

Black Boxes, Incorporated.

Mohammad Mahmoody-Ghidary, Avi Wigderson

March 9, 2009

Introduction

The term “Black Box” often refers to a device whose functionality we understand, but whose inner workings we don’t, or choose to ignore. This term appears a lot in a large variety of contexts within theoretical computer science, and happens to be extremely convenient to capture computations with restricted knowledge about or access to certain information.

In its most basic form, a *black box* (or *oracle* which we might use interchangeably) encodes a function f , to which the computation may issue query x and get the response $f(x)$. We have no knowledge (or interest) on the implementation of f in the black box – indeed, f itself may be incomputable (e.g., the characteristic function of the Halting problem). From a programming perspective, this viewpoint is convenient when solving a problem using a subroutine for f that someone else has implemented and given us its input-output specification only. This simple idea is of wide use in almost any large software development project, as well as in algorithm design. From a theoretical perspective, the ability to efficiently solve a given problem g using an oracle to f may constitute a *reduction* from g to f , showing that computing g “is not much harder than” computing f (a statement of relevance even if we have no idea how complex to compute either of these functions are). The most powerful example for black-box reductions is of course in **NP**-completeness proofs.

However, the abstraction of information access via black boxes and the roles this abstraction plays in different settings is far more varied and sometimes subtle. It might be that the black box encodes the input itself, or it is a powerful subroutine which helps to solve the problem on the given input. Sometimes observing that the algorithm A has access to its input, or a certain subroutine in a black-box way leads to unexpected conclusions/ways on how to generalize the algorithm or even solving problems that seem irrelevant to the purpose of A at the first glance. .

It is also possible to have “levels” of black-box-ness! That is, in some cases, one uses a subroutine ignoring how the computation is done in general, but still consider some restrictions on the complexity of the oracle. Furthermore, in some occasions (e.g., cryptographic constructions) different elements of the system (e.g., the implementation of the primitive and/or the adversary) may or may not be given as a black box and each setting leads to different possibilities.

In what follows we discuss, in no particular order, a collection of scenarios in which the notion of black box is a central player, exhibiting different facets of this abstraction, its implied power and limitations. Since we mostly go over the main results in different areas and will not have enough space to discuss the technical aspects thoroughly, we will keep the policy of giving general references for the topics discussed for further details. The topics discussed in this survey are as follows.

- **Section 1: Decision Trees.** Decision trees capture the notion of computing a function while the only restriction is having black-box access to the bits of the input. We will briefly compare

one's computational power in this framework when they have access to different resources: randomness, nondeterminism, or quantum computation.

- **Section 2: The Polynomial-Time Hierarchy.** We will see the different consequences of having black-box vs. non-black-box access to a program which solves the Boolean satisfiability problem SAT over the polynomial-time hierarchy.
- **Section 3: Diagonalizing Approach to $P \neq NP$.** We will see that the basic diagonalizing techniques are incapable of resolving the $P \neq NP$ problem, and the reason is that they hold relative to any oracle.
- **Section 4: Cryptographic Constructions.** We will see the results about the power of cryptographic constructions with black-box vs. non-black-box access to the implementation of the primitive and/or the adversary.
- **Section 5 Worst-Case vs. Average-Case Complexity.** We will see some positive and negative results about our abilities to prove average-case hardness (and in particular one-way functions) based on worst-case assumptions while the reduction treats the adversary in a black-box way.
- **Section 6: Learning: Opening Black Boxes.** We will see some results from computation learning, a field which aims at extracting information about a function hidden in a black box. In particular we will discuss the Goldreich-Levin lemma and its consequences and also an algorithm for black-box learning of SAT solvers.
- **Section 7: Derandomization.** We see that under reasonable assumptions about hardness of problems like SAT any algorithm A for a language in **BPP** can be derandomized to an equivalent deterministic and efficient algorithm, using pseudo-random generators which treat the algorithm A (almost) as a black box. We will also see that certain “black box properties” of the proofs in this context led to developments in the extractor theory.
- **Section 8: The Ellipsoid Method.** We will see that observing that the Ellipsoid method for solving linear programming uses a certain subroutine in a black-box way leads to generalizing the method to solve a broader class of optimization problems known as Semidefinite Programming.
- **Section 9: Volume Computation.** This section is about some possibility (and impossibility) results about the problem of computing the volume of convex bodies, when the access to the input is black-box.

Note on the notation. We will use the following basic notation. Let B be some function on binary strings. The function computed by an algorithm A with black-box access to B will be denoted A^B (the best way to think about it is that A can write a string z in a special “register”, and gets in unit time the value of $B(z)$). A *relativized* world is one in which everyone has access to such an oracle B . This can be generalized to classes of algorithms \mathcal{A} and to classes of functions \mathcal{B} . We will use the term PTM instead of “polynomial-time Turing machine”, and PPTM instead of “probabilistic PPT”. The term “efficient” will always mean “polynomial-time”.

Note on complexity classes. We generally will not define the complexity classes that we refer to, unless there are details there that matter for our discussion, and we refer the reader to the

great books on complexity theory [Gol08, AB07, Pap94] and in particular [Gol08] Appendix A for the definitions.

1 Decision Trees

The most basic use of a black box is in controlling access to the input to a problem. Here we are interested in computing some (say Boolean) function¹ $f: \{0,1\}^n \rightarrow \{0,1\}$. The algorithm knows f , and should compute $f(x)$. To gather information on the input $x = (x_1, x_2, \dots, x_n)$ it adaptively queries a sequence of indices in $[n] = \{1, 2, \dots, n\}$, and for each index i the oracle provides it with the value of x_i . When the algorithm has enough information about the input, it will announce the value of $f(x)$.

This model, often called *decision tree*, is the simplest and most basic computational model. Indeed, it is purely information theoretic, in that there is no charge for computation at all – the algorithm is charged only for information access, and its complexity is defined to be the largest number of input queries it ever makes. As simple as it is, while quite a bit is known about it, many problems regarding it are still open.

In this section our concern will be to see if there is any gain in using resources like randomness, quantum superposition, or even nondeterminism in order to compute $f(x)$ in this model, where the only restriction is black-box access to the input bits. For more details refer to the survey of Buhrman and de Wolf [BdW02] which does this comparison in more details.

Let begin by formally defining the deterministic and nondeterministic decision tree complexity of a function.

Definition 1.1. For a Boolean function $f: \{0,1\}^n \rightarrow \{0,1\}$, we define $D(f)$, the *deterministic decision tree complexity (DTC)* of f , to be the required number of (possibly adaptive) queries about the values of the bits of an arbitrary input $x = (x_1, x_2, \dots, x_n)$ by which we can determine $f(x)$. Each question is simply an $i \in [n]$, and the answer is x_i . Using nondeterminism, we define $N(f)$, the *nondeterministic DTC* of f , to be the smallest number such that whenever $f(x) = 1$, there are $N(f)$ bits of x that $f(y) = 1$ holds whenever y agrees with x on those $N(f)$ bits.²

Depicting the algorithm as a binary tree, whose internal nodes are labeled by queries, edges by answers to the queries, and leaves by outputs, the complexity is the depth of the shallowest tree for f . It is trivial that for every function f on n bits $D(f) \leq n$, and there are functions which $D(f) = n$ (e.g., the parity function: $f(x_1, \dots, x_n) = \sum_i x_i \bmod 2$), but it is also easy to find a function f depending on all bits whose complexity is only $D(f) = O(\log n)$.

For probabilistic algorithms, we demand only that the output is correct with probability at least $2/3$ on every input. Here a convenient way to think of an algorithm is a distribution (corresponding to the coin tosses performed in advance) over deterministic decision trees. The complexity of an algorithm is the maximum depth of any tree in the support, and $R(f)$, the randomized DTC of f , is the minimum complexity over all probabilistic algorithms computing f in this sense. Clearly $R(f) \leq D(f)$ for every function f , and there are known examples where the gap can be almost

¹As usual, we'll often think asymptotically of f being one of an infinite family of functions, and describe its complexity asymptotically in the input length n .

²So equivalently one can nondeterministically choose $N(f)$ bits of x to ask, and whenever $f(x) = 1$ there is a chance to ask the queries in a way that their answers confirm $f(x) = 1$.

quadratic: $D(f) = n$ but $R(f) = O(\sqrt{n \log n})$ [BSW05]. Interestingly, the gap can never be super-polynomial: for every f we have $D(f) \leq O(R(f)^3)$ [Sni85, Nis91].

For quantum algorithms, it stops making sense to view these algorithms as trees, known as the inherent cancelations of computation paths due to “interference”. More formally, quantum algorithms iterate making adaptive queries to the oracle, just like their deterministic and probabilistic counterparts. However, a query is actually a linear superposition $\sum_i \alpha_i |i\rangle$ accessing each input bit i with complex *amplitude* α_i , “in parallel”. The answer to such a query is $\sum_i \alpha_i |x_i\rangle$. The amplitudes α_i satisfy $\sum_i |\alpha_i|^2 = 1$ (so $|\alpha_i|^2$ may be viewed as the probability of asking the query i , as in the probabilistic model). Between queries the algorithm can perform arbitrary unitary operations on the amplitudes. After all the queries, the algorithm performs a “measurement”, and outputs its (random) outcome, which we demand again to be correct with probability at least $2/3$.³

It is not hard to see that quantum algorithms can simulate probabilistic ones, and so for every function f we have $Q(f) \leq R(f)$. In this query model, however, even quantum computing cannot be super-polynomially stronger than deterministic computation:

Theorem 1.2 ([BBC⁺01]). *There is an absolute constant c such that for every function f we have $D(f) \leq Q(f)^c$.*

This is one of the few models for which such a relation (i.e., that quantum power does not lead to super-polynomial advantage) is known (or even believed), and its proof is quite remarkable in its “indirectness” and the mathematical tools it uses. We give here a sketch of the ideas of the proof.

Proof Sketch: The proof has three steps. The first step shows that if a Boolean function f is computed with complexity d_1 by a quantum algorithm, then f can be approximated, in the L_∞ norm, by a multivariate real polynomial p of total degree $d_2 = O(d_1)$. Namely, for every Boolean vector x we have $|p(x) - f(x)| \leq 1/3$. The second step is the remarkable fact that for every such polynomial p , there exists another polynomial q , of degree $d_3 = d_2^{O(1)}$ which computes f exactly (i.e. $q(x) = f(x)$, for every Boolean x). Finally, every f with such a polynomial representation has a deterministic decision tree whose depth is at most $d_4 = d_3^{O(1)}$ [NS94]!⁴ \square

So the theorem shows that that $D(f)$, $R(f)$, and $Q(f)$ are all polynomially related.

Now we consider the very simple function OR , the disjunction of n bits, and show how useful studying even such simple functions in this simple model can be. It is not hard to see that $D(OR) = n$, and that $R(OR) = \Omega(n)$. One of the few examples of the power of quantum computation is Grover’s “database search” algorithm [Gro96] establishing that $Q(OR) = O(\sqrt{n})$, giving a quadratic speed-up over probabilistic algorithms, and we know [BBBV97] that this bound is tight $Q(OR) = \theta(\sqrt{n})$. But we have $N(OR) = 1$, and it shows that the gap between the nondeterministic DTC and the other measures defined above (i.e., deterministic, randomized, and quantum DTC) could indeed be super-polynomial.

Finally, let us have a look on the non-black-box version of the OR problem. That is, let assume that we are given a Boolean formula ϕ on k variables and are asked if ϕ is satisfiable or not. Let $n = 2^k$ and denote $x_i = \phi(i)$ for $1 \leq i \leq n$ to be the i^{th} bit of the n -bit input for the OR function. As x_i is easily computable from ϕ and one can build a quantum circuit for the same function,

³We were vague about how the answer to the queries is returned and how the last measurement is done. See [BBC⁺01] for a complete discussion.

⁴[BBC⁺01] proves $c \approx 6$ by using “block sensitivity” (another complexity measure) of the Boolean function f as the middle parameter, rather than the minimum degree of a polynomial representing f .

Grover’s search method gives a quantum algorithm for SAT which is quadratically faster than the best known deterministic or probabilistic one. But the exponential lower-bound of $\Omega(n) = \Omega(2^k)$ does not apply here and the question of a possible poly(k)-time algorithm in the \mathbf{P} vs. \mathbf{NP} question.

2 The Polynomial-Time Hierarchy

The question of whether $\mathbf{P} \neq \mathbf{NP}$ is perhaps the most central problem in complexity theory. The importance is implied by the fact that the equality of \mathbf{P} and \mathbf{NP} versus their difference leads to drastically different worlds regarding efficient computation and cryptography. That is, it would be a totally different setting, compared to the current situation (that $\mathbf{NP} \neq \mathbf{P}$ is assumed for cryptography), if someone came up with an efficient algorithm which solves a complete problem in \mathbf{NP} . In this section, we will see some of the different consequences of having non-black-box access to an efficient algorithm which solves such a problem vs. the case that we have only black-box access to an oracle for it (without the guarantee that the oracle hides an efficient algorithm inside). In particular, we study the different implications that these two settings imply about the polynomial-time hierarchy.

Let first have a look on the definition of the polynomial-time hierarchy. The class \mathbf{NP} is defined similar to \mathbf{P} by adding a quantifier. I.e., \mathbf{NP} is the set of all languages L for which there is a PTM V (the verifier) such that $x \in L$ iff $\exists y \in \{0, 1\}^{\text{poly}(|x|)}, V(x, y) = 1$. By adding more quantifiers before the final check we get larger complexity classes. Formally the k^{th} level of the polynomial-time hierarchy is defined as follows.

Definition 2.1 ([MS72]). The class Σ_k contains all the languages L for which there is a PPT V and a polynomial p such that $x \in L$ iff

$$\exists y_1 \in \{0, 1\}^{p(|x|)} \forall y_2 \in \{0, 1\}^{p(|x|)} \dots y_k \in \{0, 1\}^{p(|x|)}, V(x, y_1, y_2, \dots, y_k) = 1 \ .$$

The class Π_k contains the complements of the languages in Σ_k , and can be described similarly using quantifiers with the difference that now the odd quantifiers are universal. The polynomial-time hierarchy \mathbf{PH} , as a single class, is defined as $\mathbf{PH} = \bigcup_i \Sigma_i$.

Before continuing our main discussion, let see one natural way of looking at the levels of the hierarchy.

A game-theoretic perspective. Assume A and B are two players, playing a finite game, and A plays first. For a moment suppose that the game has only two moves. Therefore, A has a winning strategy iff there is a move for A such that for all moves of B the result is win for A . If we extend the number of moves in the game to any constant k , then A wins iff there is a move for A such that for every move of B there is a move of A ... such that at the end the result of the game is win for A . Suppose that all initial positions of the game can be described by some binary string x , and each player’s moves can be described by binary strings of length poly($|x|$), and the game finishes in at most k moves. Suppose also that there is a PTM which decides the winner given the initial position and all the moves. Then by definition, the set of all binary strings describing an initial position of the game in which A has a winning strategy builds a language in Σ_k , and the complement — the cases that B has a winning strategy — will therefore be language in Π_k . On the reverse direction, any language $L \in \Sigma_k$ (resp. $L \in \Pi_k$) can be interpreted as the initial winning cases for A (resp. B) in a k -move game. There is a similar connection between the games with polynomially many moves and the class \mathbf{PSPACE} . For more discussion on this connection see [Pap94] Section 19.2.

Going back to our discussion, one can ask about the computational power of deterministic polynomial-time algorithms in a world in which SAT is efficiently solvable (or say we have access to the code of a program solving SAT). As it turns out, in this case we can solve the whole class of **PH** efficiently in polynomial time.

Theorem 2.2. *If $\text{SAT} \in \mathbf{P}$, then $\mathbf{PH} = \mathbf{P}$.*

Proof. Suppose A is an efficient algorithm for SAT. Let $L \in \mathbf{NP}$ and $x \in L$ iff $\exists y \in \{0, 1\}^{\text{poly}(|x|)}, V(x, y) = 1$. We can plug in $x \in \{0, 1\}^n$ into $V(\cdot, \cdot)$, and run the Cook-Levin reduction [Coo71, Lev73] over $V(x, \cdot)$ to get a SAT instance $s(x)$ which is satisfiable iff $\exists y, V(x, y) = 1$. Then we can use A to see if $s(x)$ is satisfiable or not. Concatenating the the two steps, we get an efficient algorithm to decide membership in L . We can do similarly for $L \in \mathbf{co-NP}$.

Now suppose $L \in \Sigma_i$, and membership in L has a description of the form:

$$x \in L \text{ iff } \exists y_1 \forall y_2 \cdots Q y_k, V(x, y_1, y_2, \dots, y_k) = 1 \text{ where } Q \text{ is } \forall \text{ or } \exists.$$

By the above method one can substitute $Q y_k, V(x, y_1, \dots, y_k) = 1$ by $U(x, y_1, \dots, y_{k-1}) = 1$ for PPT U . Repeating the procedure $i - 1$ more times eliminates all the quantifiers and give us a PTM to decide L .⁵ \square

Digest. In each step of the proof that we remove the last quantifier, we need the code of the verifier V to feed into the Cook-Levin reduction. We need the code of A to get the code of U (which plays the role of V in the next round). So when we want to remove the second quantifier, the code of A (as a part of the code of the new verifier) will be used explicitly to feed in to Cook-Levin reduction.

The common believe is that $\Sigma_i \neq \Sigma_{i-1}$ for all $i \geq 1$, and in particular $\mathbf{P} \neq \mathbf{NP}$. So it is likely that we never get access to the code of a SAT-solver PPT, but even if $\mathbf{P} \neq \mathbf{NP}$, it still makes sense to imagine having access to an oracle which solves SAT (say for theoretical purposes). So one might wonder if again the whole hierarchy collapses to \mathbf{P} . The following theorem by Meyer and Stockmeyer [MS73] answers our question indirectly by giving another characterization of the levels of the polynomial-time hierarchy.

Theorem 2.3 ([MS73]). $\mathbf{NP}^{\Sigma_i} = \Sigma_{i+1}$.

Look at [Gol08] Section 3.2.2 for a proof.

So having \mathbf{NP} -access to a Σ_i oracle, one's computational power is limited to the languages in Σ_{i+1} . This shows that $\mathbf{P}^{\mathbf{NP}} \subset \mathbf{NP}^{\mathbf{NP}} = \Sigma_2$, and so we can not solve the higher levels of **PH** in polynomial time (in case that they are not equal in the first place!).

Now suppose that we somehow get black-box access to the code of a program that we know is solving SAT *efficiently*. In Section 6.2 we will see that there is in fact an efficient way to learn and reconstruct this program by just asking queries from it!

3 Diagonalizing Approach to $\mathbf{P} \neq \mathbf{NP}$

Noting the goal in computational complexity theory which is to determine the complexity class of computational problems, we shall eventually be able to separate different complexity classes (or

⁵Note that this process can not be applied to more than a constant number of quantifiers, because of the polynomial blow-up in the running time of U compared to V .

perhaps prove that they are equal). Although our success in doing so has been far below our wish, there has been moments in the history of the field that new techniques rose the hope to being able to even separate \mathbf{P} from \mathbf{NP} . Among those techniques was the *Diagonalization* method. The problem with Diagonalization (as we will see shortly) was that, in case being applied, they do “too much” for us, in the sense that the result relativizes (i.e., hold relative to any oracle O).

To be more formal let us look at a classical example of using Diagonalization. Hartmanis and Stearns [HS65] showed that granting more resource of time or space leads to more computational power.

Theorem 3.1 ([HS65]). *For any integer $k \geq 1$, $\mathbf{Dtime}(n^{k+1}) \not\subseteq \mathbf{Dtime}(n^k)$.*

The theorem is known as the time hierarchy theorem. There is an analogous space hierarchy theorem as well. We first sketch the proof of Theorem 3.1, and then will see why such techniques are unable to tell us whether $\mathbf{P} = \mathbf{NP}$ or not. The proof is reminiscent of classical arguments in mathematics from long time ago: Cantor’s proof for uncountability of \mathbb{R} [Can74] and Turing’s proof for undecidability of the Halting problem [Tur36].

Proof Sketch: The proof uses the notion of *Universal Turing machine* — a machine which simulates the running of a given machine on a given input. For simplicity let $k = 2$. We try to define a language L such that by definition $L \in \mathbf{Dtime}(n^3)$ but $L \notin \mathbf{Dtime}(n^2)$. Given an input x we can interpret it as an encoding of a Turing machine M_x (in a way that all Turing machines have some encoding), and we run $M_x(\cdot)$ over x itself for n^2 steps. If at the end it turned out that $M_x(x) = 1$, we would define $x \notin L$, and otherwise (even if the computation does not end) we define $x \in L$. This way of defining L prevents any Turing machine M to decide L on all inputs in time n^2 . That is because if there were such a Turing machine M , the behavior of M on any of its own representations M_x is not well defined. On the other hand, by using some efficient implementations of the Universal Turing machine (e.g., see [AB] Section 1.2) we have $L \in \mathbf{Dtime}(n^3)$. \square

The argument above relies on two facts: (1) One can represent Turing machines with strings and treat them as input to another machine, (2) One can simulate a given Turing machine on a given input (with a small overhead in the time). One can still simulate a program which uses oracle access to some oracle O , if the simulator itself is also provided access with the oracle O . This means that the proof indeed shows that $\mathbf{Dtime}(n^{k+1})^O \not\subseteq \mathbf{Dtime}(n^k)^O$ for any oracle O . So this way of Diagonalization relativizes. There are exception to this rule which could be found in Fortnow’s survey [For00].

The following theorem by Baker et al [BGS75] showed that relativizing techniques come short in proving (or disproving) $\mathbf{P} \neq \mathbf{NP}$.

Theorem 3.2 ([BGS75]). *There are oracles A and B such that $\mathbf{P}^A \neq \mathbf{NP}^A$, but $\mathbf{P}^B = \mathbf{NP}^B$.*

Proof Sketch: Since the common believe is that $\mathbf{P} \neq \mathbf{NP}$, perhaps the oracle B is the one who shows the issue with relativization here, and it is in fact this half of the theorem is easier to prove! One possibility is to take $B = \mathbf{EXP}$ (i.e., the class of problems computable in exponential-time⁶). Any language in \mathbf{EXP} is obviously in $\mathbf{P}^{\mathbf{EXP}}$. In addition, given any input x , in order to decide its membership in a language in $\mathbf{NP}^{\mathbf{EXP}}$, we can go over all all nondeterministic choices of the machine and also find the needed oracle answers in exponential time. So we have $\mathbf{EXP} \subset \mathbf{P}^{\mathbf{EXP}} \subset \mathbf{NP}^{\mathbf{EXP}} \subset \mathbf{EXP}$, and hence $\mathbf{P}^{\mathbf{EXP}} = \mathbf{NP}^{\mathbf{EXP}}$.

⁶One complete problem in this class is the language $\{\langle M, x, 1^n \rangle \mid M \text{ accepts } x \text{ in } 2^n \text{ steps}\}$.

As for the separating oracle, note that for any choice of A the language $L_A(x)$ defined as $\exists y \in \{0, 1\}^{|x|}, A(y) = 1$ (i.e., the *OR* function applied to the truth table of A for length $|x|$) is in \mathbf{NP}^A . Using Diagonalization⁷, A can be chosen in a way that no polynomial-time oracle machine M^A can decide membership in L_A for all x 's. \square

In the next section we will see more examples from cryptography that relativization poses limits over the standard techniques we use.

4 Cryptographic Constructions

The theory of cryptography is concerned with formally defining the notion of security for different tasks and designing provably secure protocols for them. Since even very basic cryptographic tasks need $\mathbf{P} \neq \mathbf{NP}$ to be possible, the current state in cryptography is to prove the security based on the assumptions that certain primitives exist. We do not formally define all the cryptographic notions discussed in this section and refer the reader to the great texts [Gol01, Gol04, KL07] for details.

The existence of many cryptographic primitives (e.g. secret-key encryption (SKE)) is known to be implied by the existence of one-way function (OWF) [GM84, GGM86, LR85, Rac88] and even equivalent to that [IL89, OW93].

Definition 4.1. A *one-way function* is a function $f: \{0, 1\}^n \rightarrow \{0, 1\}^n$ such that:

- **Easy to compute:** There is a PTM A computing f .
- **Hard to invert:** For any PPTM adversary E , $\Pr_{x \leftarrow_{\mathbf{R}} \{0, 1\}^n} [f(E(f(x))) = x] < \epsilon(n)$, where $\epsilon(n)$ is a “negligible” function, meaning that it is smaller than any inverse of polynomial for large enough n .

On the other hand, for some other cryptographic primitives (e.g. public-key encryption (PKE)), we have not been able to prove they exist assuming the existence of OWF, and so stronger primitives are used [DH76, Rab79, GM84, NY90, CS98]. Noting the effort spent for proving $OWF \Rightarrow PKE$ the failure so far one might wonder if there is an “explanation” for it, or cast doubt that even it is possible to prove such a statement. Noting that both the assumption and conclusion are widely believed to be true, the question seems subtle. The fact is that the way we (almost always) prove theorems like $OWF \Rightarrow SKE$ in cryptography follows a narrow form known as *black-box constructions* (which we will define shortly). So it makes sense to question if this proof method is limited to prove statements like $OWF \Rightarrow PKE$ or not.

A construction a primitive Q (e.g. PKE) from a primitive P (e.g. OWF) would consists of two reductions. The first one (the implementation reduction) gives an implementation for Q assuming that there is a way to implements P , and the second (the security reduction) should prove that the constructed implementation for Q is secure assuming that the used implementation of P was secure. Such a reduction is called black-box if both of the implementation and security reductions treat the components they access to, in a black-box way.

Definition 4.2. There is a *black-box construction* from a cryptographic primitive Q to another primitive P , if there exist efficient probabilistic oracle machines I, S such that for every oracle f implementing P we have the following.

⁷Note that here we want to show some limitations of the Diagonalization approach!

- **Correctness:** I^f is an implementation for Q .
- **Security:** For every function (adversary) g breaking the security of I^f , $S^{g,f}$ breaks the security of f .

We refer the reader to [RTV04] for a thorough elaboration on the notion of black-box constructions and their variants.⁸

One important point about black-box constructions is that they relativize. That is, assuming Q can be reduced to P in a black-box way, then for any oracle O , a secure implementation of P relative to O , implies that this is the case also for Q . The reason is that the implementation and security reductions in a black-box construction treat the used primitive and its adversary as an oracle. In the next section we will see that black-box constructions are not able to prove certain cryptographic implications that we would like to prove and believe that are true. Then we will see a few examples about the ways that we know how to detour this barrier.

4.1 Black-Box Separation

In their seminal paper [IR88], for the first time, Impagliazzo and Rudich showed that black-box constructions are not able to prove certain cryptographic statements. The question that they studied was the possibility of building secure key agreement (KA) protocols assuming that one-way permutations (OWP) exist⁹. In a KA protocol, Alice and Bob agree on a key by communicating through a public channel while it is not feasible for any efficient adversary who has access to the public messages sent between Alice and Bob to find the key.

Theorem 4.3 ([IR88]). *There exists an oracle relative to which OWP exists but KA does not. Therefore $OWP \Rightarrow KA$ can not be proven by black-box constructions.*

Proof Sketch: A random permutation with high probability is hard to invert by any PPTM. So the first step is to add a random permutation (RP) oracle for OWP to exist. The second step is to add some features to the oracle in a way that any KA can be broken efficiently. If the new part of the oracle is independently of the random permutation, the latter still remains hard to invert. The main technical part of [IR88] is to show that in an RP-relativized world, any KA can be broken by a computationally unbounded adversary who asks polynomially many number of queries to the oracle¹⁰, and they show that this adversary can be implemented efficiently if $\mathbf{P} = \mathbf{NP}$. This already shows that giving black-box construction of KA from OWP needs to prove $\mathbf{P} \neq \mathbf{NP}$. Then they add one more oracle O (e.g. $O = \mathbf{EXP}$) relative to which $\mathbf{P} = \mathbf{NP}$. \square

Now that we know black-box constructions are not able to build fully-secure KA from OWP, one can ask: Is it still possible to prove that KA is possible with some relaxed notion of security? Merkle was the first who gave such a protocol.

A weak KA protocol [Mer82]. In this protocol, Alice and Bob (and the adversary Eve) have access to a random permutation p over the set $[n^2]$. Alice and Bob will run in time $\approx n$, but if Eve runs in time $o(n^2)$, she can guess the key with probability only $o(1)$. Alice queries p on n different random points: x_1, \dots, x_n , and gets the answers $p(x_1) = a_1, \dots, p(x_n) = a_n$, and sends all of them to Bob. Now, Bob asks different random queries from p : y_1, y_2, \dots until he asks a query

⁸What we call black-box construction here, is called “fully black-box” in [RTV04].

⁹A OWP is a OWF which is a permutation when fix the length. Also note that PKE implies KA.

¹⁰This is actually enough to rule out black-box constructions.

that its answer $p(y_j) = b$ is equal to a_i for some i , and sends the index i to Alice. The key will be x_i which is equal to y_j . The birthday paradox says that Bob will ask also $\approx n$ queries. The best thing that Eve can do after observing the messages (i.e. the list of a_1, \dots, a_n and i), is to ask queries from p till he gets to know the preimage of a_i . Since p was a random permutation, Eve needs to ask $\approx n^2/2$ queries to find out $p^{-1}(a_i)$.

A simple modification of the protocol needs p to be only a random *function*, and [BGI08] shows how to use exponentially hard (standard) OWF to get quadratic security (rather than the ideal primitive like random function).

Following [IR88] many other black-box separation results followed [Sim98, GKM⁺00, GMR01, Fis02, HR04, HH09]. Another trend of results is to prove lower-bounds on the efficiency of the implementation reduction in black-box constructions [KST99, GT00, GGK03, HK05, LTW05, HHRS07, BMG07, BMG08]. The latter [BMG08] showed that quadratic security for KA is the best one can get from OWF (or even random functions) using a black-box construction.

4.2 Non-Black-Box Reductions

In this section we will talk about the less common type of results in cryptography in which either the implementation or the security reduction is non-black-box. Both types of these results deal with the concept of “zero knowledge proofs” which we will discuss briefly before getting to the non-black-box results.

4.2.1 Zero Knowledge Proofs; A Brief Introduction

The class **NP** contains all the languages L that given a witness membership in L can be verified by a PTM. If one is given a witness w for the membership of $x \in L$, they can prove $x \in L$ to anyone else by using the same witness w , and in some situations this might be a kind of “knowledge” we did not want to reveal. But is there any way at all to prove $x \in L$ to someone in a way that our proof does not carry any knowledge more than $x \in L$?

In an influential paper, Goldwasser, Micali, and Rackoff [GMR89] brought up, for the first time, the notion of an *interactive proof*. In such a proof system, the “verifier” V interacts dynamically with a “prover” P to verify if $x \in L$. The verifier speaks to the prover and might ask questions depending on the answers to the previous questions. Now the interaction plays a role for the proof being convincing, and it is not clear anymore how a proof of $x \in L$ can be learned and then repeated by the verifier for a third party. More formally we have the following definition for the class of languages for which $x \in L$ can be proved through interaction.

Definition 4.4 ([GMR89]). A language L is in **IP** if there is an efficient probabilistic interactive Turing machine V (the verifier) and a computationally unbounded interactive Turing machine P (the prover) such that:

- **Completeness:** For every $x \in L$ we have $\Pr[\langle V, P \rangle(x) = \text{accept}] = 1$ where by $\langle V, P \rangle(x)$ we mean the result of the interaction between V and P (announced by V).
- **Soundness:** If $x \notin L$, for *any* prover strategy P^* , we have $\Pr[\langle V, P^* \rangle(x) = \text{accept}] \leq 1/2$.

If the soundness property holds only against the efficient cheating provers, the protocol will be called an interactive *argument* system. If the verifier’s message consist only of public coin tosses, the protocol is called public coin or **AM** (standing for king Arthur and his wizard Merlin [Bab90]).

Perhaps surprisingly, the class **IP** turned out to be another way to define the class **PSPACE** [Sha92, LFKN92]. But since the perspective in the definition is different here, it lets us to extend the notion of interactive proofs in directions not possible to do starting from **PSPACE**.

To decrease the soundness error the verifier might decide to repeat the protocol many times and accept at the end if all the branches of interaction accept. But there are different ways to do the repetition. In a *sequential repetition* the i^{th} instance of the protocol starts after the $(i - 1)^{\text{th}}$ one ends, while in a *parallel repetition* the j^{th} round of all the branches are done at the same time. So composing k copies of a protocol of r rounds with sequential (resp. parallel) way gives a new protocol with kr (resp. r) rounds. It is not hard to see that k sequential repetition of proof and argument systems decreases the soundness error down to $\frac{1}{2^k}$. It is also true for parallel repetition of proof systems, but is *not* true for argument system [BIN97].

In the same paper [GMR89] Goldwasser et al introduced the notion of “knowledge” gained by the verifier through the interaction with the prover. The “Simulation paradigm” was the framework used to model the concept of knowledge. Roughly speaking, an interactive proof/argument system has *zero knowledge* for a potentially malicious verifier V^* if whatever he can compute at the end of the interaction with the prover P (other than $x \in L$ or not) can be simulated by an efficient simulator S , and hence can be gained without interacting with the prover.

Definition 4.5. A language L has a *zero knowledge* (ZK) proof system if there are a verifier V and a prover P as in Definition 4.4 such that:

- **Completeness and Soundness:** The same as in Definition 4.4.
- **Zero Knowledge:** There is a PPTM simulator S which for every (potentially cheating) verifier V^* , and every $x \in L$, the output of $S(V^*, x)$ (when V^* is given to S as input) is computationally indistinguishable from $\langle V^*, P \rangle(x)$.

So far we only defined the concept of ZK proofs, but have not seen if such a proof system actually exists for any non trivial language¹¹. The work of Goldreich et al [GMW86] answered this question.

Theorem 4.6 ([GMW86]). *For any language $L \in \mathbf{NP}$, there is a ZK proof system if OWF exists.*

Proof Sketch: Using Cook-Levin reduction, it would be enough to give a ZK proof system for an **NP**-complete problem. We will do it for 3-coloring problem.

Suppose for a moment that there are “digital lockable boxes” available to the parties. The prover who knows a 3-coloring for the input graph G , puts the color of all vertices c_1, \dots, c_n into locked boxes and gives them to the verifier. Now the verifier chooses a random edge $e \leftarrow |E(G)|$ from G , which connects the vertices u, v and asks the prover to open the envelopes containing the colors of u and v . If the prover is cheating, he will be caught with probability at least $\frac{1}{|E(G)|}$ (which as we will see is big enough). But what about the ZK property? If the prover is truthful, all the verifier can see is two different colors, and if the prover randomly permutes the 3 colors of the vertices before putting them into boxes, the verifier only gets to see two different random colors. This shows at the intuitive level that the protocol is ZK, and in fact it can be formalized using a simulator.

¹¹ Note that languages in **BPP** trivially have zero knowledge proof systems (with no interaction).

The existence of OWF is used to build digital lockable boxes. These boxes are protocols that let a sender (here the prover) to commit to a value v (here the color) and although the commitment scheme determines v information theoretically, a bounded receiver (i.e. the verifier) can not distinguish between the commitment of different values of v . The prover can decommit the value of v by revealing its randomness used in the commitment phase. \square

The ZK protocol of [GMW86] has a constant number of rounds, but has a high soundness error (i.e. $1 - \frac{1}{\text{poly}(n)}$) which can be decreased by sequential repetition of the protocol. But that increases the number of rounds which is perhaps the main efficiency measure of the protocol. One might suggest using parallel repetition. The bad news is that ZK property does not necessarily holds under the parallel repetition (see [Gol01] Proposition 4.5.9). The later work [FS89, BCY89] showed that for any language in **NP** there is a constant round ZK proof system with constant soundness error.

4.2.2 Non-Black-Box Implementation Reductions

An identification (ID) scheme is a protocol with n parties which lets them to identify each other after an initial setup phase. In this phase there is some “public information” p announced by all the parties, and the person i gets to know a secret s_i which he can use later to prove his identity. In addition, if person i proves its identity to person j , the information revealed by that should not let the person j to forge the identity of i in the future.

An ID scheme based on OWF. In [FFS87], an ID scheme was designed based on the assumption that there is a OWF f . In the initial phase the person i takes a random private value s_i , and announce the public value of $p_i = f(x_i)$. Later, to prove that he is indeed the person i , he can show that he knows some x such that $f(x) = p_i$, and it proves his identity because f is a OWF. If he reveals x_i , it would prove his identity, but will let others to use it later and forge his identity. But since the OWF f is known to everyone, and noting that the person i only needs to prove an **NP** statement, the person i can use the ZK proof of Theorem 4.6 rather than revealing x_i .¹² It prevents others to gain any knowledge about x_i .

Note that the actual code of f (which is the implementation of the primitive used) is needed to be fed into Cook-Levin reduction (of the ZK proof for the 3-coloring problem). That means in the construction of this identification scheme the implementation reduction uses the implementation of OWF in a non-black-box way.

There are also other cryptographic constructions that by the same reason (i.e. using ZK proofs for **NP** statements) their implementation reduction is non-black-box.

4.2.3 Non-Black-Box Security Reductions

At the end of Section 4.2.1 we mentioned that for any language in **NP** there is a constant-round ZK proof system with constant soundness error. The next natural question is if there are constant round protocols with negligible soundness error or not. Goldreich and Krawczyk [GK96] showed that it is impossible to do (for nontrivial languages), if **(1)** the verifier is public coin **(2)** The simulator S uses the cheating verifier V^* in a black-box way (i.e. as an oracle – look at Definition 4.5).

¹²This is, actually, a proof of knowledge – and extension to the basic proof systems of Definition 4.4.

Theorem 4.7 ([GK96]). *If for a language L we have a ZK argument (or proof) system with the following properties:*

- *It is constant round.*
- *The soundness error is negligible.*
- *The verifier is public coin.*
- *The simulator uses the cheating verifier as a black box.*

then $L \in \mathbf{BPP}$.

In a breakthrough, Barak [Bar01, Bar02] showed that the simulator can use the code of a the cheating verifier, to get the other properties.

Theorem 4.8 ([Bar01]). *For any language $L \in \mathbf{NP}$ there is a constant round ZK argument system with properties in Theorem 4.7 (assuming collision resistant hash functions¹³ exist).*

The work of Barak [Bar01, Bar02] so far is the only cryptographic construction with non-black-box proof of security.

5 Worst-Case vs. Average-Case Complexity

The existence of OWF is necessary for cryptography [IL89, OW93], and it is easy to see that $\mathbf{P} \neq \mathbf{NP}$ and even $\mathbf{NP} \not\subseteq \mathbf{BPP}$ are necessary for the existence of OWF. So the holy grail of cryptography could be to base the security of the protocols on the widely believed worst-case assumption of $\mathbf{NP} \not\subseteq \mathbf{BPP}$. Doing that using black-box methods might seem impossible at first, because as we saw in Theorem 4.3 even starting from the existence of OWF (let alone a necessary condition for it) black-box methods are not able to build secure KA. But since \mathbf{NP} has complete problems, we do not need to start from a *general* problem in \mathbf{NP} but could rather base the security on the hardness of an \mathbf{NP} -complete problem like SAT. That means, we can hope to construct a specific OWF f whose security relies on $\mathbf{SAT} \not\subseteq \mathbf{BPP}$ and construct cryptographic protocols whose security relies on the security of f . This way the implementation reduction for the primitives will access the OWF in a non-black-box way (f is known to the reduction), and the impossibility results of Section 4.1 do not apply here.

There are both positive and negative evidence for for the possibility of basing OWF on $\mathbf{NP} \not\subseteq \mathbf{BPP}$ with a black-box proof of security and this is the subject of the rest of this section. For a general discussion on average-case complexity and in particular its connection to worst-case complexity look at the great survey by Bogdanov and Trevisan [BT06].

5.1 Positive results

5.1.1 Locally Random Reductions

OWF is an average-case hard object. Namely, if OWF exists, then (using Cook-Levin reduction), we can sample SAT instances that are hard to solve with non-negligible probability. So the first

¹³CRHF is a function which compresses the input (i.e., the output has smaller length) and therefore have many collisions (i.e. different inputs with the same output), but it is hard to find such a pair.

step to get OWF from worst-case assumptions would be to give average-case hard samplable (e.g., uniform) distributions for SAT instances assuming its worst-case hardness. (The next step would be to do this sampling in a way that the sampler knows the answer to the sampled SAT instances.) One way to base average-case harness of a problem A on its worst-case hardness is through random self reduction.

Definition 5.1. A locally random reduction from a computational problem A to problem B is a PPTM R which given any input instance x , if $R(x)$ is given access to any oracle who solves B , it can solve the problem A on x (i.e. $\Pr[R^B(x) = A(x)] = 1$). Moreover, $R^B(x)$ asks its queries from the B -oracle according to the uniform distribution over $\{0, 1\}^{|x|}$. If A and B are the same problems, R is called a random self reduction.

Suppose we have a random self reduction for a problem A which given x asks $q = \text{poly}(|x|)$ number of oracle queries. It means that if an algorithm G solves A on $1 - \frac{1}{4q}$ fraction of the inputs, then we can solve A with probability at least $3/4$ on *any* input. For doing so we can use G as the oracle for the reduction $R^G(x)$, and the reduction will ask a query that G fails to answer with probability at most $q(\frac{1}{4q}) = \frac{1}{4}$.

Now we give two examples of random self reducible problems.

Random self reduction for the discrete logarithm DL problem. Each DL problem is parameterized with a (family of) cyclic group G and a generator g for G . For the input x we want to find the unique i , $0 \leq i \leq |G| - 1$ such that $g^i = x$. The random self reduction is as follows. Given x we choose $j \leftarrow_{\text{R}} [n]$ at random and take $z = xg^j$. Note that z has uniform distribution over G . Now given $i = \text{DL}(z)$ (i.e. $g^i = z$), we get $\text{DL}(x) = (i - j) \bmod n$.

The next example is about one of the successful cases for which we could base average-case complexity of a complexity class ($\#\mathbf{P}$) over its worst-case hardness. Since the problem of computing the permanent PERM over finite fields is complete for $\#\mathbf{P}$, a random self reduction for PERM implies that the uniform distribution over PERM is a hard distribution if $\#\mathbf{P}$ has worst-case hard problems. Note that the permanent of a matrix $M_{n \times n}$ is a multilinear polynomial (with $n!$ monomials) over the n^2 entries of $M_{n \times n}$. In fact random self reductions is possible for the more general problem of low-degree polynomials (with PERM as a special case).

Random self reduction for polynomials [BF90, Lip91]. Suppose $p(x_1, x_2, \dots, x_n)$ is a polynomial on n variables of total degree d over the field \mathbb{F} , $|\mathbb{F}| \geq d + 2$, and we want to compute $p(\vec{a})$, $\vec{a} = (a_1, \dots, a_n)$. Let $\vec{b} \in \mathbb{F}^n$. For a variable t , $p(\vec{a} + t\vec{b}) = q(t)$ is a univariate polynomial of degree d which at zero gives us the answer $q(0) = p(\vec{a} + 0\vec{b}) = p(\vec{a})$. Now if we fix $t \neq 0$ and choose $\vec{b} \leftarrow_{\text{R}} \mathbb{F}^n$ at random, the result is uniformly distributed over \mathbb{F}^n . So reduction does the following. Given \vec{a} choose t_1, t_2, \dots, t_{d+1} to be $d + 1$ different members of \mathbb{F} such that $t_i \neq 0$. Then choose $\vec{b} \leftarrow_{\text{R}} \mathbb{F}^n$ uniformly at random, and ask for $p(\vec{a} + t_i\vec{b}) = s_i$ for $1 \leq i \leq d + 1$. Knowing $q(t)$ on $d + 1$ points, we can interpolate the polynomial $q(t)$ and get $q(0)$.

Random self correction has been used to show the equivalence of average-case and worst-case hardness for \mathbf{PSPACE} and \mathbf{EXP} classes as well [STV01], but here we would like to have this result for the class \mathbf{NP} . Unfortunately, as we will see in the next section random self reductions are not able to do so if they are non-adaptive. (The reductions mentioned above were all non-adaptive.)

5.1.2 Lattice-Based Cryptography

Although we still do not know how to base cryptography on $\mathbf{NP} \not\subseteq \mathbf{BPP}$, but we have been able to base it on the worst-case hardness of problems from lattices. Yet the worst-case harness used in

this type of results do not let them to be **NP**-hard.

More formally for a set of n independent vectors $B = \{b_1, \dots, b_n\}$ in \mathbb{R}^n one can define a lattice $L(B) = \{\sum_i a_i b_i \mid a_i \in \mathbb{Z} \text{ for all } i\}$ with B as its basis. The shortest vector problem **SVP** is to compute the shortest vector in $L(B)$ for a given B . The **SVP** is **NP**-hard (even to approximate within a factor of $\sqrt{2}$ [Mic01]).

The celebrated work of Ajtai [Ajt96] showed how to generate hard instances (while the generator knows the answer) for **SVP** assuming that **SVP** is hard to approximate *in the worst-case* within a factor of $\text{poly}(n)$. Further work [AD96, GGH96, GGH96, GPV07, Pei08] showed how to get stronger cryptographic primitives (e.g. public-key encryption) from the same assumptions. Unfortunately the worst-case approximation factor in assumptions needed for these constructions was big in the sense that approximating **SVP** for that range cannot be **NP**-hard (unless $\mathbf{NP} = \mathbf{co-NP}$) [GG98].

So it remains an open question either to close the gap between the range of approximations for **SVP** known to be **NP**-hard and useful to cryptography or to show that it is not possible with black-box methods (which we will explain more in the next section).

5.2 Negative Results

In this section we will see some evidence suggesting that basing cryptography on **NP**-hardness might not be possible as long as the security reduction is black-box. In particular, we see that basing average-case hardness of **NP** on its worst-case assumption through certain forms of reductions is not possible (unless the polynomial-time hierarchy collapses).

Feigenbaum and Fortnow [FF93] showed that non-adaptive random self reductions are unlikely to be able to base the average-case hardness of **NP** on its worst-case hardness.

Theorem 5.2 ([FF93]). *There is no non-adaptive random self reduction R for SAT, unless the polynomial-time hierarchy collapses to the third level $\mathbf{PH} = \Sigma_3$.*

Proof Sketch: The idea is to use the reduction R and design an (advised) **AM** proof system (see Definition 4.4) for **co-SAT** which would imply $\mathbf{PH} = \Sigma_3$ [Yap83, GS86, BM88].

Suppose for a given input x , $|x| = n$ (for which we want to know if $x \in \text{SAT}$ or not), the verifier asks the prover the answers to the queries asked by $R(x)$ to the SAT oracle. If the prover sends back the *true* answers, the verifier gets to know $\text{SAT}(x)$. But there is no guarantee that the prover would do so. On the other hand, whenever the prover claims that an oracle query is satisfiable (i.e. a YES instance), the verifier can ask for a witness. So if we know how many of the answers should be YES, the prover can not cheat. We can not know this number, because it might depend on the instance x , but we can know the *average* number of them a_n for a random running of $R(x)$ as an advice (because a_n only depends on n). So if the reduction $R(x)$ asks $q = \text{poly}(n)$ number of queries, the verifier will run $k = q^6$ instances of $R(x)$ in parallel and ask for the answers to all of the queries she gets in the reductions. Using concentration bounds, the verifier would know that with high probability the total number of YES instances is $\approx a_n k q \pm q k^{2/3} = a_n q^7 \pm q^5$ (and she would abort the protocol if this does not happen). So the number of queries that the prover can cheat is bounded by q^5 and therefore totally there are only $q^5/q^6 = o(1)$ instances of the reduction $R(x)$ where the prover has given a wrong answer to one of its queries. Therefore for almost all of the k instances of $R(x)$ the verifier gets to know the right answer! \square

This result of [FF93] talks about reductions that are restricted in two ways: **(1)** They are non-adaptive **(2)** Each query is asked according to the uniform distribution. Bogdanov and Trevisan

extended the result to the reductions that have only the first restriction. Namely, the reduction asks its queries with arbitrary distributions, but is supposed to solve SAT whenever the oracle answers $1 - \frac{1}{\text{poly}(n)}$ fraction of the queries of length n correctly. Such a reduction is called a *self corrector*.

Theorem 5.3 ([BT03]). *There is no non-adaptive self corrector R for SAT, unless the polynomial-time hierarchy collapses to the third level $\mathbf{PH} = \Sigma_3$.*

Proof Sketch: Since this time (compared to the proof of Theorem 5.2) the queries of the reduction R are not necessarily asked according to any particular distribution we can not have the average number of YES answers a_x as an advice, and it can depend on the input x .

For a fixed input x , $|x| = n$ suppose the reduction $R(x)$ asks only queries of length n . Define a query $y \in \{0, 1\}^n$ to be α -heavy (for $\alpha = \text{poly}(n)$), if the probability that $R(x)$ asks y is at least $\alpha 2^{-n}$. If the reduction asks $q = \text{poly}(n)$ queries, among all possible queries (of length n) there can not be more than q/α fraction of them which are heavy. So for large enough α , if we simulate the oracle answers to the heavy queries by “I do not know”, the reduction is still supposed to solve SAT correctly (with high probability). The main ideas in [BT03] are the following:

- We can force the prover to tell us which queries of the reduction are heavy. This is done by using protocols to prove lower-bound and upper-bound on the probability of asking a specific query [GS86, For87, AH91].
- If we put a light query in a random position among $\Omega(\frac{1}{\alpha^2})$ queries which are sampled according to the uniform distribution, the prover can not tell which one is the special query that we hid among the rest. Therefore, if we know the fraction of YES instances for a uniformly random query (which we can, using some advice), the prover has to answer the query that we hid among the random ones correctly or otherwise would be caught with high probability! This way we can force the prover to answer the light queries correctly, and we can still run the reductions using an oracle which answers the light queries correctly.

□

Using the reductions of [IL90, BDCGL92] Bogdanov and Trevisan extend their result to the case that the reduction R relies on oracles which solve search version of SAT over *some* samplable (rather than the uniform) distribution. (Note that if we simply ask the prover for the witnesses to use them in the running of the reduction, he might *choose* these answers depending on the randomness of the reduction and fool our reduction).

The possibility of extending the result of [BT03] to reductions with more than one round of adaptivity (and perhaps any polynomial number of rounds) remains an important open question.

6 Learning: Opening Black Boxes

Most of the times, a learning problem has the following form. There is a black box hiding a function $f: \{0, 1\}^n \rightarrow \{0, 1\}$ inside. We know a priori that the function f comes from a family of functions F , and we want to know more about f by asking certain form of queries from the black box. We might want to reconstruct f completely, approximate it, or find some properties of it. For formal definitions look at [KV94].

In this section we will first see the Goldreich-Levin lemma which was originally described for cryptographic purposes but turned out to be extremely useful in different fields including computational learning and will see some applications of it. Then we will see a perhaps surprising result that using black-box access to an efficient program which solves SAT we can reconstruct this program completely!

6.1 Goldreich-Levin Lemma

Although originally described in cryptographic context, one way to look at the Goldreich-Levin lemma is that it shows how to learn large Fourier coefficients of a Boolean function f by asking $f(\cdot)$ queries. The connection to Fourier analysis and learning was noticed later by Kushilevits and Mansour [KM93]. First we see the theorem formally, then we will see different ways to look at it, and then in Section 6.1.1 we will see some applications of it to the learning theory. It simplifies our job if we think of Boolean functions to output from $\{+1, -1\}$ (-1 instead of 1 and $+1$ instead of 0).

Lemma 6.1 ([GL89]). *There is a probabilistic algorithm GL which by having oracle access to any $f: \{0, 1\}^n \rightarrow \{+1, -1\}$ runs in time $(\frac{n}{\epsilon})^{O(1)}$ and finds (with probability 0.99) the set $F = \{a \mid \hat{f}(a) \geq \epsilon\}$ of all Fourier coefficients of f which are at least ϵ .*¹⁴

Note that $|F|$ can not be too large, because by the Parseval's equality we have $\sum_a \hat{f}(a)^2 = \mathbb{E}_x[f(x)^2] = 1$ which implies $|F| \leq \frac{1}{\epsilon^2}$. The original proof of Goldreich and Levin used pseudo-randomness techniques (look at [Gol99] Appendix C), and [KM93] gave a Fourier analytic proof. Now let see some of the corollaries of GL-lemma.

- **List decoding of Hadamard.** The Hadamard encoding H_m of a message $m \in \{0, 1\}^n$ is a string of length 2^n whose a^{th} bit is equal to $(-1)^{\langle m, a \rangle}$ where $\langle m, a \rangle = \sum_i m_i a_i \bmod 2$. Suppose we receive a corrupted Hadamard codeword $f \in \{+1, -1\}^{2^n}$ of the message $m \in \{+1, -1\}^n$ where we know that only slightly less than half of the bits are corrupted: $\Pr_{x \leftarrow_R \{0, 1\}^n} [f(x) = H(x)] \geq \frac{1+\epsilon}{2}$. Note that since different Hadamard codewords differ in $1/2$ fraction of the points, unique decoding is not possible when the fraction of error is more than $1/4$. It is easy to see that $\Pr_{x \leftarrow_R \{0, 1\}^n} [f(x) = H_m(x)] \geq \frac{1+\epsilon}{2}$ iff $\hat{f}(m) \geq \epsilon$. So the GL algorithm gives us a “short” list containing all the possible messages that we are looking for!
- **Pseudo-random generators from one-way permutation.** Suppose $p: \{0, 1\}^n \rightarrow \{0, 1\}^n$ is a OWP and therefore the function $q: \{0, 1\}^{2n} \rightarrow \{0, 1\}^{2n}$ where $q(x, y) = (p(x), y)$ (for $x, y \in \{0, 1\}^n$) is also a OWP. An interesting corollary of the GL-lemma is that given $q(x, y) = (p(x), y)$ for $(x, y) \leftarrow_R \{0, 1\}^{2n}$, it is hard to compute $\langle x, y \rangle$ with non-negligible probability more than $1/2$. The reason is that if we can do so with probability $\geq 1/2 + \epsilon$, then with probability at least $\epsilon/2$ over $x \leftarrow_R \{0, 1\}^n$, if we fix x , we have that for at least $\frac{1+\epsilon}{2}$ fraction of y 's we can compute $\langle x, y \rangle$ from $(p(x), y)$.¹⁵ But it means that for such “good” x 's we have (implicit) access to a (corrupted) Hadamard codeword encoding x , which we can list-decode and output x with non-negligible probability, contradicting the fact that p was one-way.

¹⁴More precisely, GL finds F containing all such coefficient indexes, but F might also contain some b where $0.99\epsilon < \hat{f}(b) < \epsilon$. It still suffices for the applications that we will see.

¹⁵Here we assumed that the algorithm is deterministic, although similar argument works for randomized algorithms as well.

- **Approximating under uniform distribution.** Let f_ϵ be the approximation of f by keeping the large Fourier coefficients $f_\epsilon(x) = \sum_{\hat{f}(a) \geq \epsilon} \hat{f}(a) \cdot (-1)^{\langle x, a \rangle}$. Let also $L_1(f) = \sum_a \hat{f}(a)$. It is easy to see that if $\epsilon = \frac{\delta}{L_1(f)}$, then $\mathbb{E}[(f(x) - f_\epsilon(x))^2]$ is at most δ . So f_ϵ approximates f in ℓ_2 norm, and we can find such approximation using GL-lemma if we have a bound on $L_1(f)$.

6.1.1 Learning Decision Trees and DNF Formulas

In this section, we will see two applications of GL-lemma to learn specific families of functions under the uniform distribution.

Theorem 6.2 ([KM93]). *Polynomial-size decision trees can be learned efficiently under the uniform distribution.*

Proof Sketch: The first steps of the proof shows that for any function f which can be computed by a decision tree with m leaves we have $L_1(f) \leq m$. Then as we said before, we can use GL-lemma to find all Fourier coefficients of f which are at least $\epsilon = \delta/m$ (in time $(\frac{m}{\delta})^{O(1)}$), and can efficiently represent/compute the function f_ϵ which approximates f : $\mathbb{E}_x[(f(x) - f_\epsilon(x))^2] \leq \delta$. But since $f(x) \in \{+1, -1\}$, the function $\text{sign}(f_\epsilon)$ (which outputs +1 or -1) is what we are looking for: $\Pr_x[\text{sign}(g)(x) \neq f(x)] \leq \mathbb{E}_x[(f(x) - g(x))^2] \leq \delta$. □

Theorem 6.3 ([Jac97]). *Polynomial-size DNF formulas can be learned efficiently under the uniform distribution.*

Proof Sketch: A result of [BFJ⁺94] shows that for any DNF function f over n variables has a non-negligibly large Fourier coefficient. Using GL-lemma we can find such coefficient and approximate f “weakly” with probability slightly more than 1/2.¹⁶

Started by Schapire [Sch90], we now have powerful tools for “boosting” a weak learning algorithm to a strong one which approximates the function with probability ≈ 1 . But to get a strong learner under a distribution D , one needs to have weak learners under a *set* of distributions \mathcal{D} (depending on D). In the case of D being the uniform distribution Jackson showed how to generalize the work of [BFJ⁺94] to get weak learners for all the distributions in \mathcal{D} to use boosting and get a strong learner for DNF functions. □

6.2 Learning SAT-Solver Circuits

Although we believe that there is no family of polynomial-size circuits solving SAT [KL80], but if some day someone claims that they have found an efficient algorithm for SAT and they let us ask them arbitrary queries (perhaps to show that they indeed have such an algorithm) are we able to “reconstruct” the algorithm using only black-box access to the algorithm? As Bshouty et al [BCG⁺95] showed the answer is yes!

Theorem 6.4 ([BCG⁺95]). *Let $\mathcal{C} = (C_1, C_2, \dots)$ be a sequence of circuits that C_n solves SAT on instances of length n and $|C_n| \leq s(n)$. There is a learning algorithm which given 1^n and having input/output access to members of \mathcal{C} , runs in expected $s(n)^{O(1)}$ time and outputs a circuit D which solves SAT on instances of length n (like C_n).*

¹⁶We do not have $L_1(f) \leq \text{poly}(n)$ here anymore, otherwise we could already get a strong approximation.

Proof Sketch: First assume somehow artificially that the learning algorithm is granted an extra feature: It can ask “equivalence queries”. That is, it can query a circuit C , and will be answered if C solves all SAT instances of length n or not, and in case it does not, the answer will contain a counter example x (i.e. $C(x) \neq C_n(x)$).

At the beginning of the learning process, we know that the circuit C_n (or an equivalent one) which we are looking for has size at most $s(n)$. Since any circuit of size s can be represented with $O(s^2)$ bits, we should look for our needle in a haystack of size at most $2^{O(s(n)^2)}$. We will ask the equivalence queries in such a way that in each step the number of possible candidates reduces by a constant factor, and doing so we will find the circuit in $\log(2^{O(s(n)^2)}) = O(s(n)^2)$ steps.

Suppose we have asked i equivalence queries and got i counter examples x_1, \dots, x_i . Let F be the set of circuits of size at most $s(n)$ which is consistent with our current knowledge about C_n . That is members of F do answer x_1, \dots, x_i correctly when considered as SAT instances. We will ask the next query C such that either it is equivalent to C_n , or if not, any possible counter example x to C is a “good” one: adding x to our current knowledge about C_n , will shrink F by a factor of $3/4$.

Now let see when x is a good counter example. It is so whenever at most $3/4$ fraction of F agree about the satisfiability of x . Namely if we define $\delta_0(x) = \frac{|\{E \in F \mid E(x)=0\}|}{|F|}$ and $\delta_1(x) = 1 - \delta_0(x)$, x will definitely be a good counter example (regardless of our query circuit C) if $\delta_0(x) < 3/4$ and $\delta_1(x) < 3/4$. So let x be such that say $\delta_0(x) > 3/4$ (to have the potential of not being good). But that itself means that if our query circuit C is a random member of F , then x will be a good counter example with probability at least $3/4$. In fact since we can verify membership in F , we are able to sample from F uniformly using an NP oracle [JVV86, BGP00]¹⁷ which we do have here: our circuit family \mathcal{C} ! But we want to ask our equivalence query C in a way that *every* x can only be a good counter example. For doing so, we sample many random instances $E_1, E_2, \dots, E_{\text{poly}(n)}$ from F , and take C to be their majority. This way, any x has chance at most 2^{-2^n} of being a bad counter example, and by the union bound there cannot be any bad counter example (with probability at least $1 - 2^{-n}$).

But our learning algorithm so far uses equivalence queries which is an unrealistic assumption. A nice point is that using the Cook-Levin reduction we can write any statement of the form: “*there is a counter example to C as a SAT solver*” as a SAT instance and by the self-reducibility of SAT we can find such a counter example by using standard queries to \mathcal{C} !

□

7 Derandomization

As we saw in Section 4.2.1 randomness plays a crucial role in interactive proofs and it is in fact necessary for cryptographic protocols in general. Two other areas in which randomness seems to be of great use are the following.

- Probabilistic method in combinatorics (see [AS92]) is a way to prove the existence of objects with nice properties by constructing the objects in a random way and arguing that most of them are having the nice property we want. A natural question is when this process can

¹⁷The result of [BGP00], which allows uniform sampling, was not known at the time [BCG⁺95] was published. They just used approximately-uniform generation of [JVV86] which is enough for their purpose.

be turned into a deterministic construction of the object with the needed property (in time polynomial over the size of the object).

Example: Finding hard functions. For function $f: \{0,1\}^n \rightarrow \{0,1\}$ define $C(f)$ to be the size of the smallest circuit computing f . It is easy to see that a random function f sampled from the set of all 2^{2^n} possibilities with high probability has circuit complexity $C(f) \geq 2^{\Omega(n)}$. Is there a way to construct such a function deterministically in a time which is polynomial over its size (i.e. in time $\text{poly}(2^n) = 2^{O(n)}$). Equivalently is there a language $L \in \mathbf{E} = \mathbf{Dtime}(2^{O(n)})$ which $C(f) \geq 2^{\Omega(n)}$?

- There are computational problems that randomness seems to play a great role in the efficiency of solving them. Can we derandomize such algorithms and show that $\mathbf{BPP} = \mathbf{P}$?

Example: Testing zero polynomials. Given a polynomial P over a field F represented by an algebraic circuit¹⁸ C_p , is $P(x) = 0$ for all $x \in F$ or not? It can be shown that if $\deg(P) < \frac{F}{2}$, and P is not always zero, then a random assignment $P(x), x \leftarrow_{\mathbf{R}} F$ is nonzero with probability at least $1/2$. So we can test if such a polynomial is zero or not in \mathbf{BPP} , but still do not know if the problem $\mathbf{ZEROP} \in \mathbf{P}$ or not.¹⁹

In this section we will see that solving the first example (by constructing hard functions in \mathbf{E}) answers the second question positively: $\mathbf{BPP} = \mathbf{P}$. This is done by using a hard function $f \in \mathbf{E}$ and constructing a *Pseudo-Random Generator (PRG)* G_f (which use short truly random seeds and expand them) for any \mathbf{BPP} algorithm. This approach derandomizes all algorithms in \mathbf{BPP} while treats them as a black box (with the only restriction that the black box has a polynomial-time algorithms in). The proof of the construction for PRG is by a reduction that takes a potential distinguisher D (which can differentiate the pseudo-random distribution from a truly one) and turning it into a short circuit that computes f . The proof uses the adversary D in a black-box way, and this fact is used to give new constructions for *Randomness Extractors* — objects not known to be relevant to PRG at the time.

For a complete discussion look at [AB] Chapter 20, [Gol08] Chapters 7, 8, and [Kab02].

7.1 Derandomizing BPP with PRG's

Definition 7.1. An (ℓ, T) -PRG G is a function $G: \{0,1\}^\ell \rightarrow \{0,1\}^T$ computable in time $2^{O(\ell)}$ such that for any circuit D of size at most T ,²⁰ we have $|\Pr[D(U_T) = 1] - \Pr[D(G(U_\ell)) = 1]| \leq \frac{1}{T}$.

Since G can always be computed by a circuit of size $2^{O(\ell)}$, there is always a circuit of size $2^{O(\ell)}$ which outputs 1 only on the images of G . So we can not take $\ell = o(\log T)$. Now suppose G is a $(\theta(\log T), T)$ -PRG. We can use G to derandomize any algorithm A for a language in \mathbf{BPP} : Given an input $x, |x| = n$ to A , we can hardwire x into A and using Cook-Levin reduction get a circuit C_x of size $\text{poly}(n)$ which given a random seed r , outputs in $\{0,1\}$ and for at least $2/3$ fraction of r 's gives the right answer. If we take $T = \text{poly}(n)$ to be the size of C_x , by the definition of G we have $|\Pr[C_x(U_T) = 1] - \Pr[C_x(G(U_\ell)) = 1]| \leq \frac{1}{T} < \frac{1}{10}$, and so by enumerating all pseudo-random

¹⁸It is a circuit over F with addition and multiplication gates.

¹⁹Testing if a given number is prime or not used to be such an example [Mil76, SS77, Rab80], but now we know how to do it deterministically [AKS02].

²⁰Here we do not count the input gates for the circuit size.

inputs to C_x , and taking the majority we can find out if x is in the language defined by A or not in time $2^\ell \times 2^{O(\ell)} \times T = \text{poly}(n)$.

So PRG's can be used for derandomizing **BPP** algorithms, but do such objects exist? The following theorem is the result of a series of exiting work [BFNW93, NW88, IW97, ACR98, STV98, SU01, Uma03] and shows that any degree of (non-uniform) hardness in the class **E** translates into PRG's of related parameters.

Theorem 7.2 ([Uma03]). *If there exist $f \in \mathbf{E}$ with $C(f) \geq S(n)$, then there exist an $(\ell = O(n), T = S(n)^{\Omega(1)})$ -PRG G_f .*

Proof Sketch: We first sketch the proof for the extreme special case of $S(n) = 2^{cn}$ for a constant c .

Starting from $f_1 \in \mathbf{E}$ with $C(f) \geq 2^{\Omega(n)}$ the first step is to construct another function $f_2: \{0, 1\}^{O(1)} \rightarrow \{0, 1\}$ which is still computable in time $2^{O(n)}$ but is mildly hard on average. Namely g will be $2^{\Omega(n)}$ mildly average-hard, where an S mildly average-hard function h is one which for every circuit C of size at most S we have $\Pr_{x \leftarrow_{\mathbf{R}} \{0, 1\}^n} [C(x) = h(x)] \leq 1 - \frac{1}{n}$. This step uses polynomial-size error correcting codes with sublinear-time decoding algorithms (which are based on random self reducibility of multivariate polynomials – look at Section 5.1.1).

The second step is to take f_2 and construct $f_3: \{0, 1\}^{O(n)} \rightarrow \{0, 1\}$ which is again computable in time $2^{O(n)}$, but is $2^{\Omega(n)}$ strongly average-hard, where an S strongly average-hard function h is one which for every circuit C of size at most S we have $\Pr_{x \leftarrow_{\mathbf{R}} \{0, 1\}^n} [C(x) = h(x)] \leq \frac{1}{2} + \frac{1}{S}$. This step is based on a derandomized version of Yao's XOR lemma [IW97]. [STV98] shows how to do the first two steps at once using certain list-decodable codes.

Having f_3 defined on inputs of length say $\{0, 1\}^{c'n}$, the final step is to take an input (to the PRG) of length $10c'n$ and apply f_3 to $2^{\Omega(n)}$ number of $c'n$ -sized subsets of the input bits. This steps uses certain combinatorial designs [NW88] which keeps the output bits sort of “independent” and can deduce the pseudo-randomness of the output bits from the hardness of f_3 by a hybrid argument [Yao82].

The proof of all above steps is by reduction and combining them, the proof (also in the general case) shows that for every function f and every distinguisher D which $\frac{1}{T}$ -distinguishes U_T from $G_f(U_\ell)$, there is a circuit d of size $T^{O(1)}$ which given oracle access to D describes f fully. This shows that if the distinguisher D can be implemented with a circuit of size T , d^D will have circuit size $T^{O(1)}$ which by taking T small enough $T^{O(1)}(n) < S(n)$ we get a contradiction. \square

But is proving circuit lower-bounds necessary for derandomization? After all, circuit lower-bounds are notoriously hard to prove [RR97] and we might still be able to derandomize **BPP** algorithms to some extent with another method. The subsequent works [IKW02, KI04] showed that derandomizing **BPP** implies some circuit lower-bounds! (The class **AlgP/poly** contains functions over integers that can be computed using $+$, $-$, and \times gates.)

Theorem 7.3 ([KI04]). *If $\text{ZEROP} \in \mathbf{P}$ then either $\text{NEXP} \not\subseteq \mathbf{P}/\text{poly}$ or $\text{PERM} \not\subseteq \mathbf{AlgP}/\text{poly}$.*

7.2 Trevisan's Extractor

A theory which speaks about how to use random bits in computation without talking about how to find/prepare them is not a complete one. The goal in extractor theory is to extract randomness from sources with high entropy (which could be found in nature) into independent unbiased random bits. Here we will focus on the case of single source extractors and for a broad discussion of the subject

refer the reader to the excellent survey of Shaltiel [Sha02] and references there. (The *min-entropy* of a random variable X is defined as $H_\infty(X) = -\log(\min_x \Pr[X = x])$.)

Definition 7.4. A function $f: \{0, 1\}^n \times \{0, 1\}^\ell \rightarrow \{0, 1\}^T$ is a (k, ϵ) -extractor, if for any random variable X over $\{0, 1\}^\ell$ with $H_\infty(X) \geq k$ the statistical distance between U_T and $f(U_n, X)$ is at most ϵ .

We are interested in *explicit* constructions for f , and prefer the parameters to be: T as large as $\approx k$ and ϵ as small as possible.

In a breakthrough [Tre01] which led to many subsequent improved constructions Trevisan showed that PRG's constructed in Theorem 7.2 are indeed randomness extractors! The connection is surprising because extractors are information-theoretic objects and the more anticipated direction would be to find a computational version of an information theoretic phenomenon (e.g. Yao's XOR lemma [Yao82, GNW95] for hardness amplification).

Theorem 7.5 ([Tre01]). *The PRG of Theorem 7.2 is a $(T^{O(1)}, O(\frac{1}{T}))$ -extractor.*

Proof. We say that a statistical test (i.e. distinguisher) D breaks f if it $\frac{1}{T}$ -distinguishes U_T from $G_f(U_\ell)$. The proof of Theorem 7.2 shows that whenever D breaks f there is a circuit d_f of size $W = T^{O(1)}$ such that d_f^D computes f . Since there are at most $2^{O(W^2)}$ circuits of size at most W . The key point is that d uses the adversary D as a black box, so for a *fixed* D , there can not be more than $2^{O(W^2)}$ number of functions f that D breaks. So if the distribution over f has min-entropy at least $T \times W^2$, with probability at most $O(\frac{1}{T})$ over choice of f , D can break it. Finally whenever D does not break f , it can only ϵ -distinguish U_T from $G_f(U_\ell)$ for $\epsilon \leq \frac{1}{T}$. So D can not distinguish U_T from $G_f(U_\ell)$ with better than a gap of $\frac{1}{T} + O(\frac{1}{T}) = O(\frac{1}{T})$. □

8 Ellipsoid Method

Due to their vast applications in computer science and engineering, finding efficient algorithms to solve linear optimization problems is an important research area in the recent decades. In this section we will see how an observation about the fact that an optimization method uses a subroutine in a black-box way led to a generalization to the algorithm for solving a much bigger class of problems.

In theoretical computer science, one of the most important applications of linear programming is in combinatorial optimization. For instance, for a large number of combinatorial problems (e.g. Maximum Flow and Shortest Path) one can easily reduce the problem to a linear optimization problem. We refer the reader to the books [Sch03, PS98] for comprehensive discussion of the subject.

To be more formal, a *Linear Programming* (LP) problem is one in which the constraints and the optimization function are all linear. Namely we want to find a vector $x \in \mathbb{R}^n$ maximizing $c^T \cdot x$ under the condition that $Ax \leq b$, for $c \in \mathbb{R}^n, A \in \mathbb{R}^{m \times n}, b \in \mathbb{R}^m$. From a geometric point of view, in an LP instance, the constraints $Ax \leq b$ which form the “feasible domain” of x , describe a polytope in \mathbb{R}^n and we want to find the “last point” of this polytope in some direction specified by c .

The most widely used method for solving LP problems is the Simplex method [Dan51] of Dantzig. The intuition behind the Simplex method is to move along the vertices of the polytope while in each step one moves to an “adjacent” vertex which improves the objective function. Although

the number of vertices for a high dimensional polytope (which can be described by polynomially many hyperplanes) could be exponential, and in fact one can find instances on which the Simplex method takes exponential time, it is still an effective algorithm for instances that come up in real life applications. In [ST04] Spielman and Teng gave an explanation for this phenomenon.

Later work showed how to use randomization in the Simplex method to guarantee polynomial time [KS06], but it was first in 70's when Khachiyan [Kha79] gave the first polynomial-time algorithm for solving LP. The method is called *Ellipsoid Method*, because of the elegant way it uses high dimensional ellipsoids to give an iteratively improving bound on the region that the optimal point is. First, we describe the framework of the Ellipsoid method and then will see how a more careful look at it leads to a generalization of the applicability of the method.

Sketch of the Ellipsoid Method. By adding an artificial constraint and using binary search, solving an LP problem can be reduced to checking if there is a feasible solution (i.e. x for which $Ax \leq b$) without caring about the optimality. So we would like to know if a given polytope described by (A, b) is empty or not.²¹ We begin by finding a not-too-big ellipsoid E_1 which contains the polytope²². The algorithm now works in iterations. In the beginning of the j^{th} iteration, the ellipsoid E_j contains the polytope. Let the point $x_j \in \mathbb{R}^n$ be the center of E_j . We will check if x_j is a feasible solution or not by seeing if $Ax_j \leq b$. If it was feasible, we are done. Otherwise it means that for some i , $a_i^T \cdot x_j > b_i$ where the vector a_i^T is the i^{th} row of A . This means that the whole polytope is inside the half space $X_j = \{x \mid a_i^T \cdot x \leq a_i^T \cdot x_j\}$, and by the convexity of the polytope, x_j is indeed in $E_j \cap X_j$. One interesting geometric fact is that the smallest ellipsoid which contains $E_j \cap X_j$ (for X_j being *any* half-space with the center of E_j on its border) has a smaller volume than E_j by some noticeable factor depending on the dimension n . Fortunately this smaller ellipsoid can be found efficiently, and we take E_{j+1} to be the ellipsoid with smallest volume which contains $E_j \cap X_j$. It can be shown that in case of being non-empty, the volume of the polytope described by (A, b) can not be “too small” (depending on the number of bits for representing (A, b)). Therefore after polynomially many number of iterations, we either find a feasible point, or can make sure that it does not exist.

In [KP82] Karp and Papadimitriou pointed out that the idea used in the Ellipsoid method can be applied to more general forms of optimization problems. In particular, if we want to check the emptiness of the feasible domain, what we want are the following features of the algorithm.

1. **Convexity:** The feasible domain is convex.
2. **Membership Oracle:** For a given input point x_0 , we are able to check the membership of x_0 in the feasible domain in polynomial-time.
3. **Separating Oracle:** If the answer to the question in the previous item was negative, we can efficiently find a hyperplane which separates our query point from the feasible domain.

As the first application of this observation, in [KP82] they gave examples of LP instances with exponentially many constraints for which the membership and separation oracles can be efficiently implemented.

Moreover, by this approach one can get polynomial-time algorithms for solving *Semi-Definite Programming* (SDP) [GLS81]. In a SDP, the vector of variables come from the space PSD_n – the set of positive semidefinite matrices in $\mathbb{R}_{n \times n}$. In addition to this implicit constraint over the variables,

²¹Here we ignore the case that the polytope is unbounded

²²We assume here that the polytope is bounded.

we can also have arbitrary linear constraints and the objective function is also linear (similar to LP). Any LP can be rephrased as an SDP problem, so it is a stronger form of an optimization problem. The space of matrices in PSD_n is convex. In addition the membership and separation oracles can be implemented efficiently for the feasible domain described by SDP instances. Therefore using the Ellipsoid method SDP instances can be solved efficiently. In a breakthrough [GW95], Goemans and Williamson illustrated the power of SDP by giving improved approximation algorithms for the Max Cut and Max 2-SAT problems and since then the applications of SDP for approximation algorithms and optimization problems have been numerous.

9 Volume Computation

In this section, like Section 1 we again will compare the computational power provided by randomness (compared to the deterministic computation) when the access to the input is in a black-box way. But this time we focus on the specific problem of computing the volume, and the input is also of a special format. Namely, the input is a convex high dimensional body B and we have membership and separation oracles (look at Section 8) for B . (Also, at the beginning we have an initial point inside the body to start with.) We call this special case of the volume computation VC problem (where the input is accessible by membership and separation oracles) the black-box VC problem.

A note on the non-black-box VC. Let us have a look at the non-black-box version of VC where we are given the input completely at the beginning. In particular we want to compute the volume of the polytope described by $Ax \leq b$ where $A \in \mathbb{R}_{m \times n}$ and $b \in \mathbb{R}^m$. Although this problem is easy in two dimension, the curse of dimensionality is here with us, and the problem in arbitrary dimension is in fact $\#\mathbf{P}$ -complete [DF88]!

Going back to our black-box VC problem, the problem is not definitely easier compared to the non-black-box case (because we can implement the membership and separation oracles for the polytope described by (A, b) on our own). So what happens if we relax the problem and look only for an approximation of the volume? In [BF86], Barany and Furedi showed that even approximation is not possible!

Theorem 9.1 ([BF86]). *There is no polynomial-time deterministic approximation algorithm for the black-box VC problem with approximation factor better than $\Omega((\frac{n}{\log n})^n)$.*

The result is almost tight because an approximation algorithm with factor n^n for VC had already been known [M. 88].

The next idea could be to relax the resources rather than the aim: can randomness help us to find the volume with high probability? Again noting that the non-black-box version of the problem is $\#\mathbf{P}$ -complete we should not be so hopeful. Dyer et al [DFK91] showed that using *both* of the relaxations (i.e. having randomness and looking for approximation) things do change very much by giving a (randomized) fully polynomial approximation scheme for VC!

Theorem 9.2 ([DFK91]). *There is a randomized algorithm A which given ϵ and black-box access to the body K , runs in time $(\frac{n}{\epsilon})^{O(1)}$ and outputs v such that $\text{VOL}(K) \leq v \leq (1 + \epsilon)\text{VOL}(K)$ with probability 0.99.*

Proof Sketch: Let $0 < c < 1$ be a parameter that we choose later. Let L_c be an n -dimensional lattice defined with the base $B_c = \{(c, 0, \dots, 0), (0, c, 0, \dots, 0), \dots, (0, \dots, 0, c)\}$ (look at Section

5.1.2). For each convex body K , if we take c to be small enough and count the number of lattice points inside the body, it gives us a good approximation of $\text{VOL}(K)$. The first idea to solve the problem lies in the connection between counting and sampling. Jerrum et al [JVV86] showed that in verity of settings (which covers our case here) these two tasks are essentially equivalent (up to an approximation factor).

Now we only need to sample an (approximately) uniform lattice point inside K . If we run a random walk over the lattice nodes inside the body (by defining the lattice points with distance at most c as adjacent), the distribution over the nodes eventually converges to the uniform. So if we can show that this convergence happens in polynomial time we can do the sampling using a long enough random walk and then choosing the last node. The tools developed by Jerrum and Sinclair [JS88] showed that this random walk, does converges in polynomial time. It is interesting to know that the problem for which those tools were developed was to approximate the Permanent of Boolean matrices! \square

References

- [AKS02] M. Agrawal, N. Kayal, and N. Saxena. PRIMES is in P. Report, Department of Computer Science and Engineering, Indian Institute of Technology Kanpur, Kanpur-208016, India, Aug. 2002.
- [AH91] W. Aiello and J. Hastad. Statistical Zero-Knowledge Languages Can Be Recognized in Two Rounds. *Journal of Computer and System Sciences*, 42(3):327–345, June 1991.
- [Ajt96] Ajtai. Generating Hard Instances of Lattice Problems. In *ECCCTR: Electronic Colloquium on Computational Complexity, technical reports*, 1996.
- [AD96] Ajtai and Dwork. A Public-Key Cryptosystem with Worst-Case/Average-Case Equivalence. In *ECCCTR: Electronic Colloquium on Computational Complexity, technical reports*, 1996.
- [AS92] N. Alon and J. H. Spencer. *The probabilistic method*. Wiley, New York, 1992.
- [ACR98] Andreev, Clementi, and Rolim. A New General Derandomization Method. *JACM: Journal of the ACM*, 45, 1998.
- [AB] S. Arora and B. Barak. *Computational Complexity, A Modern Approach*.
- [AB07] S. Arora and B. Barak. *Computational Complexity: A Modern Approach (Draft)*. January 2007.
- [BFNW93] Babai, Fortnow, Nisan, and Wigderson. BPP Has Subexponential Time Simulations Unless EXPTIME Has Publishable Proofs. *CMPCMPL: Computational Complexity*, 3, 1993.
- [BM88] Babai and Moran. Arthur-Merlin Games: A Randomized Proof System, and a Hierarchy of Complexity Classes. *JCSS: Journal of Computer and System Sciences*, 36, 1988.

- [Bab90] L. Babai. E-mail and the Unexpected Power of Interaction. In *Structure in Complexity Theory Conference*, pages 30–44, 1990.
- [BGS75] Baker, Gill, and Solovay. Relativizations of the $P =? NP$ Question. *SICOMP: SIAM Journal on Computing*, 4, 1975.
- [BMG07] Barak and Mahmoody-Ghidary. Lower Bounds on Signatures From Symmetric Primitives. In *FOCS: IEEE Symposium on Foundations of Computer Science (FOCS)*, 2007.
- [Bar01] B. Barak. How to Go Beyond the Black-Box Simulation Barrier. In *FOCS*, pages 106–115, 2001.
- [Bar02] B. Barak. Constant-Round Coin-Tossing with a Man in the Middle or Realizing the Shared Random String Model. In *Proceedings of the 43rd Symposium on Foundations of Computer Science (FOCS-02)*, pages 345–355, Los Alamitos, Nov. 16–19 2002. IEEE COMPUTER SOCIETY.
- [BMG08] B. Barak and M. Mahmoody-Ghidary. Merkle Puzzles are Optimal. *Arxiv preprint arXiv:0801.3669*, 2008. Preliminary version of this paper. Version 1 contained a bug that is fixed in this version.
- [BF86] Barany and Furedi. Computing the Volume is Difficult. In *STOC: ACM Symposium on Theory of Computing (STOC)*, 1986.
- [BBC⁺01] Beals, Buhrman, Cleve, Mosca, and de Wolf. Quantum Lower Bounds by Polynomials. *JACM: Journal of the ACM*, 48, 2001.
- [BF90] Beaver and Feigenbaum. Hiding Instances in Multioracle Queries. In *STACS: Annual Symposium on Theoretical Aspects of Computer Science*, 1990.
- [BIN97] Bellare, Impagliazzo, and Naor. Does Parallel Repetition Lower the Error in *Computationally Sound* Protocols? In *FOCS: IEEE Symposium on Foundations of Computer Science (FOCS)*, 1997.
- [BGP00] M. Bellare, O. Goldreich, and E. Petrank. Uniform Generation of NP-Witnesses Using an NP-Oracle. *Inf. Comput*, 163(2):510–526, 2000.
- [BDCGL92] Ben-David, Chor, Goldreich, and Luby. On the Theory of Average Case Complexity. *JCSS: Journal of Computer and System Sciences*, 44, 1992.
- [BSW05] Benjamini, Schramm, and Wilson. Balanced Boolean Functions that can be Evaluated so that Every Input Bit is Unlikely to be Read. In *STOC: ACM Symposium on Theory of Computing (STOC)*, 2005.
- [BBBV97] Bennett, Bernstein, Brassard, and Vazirani. Strengths and Weaknesses of Quantum Computing. *SICOMP: SIAM Journal on Computing*, 26, 1997.
- [BGI08] E. Biham, Y. J. Goren, and Y. Ishai. Basing Weak Public-Key Cryptography on Strong One-Way Functions. In R. Canetti, editor, *TCC*, volume 4948 of *Lecture Notes in Computer Science*, pages 55–72. Springer, 2008.

- [BFJ⁺94] A. Blum, M. Furst, J. Jackson, M. Kearns, Y. Mansour, and S. Rudich. Weakly Learning DNF and Characterizing Statistical Query Learning Using Fourier Analysis. In *Proc. of Twenty-sixth ACM Symposium on Theory of Computing*, 1994.
- [BT03] Bogdanov and Trevisan. On Worst-Case to Average-Case Reductions for NP Problems. In *FOCS: IEEE Symposium on Foundations of Computer Science (FOCS)*, 2003.
- [BT06] Bogdanov and Trevisan. Average-Case Complexity. In *Foundations and Trends in Theoretical Computer Science, Now Publishers or World Scientific*, volume 2. 2006.
- [BCY89] Brassard, Crepeau, and Yung. Everything in NP Can Be Argued in Perfect Zero-Knowledge in a Bounded Number of Rounds. In *EUROCRYPT: Advances in Cryptology: Proceedings of EUROCRYPT*, 1989.
- [BCG⁺95] Bshouty, Cleve, Gavaldà, Kannan, and Tamon. Oracles and Queries that are Sufficient for Exact Learning. In *ECCCTR: Electronic Colloquium on Computational Complexity, technical reports*, 1995.
- [BdW02] Buhrman and de Wolf. Complexity Measures and Decision Tree Complexity: A Survey. *TCS: Theoretical Computer Science*, 288, 2002.
- [Can74] G. F. L. P. Cantor. Über eine Eigenschaft des Inbegriffs aller reellen algebraischen Zahlen. *Journal f. reine und angew. Math*, 77:258–262, 1874.
- [Coo71] S. A. Cook. The complexity of theorem proving procedures. In *stoc71*, pages 151–158, 1971.
- [CS98] R. Cramer and V. Shoup. A Practical Public Key Cryptosystem Provably Secure against Adaptive Chosen Ciphertext Attack. In *Crypto '98*, pages 13–25, 1998. LNCS No. 1462.
- [Dan51] G. B. Dantzig. Maximization of a Linear Function of Variables Subject to Linear Inequalities. In T. C. Koopmans, editor, *Activity Analysis of Production and Allocation*, pages 339–347. John Wiley, New York, 1951.
- [DH76] W. Diffie and M. E. Hellman. New Directions in Cryptography. *IEEE Transactions on Information Theory*, 22(5):644–654, 1976.
- [DF88] Dyer and Frieze. On the Complexity of Computing the Volume of a Polyhedron. *SICOMP: SIAM Journal on Computing*, 17, 1988.
- [DFK91] Dyer, Frieze, and Kannan. A Random Polynomial Time Algorithm for Approximating the Volume of Convex Bodies. *JACM: Journal of the ACM*, 38, 1991.
- [FS89] Feige and Shamir. Zero Knowledge Proofs of Knowledge in Two Rounds. In *CRYPTO: Proceedings of Crypto*, 1989.
- [FFS87] U. Feige, A. Fiat, and A. Shamir. Zero Knowledge Proofs of Identity. In *stoc87*, pages 210–217, 1987.

- [FF93] Feigenbaum and Fortnow. Random-Self-Reducibility of Complete Sets. *SICOMP: SIAM Journal on Computing*, 22, 1993.
- [Fis02] Fischlin. On the Impossibility of Constructing Non-interactive Statistically-Secret Protocols from Any Trapdoor One-Way Function. In *CTRSA: CT-RSA, The Cryptographers' Track at RSA Conference, LNCS*, 2002.
- [For87] L. Fortnow. The Complexity of Perfect Zero Knowledge. In *stoc87*, pages 204–209, 1987.
- [For00] L. Fortnow. Diagonalization. *Bulletin of the EATCS*, 71:102–113, 2000.
- [GGK03] R. Gennaro, Y. Gertner, and J. Katz. Lower Bounds on the Efficiency of Encryption and Digital Signature Schemes. In *Proc. 35th STOC*. ACM, 2003.
- [GT00] R. Gennaro and L. Trevisan. Lower Bounds on the Efficiency of Generic Cryptographic Constructions. In *Proc. 41st FOCS*, pages 305–313. IEEE, 2000.
- [GPV07] C. Gentry, C. Peikert, and V. Vaikuntanathan. Trapdoors for Hard Lattices and New Cryptographic Constructions. *Electronic Colloquium on Computational Complexity (ECCC)*, 14(133), 2007.
- [GKM⁺00] Y. Gertner, S. Kannan, T. Malkin, O. Reingold, and M. Viswanathan. The relationship between public key encryption and oblivious transfer. In *Proc. 41st FOCS*, pages 325–338. IEEE, 2000.
- [GMR01] Y. Gertner, T. Malkin, and O. Reingold. On the Impossibility of Basing Trapdoor Functions on Trapdoor Predicates. In *FOCS*, pages 126–135, 2001.
- [GW95] Goemans and Williamson. Improved Approximation Algorithms for Maximum Cut and Satisfiability Problems Using Semidefinite Programming. *JACM: Journal of the ACM*, 42, 1995.
- [GG98] Goldreich and Goldwasser. On the Limits of Non-Approximability of Lattice Problems. In *STOC: ACM Symposium on Theory of Computing (STOC)*, 1998.
- [GGH96] Goldreich, Goldwasser, and Halevi. Collision-Free Hashing from Lattice Problems. In *ECCCTR: Electronic Colloquium on Computational Complexity, technical reports*, 1996.
- [GGM86] Goldreich, Goldwasser, and Micali. How to Construct Random Functions. *JACM: Journal of the ACM*, 33, 1986.
- [GK96] Goldreich and Krawczyk. On the Composition of Zero-Knowledge Proof Systems. *SICOMP: SIAM Journal on Computing*, 25, 1996.
- [GNW95] Goldreich, Nisan, and Wigderson. On Yao's XOR-Lemma. In *ECCCTR: Electronic Colloquium on Computational Complexity, technical reports*, 1995.

- [Gol99] O. Goldreich. *Modern cryptography, probabilistic proofs and pseudorandomness*, volume 17 of *Algorithms and Combinatorics*. Springer-Verlag, Berlin-Heidelberg-New York-Barcelona-Budapest-Hong Kong-London-Mailand-Paris-Santa Clara-Singapur-Tokyo, 1999.
- [Gol01] O. Goldreich. *Foundations of Cryptography: Basic Tools*. Cambridge University Press, 2001. Earlier version available on <http://www.wisdom.weizmann.ac.il/~oded/frag.html>.
- [Gol04] O. Goldreich. *Foundations of Cryptography: Basic Applications*. Cambridge University Press, 2004.
- [Gol08] O. Goldreich. *Computational Complexity: A Conceptual Perspective*. Cambridge University Press, 1 edition, April 2008.
- [GGH96] O. Goldreich, S. Goldwasser, and S. Halevi. Public-Key Cryptosystems from Lattice Reduction Problems. Technical Report MIT/LCS/TR-703, Massachusetts Institute of Technology, Nov. 1996.
- [GL89] O. Goldreich and L. A. Levin. A Hard-Core Predicate for all One-Way Functions. In *STOC*, pages 25–32. ACM, 1989.
- [GMW86] O. Goldreich, S. Micali, and A. Wigderson. Proofs that Yield Nothing but their Validity and a Methodology of Cryptographic Design. In *focs86*, 1986.
- [GMR89] Goldwasser, Micali, and Rackoff. The Knowledge Complexity of Interactive Proof Systems. *SICOMP: SIAM Journal on Computing*, 18, 1989.
- [GM84] S. Goldwasser and S. Micali. Probabilistic Encryption. *JCSS*, 28(2):270–299, Apr. 1984.
- [GS86] S. Goldwasser and M. Sipser. Private coins versus public coins in interactive proof systems. In *STOC '86: Proceedings of the eighteenth annual ACM symposium on Theory of computing*, pages 59–68, New York, NY, USA, 1986. ACM Press.
- [GLS81] M. Grötschel, L. Lovasz, and A. Schrijver. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1:169–197, 1981.
- [Gro96] L. K. Grover. A Fast Quantum Mechanical Algorithm for Database Search. In *STOC*, pages 212–219, 1996.
- [HHR07] Haitner, Hoch, Reingold, and Segev. Finding Collisions in Interactive Protocols – A Tight Lower Bound on the Round Complexity of Statistically-Hiding Commitments. In *ECCCTR: Electronic Colloquium on Computational Complexity, technical reports*, 2007.
- [HH09] I. Haitner and T. Holenstein. On the (Im)Possibility of Key Dependent Encryption. In O. Reingold, editor, *TCC*, volume 5444 of *Lecture Notes in Computer Science*, pages 202–219. Springer, 2009.

- [HS65] J. Hartmanis and R. E. Stearns. On the Computational Complexity of Algorithms. *Trans. Amer. Math. Soc.*, 117:285–306, 1965.
- [HK05] Horvitz and Katz. Bounds on the Efficiency of “Black-Box” Commitment Schemes. In *ICALP: Annual International Colloquium on Automata, Languages and Programming*, 2005.
- [HR04] Hsiao and Reyzin. Finding Collisions on a Public Road, or Do Secure Hash Functions Need Secret Coins? In *CRYPTO: Proceedings of Crypto*, 2004.
- [IKW02] Impagliazzo, Kabanets, and Wigderson. In Search of an Easy Witness: Exponential Time vs. Probabilistic Polynomial Time. *JCSS: Journal of Computer and System Sciences*, 65, 2002.
- [IL89] Impagliazzo and Luby. One-way Functions are Essential for Complexity Based Cryptography. In *FOCS: IEEE Symposium on Foundations of Computer Science (FOCS)*, 1989.
- [IR88] Impagliazzo and Rudich. Limits on the Provable Consequences of One-Way Permutations. In *CRYPTO: Proceedings of Crypto*, 1988.
- [IL90] R. Impagliazzo and L. A. Levin. No Better Ways to Generate Hard NP Instances than Picking Uniformly at Random. In *FOCS*, volume II, pages 812–821. IEEE, 1990.
- [IW97] R. Impagliazzo and A. Wigderson. $\mathbf{P} = \mathbf{BPP}$ if \mathbf{E} Requires Exponential Circuits: Derandomizing the XOR Lemma. In *Proc. 29th STOC*, pages 220–229. ACM, 1997.
- [Jac97] J. C. Jackson. An Efficient Membership-Query Algorithm for Learning DNF with Respect to the Uniform Distribution. *J. Comput. Syst. Sci.*, 55(3):414–440, 1997.
- [JS88] Jerrum and Sinclair. Conductance and the Rapid Mixing Property for Markov Chains: The Approximation of the Permanent Resolved. In *STOC: ACM Symposium on Theory of Computing (STOC)*, 1988.
- [JVV86] Jerrum, Valiant, and Vazirani. Random Generation of Combinatorial Structures from a Uniform Distribution. *TCS: Theoretical Computer Science*, 43, 1986.
- [KI04] Kabanets and Impagliazzo. Derandomizing Polynomial Identity Tests Means Proving Circuit Lower Bounds. *CMPCMPL: Computational Complexity*, 13, 2004.
- [Kab02] V. Kabanets. Derandomization: a brief overview. *Bulletin of the EATCS*, 76:88–103, 2002.
- [KL80] Karp and Lipton. Some Connections Between Nonuniform and Uniform Complexity Classes. In *STOC: ACM Symposium on Theory of Computing (STOC)*, 1980.
- [KP82] Karp and Papadimitriou. On Linear Characterizations of Combinatorial Optimization Problems. *SICOMP: SIAM Journal on Computing*, 11, 1982.
- [KL07] J. Katz and Y. Lindell. *Introduction to Modern Cryptography (Chapman & Hall/CRC Cryptography and Network Security Series)*. Chapman & Hall/CRC, 2007.

- [KV94] M. J. Kearns and U. V. Vazirani. *An Introduction to Computational Learning Theory*. MIT press, Cambridge, Massachusetts, 1994.
- [KS06] J. A. Kelner and D. A. Spielman. A randomized polynomial-time simplex algorithm for linear programming. In ACM, editor, *Proceedings of the Thirty-Eighth Annual ACM Symposium on Theory of Computing 2006, Seattle, WA, USA, May 21–23, 2006*, pages 51–60, pub-ACM:adr, 2006. ACM Press.
- [Kha79] L. G. Khachian. A polynomial time algorithm for linear programming. *Soviet Math. Dokl.*, 20:191–194, 1979.
- [KST99] J. H. Kim, D. R. Simon, and P. Tetali. Limits on the Efficiency of One-Way Permutation-Based Hash Functions. In *FOCS*, pages 535–542, 1999.
- [KM93] Kushilevitz and Mansour. Learning Decision Trees Using the Fourier Spectrum. *SICOMP: SIAM Journal on Computing*, 22, 1993.
- [Lev73] Levin. Universal Sequential Search Problems. *PINFTRANS: Problems of Information Transmission (translated from Problemy Peredachi Informatsii (Russian))*, 9, 1973.
- [LTW05] Lin, Trevisan, and Wee. On Hardness Amplification of One-Way Functions. In *Theory of Cryptography Conference (TCC), LNCS*, volume 2, 2005.
- [Lip91] R. J. Lipton. New directions in testing. In *Distributed Computing and Cryptography*, volume 2 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 191–202. American Mathematics Society, 1991.
- [LR85] Luby and Rackoff. How to Construct Pseudo-random Permutations from Pseudo-random Functions. In *CRYPTO: Proceedings of Crypto*, 1985.
- [LFKN92] Lund, Fortnow, Karloff, and Nisan. Algebraic Methods for Interactive Proof Systems. *JACM: Journal of the ACM*, 39, 1992.
- [M. 88] M. Grötschel and L. Lovász and A. Schrijver. *Geometric algorithms and combinatorial optimization*. Springer-Verlag, 1988.
- [Mer82] Merkle. Secure Communications over Insecure Channels. In *SIMMONS: Secure Communications and Asymmetric Cryptosystems*, 1982.
- [MS72] Meyer and Stockmeyer. The Equivalence Problem for Regular Expressions with Squaring Requires Exponential Space. In *FOCS: IEEE Symposium on Foundations of Computer Science (FOCS)*, 1972.
- [MS73] Meyer and Stockmeyer. Word Problems Requiring Exponential Time. In *STOC: ACM Symposium on Theory of Computing (STOC)*, 1973.
- [Mic01] Micciancio. The Shortest Vector in a Lattice is Hard to Approximate to within Some Constant. *SICOMP: SIAM Journal on Computing*, 30, 2001.
- [Mil76] G. L. Miller. Riemann’s Hypothesis and Tests for Primality. *Journal of Computer and System Sciences*, 13(3):300–317, 1976.

- [NY90] M. Naor and M. Yung. Public-Key Cryptosystems Provably Secure against Chosen Ciphertext Attacks. In *In Proc. of the 22nd STOC*, pages 427–437. ACM Press, 1990.
- [Nis91] N. Nisan. CREW PRAMs and Decision Trees. *SIAM Journal on Computing*, 20(6):999–1007, Dec. 1991.
- [NS94] N. Nisan and M. Szegedy. On the Degree of Boolean Functions as Real Polynomials. *Computational Complexity*, 4(4):301–313, 1994.
- [NW88] N. Nisan and A. Wigderson. Hardness vs Randomness. *J. Comput. Syst. Sci.*, 49(2):149–167, Oct. 1994. Preliminary version in FOCS’ 88.
- [OW93] R. Ostrovsky and A. Wigderson. Nontrivial zero-knowledge implies one-way functions. In *Proceedings of the 2nd ISTCS*, pages 3–17, 1993.
- [Pap94] C. H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.
- [PS98] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Dover Pub., 1998.
- [Pei08] C. Peikert. Public-Key Cryptosystems from the Worst-Case Shortest Vector Problem. *Electronic Colloquium on Computational Complexity (ECCC)*, 15(100), 2008.
- [Rab79] M. O. Rabin. Digitalized Signatures and Public-Key Functions as Intractable as Factorization. Technical Report MIT/LCS/TR-212, Massachusetts Institute of Technology, Jan. 1979.
- [Rab80] M. O. Rabin. Probabilistic Algorithm for Testing Primality. *Journal Of Number Theory*, 12:128–138, 1980.
- [Rac88] C. Rackoff. A Basic Theory of Public and Private Cryptosystems. In *Proc. Advances in Cryptology – CRYPTO ’88*, pages 249–255, 1988.
- [RR97] Razborov and Rudich. Natural Proofs. *JCSS: Journal of Computer and System Sciences*, 55, 1997.
- [RTV04] O. Reingold, L. Trevisan, and S. P. Vadhan. Notions of Reducibility between Cryptographic Primitives. In M. Naor, editor, *TCC*, volume 2951 of *Lecture Notes in Computer Science*, pages 1–20. Springer, 2004.
- [Sch90] Schapire. The Strength of Weak Learnability. *MACHLEARN: Machine Learning*, 5, 1990.
- [Sch03] A. Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*, volume 24 of *Algorithms and Combinatorics*. Springer, 2003.
- [Sha02] R. Shaltiel. Recent developments in extractors. *Bulletin of the European Association for Theoretical Computer Science*, 2002. Available from <http://www.wisodm.weizmann.ac.il/~ronens>.
- [SU01] R. Shaltiel and C. Umans. Simple extractors for all min-entropies and a new pseudo-random generator. In *Proc. 42nd FOCS*, pages 648–657. IEEE, 2001.

- [Sha92] A. Shamir. IP = PSPACE. *J. ACM*, 39:869–877, 1992.
- [Sim98] Simon. Finding Collisions on a One-Way Street: Can Secure Hash Functions be Based on General Assumptions? In *EUROCRYPT: Advances in Cryptology: Proceedings of EUROCRYPT*, 1998.
- [Sni85] M. Snir. Lower Bounds on Probabilistic Linear Decision Trees. *Theoretical Computer Science*, 38(1):69–82, May 1985.
- [SS77] R. Solovay and V. Strassen. A Fast Monte-Carlo Test for Primality. *SIAM J. Comput.*, 6(1):84–85, 1977.
- [ST04] Spielman and Teng. Smoothed Analysis of Algorithms: Why the Simplex Algorithm Usually Takes Polynomial Time. *JACM: Journal of the ACM*, 51, 2004.
- [STV01] Sudan, Trevisan, and Vadhan. Pseudorandom Generators without the XOR Lemma. *JCSS: Journal of Computer and System Sciences*, 62, 2001.
- [STV98] M. Sudan, L. Trevisan, and S. Vadhan. Pseudorandom generators without the XOR Lemma. ECCC Report TR98-074, 1998. <http://www.eccc.uni-trier.de/eccc/>.
- [Tre01] Trevisan. Extractors and Pseudorandom Generators. *JACM: Journal of the ACM*, 48, 2001.
- [Tur36] A. M. Turing. On Computable Numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, 42(2):230–265, 1936.
- [Uma03] C. Umans. Pseudo-random generators for all hardnesses. *J. Comput. Syst. Sci.*, 67(2):419–440, 2003.
- [Yao82] A. C. Yao. Theory and Applications of Trapdoor Functions. In *Proc. 23rd FOCS*, pages 80–91. IEEE, 1982.
- [Yap83] C. K. Yap. Some Consequences of Non-Uniform Conditions on Uniform Classes. *Theoretical Computer Science*, 26(3):287–300, Oct. 1983.