Nisan and Wigderson's hard problem can be equally hard for all.

To derandomize an algorithm A, it suffices to, given x, estimate the fraction of strings r that cause probabilistic algorithm $A(x, r)$ to output 1. If A runs in $t(|x|)$ steps, we can construct an approximately $t(|x|)$ size circuit C which on input r simulates $A(x, r)$. So the problem reduces to: given a size t circuit $C(r)$, estimate the fraction of inputs on which it accepts. Note that solving this circuit-estimation problem allows us to derandomize $Promise - BPP$ as well as BPP, i.e., we don't use a global guarantee that the algorithm either overwhelmingly accepts or rejects *every instance*, only that it does so *on the particular instance that we are solving*.

We could solve this by searching over all 2^t t-bit strings, but we'd like to be more efficient. Instead, we'll search over a specially chosen small *low discrepancy set* $S = \{r_1, ...r_m\}$ of such strings, as defined earlier. The average value over $r_i \in S$ of $C(r_i)$ approximate the average over all r's for any small circuit C. This is basically the same as saying that the task of distinguishing between a random string and a member of S is so computationally difficult that it lies beyond the abilities of size t circuits. We call such a sample set *pseudo-random*. Pseudo-random sample sets are usually described as the range of a function called a *pseudo-random generator*. For cryptographic purposes, it is important that this generator be easily computable, say polynomial-time in its output length. However, for derandomization, we can settle for it to be computable in $poly(m)$ time, i.e., in exponential time in its input size. Since such a slow process is hardly describable as a generator, we will avoid using the term, and stick to low discrepancy set.

We will take an algorithmic point of view, where we show explicitly how to construct the discrepancy from the truth table of a hard function f. To show the relationship, we will denote the set as S_f.

9.1 The standard steps

The canonical outline for constructing the low discrepancy set from f was first put together in [BFNW93]; however, each of their three steps was at least implicit in earlier papers, two of which we've alredy discussed. Later constructions either improve one of the steps, combine steps, or apply the whole argument recursively. However, a conceptual break-through that changed the way researchers looked at these steps is due to [Tre01] and will be explored in more detail in the next subsection.

1. Extension and random-self-reduction. Construct from f a function $\hat{f}$ so that, if $\hat{f}$ has a circuit that computes its value correctly on *almost all* inputs, then f has a small circuit that is correct on *all* inputs. This is usually done by the multilinearization method of [BF90] or variants, that we discussed previously in the context of arithmetization. However, the real need here is that $\hat{f}$ be a *locally decodeable error-correcting code* of message f. Then, if we have a circuit that computes $\hat{f}$ *most of the time*, we can view it as a corrupted code word, and "decode" it to obtain a circuit that computes f *all of the time*.
   The key to efficiency here is to not make the input size for $\hat{f}$ too much larger than that for f, since all known constructions for S_f depend at least exponentially on this input size. This corresponds to a code with as high a rate as possible, although inverse polynomial rate is fine here, whereas it is terrible for most coding applications.
2. Hardness Amplification: From $\hat{f}$, construct a function $\overline{f}$ on inputs of size $\overline{\eta}$ so that, from a circuit that can predict $\overline{f}$ with an ϵ advantage over guessing, we can construct a circuit that computes $\hat{f}$ on almost all inputs.
   The prototypical example of a hardness amplification construction is the exclusive-or lemma [Yao82,Lev86], that we have discussed. Here $\overline{f(y_1 \circ y_2 ... \circ y_k)} = \hat{f}(y_1) \oplus \hat{f}(y_2) ... \oplus \hat{f}(y_k)$. As mentioned earlier, the generalization is to use any *approximate local list decodeable* code. Again, the key to efficiency is to minimize the input size, which is the same as maximizing the rate of the code. This is rather tricky, since direct products seem to need multiple inputs. [I95,IW97] solve this by using correlated inputs that are still "unrelated" enough for a direct product result to hold.
3. Finding quasi-independent sequences of inputs. Now we have a function $\overline{f}$ whose outputs are almost as good as random bits at fooling a size-limited guesser. However, to determine a t bit sequence to put in S, we need t output bits that look mutually random. In this step, a small sets of input vectors V is constructed so that for $(v_1, ... v_t) \in_U V$, guessing $\overline{f}$ on v_i is hard and in some sense independent of the guess for v_j.
   Then the sample set will be defined as: $S = \{(\overline{f}(v_1), ... \overline{f}(v_t)) | (v_1, ... v_t) \in V\}$
   The classical construction for this step is from [NW94]. This construction starts with a *design*, a family of subsets $D_1, .. D_t \subseteq [1, .. \mu], |D_i| = \overline{\eta}$, and $|D_i \cap D_j| \leq \Delta$ for $i \neq j$. Then for each $w \in \{0,1\}^\mu$ we construct $v_1, ... v_t$, where v_i is the bits of w in D_i, listed in order. Intuitively, each v_i is "almost independent" of the other v_j, because of the small intersections. More precisely, if a test predicts $\hat{f}(v_i)$ from the other v_j, we can restrict the parts of w outside D_i. Then each restricted v_j takes on at most 2^Δ values, but we haven't restricted v_i at all. We can construct a circuit that knows these values of $\hat{f}$ and uses them in the predictor.

The size of S_f is 2^μ, so for efficiency we wish to minimize μ. However, our new predicting circuit has size $2^\Delta poly(t)$, so we need $\Delta \in O(\log t)$. Such designs are possible if and only if $\mu \in \Omega(\overline{\eta}^2/\Delta)$. Thus, the construction will be poly-time if we can have $\overline{\eta} = O(\eta) = O(\log t)$.

[BFNW93] use this outline to get a *low-end* hardness-randomness tradeoff, meaning the hardness assumption is relatively weak and so is the derandomization result. They prove that, if there's a function in EXP that requires more than polynomial-sized circuit size, then (promise)-BPP problems are solvable in deterministic sub-exponential time. [IW97] prove a pretty much optimal *high-end* hardness-randomness tradeoff. If there is an $f \in E$ that requires exponential-sized circuit size, then (promise)-$BPP = P$. [STV01] obtains a similar high-end result, but combines the first two steps into a single step by using algebraic *local list-decodeable codes* directly, rather than creating such a code artificially by composing a local decodeable code for low noise (multivariate extension, etc.) with a local approximately list-decodeable code for high noise (xor lemma).

9.2 Extractors and Hardness vs. Randomness

At this point, Trevisan ([Tre01]) changed our perspective on hardness vs. randomness and extractors entirely. He observed that these two questions were *fundamentally identical*. This observation allowed ideas from one area to be used in the other, which resulted in tremendous progress towards optimal constructions for both.

Look at any hardness to randomness construction. From a function f we create a set S_f. If we have a test T that distinguishes a random element of S_f from random, then there is a small circuit using T as an oracle that computes f. Look at the extractor that, treats the output of the flawed source as f and uses its random seed to pick an element of S_f. If the resulting distribution were not close to random, there would be a test T that would be far from random for many S_f's from our source. Then each such f would be computable by a small circuit using T as an oracle. Since there are not many such circuits, there must be such an f with a relatively high probability of being output from the source. Contrapositively, this means that any sufficiently high min-entropy source the extracted string will be close to random.

Trevisan used this observation to use variants of the [BFNW93] and [IW97] as new constructions of extractors with better parameters than previous ones. This started a flurry of work culminating in asymptotically optimal extractors ([SU01]) for all min-entropies and optimal hardness-randomness constructions for all hardness functions ([Uma02]).

10 Hardness from Derandomization

Recently, derandomization has caused our supply of natural examples where randomness seems to help to dwindle. For example, Agrawal, Kayal, and Saxena ([AKS02]) have come up with a deterministic polynomial-time algorithm for primality, and Reingold has a deterministic logspace algorithm for undirected connectivity. Is it possible that we can simply derandomize all probabilistic algorithms without any complexity assumptions?

In particular, are circuit lower bounds necessary for derandomization? Some results that suggested they might not be are [IW98] and [Kab01], where average-case derandomization or derandomization vs. a deterministic adversary was possible based on a uniform or no assumption. However, intuitively, the instance could code a circuit adversary in some clever way, so worst-case derandomization based on uniform assumptions seemed difficult. Recently, we have some formal confirmation of this: Proving worst-case derandomization results automatically prove new circuit lower bounds.

These proofs usually take the contrapositive approach. Assume that a large complexity class has small circuits. Show that randomized computation is unexpectedly powerful as a result, so that the addition of randomness to a class jumps up its power to a higher level in a time hierarchy. Then derandomization would cause the time hierarchy to collapse, contradicting known time hierarchy theorems.

An example of unexpected power of randomness when functions have small circuits is the following result from [BFNW93]:

Theorem 1. *If $EXP \subseteq P/poly$, then $EXP = MA$.*

This didn't lead directly to any hardness from derandomization, because MA is the probabilistic analog of NP, not of P. However, combining this result with Kabanet's easy witness idea ([Kab01]), [IKW01] managed to extend it to $NEXP$.

Theorem 2. *If $NEXP \subseteq P/poly$, then $NEXP = MA$.*

Here, MA is the class of problems with non-interactive proofs certifiable by a probabilistic polynomial-time verifier. It is easy to show that derandomizing $Promise-BPP$ collapses MA with NP. It follows that full derandomization is not possible without proving a circuit lower bound for $NEXP$.

Corollary 1. *If $Promise - BPP \subseteq NE$, then $NEXP \not\subseteq P/poly$.*

Kabanets and Impagliazzo [KI] used a similar approach to show that we cannot derandomize the classical Schwartz-Zippel ([Sch80] [Zip79]) algorithm for polynomial identity testing without proving circuit lower bounds. Consider the question: given an arithmetic circuit C on n^2 inputs, does it compute the permanent function? This problem is in BPP, via a reduction to polynomial identity testing. This is because one can set inputs to constants to set circuits that should compute the permanent on smaller matrices, and then use the Schwartz-Zippel test ([Sch80], [Zip79]) to test that each function computes the expansion by minors of the previous one. Then assume $Perm \in AlgP/poly$. It follows that $PH \subseteq P^{Perm} \subseteq NP^{BPP}$, because one could non-deterministically guess the algebraic circuit for Perm and then verify one's guess in BPP. Thus, if $BPP = P$ (or even $BPP \subseteq NE$) and $Perm \in AlgP/poly$, then $PH \subseteq NE$. If in addition,

$NE \subseteq P/poly$, we would have $Co - NEXP = NEXP = MA \subseteq PH \subseteq NE$, a contradiction to the non-deterministic time hierarchy theorems. Thus, if $BPP \subseteq NE$, either $Perm \notin AlgP/poly$ or $NE \not\subseteq P/poly$. In either case, we would obtain a new circuit lower bound, although it is not specified whether the bound is for Boolean or arithmetic circuits.

Thus, the question of derandomization and circuit lower bounds are inextricably linked. We cannot make substantial progress on one without making progress on the other.

11 Natural proofs

If we cannot eliminate the need for circuit lower bounds, can we prove them? Why did circuit complexity fizzle in the late 80's?

The natural proofs paradigm of Razborov and Rudich ([RR97]) explains why the approaches used then died out, and give us a challenge to overcome in proving new lower bounds. An informal statement of the idea of natural proofs is that computational hardness might also make it hard to prove lower bounds. When we prove that a function requires large circuits, we often characterize what makes a function hard. In other words, insight into an existence of hard functions often gives us insight into the computational problem of recognizing hard functions. On the other hand, if cryptographic pseudo-random function generators can be computed (in a class of circuits), then it is computationally hard to recognize hard functions reliably. By definition, a pseudo-random function is easy to compute any bit, for an algorithm knowing the seed (also called the key). Thus, hardwiring the key, each such function has low complexity. However, random functions have high complexity. If we could reliably, given the truth table of functions, compute their complexity, this would give a way to distinguish between pseudo-random functions and truly random functions, contradicting the definition of pseudo-randomness. Unfortunately, for almost all circuit classes where we don't have lower bounds, there are plausibly secure pseudorandom function generators computable in the class. That means that our lower bounds for these classes will either have to be less constructive (not giving an effective characterization), or tailored to a specific hard function and so not give a general classification of hard functions.

Optimistically, there is no known analog of natural proofs for arithmetic circuits. Maybe we (as Valliant suggests in [Val92]) should strive for arithmetic circuit lower bounds first, before tackling Boolean circuits.

12 Conclusion

Recently, I told a long-time friend who isn't a mathematician what I was working on. His shocked reply was that that was the exact same topic I was working on my first month of graduate school, which was very close to the truth. The classical challenges that have been with us for over two decades remain. We know so much more about the questions, but so little about the answers. (Of course, I have shortchanged the genuinely new areas of complexity, such as quantum complexity, in my pursuit of links with the past.)

The more we study these problems, the closer the links between them seem to grow. On the one hand, it is hard to be optimistic about our area soon solving any one of these problems, since doing so would cut through the Gordian knot and lead to progress in so many directions. It seems that randomness does not make algorithms more powerful, but that we need to prove lower bounds on circuits to establish this. On the other hand, it seems that if cryptographic assumptions hold, proving circuit lower bounds is difficult.

On the other hand, it is hard to be pessimistic about an area that has produced so many fascinating and powerful ideas. It is hard to be pessimistic when every year, substantial progress on another classical complexity progress is made. There are no safe bets, but if I had to bet, I would bet on the longshots. The real progress will be made in unexpected ways, and will only be perfectly reasonable in hindsight.

References

[Adl78] L. Adleman Two Theorems on Random Polynomial Time. *FOCS*, 1978, pp. 75-83.

[AGH] W. Aiello, S. Goldwasser, and J. Hstad, On the power of interaction. *Combinatorica*, Vol 10(1), 1990, pp. 3-25.

[AKS02] M. Agrawal, N. Kayal, and N. Saxena, Primes is in P. *Annals of Mathematics*, Vol. 160, No. 2, 2004, pp. 781-793.

[Aj83] M. Ajtai. $\Sigma_{1,1}$ formulas on finite structures. *Annals of Pure and Applied Logic*, 1983

[AKS87] M. Ajtai, J. Komlos, and E. Szemeredi, Deterministic Simulation in LOGSPACE. *19'th STOC*, 1987, pp. 132-140.

[AGS] A. Akavia, S. Goldwasser, and S. Safra, Proving Hard-core Predicates Using List Decoding. *FOCS*, 2003, pp. 146-156.

[AKLLR] R. Aleliunas, R. Karp, R. Lipton, L. Lovasz, and C. Rackoff, Random Walks, Universal Traversal Sequences, and the Complexity of Maze Problems *20th FOCS*, 1979, pp. 218-223.

[ACR98] A.E. Andreev, A.E.F. Clementi, and J.D.P. Rolim. A new general derandomization method. *Journal of the Association for Computing Machinery*, 45(1):179–213, 1998. (preliminary version in ICALP'96).

[ACRT] A. Andreev, A. Clementi, J. Rolim, and L. Trevisan, "Weak random sources, hitting sets, and BPP simulation", *38th FOCS*, pp. 264-272, 1997.

[AL] S. Arora and C. Lund, Hardness of Approximations In, *Approximation Algorithms for NP-hard Problems*, D. Hochbaum, editor, PWS Publishing, 1996.

[ALM$^+$98] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and the hardness of approximation problems. *Journal of the Association for Computing Machinery*, 45(3):501–555, 1998. (preliminary version in FOCS'92).

[AS97] S. Arora and M. Sudan. Improved low-degree testing and its applications, In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, pages 485–495, 1997.

[AS98] S. Arora and S. Safra. Probabilistic checking of proofs: A new characterization of NP. *Journal of the Association for Computing Machinery*, 45(1):70–122, 1998. (preliminary version in FOCS'92).

[BFLS91] L. Babai, L. Fortnow, L. A. Levin, M. Szegedy, Checking Computations in Polylogarithmic Time 23rd STOC, 1991, pp. 21-31

[BFL91] L. Babai, L. Fortnow, and C. Lund. Non-deterministic exponential time has two-prover interactive protocols. *Computational Complexity*, 1:3–40, 1991.

[BFNW93] L. Babai, L. Fortnow, N. Nisan, and A. Wigderson. BPP has subexponential time simulations unless EXPTIME has publishable proofs. *Complexity*, 3:307–318, 1993.

[BMor] L. Babai and S. Moran Arthur-Merlin games: a randomized proof system, and a hierarchy of complexity class *JCSS*, Vol 36, Issue 2, 1988, pp. 254-276.

[BIW] B. Barak, R. Impagliazzo, and A. Wigderson, Extracting Randomness Using Few Independent Sources, *45th FOCS*, 2004, pp. 384-393.

[BKSSW] B. Barak, G. Kindler, R. Shaltiel, B. Sudakov, and A. Wigderson, Simulating independence: new constructions of condesnsors, ramsey graphs, dispersers and extractors. *37th STOC*, 2005, pp. 1-10.

[BF90] D. Beaver and J. Feigenbaum. Hiding instances in multioracle queries. In *Proceedings of the Seventh Annual Symposium on Theoretical Aspects of Computer Science*, volume 415 of *Lecture Notes in Computer Science*, pages 37–48, Berlin, 1990. Springer Verlag.

[BGKW] M. Ben-Or, S. Goldwasser, J. Kilian, and A. Wigderson Multi-Prover Interactive Proofs: How to Remove Intractability Assumptions. *STOC*, 1988, pp. 113-131.

[Ber72] E.R. Berlekamp. Factoring Polynomials. *Proc. of the 3rd Southeastern Conference on Combinatorics, GRAPH THEORY AND COMPUTING* 1972, pp. 1-7.

[B86] M. Blum Independent Unbiased Coin Flips From a Correlated Biased Source: a Finite State Markov *Combinatorica*, Vol. 6, No. 2, 1986, pp. 97-108. Chain FOCS 1984: 425-433

[BK] M. Blum and S. Kannan, Designing Programs That Check Their Work. *STOC*, 1989, pp. 86-97.

[BLR] M. Blum, M. Luby, and R. Rubinfeld Self-Testing/Correcting with Applications to Numerical Problems. *J. Comput. Syst. Sci.* Vol 47(3), 1993, pp. 549-595.

[BM] M. Blum and S. Micali. "How to Generate Cryptographically Strong Sequences of Pseudo-Random Bits", *SIAM J. Comput.*, Vol. 13, pages 850–864, 1984.

[BKT] J. Bourgain, N. Katz, and T. Tao, A sum-product estimate in finite fields, and applications *Geometric and Functional Analysis*, Vol. 14, 2004, pp. 27-57.

[DH] W. Diffie and M. Hellman, New Directions in Cryptography, *IEEE Transactions on Information Theory*, Vol. IT-22, No. 6, 1976, pp. 644-654.

[CGHFRS]

1. . Chor, O. Goldreich, J. Hstad, J. Friedman, S. Rudich, R. Smolensky The Bit Extraction Problem of t-Resilient Functions. *FOCS*, 1985, pp. 396-407.

[C83] S. Cook, An Overview of Computational Complexity. *Communications of the ACM*, Volume 26, Number 3, pp. 401-407.

[D05] I. Dinur, The PCP Theorem by gap amplification. ECCC tech. report TR05-046, 2005.

[FGLSS] U. Feige, S. Goldwasser, L. Lovasz, S. Safra, and M. Szegedy, Approximating Clique is Almost NP-complete. *FOCS*, 1991, pp. 2-12.

[For01] L. Fortnow. Comparing notions of full derandomization. In *Proceedings of the Sixteenth Annual IEEE Conference on Computational Complexity*, pages 28–34, 2001.

[FSS] M. Furst, J. B. Saxe, and M. Sipser. Parity, Circuits, and the Polynomial-Time Hierarchy. *Mathematical Systems Theory*, 17(1), 1984, pp. 13-27.

[GG] O. Gabber and Z. Galil. Explicit Constructions of Linear-Sized Superconcentrators. *J. Comput. Syst. Sci.* Vol. 22(3), 1981, pp. 407-420.

[Gill] J. Gill. Computational complexity of proabilistic Turing machines. *SIAM J. Comput.*, Vol. 6, 1977, pp. 675-695.

[GL89] O. Goldreich and L.A. Levin. "A Hard-Core Predicate for all One-Way Functions", in *ACM Symp. on Theory of Computing*, pp. 25–32, 1989.

[GGM] O. Goldreich, S. Goldwasser, and S. Micali. How to construct random functions. *J. ACM*, Vol. 33(4), 1986, pp. 792-807.

[GS] S. Goldwasser and M. Sipser, Private Coins versus Public Coins in Interactive Proof Systems *STOC*, 1986, pp. 59-68.

[GMR] S. Goldwasser, S. Micali, and C. Rackoff The Knowledge Complexity of Interactive Proof Systems. *SIAM J. Comput.* 18(1), 1989, pp. 186-208.

[Has86] J. Hstad Almost Optimal Lower Bounds for Small Depth Circuits. *STOC*, 1986, pp. 6-20.

[HILL] J. Hstad, R. Impagliazzo, L. A. Levin and M. Luby. A Pseudorandom Generator from any One-way Function. *SIAM J. Comput.*, 28(4), 1999, pp. 1364-1396.

[HS] J. Heintz and C.-P. Schnorr. Testing Polynomials which Are Easy to Compute. *STOC*, 1980, pp. 262-272.

[I95] R. Impagliazzo, "Hard-core Distributions for Somewhat Hard Problems", in *36th FOCS*, pages 538–545, 1995.

[I03] R. Impagliazzo. Hardness as randomness: a survey of universal derandomization. CoRR cs.CC/0304040, 2003.

[IKW01] R. Impagliazzo, V. Kabanets, and A. Wigderson. In search of an easy witness: Exponential time vs. probabilistic polynomial time. In *Proceedings of the Sixteenth Annual IEEE Conference on Computational Complexity*, pages 1–11, 2001.

[IR] R. Impagliazzo and S. Rudich Limits on the Provable Consequences of One-Way Permutations. *STOC*, 1989, pp. 44-61.

[IW97] R. Impagliazzo and A. Wigderson. P=BPP if E requires exponential circuits: Derandomizing the XOR Lemma. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, pages 220–229, 1997.

[IW98] R. Impagliazzo and A. Wigderson. Randomness vs. time: De-randomization under a uniform assumption. In *Proceedings of the Thirty-Ninth Annual IEEE Symposium on Foundations of Computer Science*, pages 734–743, 1998.

[J84] D. Johnson, The NP-completeness column: An ongoing guide. (12th article) *Journal of Algorithms*, Vol. 5, 1984, pp. 433-447.

[Kab01] V. Kabanets. Easiness assumptions and hardness tests: Trading time for zero error. *Journal of Computer and System Sciences*, 63(2):236–252, 2001. (preliminary version in CCC'00).

[Kab02] V. Kabanets. Derandomization: A brief overview. *Bulletin of the European Association for Theoretical Computer Science*, 76:88–103, 2002. (also available as ECCC TR02-008).

[KI] V. Kabanets and R. Impagliazzo, Derandomizing Polynomial Identity Tests Means Proving Circuit Lower Bounds *Computational Complexity*, Vol. 13, No. 1-2, 2004, pp. 1-46.

[Kal92] E. Kaltofen. Polynomial factorization 1987–1991. In I. Simon, editor, *Proceedings of the First Latin American Symposium on Theoretical Informatics*, Lecture Notes in Computer Science, pages 294–313. Springer Verlag, 1992. (LATIN'92).

[K86] R. M. Karp. Combinatorics, Complexity, and Randomness. *Commun. ACM*, Vol. 29(2), 1986, pp. 97-109.

[KL] R. M. Karp and R. J. Lipton, "Turing Machines that Take Advice", *L'Ensignment Mathematique*, 28, pp. 191–209, 1982.

[KPS] R. M. Karp, N. Pippenger, and M. Sipser, A time randomness tradeoff *AMS Conference on Probabilistic Computational Complexity*, 1985.

[Kon] S. Konyagin, A sum-product estimate in fields of prime order Arxiv technical report 0304217, 2003.

[Lev87] L. A. Levin, One-Way Functions and Pseudorandom Generators. *Combinatorica*, Vol. 7, No. 4, pp. 357–363, 1987.

[Lev86] L. A. Levin, Average Case Complete Problems. *SIAM J. Comput.* Vol. 15(1), 1986, pp. 285-286.

[Lip91] New directions in testing. Distributed Computing and Cryptography, 1991.

[LR] M. Luby and C. Rackoff How to Construct Pseudorandom Permutations from Pseudorandom Functions. *SIAM J. Comput.* 17(2), 1988, pp. 373-386.

[LFKN92] C. Lund, L. Fortnow, H. Karloff, and N. Nisan. Algebraic methods for interactive proof systems. *Journal of the Association for Computing Machinery*, 39(4):859–868, 1992.

[Lip91] R. Lipton. New directions in testing. In J. Feigenbaum and M. Merrit, editors, *Distributed Computing and Cryptography*, pages 191–202. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Volume 2, AMS, 1991.

[MP] M. Minsky and S. Pappert, *Perceptrons: An Introduction to Computational Geometry*, MIT Press, Cambridge, MA, 1969. (Expanded edition, 1988.)

[NW94] N. Nisan and A. Wigderson. Hardness vs. randomness. *Journal of Computer and System Sciences*, 49:149–167, 1994.

[NZ] N. Nisan and D. Zuckerman. Randomness is Linear in Space. *JCSS*, Vol 52, No. 1, 1996, pp. 43-52.

[Pap94] C.H. Papadimitriou. *Computational Complexity.* Addison-Wesley, Reading, Massachusetts, 1994.

[PY88] C.H. Papadimitriou and M. Yannakakis. Optimization, Approximation, and Complexity Classes *STOC*, 1988, pp. 229-234. *Computational Complexity.* Addison-Wesley, Reading, Massachusetts, 1994.

[Rab80] M. O. Rabin. Probabilistic Algorithm for Testing Primality. *Journal of Number Theory*, 12:128–138, 1980.

[Raz85] A.A. Razborov, Lower bounds for the monotone complexity of some Boolean functions, *Doklady Akademii Nauk SSSR*, Vol. 281, No 4, 1985, pages 798-801. English translation in *Soviet Math. Doklady*, 31:354-357, 1985.

[Raz87] A.A. Razborov, Lower bounds on the size of bounded-depth networks over a complete basis with logical addition. *Mathematicheskie Zemetki*, Vol. 41, No 4, 1987, pages 598-607. English translation in Notes of the Academy of Sci. of the USSR, 41(4):333-338, 1987.

[RR97] A.A. Razborov and S. Rudich. Natural proofs. *Journal of Computer and System Sciences*, 55:24–35, 1997.

[RS] J. Riordan and C. Shannon, The Number of Two-Terminal Series-Parallel Networks. *Journal of Mathematics and Physics*, Vol. 21 (August, 1942), pp. 83-93.

[RSA] R. Rivest, A. Shamir, and L. Adleman, A Method for Obtaining Digital Signatures and Public-Key Cryptosystems, *Communications of the ACM*, Vol.21, No. 2, 1978, pp.120-126.

[Rud91] S. Rudich The Use of Interaction in Public Cryptosystems. *CRYPTO*, 1991, pp. 242-251.

[SV] M. Santha and U. V. Vazirani, Generating Quasi-Random Sequences from Slightly Random Sources, *25th FOCS*, 1984, pp. 434-440.

[Sch80] J.T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *Journal of the Association for Computing Machinery*, 27(4):701–717, 1980.

[SU01] R. Shaltiel and C. Umans. Simple extractors for all min-entropies and a new pseudo-random generator. In *Proceedings of the Forty-Second Annual IEEE Symposium on Foundations of Computer Science*, pages 648–657, 2001.

[Sha92] A. Shamir. IP=PSPACE. *Journal of the Association for Computing Machinery*, 39(4):869–877, 1992.

[Sip88] M. Sipser Extractors, Randomness, or Time versus Space. *JCSS*, vol 36, No. 3, 1988, pp. 379-383.

[Smol87] R. Smolensky, Algebraic Methods in the Theory of Lower Bounds for Boolean Circuit Complexity. *STOC*, 1987, pp. 77-82.

[SS79] R. Solovay and V. Strassen, A fast Monte Carlo test for primality *SIAM Journal on Computing* 6(1):84-85, 1979.

[STV01] M. Sudan, L. Trevisan, and S. Vadhan. Pseudorandom generators without the XOR lemma. *Journal of Computer and System Sciences*, 62(2):236–266, 2001. (preliminary version in STOC'99).

[Sud97] M. Sudan. Decoding of Reed Solomon codes beyond the error-correction bound. *Journal of Complexity*, 13(1):180–193, 1997.

[Tar88] . Tardos The gap between monotone and non-monotone circuit complexity is exponential. *Combinatorica* 8(1), 1988, pp. 141-142.

2. , S. Toda, "On the computational power of PP and $\oplus P$", in *30th FOCS*, pp. 514–519, 1989.

[Tre01] L. Trevisan. Extractors and pseudorandom generators. *Journal of the Association for Computing Machinery*, 48(4):860–879, 2001. (preliminary version in STOC'99).

[Tre03] L. Trevisan, List Decoding Using the XOR Lemma. Electronic Colloquium on Computational Complexity tech report 03-042, 2003.

[Uma02] C. Umans. Pseudo-random generators for all hardnesses. In *Proceedings of the Thirty-Fourth Annual ACM Symposium on Theory of Computing*, 2002.

[Val92] L. Valiant. Why is Boolean complexity theory difficult? In M.S. Paterson, editor, *Boolean Function Complexity*, volume 169 of *London Math. Society Lecture Note Series*, pages 84–94. Cambridge University Press, 1992.

[vN51] J. von Neumann, Various Techniques Used in Relation to Random Digits *Applied Math Series*, Vol. 12, 1951, pp. 36-38.

[We87] I. Wegener *The Complexity of Boolean Functions.* Wiley-Teubner, 1987.

[Yab] S. Yablonski, The algorithmic difficulties of synthesizing minimal switching circuits. *Problemy Kibornetiki 2*, 1959, pp. 75-121.

[Yao82] A.C. Yao. Theory and applications of trapdoor functions. In *Proceedings of the Twenty-Third Annual IEEE Symposium on Foundations of Computer Science*, pages 80–91, 1982.

[Yao85] A.C. Yao. Separating the Polynomial-Time Hierarchy by Oracles. *FOCS*, 1985, pp. 1-10.

[Zip79] R.E. Zippel. Probabilistic algorithms for sparse polynomials. In *Proceedings of an International Symposium on Symbolic and Algebraic Manipulation (EUROSAM'79)*, Lecture Notes in Computer Science, pages 216–226, 1979.

[Z90] D. Zuckerman, General Weak Random Sources *31st FOCS*, 1990, pp. 534-543.

[Z91] D. Zuckerman, Simulating BPP Using a General Weak Random Source, *FOCS*, 1991, pp. 79-89.