Black Boxes, Incorporated

Mohammad Mahmoody Avi Wigderson

July 26, 2012

Abstract

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* (also called an *oracle*) 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 computationally hard or even not computable. From a programming perspective, this viewpoint is convenient when solving a problem using a subroutine for f that someone else has implemented and we are given 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 computational problem g using an oracle to f may constitute a *reduction* from g to f, showing that computing (or solving) g "is not much harder than" computing f (a statement of relevance even if we have no idea how complex computing either of these functions are). The most powerful example for black-box reductions is of course in **NP**-completeness proofs.

The abstraction of information access via black boxes and the roles this abstraction plays in different settings is varied and rather subtle in some occasions. E.g., it might be that the black box encodes the input itself, or that it is a powerful subroutine which helps to solve a computational problem. Sometimes, the mere observation that a particular algorithm is using one of its subroutines in a black-box manner has led to unexpected new applications of the original proof techniques. One can also think of "levels" of black-boxness. That is, in some cases one might deal with a subroutine and ignore how it is implemented in general but still assume some restrictions on its complexity. Furthermore, in situations such as cryptographic constructions, different elements of the system (e.g., the implementation and/or the security reductions) may or may not be of a black-box form and various settings lead to different possibilities or impossibilities.

In what follows we will 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.

Contents

1 Decision Trees **3**
- 1.1 The Black-Box Power of Nondeterminism, Randomness, and Quantum Computation . . . 3

2 Effect of P = NP on Polynomial-Time Hierarchy **5**
- 2.1 Non-Black-Box Use of a SAT Solver . . . 6
- 2.2 Black-Box Use of a SAT Solver . . . 7

3 Learning: Opening Black Boxes **8**
- 3.1 Learning SAT-Solvers . . . 8
- 3.2 Goldreich-Levin Lemma . . . 11
- 3.3 Applications of Goldreich-Levin Lemma to Learning . . . 12

4 Limits of Black-Box Diagonalization **13**

5 Black-Box Ideas in Geometry **15**
- 5.1 Ellipsoid Method . . . 15
- 5.2 Volume Computation . . . 17

6 Derandomization **18**
- 6.1 Derandomizing **BPP** Almost as a Black-Box . . . 19
- 6.2 Reusing Black-Boxes: Trevisan's Extractor . . . 21

7 Black-Box Worst-Case to Average-Case Reductions for NP **22**
- 7.1 Positive Side . . . 23
- 7.2 Negative Side . . . 24

8 Limits of Black-Box Techniques in Cryptography **27**
- 8.1 Black-Box Separations . . . 28
- 8.2 Zero Knowledge Proofs and Non-Black-Box Constructions . . . 30
  - 8.2.1 A Non-Black-Box Identification Scheme . . . 31
  - 8.2.2 Non-Black-Box Proofs of Security . . . 31

9 References **32**

Notation. We will use the following basic notation. Let $B\colon \{0,1\}^* \mapsto \{0,1\}^*$ 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 *B-relativized* world is one in which every algorithm has access to such an oracle B. This can be generalized to classes of algorithms $\mathbf{A}$ and to classes of functions $\mathbf{B}$. By PPT or "efficient" we denote a probabilistic polynomial time algorithm.

Complexity Classes. By default, we will not define the complexity classes and refer the reader to the great books in complexity theory [Gol08, AB07, Pap94] (particularly Appendix A [Gol08]) for the formal definitions.

1 Decision Trees

The Role of Black-Box Notion. 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) function[1] $f\colon \{0,1\}^n \mapsto \{0,1\}$. The algorithm knows f, and should compute $f(x)$. To gather information on the input $x = (x_1, x_2, \ldots, x_n)$ it adaptively queries a sequence of indices in $[n] = \{1, 2, \ldots, n\}$, 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 for each input length n. As simple as this model of computation is, while quite a bit is known about it, many problems regarding it are still open. Here we focus on comparing the power of various standard models of computation (i.e., nondeterministic, randomized, quantum) when the access to the input is black-box.

1.1 The Black-Box Power of Nondeterminism, Randomness, and Quantum Computation

A basic question about the power of this computational model is to now if there is any gain in using resources like randomness, quantum superposition, or even nondeterminism in order to compute $f(x)$ through a decision tree (where the only restriction is the black-box access to the input bits). We We refer the interested reader to the survey of Buhrman and de Wolf [BdW02] for a comprehensive study and describe some of the main results here.

First we formally define the variants of decision tree complexities of a function.

Definition 1.1. For a Boolean function $f\colon \{0,1\}^n \mapsto \{0,1\}$, $D(f)$ denotes the (deterministic) *decision tree complexity* (DTC) of f which is the minimum number of (possibly adaptive) queries to the bits of an arbitrary input $x = (x_1, x_2, \ldots, x_n)$ by which one can determine $f(x)$ where each query is simply an index $i \in [n]$ and the answer is x_i. Using nondeterminism, by $N(f)$ we denote the *nondeterministic* decision tree complexity (NTC) of f which is the smallest number of bits such that whenever if $f(x) = 1$, then there exist $N(f)$ bits of x that witness that $f(x) = 1$.

Depicting the decision tree algorithm as a binary tree whose internal nodes are labeled by queries, edges by answers to the queries, and leaves by outputs, $D(f)$ is the depth of the

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

shallowest tree for deciding f. It is clear that for every function f on n bits it holds that $D(f) \leq n$, and there are functions for which we have $D(f) = n$ (e.g., the parity function: $f(x_1, \ldots, x_n) = \sum_i x_i \mod 2$), but it is also easy to find functions f that depend on all bits (and so $D(f) = n$), yet their NTC is only $N(f) = O(\log n)$.

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 a randomized 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 decision-tree complexity (RTC) of f is the minimum depth over all probabilistic decision trees computing f with probability $\geq 2/3$. Clearly we have $R(f) \leq D(f)$ for every function f, but there are known examples where the gap can be almost quadratic; namely it holds that $D(f) = n$ but $R(f) = O(\sqrt{n \log n})$ [BSW05]. Interestingly, this gap can never be super-polynomial, and for every function f it holds that $D(f) \leq O(R(f)^3)$ [Sni85, Nis91].

For quantum algorithms, it stops making sense to view these algorithms as trees because of the inherent cancelations of computation paths due to "interference". In particular, quantum algorithms iteratively make queries to the oracle, just like their deterministic and probabilistic counterparts. However, a query in the quantum setting is actually a linear superposition $\sum_i \alpha_i |i\rangle|0\rangle$ accessing each input bit i with complex *amplitude* α_i, *all in parallel*. The answer to such a query is the vector $\sum_i \alpha_i |i\rangle|x_i\rangle$. The amplitudes $\{\alpha_i\}$ 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 asking the queries, the algorithm can perform arbitrary unitary operations on the amplitudes. After all the queries are asked, the algorithm performs a "measurement" and outputs its (random) outcome, which we again demand to be correct with probability at least $2/3$.[2]

It is not hard to see that quantum decision trees can simulate the 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 it holds that $D(f) \leq Q(f)^c$.*

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.

Ideas Used in the Proof of Theorem 1.2. The proof has three steps. The first step shows that if a Boolean function f is computed with complexity $Q(f) \leq d_1$ by a quantum decision tree, 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 it would hold that $|p(x) - f(x)| \leq 1/3$. 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., $q(x) =$

[2] See [BBC+01] for a formal definition of quantum decision trees.

$f(x)$, for every Boolean vector 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].[3]

Theorem 1.2 shows that $D(f), R(f)$, and $Q(f)$ are indeed all polynomially related and it remains to compare them with $N(f)$.

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 $D(\mathsf{OR}) = n$, and that even randomization does not provide a major advantage $R(\mathsf{OR}) = \Omega(n)$. One of the few examples where quantum computation is *provably* faster is Grover's search algorithm [Gro96] establishing a quadratic speed-up over the probabilistic algorithms $Q(\mathsf{OR}) = O(\sqrt{n})$. Moreover, it is known [BBBV97] that Grover's search algorithm is in fact tight $Q(\mathsf{OR}) = \Theta(\sqrt{n})$. However, it clearly holds that $N(\mathsf{OR}) = 1$ which shows a super-polynomial gap between the nondeterministic DTC and the other modes of computation: deterministic, randomized, and quantum.

Non-Black-Box Input Access for OR Problem. Finally, let us have a look at the "non-black-box" version of the OR problem discussed above. That is, we assume that we are given a Boolean formula φ on k variables which defines an input x of length $n = 2^k$ to the OR problem by letting $x_i = \varphi(i)$ for $1 \leq i \leq n$. To solve the OR problem over $x = (x_1, \ldots, x_n)$ one shall decide whether φ is a satisfiable formula or not which is the same as the Boolean satisfiability problem SAT. The lower-bound of $\Omega(n) = \Omega(2^k)$ over the running time of deterministic, probabilistic, and quantum decision trees in solving SAT does not apply to the new non-black-box setting anymore, and in the question suddenly becomes as hard as the renowned **P** vs. **NP** question. In particular, there is a nondeterministic algorithm that decides the OR problem in linear time over the size of φ, but whether a similar thing can be done using randomized or even quantum algorithms is widely open. We note, however, that since x_i is efficiently computable from φ, one can also build a polynomial-sized quantum circuit that computes φ, and therefore Grover's search method gives a quantum algorithm for solving the Boolean satisfiability problem SAT (i.e., the non-black-box version of the OR problem) which is quadratically faster than the best known deterministic or probabilistic one.

2 Effect of P = NP on Polynomial-Time Hierarchy

The question of weather $\mathbf{P} \neq \mathbf{NP}$ or not is a central problem in complexity theory. The importance of this question comes from the fact that whether $\mathbf{P} = \mathbf{NP}$ or $\mathbf{P} \neq \mathbf{NP}$ would lead to drastically different "worlds" regarding what is efficiently computable and what tasks can be performed "securely" (i.e., whether cryptography is possible).

[3] To obtain a better constant c, Beals et al. [BBC+01] proves Theorem 1.2 also by using "block sensitivity" (another complexity measure) of the Boolean function f as the middle parameter, rather than using the minimum degree of a polynomial representing f; this proof obtains $c \approx 6$.

The Role of Black-Box Notion. In this section, we will see the different consequences of these two scenarios: **(1)** Having an actual (non-black-box) access to an efficient algorithm that solves an **NP**-complete problem like SAT versus **(2)** the case that such efficient algorithm does not exist, but we still have oracle access to a magical black-box which solves SAT. In particular, we study the different implications of these two scenarios regarding the polynomial-time hierarchy. We shall also note that there is a third possibility as well: **(3)** that there exists an efficient SAT solver algorithm A, but we are only given a black-box access to A. In Section 3.1 we show how to reduce this case to case **(1)** by reconstructing such algorithm A only by oracle access to it!

We first formally define the polynomial-time hierarchy. Recall that the class **NP** is the set of all languages L for which there is a polynomial-time membership verifier V such that $x \in L$ iff $\exists\, y \in \{0,1\}^{\mathrm{poly}(|x|)}, V(x,y) = 1$. Adding more quantifiers before the final verification leads to larger complexity classes.

Definition 2.1 ([MS72])**.** The class $\mathbf{\Sigma}_k$ contains all the languages L for which there is a polynomial-time verifier V and a polynomial $p = \mathrm{poly}(n)$ such that $x \in L$ iff

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

The class $\mathbf{\Pi}_k$ contains the complements of the languages in $\mathbf{\Sigma}_k$, and can be described similarly by using k quantifiers starting with a $\forall$.

By definition it holds that $\mathbf{\Pi}_i \subseteq \mathbf{\Sigma}_{i+1}$ and $\mathbf{\Sigma}_i \subseteq \mathbf{\Pi}_{i+1}$, the polynomial-time hierarchy **PH**, as a single complexity class is defined as the union of these levels: $\mathbf{PH} = \bigcup_i \mathbf{\Sigma}_i = \bigcup_i \mathbf{\Pi}_i$. An interesting interpretation of the class **PH** comes from the complexity of finite transparent games (see below). Meyer and Stockmeyer [MS73] gave another characterization of the levels of the polynomial-time hierarchy (see Section 3.2.2 of [Gol08] for a proof.)

Theorem 2.2 ([MS73])**.** $\mathbf{NP}^{\mathbf{\Sigma}_i} = \mathbf{\Sigma}_{i+1}$.

2.1 Non-Black-Box Use of a SAT Solver

Which languages can be decided efficiently if SAT $\in$ **P** (i.e., **P** = **NP**)? As it turns out, in that case, all of the languages in **PH** could be solved in polynomial time by using the SAT-solving algorithm A in a heavily non-black-box manner.

Theorem 2.3. *If* SAT $\in$ **P**, *then* **PH** = **P**.

Proof. The proof is by eliminating the quantifiers of a language in **PH** one by one. Let A be an efficient algorithm for SAT. A can be used to solve any language $L \in \mathbf{NP}$ using the Cook-Levin reduction [Coo71, Lev73]. Namely if $L = \{x \mid \exists\, y, V_L(x,y) = 1\}$, then one can efficiently map any input x to an instance $s(x)$ which is satisfiable iff $\exists\, y, V(x,y) = 1$. Note that this mapping does not use the code of A in a non-black-box way, but rather uses the code of $V_L(\cdot,\cdot)$. So having oracle access to such algorithm A we can decide any language in **NP** and **coNP** efficiently.

Now suppose $L \in \mathbf{\Sigma}_i$ and L is defined as

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

By the above method one can substitute $[Q\, y_k\, V(x, y_1, \ldots, y_k) = 1]$ by $[U_k(x, y_1, \ldots, y_{k-1}) = 1]$ for some polynomial-time algorithm U_k which depends on A. Repeating this procedure $i-1$ more times eliminates all the quantifiers and give us an algorithm U_1 which decides L in polynomial-time.[4] □

In the second iteration of the proof above when we want to remove the second quantifier, we will need to use the code of A (invoked inside U_k) in a non-black-box way to feed it into the Cook-Levin reduction.

2.2 Black-Box Use of a SAT Solver

As we described above, the proof of Theorem 2.3 is non-black-box. But is this non-black-box use inherent? Suppose we are given a SAT-solver only in a black-box. In Section 3.1 we will see that there is in fact an *efficient* way to learn and reconstruct such algorithm A in a universal way by just asking polynomially many oracle queries to A! In this section we deal with the (more plausible) case that $\mathbf{P} \neq \mathbf{NP}$ and thus no such efficient algorithm exists.

Note that even if $\mathbf{P} \neq \mathbf{NP}$, it still makes sense to ask which problems can be solved efficiently given *oracle* access to a SAT solver. In particular, can we still solve all of the languages in the polynomial-time hierarchy using such oracle (i.e., $\mathbf{PH} \subseteq \mathbf{P}^{\mathbf{NP}}$)? Of course the answer is yes if $\mathbf{PH} = \mathbf{P}$ holds already (in which case one can simply ignore the oracle and solves **PH** efficiently from the scratch), so here we also assume that the levels of **PH** do not collapse and ask: what is the set of languages in $\mathbf{P}^{\mathbf{NP}}$.

Interestingly, Theorem 2.2 answers our question indirectly! The reason is that by Theorem 2.2 having *even* nondeterministic access to a complete problem in $\mathbf{\Sigma}_i$ provides a computational power which is still limited to the languages in $\mathbf{\Sigma}_{i+1}$. In particular it holds that $\mathbf{P}^{\mathbf{NP}} \subseteq \mathbf{NP}^{\mathbf{NP}} = \mathbf{\Sigma}_2$. Therefore, by having oracle access to **NP** we can not go even beyond the second level of **PH**!

Open Question. Does $\mathbf{P} = \mathbf{NP}$ imply the collapse of any other complexity class other than the polynomial-time hierarchy?

A Game-Theoretic Perspective on PH. Let A and B be two players, playing a finite game of only two moves where, in order, A and B choose their moves y_1, y_2 from the set $U = \{0,1\}^{\text{poly}(n)}$ and n is a parameter controlling the "size" and description of the game. Also suppose the predicate $V(y_1, y_2)$ is equal to one iff A is the winner. In this game, the player A has a winning strategy if and only if there is a move y_1 for A such that for all moves y_2 of B the result is a win for A: $V(y_1, y_2) = 1$. By extending the set of games and

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

allowing the number of moves in the game to be a constant k, the player A would have a winning strategy in an extended version of the game iff there exist a move y_1 for A such that for every move y_2 for B there is a move y_3 for A ... such that at the end it holds that $V(y_1, \ldots, y_k) = 1$. Now suppose we generalize such games one more step and let the game to have initial position described by some $x \in \{0,1\}^n$. It is easy to see that for any such game G, if the final verdict for the game $V(x, y_1, \ldots, y_k)$ is computable in polynomial time, by definition, the set W_A^G of all x's describing a winning initial position for A builds up a language in $\mathbf{\Sigma}_k$. The complement set $\overline{W_A^G} = W_B^G$ which contains the initial positions over which B has a winning strategy will be a language in $\mathbf{\Pi}_k$. In fact, the reverse connection from games to the polynomial-time hierarchy holds as well. Any language $L \in \mathbf{\Sigma}_k$ (resp. $L \in \mathbf{\Pi}_k$) can be interpreted as the initial positions of a k-move game over which the first party (resp. the second party) has a winning strategy. There is no inherent reason for only considering games with $k = O(1)$ number of moves and studying the analogy only with the polynomial-time hierarchy. In fact one can think of a similar connection between the games with polynomially many moves and the languages solvable in polynomial space **PSPACE**. For further discussion on this connection Section 19.2 of [Pap94].

3 Learning: Opening Black Boxes

At a high level, many results in learning theory can be described as follows. There is a black box hiding a function $f\colon \{0,1\}^n \mapsto \{0,1\}$ inside. We know a priori that f comes from a family of functions F, and we want to know more about f by asking certain form of queries to the black box. We might want to **(1)** reconstruct f completely, **(2)** find out some properties or parameters of f (e.g., its average, or its large Fourier coefficients), or **(3)** approximate it (under some definition of what it means to approximate), or . For a great introduction to the field of learning theory we refer the reader to [KV94].

In this section we first describe a perhaps surprising result due to Bshouty et al. [BCG+95] which states that using black-box access to an efficient SAT-solver we can reconstruct any such algorithm (or an equivalent alternative) efficiently! This falls into the first category described above. Then we will see the Goldreich-Levin (GL) lemma [GL89] which was originally proposed in the context of cryptographic applications but later turned out to be very useful also in other areas of computer science such as computational learning theory. The GL lemma shows how to learn the "large" Fourier coefficients of a function $f\colon \{0,1\}^n \mapsto \{0,1\}$ in probabilistic polynomial-time by only invoking black-box queries to f. This task falls into the second category above. Finally we describe two applications of GL lemma to learn decision trees and DNF formulas, which falls into the third category above.

3.1 Learning SAT-Solvers

Although the common believe is that there is no family of polynomial-size circuits solving SAT (otherwise the polynomial-time hierarchy collapses to the second level [KL80]), but it is still meaningful to ask the following question. Suppose someone claims to have an efficient

algorithm for SAT and they only provide answers to SAT queries rather than revealing the algorithm itself. Can we extract any useful information about this algorithm by asking only oracle queries to it? More ambitiously, can we "reconstruct" the whole algorithm completely? As Bshouty et al. [BCG+95] showed, it is in fact possible to learn any SAT-solving algorithm! The result in [BCG+95] is proved in a more general context and the theorem below can be derived from the results proved there.

Theorem 3.1 ([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 $|C_n| \leq s(n)$. Also, suppose that* SAT *instances are encoded in a way which allows padding and so C_m can be used to solve* SAT *on instances x where $|x| < m$ as well. Then there is a universal learning algorithm which given 1^n as input gets only black-box access to the family $\mathcal{C}$, runs in (expected) time $\mathrm{poly}(s(n))$, and it outputs a circuit D_n which solves* SAT *on all instances of length n (i.e., $D_n(x) = C_n(x)$ for all $x \in \{0,1\}^n$).*

In the rest of this subsection we prove Theorem 3.1.

For any two circuits C, D with the same input length n we define $C \approx D$ if and for all $x \in \{0,1\}^n$ it holds that $C(x) = D(x)$. The high level structure of the proof is as follows.

1. Suppose the learning algorithm is granted an extra (imaginary) feature which allows it to ask "equivalence queries" rather than regular SAT queries to C_i's. Namely, in this model, we can query a circuit D with input length n, and the oracle answers back whether it holds that $C_n \approx D$ or not. In case $C_n \not\approx D$, the answer will include a "counter example" x such that $D(x) \neq C_n(x)$. The first step is to show that in the model of equivalence-queries there is a way to reconstruct a SAT solver in (expected) polynomial time with the help of an **NP** oracle.

2. The second step uses C_m (for large enough m) both as an **NP** oracle, and also simulate the equivalence queries for circuits of input length n by using only regular SAT queries (which in turn will be asked to C.

Now we go over more details of the outline above. At the beginning of the learning process, we know that there exists at least one circuit D of size at most $s(n)$ such that $D \approx C_n$. Since any circuit of size s can be represented with $O(s^2)$ bits, the goal of the learning algorithm can be seen as aiming to find a needle $D \approx C_n$ in a haystack of size at most $2^{O(s(n)^2)}$.

Suppose we have asked i equivalence queries $D_1, \dots, D_i$ so far and has received i counter examples $x_1, \dots, x_i$. Let $\mathcal{F}_i$ be the set of all circuits of size at most $s(n)$ which are consistent with our current knowledge about C_n based on the counter examples $\{x_1, \dots, x_i\}$, namely:

$$\mathcal{F}_i = \{D \mid D \text{ is a Boolean circuit with } n \text{ inputs}, |D| \leq s(n), D(x_j) \neq D_j(x_j) \text{ for all } j \in [i]\}.$$

The next equivalence query $D = D_{i+1}$ will be chosen in such a way that either $D \approx C_n$, or otherwise, any possible counter example x_{i+1} to D is a "shrinks" the set $\mathcal{F}_i$ by a constant factor: $|\mathcal{F}_{i+1}| \leq \frac{3}{4} \cdot |\mathcal{F}_i|$. By asking such queries, only after $\log_{4/3}(2^{O(s(n)^2)}) = O(s(n)^2)$ many steps we can find a $D \approx C_n$.

Call x an unbiased query for $\mathcal{F}_i$ if at most $3/4$ fraction of $\mathcal{F}_i$ agree about the satisfiability of a fixed query x, namely if $\delta_0(x) = \frac{|\{E \in \mathcal{F}_i | E(x)=0\}|}{|\mathcal{F}_i|}$ and $\delta_1(x) = 1 - \delta_0(x)$, then both of $\delta_0(x), \delta_1(x)$ are at most $3/4$. We call x a biased query if it is not unbiased.

It is easy to see that if x is an unbiased query and is returned as a counter example to any equivalence query D, it can only be a shrinking one, because matter what circuit D is queried, if x is returned as a counter example it will eliminate $1/4$ of $\mathcal{F}_i$. Now suppose x is a biased query. This means that if an equivalence query circuit D is chosen *at random* from $\mathcal{F}_i$, then with probability at least $3/4$, it holds that $3/4$ fraction of $\mathcal{F}_i$ agrees with D on the input x, in which case x will also be a shrinking counter example.

We would like to ask our equivalence query D in a way that *all* queries x can only be a shrinking example. For that purpose, we can sample uniformly at random from $\mathcal{F}_i$ polynomially many circuits $E_1, E_2, \ldots, E_k$, and take D to be a circuit which given the input x, outputs $\text{maj}_{j \in [k]} E_j(x)$. By choose $k = 10n$ and using Chernoff bound, any x will have a chance of at most 2^{-2n} to be a "non-shrinking" counter example. Thus by a union bound, with probability at least $1 - 2^{-n}$ over the choice of D, any returned counter example will be shrinking! But how can we sample $E \xleftarrow{\$} \mathcal{F}_i$ at random? Note that we can efficiently verify membership in $\mathcal{F}_i$. A fundamental result due to [JVV86, BGP00][5] states that efficient membership verification in a set implies that one can efficiently sample uniform members from it with the help of an **NP** oracle. But here we do have such oracle for free: the circuits of the family $\mathcal{C}$!

Changing the Equivalence Queries to SAT Queries. The only remaining step of the proof is to get rid of the equivalence queries and run the learning algorithm using regular SAT queries. Interestingly, by using the Cook-Levin reduction and the self-reducibility of SAT we can always solve any query of the form: "*Is there a counter example to circuit D as a* SAT *solver for input length n*" using an **NP** oracle! Any counter example x of the form $D(x) = 0, \mathsf{SAT}(x) = 1$ (i.e., x is a satisfiable formula, but D says it is not) can be "witnessed" by providing x and its satisfying assignment. Thus the existence of such x can be by using the Cook-Levin reduction and a single query to the **NP** oracle. Using a search-to-decision reduction one can also find the actual counter example x. It is also possible that $D(x) = 1$ but $\mathsf{SAT}(x) = 0$. This case is more subtle since it is not clear right away how such x can be efficiently witnessed. But even in this case by using the self-reducibility of SAT and substituting the variables inside x with values from $\{0, 1\}$ one by one, one can find inputs three inputs x', x'_0, x'_1 where x'_0 and x'_1 are, in order, the formulas obtained from x by substituting one of its variables v with values $v = 0$ and $v = 1$ and it holds that $D(x') = 1, D(x'_0) = 0$ and $D(x'_1) = 0$. By using the Cook-Levin reduction and the **NP** oracle one can also confirm the existence of the the set $\{x', x'_0, x'_1\}$ which witnesses that that D is not a SAT solver. Finally, again by a search-to-decision reduction and using the **NP** oracle one can find out which one of $\{x', x'_0, x'_1\}$ is the actual counter example for D.

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

3.2 Goldreich-Levin Lemma

In this section we describe a fundamental result of Goldreich and Levin [GL89], known as the Goldreich-Levin (GL) lemma, which originally was discovered in the context of cryptographic applications. Using GL lemma Goldreich and Levin showed that the existence of one-way permutations implies the existence of pseudorandom generators through an extremely efficient reduction. It turned out that GL is in fact quite useful also in other areas of computer science such as learning theory, and this is the subject of our focus here. At a high level, this results tells us how to find large Fourier coefficients of a Boolean function $f\colon \{0,1\}^n \mapsto \{0,1\}$ which is only accessible as a black-box. First we will describe the Goldreich-Levin (GL) lemma formally, then we go over some of its many. It simplifies the presentation if one thinks of Boolean functions to choose their output from $\{\pm 1\}$ (-1 representing 1 and $+1$ representing 0).

Lemma 3.2 ([GL89])**.** *There exist a probabilistic algorithm GL which given oracle access to any $f\colon \{0,1\}^n \to \{\pm 1\}$ runs in time $(\frac{n}{\varepsilon})^{O(1)}$ and finds (with probability 0.99) a set F (of size $(\frac{n}{\varepsilon})^{O(1)}$) that is a superset of $F_\varepsilon = \{a \mid \hat{f}(a) \geq \varepsilon\}$ – the set of Fourier coefficients of f which are at least ε.*[6]

Note that $|F_\varepsilon|$ can never be too large, because by the Parseval's equality we have $\sum_a \hat{f}(a)^2 = \mathbb{E}_x[f(x)^2] = 1$ and so $|F| \leq \frac{1}{\varepsilon^2}$. Some of the main applications of the GL lemma are as follows.

List Decoding of Hadamard Code. 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 $H_m(a) = (-1)^{\langle m,a\rangle}$ where $\langle m,a\rangle = \sum_i m_i a_i \mod 2$. Suppose we have received a "corrupted" Hadamard codeword $f \in \{\pm 1\}^{2^n}$ of the message $m \in \{0,1\}^n$ and we know that less than half of its bits might be different from a correct encoding: $\Pr_{x \xleftarrow{\$} \{0,1\}^n}[f(x) = H_m(x)] \geq \frac{1+\varepsilon}{2}$. Since any two Hadamard codewords differ in exactly 2^{n-1} coordinates (i.e., its normalized distance is $1/2$), unique decoding is not possible when the fraction of errors is more than $1/4$. But as we will see it is still possible to "list decode" a corrupted Hadamard codeword even when more than $1/4$ fraction of bits are corrupted, and this will be done using GL lemma! It can be seen that $\Pr_{x \xleftarrow{\$} \{0,1\}^n}[f(x) = H_m(x)] \geq \frac{1+\varepsilon}{2}$ if and only if $\hat{f}(m) \geq \varepsilon$. So for $\varepsilon = 1/\operatorname{poly}(n)$ the GL algorithm produces a set $\{m_1, \ldots, m_{\operatorname{poly}(n)}\}$ such that $m \in \{0,1\}^n$ is the message we are looking for.

Pseudorandom Generators. Here we rely on the list-decoding application GL lemma and also describe the original main application of GL lemma presented by [GL89]. Suppose $p\colon \{0,1\}^n \mapsto \{0,1\}^n$ is a one-way permutation. Namely p can be computed efficiently, yet for all efficient algorithms A it holds that $\Pr[f(A(y)) = y\colon y = f(x), x \xleftarrow{\$} \{0,1\}^n]$ is negligible. It is easy to see that the function $q\colon \{0,1\}^{2n} \mapsto \{0,1\}^{2n}$ where $q(x,y) = (p(x),y)$ for $x,y \in \{0,1\}^n$ is also a one-way permutation. We claim that given $q(x,y) = (p(x),y)$

[6] The set F might also contain some b where $\frac{99}{100} \cdot \varepsilon < \hat{f}(b) < \varepsilon$.

for $(x,y) \xleftarrow{\$} \{0,1\}^{2n}$, it is infeasible to guess $\langle x,y\rangle$ correctly with probability at least $1/2 + 1/\operatorname{poly}(n)$. This implies that $G(x,y) = (p(x), y, \langle x,y\rangle)$ is a *pseudorandom generator*, because the output of $G(U_2)$ is *not* random, yet it is indistinguishable from U_{2n+1}. The reason is that if given $q(x,y) = (p(x),y)$ for $(x,y) \xleftarrow{\$} \{0,1\}^{2n}$, an algorithm A can guess $\langle x,y\rangle$ correctly with probability at least $\geq 1/2 + \varepsilon$, then by an average argument with probability at least $\varepsilon/2$ over $x \xleftarrow{\$} \{0,1\}^n$, if we fix x, for at least $\frac{1+\varepsilon}{2}$ fraction of y's the same algorithm A is able to compute $\langle x,y\rangle$ when given $(p(x),y)$.[7] The latter means that for such "good" x's the algorithm A provides us with an implicit access to a corrupted Hadamard encoding of x with error fraction $\leq \frac{1-\varepsilon}{2}$. But as we just saw, in this case whenever $\varepsilon = 1/\operatorname{poly}(n)$, one can use GL lemma to find a list of $(\frac{n}{\varepsilon})^{O(1)} = \operatorname{poly}(n)$ candidates one of which is the right x we are looking for to invert $q(x,y)$. Thus, we are able to guess (x,y) correctly with probability at least $\frac{\varepsilon}{2} \cdot \frac{1}{\operatorname{poly}(n)} \geq \frac{1}{\operatorname{poly}(n)}$ which contradicts the fact that q was one-way.

3.3 Applications of Goldreich-Levin Lemma to Learning

In this section, we will see two applications of GL lemma to learn theory. A basic problem in learning theory is to "approximate" the value of a function f, which is given only as a black-box, under some distribution D. In this setting, the task of the learner is to query f only as an oracle at $\operatorname{poly}(n)$ many points and at the end (with high probability $1 - 1/\operatorname{poly}(n)$) output some function g such that $\Pr_{x \xleftarrow{\$} D}[f(x) \neq g(x)] \leq \varepsilon$ for a small value of $\varepsilon = 1/\operatorname{poly}(n)$. The most natural case for D is to learn f under the uniform $D = U_n$ distribution, which is the case in our following examples. If we do not know anything about f, we can never predict the next answer, even if we ask many queries to it. Thus, a learning problem in this setting is defined based on a family of functions $\mathcal{F}$ such that we already know that f belongs to $\mathcal{F}$.

Decision Trees. Recall Definition 1.1 for decision trees. Any Boolean function f has a decision tree of depth n and a total of 2^n leaves. Here we are interested in learning decision trees that have a polynomial size representation. Perhaps surprisingly, a the class of polynomial-sized decision trees can be learned efficiently under the uniform distribution. This result due to Kushilevitz and Mansour [KM93] turned out to be influential in using Fourier analytic methods in learning theory.

Theorem 3.3 ([KM93])**.** *Polynomial-size decision trees can be learned under 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 it holds that $L_1(f) \leq m$ where $L_1(f) = \sum_z |\hat{f}(z)|$. Then it is shown by straightforward calculation that for the function f_δ defined as $f_\delta(x) = \sum_{|\hat{f}(z)| > \delta/L_1} \hat{f}(z) \cdot \chi_z(x)$ it holds that f_δ approximates f in L_2 norm: $\mathbb{E}_x[(f(x) - g(x))^2] \leq \delta$. Thus, in order to approximate f in L_2 norm it is sufficient to

[7] Here we implicitly assumed that the algorithm is deterministic, although a similar argument works for randomized algorithms as well.

find all of its Fourier coefficients which are at least δ/m. Using GL lemma we can do this in time $(\frac{mn}{\delta})^{O(1)}$ and find a function f_δ which approximates f: $\mathbb{E}_x[(f(x)-f_\delta(x))^2] \leq \delta$. Finally, since $f(x) \in \{\pm 1\}$, the function $sign(f_\delta)$ (which outputs $+1$ or -1) is what we are looking for because: $\Pr_x[sign(g)(x) \neq f(x)] \leq \mathbb{E}_x[(f(x)-g(x))^2] \leq \delta$.

□

DNF Formulas. We next describe a result of Jackson [Jac97] which also uses GL lemma and Fourier techniques to show that the class of disjunctive normal form formulas can be learned efficiently under uniform distribution.

Theorem 3.4 ([Jac97])**.** *Polynomial-size DNF formulas can be learned under uniform distribution.*

Proof Sketch. A result of [BFJ+94] shows that any DNF formula 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 $1/2$ under the uniform distribution.[8]

Started by the breakthrough of Schapire [Sch90], we now have powerful tools for "boosting" a weak learning algorithm to a strong one which approximates the function with probability close to one. 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. □

4 Limits of Black-Box Diagonalization

Although after decades of research in complexity theory the question of **P** =?**NP** is still open, there has been moments when new techniques raised hopes for finally resolving this question. Introducing the diagonalization method to complexity theory was one of this moments since (as we will see below in Theorem 4.1) using this method one can separate the class of computational problems solvable with more resources such as time and space. We refer the reader to the great survey by Fortnow [For00] for many examples in which diagonalization can be used to separate complexity classes.

However, as we will see shortly, it turned out that diagonalization, at least in its basic form used in results such as Theorem 4.1 below come short in resolving **P** vs. **NP** question due to the *relativization* side-effect. Namely, these proofs hold even when the relevant complexity classes are defined relative to some arbitrary oracle O.

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

Role of Black-Box Notion. Here, the role of the notion of a black-box is indirect. That is, the main point is that some proof techniques are oblivious to the presence of any subroutine provided for free as a black box, and this makes these proof techniques limited. In other words, the relativization issue makes techniques such as diagonalization to be "too good" for the purpose of resolving certain questions such as **P** vs. **NP**.

The power of the diagonalization technique can be manifested by a classical result of Hartmanis and Stearns [HS65] who showed that more time leads to more computational power.

Theorem 4.1 ([HS65])**.** *For any integer* $k \geq 1$, $\mathbf{DTIME}(n^{k+1}) \not\subseteq \mathbf{DTIME}(n^k)$.

The proof of Theorem 4.1 is reminiscent of Cantor's proof for uncountability of real numbers [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 emulates another Turing machine M given as input over another given input x. For simplicity let $k = 2$. We define a language L such that $L \in \mathbf{DTIME}(n^3)$ but $L \notin \mathbf{DTIME}(n^2)$. Given an input x, one can always interpret it as an encoding of a Turing machine M_x and use the Universal Turing machine to run $M_x(\cdot)$ over x itself for n^2 steps. If the emulation leads to $M_x(x) = 1$, we would define $x \notin L$, and otherwise (even if the computation $M_x(x)$ does not end in n^2 steps) we define $x \in L$. This way of defining L guarantees that $L \notin \mathbf{DTIME}(n^2)$. On the other hand, by using an efficient implementation of the Universal Turing machine (e.g., see [AB] Section 1.2) we derive that $L \in \mathbf{DTIME}(n^3)$. □

Theorem 4.1, also known as the time hierarchy theorem, has an analogous version which deals with space-complexity as follows: $\mathbf{SPACE}(n^{k+1}) \not\subseteq \mathbf{SPACE}(n^k)$. Moreover Cook [Coo72] showed that allowing nondeterministic algorithms to run in longer time has the same effect: $\mathbf{NTIME}(n^{k+1}) \not\subseteq \mathbf{NTIME}(n^k)$. Thus it was very natural to pursue the same approach for showing that nondeterminism itself as a resource (compared to being deterministic) leads to more computational power, hence proving $\mathbf{NP} \not\subseteq \mathbf{P}$. In the following we will see why such a technique is not indeed powerful enough to resolve the **P** vs. **NP** question.

The proof of Theorem 4.1 relies on two facts: **((1))** one can represent a Turing machine M as a bit strings and give them as inputs to another (Universal) machine, and **((2))** that one can universally emulate the execution of M over another given input x with a small amount of overhead in the time. Both of these statements also hold in a more general scenario in which M is allowed call an oracle $\mathcal{O}$ in its execution, assuming that the Universal Turing machine can also make queries to $\mathcal{O}$. Thus the proof of Theorem 4.1 "relativizes" and we in indeed have $\mathbf{DTIME}(n^{k+1})^{\mathcal{O}} \not\subseteq \mathbf{DTIME}(n^k)^{\mathcal{O}}$ for every oracle $\mathcal{O}$. The following theorem by Baker, Gill, and Solovay [BGS75] shows that no proof technique (e.g., the diagonalization method used in the proof of Theorem 4.1) is not able to resolve the **P** vs. **NP** question.

Theorem 4.2 ([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. One possibility is to take $B = \mathbf{EXP}$ (i.e., a complete problem in the class of languages decidable in exponential time. Any language in $\mathbf{EXP}$ is also 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 do both of the following in exponential time: **(1)** go over all nondeterministic choices of the nondeterministic machine, and **(2)** find the answers to its oracle queries. Thus it holds that $\mathbf{EXP} \subseteq \mathbf{P}^{\mathbf{EXP}} \subseteq \mathbf{NP}^{\mathbf{EXP}} \subseteq \mathbf{EXP}$, and so: $\mathbf{P}^{\mathbf{EXP}} = \mathbf{NP}^{\mathbf{EXP}}$.

To obtain the desired oracle A such that $\mathbf{P}^A \neq \mathbf{NP}^A$, it is easy to see that for any choice of A the language L_A defined as $L_A = \{x : |x| = n, \exists\, y \in \{0,1\}^{|x|}, A(y) = 1\}$ (i.e., the OR function applied to the truth table of A for length $|x|$) is in $\mathbf{NP}^A$. By diagonalization[9], the oracle A can be chosen in a way that no polynomial-time oracle machine M^A can decide membership in L_A for all x. See Section 3.4 of [AB07] for the full proof. □

We will see more examples that relativization poses limits over some powerful techniques (particularly in cryptography) in Section 8.

5 Black-Box Ideas in Geometry

5.1 Ellipsoid Method

Due to their vast applications in computer science and engineering, the search for efficient optimization methods has always been an active line of research in recent decades. A prominent subarea deals with linear optimization where the input to the problem consists of a bunch of linear constraints and we would like to optimize another linear function module all these restrictions. In fact, many combinatorial problems (e.g., Maximum Flow and Shortest Path) one can be easily reduced to linear optimization problems. In a Linear Programming problem (LP for short) we want to find a vector $x \in \mathbb{R}^n$ maximizing $c^T \cdot x$ under the condition that $Ax \leq b$, for $c \in R^n, A \in \mathbb{R}_{m \times n}, b \in \mathbb{R}^m$. 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 the direction specified by c. We refer the reader to the books [Sch03, PS98] for comprehensive discussion of the subject.

Role of Black-Box Notion. In this section we will see how a simple observation about the fact that a specific optimization method uses one of its subroutines in a black-box way has eventually led to a more general approach that could solving a much bigger class of problems. We will see another example of this phenomenon in Section 6.2.

Simplex Method. The most widely used method for solving LP problems is the Simplex method of Dantzig [Dan51]. The intuition behind the Simplex method is to jump over the vertices of the polytope in a way that in each step we move form a vertex of the polytope to an *adjacent* vertex and this move improves the objective function. Unfortunately the number of vertices for a high dimensional polytope described by polynomially many hyperplanes

[9] Note that here we want to show some limitations of the Diagonalization approach!

could be exponentially large, and in fact one can find instances of the LP problem such that the Simplex method takes exponential time. Despite that, Simplex method it is still an effective method of solving LP problems over the instances that come up in real applications (Spielman and Teng [ST04] explain this phenomenon to some extent).

Subsequent work showed how to use randomization in Simplex method in order to to guarantee polynomial time [KS06], but the first polynomial time algorithm for solving linear programming emerged in the seventies when Khachiyan [Kha79] presented the Ellipsoid method. The reason behind the name *Ellipsoid Method*, is due to the elegant way it uses high dimensional ellipsoids to give an iteratively improving bound on the region that the optimal point lies in. First, we describe the Ellipsoid method at a high level and then will see how a more careful look at it reveals that certain parts of this algorithm are black-box and this feature leads to a generalization of this method to solve a wider class of linear programs.

Ellipsoid Method. Using binary search (and adding more linear constraints) solving an LP problem can be reduced to only checking the feasibility of the given LP (i.e., checking if there exits x for which $Ax \leq b$ without caring about the optimality of x). So w.l.o.g we would only want to know if a given polytope described by (A, b) is empty or not. The first step is to find an ellipsoid E_1 which contains the polytope[10]. 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 checking that $Ax_j \leq b$. If x was a feasible point, we are done. Otherwise it means that for some i it holds that $a_i^T \cdot x_j > b_i$ 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 having the center of E_j on its border) has a smaller volume than E_j by some noticeable factor depending on the dimension n. The good news is that this smaller ellipsoid can in fact be found in $\text{poly}(n)$ time. Therefore if we take E_{j+1} to be an ellipsoid with smallest volume which contains $E_j \cap X_j$, after polynomially many number of iterations, we either find a feasible point, or can make sure that it does not exist. (Here we are implicitly relying on the fact that the volume of a nonempty polytope represented by an LP in n dimension is always at least $1/\text{poly}(n)$.)

Separation Oracle Suffices. In [KP82] Karp and Papadimitriou pointed out that Ellipsoid method is in fact able to solve a wider class of convex optimization problems that just linear programming with a polynomial number of linear restrictions. In particular, all we want is a *separation oracle* which is a form of black-box access to the input convex set defined as follows.

Definition 5.1 (Separation Oracle for Convex Bodies)**.** For a convex body B and given a query $x \in \mathbb{R}^n$, if $x \in B$, return YES, otherwise return a hyperplane separating x from B.

[10] We assume here that the polytope is bounded.

Karp and Papadimitriou also presented examples of LP problem with *exponentially* many constraints for which the separation oracle can be implemented efficiently (with an implicit knowledge of the constraints that define the feasible set), and therefore Ellipsoid method could in fact be used to solve these problems efficiently!

Semidefinite Programming Using Black-Box Ellipsoid Method. In a semidefinite programming (SDP for short) problem, the vector of variables x is in fact a matrix and comes 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, there are also some input linear constraints and the objective function is also linear (similar to the case of LP). It is easy to see that any LP problem can be reduced to an instance of SDP problem, so solving SDP is a stronger optimization tool. The first point about SDP is that the space of PSD_n matrices is convex, and so the feasible set of an SDP is also a convex set. In addition, the separation oracle can be implemented efficiently for the feasible domain described by SDP instances. Therefore, the general approach of [KP82] to use Ellipsoid method through a separation oracle can be used to obtain a polynomial-time algorithm for solving SDP problems [GLS81]. In a breakthrough, Goemans and Williamson [GW95] illustrated the power of SDP by designing improved approximation algorithms for the Max Cut and Max 2-SAT problems. Ever since SDP has been a major tool in designing approximation algorithms.

5.2 Volume Computation

The Role of Black-Box Notion. Similar to Section 1, here we also deal with comparing the power of various computational models – deterministic vs. randomized, approximation vs. exact – when the access to the input is somehow black-box. This time we focus on a particular geometric problem: computing the volume of convex bodies. The input to the VC problem is a convex body $B \subset R^n$ for a polynomially large dimension n.

We will study two variants of VC problem: black-box and non-black-box. In the non-black-box version we assume that the convex body B is defined through a polynomial number of half-spaces (i.e., B is a polytope) and these half-spaces are given explicitly as part of the input. In the black-box variant of VC we assume that we have an initial point $x \in B$ and we are also provided with a separation oracle (see Definition 5.1). We will compare the hardness of these two variants of the VC problem between the randomized and deterministic models of computation. Note that since one can always implement a separation oracle efficiently for a given input polytope B, the non-black-box VC can be reduced to black-box VC and thus black-box VC is comparably a harder problem.

Solving VC Exactly. First consider the non-black-box VC problem. This problem can be solved efficiently in dimensions (e.g., $n = 2$), but unfortunately, this problem is in fact #**P**-hard when n can be polynomially large [DF88]. The black-box VC problem, will not be easier than non-black-box VC, and thus solving the problem exactly, requires solving #**P** in polynomial time!

The result of [DF88] leads us to the question of: what happens if we relax the problem and only look for an approximation of the volume? Barany and Furedi [BF86] showed that even approximation is not possible for the case of black-box VC.

Theorem 5.2 ([BF86]). *There is no polynomial-time deterministic $o((\frac{n}{\log n})^n)$-approximation algorithm for the black-box* VC *problem.*

The result is almost tight because an n^n-approximation could be computed efficiently [M. 88].

Solving Black-Box VC Randomly and Approximately. Dyer, Frieze, and Kannan [DFK91] showed that using *both* of these relaxations: **(1)** using randomness and **(2)** aiming for approximation, the situation changes drastically and VC can be approximated within an arbitrarily small polynomial factor!

Theorem 5.3 ([DFK91]). *There is a randomized algorithm A which given ε and black-box access to the body B (through a separation oracle), runs in time $(\frac{n}{\varepsilon})^{O(1)}$ and outputs v such that $\mathrm{VOL}(B) \leq v \leq (1+\varepsilon)\,\mathrm{VOL}(B)$ with probability 0.99.*

Here we only outline the structure the proof of Theorem 5.3 and mention the main ideas.

Let $0 < c < 1$ be a parameter that we choose later. Let L_c be an n-dimensional lattice defined with the base $B_c = \{(c, 0, \ldots, 0), (0, c, 0, \ldots, 0), \ldots, (0, \ldots, 0, c)\}$. For each convex body B, $|B \cap L_c|$ counts the number of L_c points inside B and for small enough c, $|B \cap L_c| \cdot c^n$ gives a good approximation of $\mathrm{VOL}(B)$. The first idea to solve the problem lies in the connection between counting and sampling. Jerrum, Valiant, and Vazirani [JVV86] showed that in verity of settings (including ours) the tasks of "counting" and "uniform-sampling" are are essentially equivalent.

Therefore, it is sufficient to show how to sample an (approximately) uniform lattice L_c point inside B. A way to sample approximately uniform points from $L_c \cap B$ is to define a graph over L_c, do a random walk of sufficient length, and choose the ending point. We define two nodes of $B \cap L_c$ connected if they have distance c. Jerrum and Sinclair [JS88] showed that a random walk over such graph does in fact converge to the uniform distributor in polynomial time. Interestingly, Jerrum and Sinclair developed these tools originally with the goal of approximating the Permanent of Boolean matrices!

6 Derandomization

One of the fundamental questions in complexity theory is to study to what extent truly random bits are inherently useful in various contexts. There are scenarios (e.g., interactive proofs) where using randomness is crucial, but we do not know yet whether randomness adds more power to deciding languages in polynomial time, namely if **BPP** = **P** or not. Although many randomized algorithms eventually were "derandomized" into equivalent deterministic polynomial-time algorithms (e.g., [AKS02] derandomizing the randomized primality tests of [SS77, Rab80]), there are known certain computational problems that still randomness seems to play a central in solving them efficiently (e.g., the polynomial identity testing problem).

Role of Black-Box Notion. In this section the role of the notion of black-box is two-fold.

1. First we describe a line of research in complexity theory which led to (conditional) derandomization of *all* languages of **BPP** *without* looking at the internal computation of the randomized algorithm. This is in contrast to derandomizing algorithms one by one in ad-hoc manners (which can be thought of as a non-black-box approach to derandomization).

2. Then we look at this general (almost) black-box way of derandomizing **BPP** and observe that one of the some components of the derandomizing process are uses as black-box. Trevisan noticed that this black-box feature of the derandomization of **BPP** can itself be used to obtain "randomness extractors". In a way, this second aspect of the notion of black-box in this section is similar to the role black-box notion in Section 5.2.

For more discussion on this subject see [AB] Chapter 20, [Gol08] Chapters 7 and 8, and [Kab02].

6.1 Derandomizing BPP Almost as a Black-Box

In this section we will describe some ideas on how to prove $\mathbf{BPP} = \mathbf{P}$ under believable complexity assumptions. This is done by using a *Pseudo-Random Generator (PRG)* which uses short truly random seeds and expand them so that they can still be used to run any **BPP** algorithm. This approach on constructing PRGs is based on on the existence of "hard functions". The surprising fact about the derandomization of **BPP** is that it is done in an *almost* black-box manner where the only non-black-box use of the algorithm is to know its running time.

Nonuniformly Hard Functions. For a function $f\colon \{0,1\}^n \mapsto \{0,1\}$ define $C(f)$ to be the (asymptotic) size of the smallest circuit computing f (over $\{0,1\}^n$). It is easy to see that a random function f chosen at random from the set of all 2^{2^n} possible functions, with high probability has circuit complexity $C(f) \geq 2^{\Omega(n)}$. This implies that there is a language L with circuit complexity $C(f) \geq 2^{\Omega(n)}$.

Theorem 6.1 ([IW97])**.** *If there is any language $L \in \mathbf{E}$ (i.e., decidable in time $2^{O(n)}$) with circuit complexity $C(L) \geq 2^{\Omega(n)}$, then $\mathbf{BPP} = \mathbf{P}$.*

A natural way to derandomize **BPP** algorithms is to use pseudorandom generators of the form defined as follows.

Definition 6.2. An (ℓ, T)-PRG G is a function $G\colon \{0,1\}^\ell \mapsto \{0,1\}^T$ computable in time $2^{O(\ell)}$ such that for any circuit D of size at most T we have

$$|\Pr[D(U_T) = 1] - \Pr[D(G(U_\ell)) = 1]| \leq 1/T.$$

Since a PRG G (or any other function on ℓ inputs) 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 hope to achieve $\ell = o(\log T)$, but as we will see, under believable complexity assumptions we can achieve $\ell = \Theta(\log T)$

Using PRGs for Derandomization. Suppose G is a $(\Theta(\log T), T)$-PRG. We can use G to derandomize any **BPP** algorithm A as follows: Given an input x (of length $|x| = n$) to A, we can hardwire x into A and use Cook-Levin reduction to get a Boolean circuit C_x of size $\text{poly}(n)$ which takes a (random) string r as input, and for at least $2/3$ fraction of r's it outputs the right answer about the membership of x. If we take $T = \text{poly}(n)$ to be the size of C_x, by the definition of G we have $|\Pr[C_x(U_T) = 1] - \Pr[C_x(G(U_\ell)) = 1]| \leq 1/T < 1/10$, and so by enumerating all pseudorandom 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 $2^\ell \times 2^{O(\ell)} \times T = \text{poly}(n)$.

As we saw, PRGs can be used for derandomizing **BPP** algorithms, but do such objects exist at all? The following theorem is the result of a series of exiting work [BFNW93, NW94, IW97, ACR98, STV98, SU01, Uma03] and shows that any level of (non-uniform) hardness in the class **E** can be transformed into PRGs of related parameters.

Theorem 6.3 ([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.*

In the following we describe some of the ideas in the proof of Theorem 6.3.

Proof Outline. We only outline the proof for the special case of $S(n) = 2^{cn}$ for a constant c, which is indeed the range of parameters needed for proving Theorem 6.1. This outline is based on the original proof of Theorem 6.1 due to [IW97].

Starting from $f_1 \in \mathbf{E}$ with $C(f_1) \geq 2^{\Omega(n)}$ the first step is to construct another function $f_2 \colon \{0,1\}^{O(n)} \mapsto \{0,1\}$ which is still computable in time $2^{O(n)}$ but is $2^{\Omega(n)}$ mildly hard on average. An S mildly hard function h is defined to be one which for every circuit C of size at most S it holds that $\Pr_{x \xleftarrow{\$} \{0,1\}^n}[C(x) = h(x)] \leq 1 - 1/n$. This step is done using polynomial-size error correcting codes with sublinear-time decoding algorithms (which can be obtained based on random self reducibility of multivariate polynomials).

The second step is to take f_2 and construct another function $f_3 \colon \{0,1\}^{O(n)} \mapsto \{0,1\}$ which is again computable in time $2^{O(n)}$, but is $2^{\Omega(n)}$ *strongly* hard, where an S strongly hard function h is one which for every circuit C of size at most S we have $\Pr_{x \xleftarrow{\$} \{0,1\}^n}[C(x) = h(x)] \leq 1/2 + 1/s$. This step is based on a derandomized version of Yao's XOR lemma. Later on, Sudan, Trevisan, and Vadhan [STV98] showed how to do both of these steps in a combined way based on certain list-decidable codes.

Now suppose f_3 is defined on inputs of length $\{0,1\}^{c'n}$ for some constat $c' < 1$. 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. The structure of these chosen subsets is based on some combinatorial design [NW94] that guarantees that the output bits obtained from different subsets are "almost independent". Pseudo-randomness of the output bits is proved by a

reduction to the hardness of f_3 and a hybrid argument. Namely, if there is a distinguisher D implemented with a circuit of size T that breaks the pseudorandomness of the output bits, D can be used to get a circuit of size $T^{O(1)}$ that breaks the strong hardness of f_2. By taking T small enough so that $T^{O(1)}(n) < S(n) = 2^{c \cdot n}$ we get a contradiction.

□

Non-Black-Box Ad-Hoc Derandomization? As mentioned, the known derandomization of **BPP** outlined above is almost black-box in the sense that the only non-black-box information needed about the derandomized algorithm is its running time. Yet, this derandomization is under computational hardness assumptions. One might wonder whether it is possible to achieve unconditional derandomization of **BPP** through a (more) non-black-box technique and eliminate the need of proving complexity lower-bounds for the purpose of derandomization. The work of [KI04, IKW02] showed that proving complexity lower-bounds are indeed *necessary* for derandomizing (for both cases of randomized deterministic **BPP** and randomized nondeterministic **AM** algorithms). Kabanets and Impagliazzo [KI04] showed that derandomizing a specific **BPP** algorithm (i.e., polynomial identity testing) implies that *either* **NEXP** $\not\subseteq$ **P/poly** or $\mathsf{PERM} \notin$ **AlgP**/*poly* (i.e., permanent does not have polynomial-sized arithmetic circuits). The latter lower-bound, in turn, was used to derandomize *all* randomized arithmetic circuits. The work of [IKW02] showed similar results for *nondeterministic* randomized computation (i.e., **AM** protocols). Namely, [IKW02] presented an **AM** protocol whose derandomization would imply lower-bounds against nondeterministic circuit. Interestingly these lower-bounds are indeed sufficient to derandomize all **AM** protocols; thus, when it comes to nondeterministic **AM** computation, derandomizing them all using PRGs in an almost black-box manner and/or derandomizing them one-by-one using ad-hoc methods are basically equivalent!

6.2 Reusing Black-Boxes: Trevisan's Extractor

In this section we describe a result due to Trevisan [Tre01] which relies on the fact that the proof of Theorem 6.3 uses the distinguisher as a black-box (see Remark 6.7) to construct randomness extractors!

Randomness extractors are efficient algorithms that allow one to extract (almost) pure randomness from sources with defected randomness. Here we will focus on the case of single source extractors and for a broad discussion of the subject refer the reader to the survey of Shaltiel [Sha02] and references there.

Definition 6.4. A function $G\colon \{0,1\}^N \times \{0,1\}^\ell \mapsto \{0,1\}^T$ is a (k,ε)-extractor, if for any random variable X over $\{0,1\}^N$ with min-entropy at least $H_\infty(X) \geq k$ the statistical distance between U_T and $f(X, U_\ell)$ is at most ε.

We are interested in *explicit* constructions for f, and would like to achieve T as large as $\approx k$ and ε as small as possible.

In a breakthrough [Tre01] Trevisan showed that PRGs constructed in Theorem 6.3 are indeed randomness extractors! The connection is surprising because extractors are

information-theoretic objects and the more common direction of transferring the ideas in complexity theory is to start from an information theoretic phenomenon and find a computational version of it (e.g., Yao's XOR lemma [Yao82, GNW95]).

Construction 6.5 (Extractors from PRGs)**.** Define the extractor $G\colon \{0,1\}^N \times \{0,1\}^\ell \mapsto \{0,1\}^T$ as follows: given $F \in \{0,1\}^N$ and $x \in \{0,1\}^\ell$, interpret F as truth table of a function $f\colon \{0,1\}^n \mapsto \{0,1\}$ for $2^n = N$ and apply G_f of Theorem 6.3 for hardness $S(n) = 2^{n/10}$ over the given ℓ bit input x to get $T = 2^{\Omega(n)} = N^{\Omega(1)}$ bits of output.

Theorem 6.6 ([Tre01])**.** *The extractor G of construction 6.5 is a $(T^{O(1)}, O(1/T))$-extractor.*

Proof of Theorem 6.6 crucially relies on the following observation.

Remark 6.7 (Proof of Theorem 6.3 is Black-Box)**.** The proof of Theorem 6.3, as a combination of its intermediate reductions, shows that for every function f and every distinguisher D which $1/T$-distinguishes U_T from the constructed $G_f(U_\ell)$, there is a circuit d of size $T^{O(1)}$ which given oracle access to D computes f correctly.

Outline of the Proof of Theorem 6.6. We say that a statistical test (i.e., distinguisher) D breaks f if it $1/T$-distinguishes U_T from $G_f(U_\ell)$. The proof of Theorem 6.3 shows that whenever D breaks f there is a circuit C_f of size $W = T^{O(1)}$ such that C_f^D computes f. Therefore, by counting the total number of circuits of size W, we conclude that for a fixed D, there can not be more than $2^{O(W^2)}$ number of functions f that D breaks. Since C uses the adversary D as a black box, such D could be computationally unbounded and could as well be the (unbounded) distinguisher that achieves the distinguishing advantage equal to the statistical distance between U_T and $G_f(U_\ell)$. Thus if the distribution over f has min-entropy at least $T \times W^2$, with probability at most $O(1/T)$ over the choice of f, D can break f. Therefore whenever D does not break f, it can only ε-distinguish U_T from $G_f(U_\ell)$ for $\varepsilon \leq 1/T$. So D can not distinguish U_T from $G_f(U_\ell)$ by an advantage more than $1/T + O(1/T) = O(1/T)$. □

7 Black-Box Worst-Case to Average-Case Reductions for NP

The existence of one-way functions is necessary for almost all basic cryptographic tasks [IL89, OW93]. A one-way function $f\colon \{0,1\}^n \mapsto \{0,1\}^n$ is a function which is efficiently computable but any polynomial time algorithm has only a negligible chance of finding a preimage for $f(U_n)$. It is easy to see that the existence of one-way functions implies that **NP** is hard *on average*. Since (in addition to being a necessary condition) it is widely believed that $\mathbf{P} \neq \mathbf{NP}$, a natural question is whether one can base the existence of one-way functions only on $\mathbf{P} \neq \mathbf{NP}$ or $\mathbf{NP} \not\subseteq \mathbf{BPP}$. Namely, the hope is to construct an efficiently computable function f whose one-way-ness relies on $\mathsf{SAT} \not\subseteq \mathbf{BPP}$. The most natural way of doing so is through a black-box reduction R that solves SAT with probability, say 2/3,

given oracle access to any adversary A who inverts f with non-negligible probability. As a relaxation one might at least hope to base the average-case hardness of **NP** on its worst-case hardness. Again, the natural way to this "hardness amplification" is through a black-box reduction.

For a general discussion on average-case complexity and in particular its connection to worst-case complexity look at the survey by Bogdanov and Trevisan [BT06].

On the positive side, there are results that show how to construct one-way functions based on worst-case complexity of certain languages, but so far none of these languages are proved to be **NP**-hard. In the following, we first describe the main known ideas on how to base average-case hardness on the worst-case hardness, and then will describe some results that provide negative evidence against the possibility of basing one-way functions (or even the average-case hardness of **NP**) over the worst-case hardness of **NP**.

7.1 Positive Side

A *black-box reduction* from a computational problem A to another computational problem B is a PPT algorithm R which given any input instance x and access to any oracle G who solves B, $R^G(x)$ solves A over x with probability 2/3. A *locally random* reduction, $R^G(x)$ asks its queries from the oracle G according to the uniform distribution over $\{0,1\}^{|x|}$. If A and B are the same, 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 follows that if an algorithm G solves A on $1 - 1/4q$ fraction of the inputs, then we can solve A with probability at least 3/4 on *any* input. Here we describe two major examples of random self reductions.

Random Self Reduction for Discrete Logarithm. An instance of Discrete Logarithm DL problem is parameterized with cyclic group (family) G and a generator g for G. For any input x we want to find the unique i in the range $0 \leq i \leq |G| - 1$ such that $g^i = x$. The random self reduction of DL is as follows. Given x choose $j \xleftarrow{\$} [n]$ at random and take $z = xg^j$. Note that z has uniform distribution over G. Now given $i = \mathsf{DL}(z)$ (i.e $g^i = z$), one can get $\mathsf{DL}(x) = (i - j) \mod n$.

The next example is about one of the successful cases for which we could base average-case hardness of a complexity class (here $\#\mathbf{P}$) over its worst-case hardness. The problem of computing the permanent over finite fields (denoted by PERM) is known to be complete for the class $\#\mathbf{P}$ [Val79]. Thus a random self reduction for PERM implies that solving PERM for a uniformly chosen instance is as hard as solving $\#\mathbf{P}$ in the worst case. Recall that the permanent of a matrix $M_{n\times n} = [m_{i,j}]$ is defined as

$$\sum_{j_1 \in [n]} \sum_{j_2 \in [n]\setminus\{j_1\}} \cdots \sum_{j_n \in [n]\setminus\{j_1,\ldots,j_{n-1}\}} \prod_{i\in[n]} m_{i,j_i}$$

and is a multilinear polynomial of degree n over the n^2 entries of $M_{n\times n} = [m_{i,j}]$. In fact, random self reduction is possible for the more general framework of low-degree polynomials (with PERM as a special case).

Random Self Reduction for Polynomials [BF90, Lip91]. Suppose $p(x_1, x_2, \ldots, x_n)$ is a polynomial on n variables of total degree d, and its coefficients are from the field $\mathbb{F}$ where $|\mathbb{F}| \geq d+2$. Suppose we want to compute $p(\vec{a})$ where $\vec{a} = (a_1, \ldots, a_n) \in \mathbb{F}^n$. Let $\vec{b} \in \mathbb{F}^n$ be an arbitrary vector. For a variable t, $p(\vec{a}+t\vec{b}) = q(t)$ is a univariate polynomial of degree d such that $q(0) = p(\vec{a}+0\vec{b}) = p(\vec{a})$. If we fix any $t \neq 0$ and choose $\vec{b} \xleftarrow{\$} \mathbb{F}^n$ at random, the vector $\vec{a}+t\vec{b}$ will be uniformly distributed over $\mathbb{F}^n$. Based on this idea, the random self reduction proceeds as follows. Given the input $\vec{a}$ choose $t_1, t_2, \ldots, t_{d+1}$ to be $d+1$ different members of $\mathbb{F}$ such that $t_i \neq 0$. Then choose $\vec{b} \xleftarrow{\$} \mathbb{F}^n$ 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 the $d+1$ points $t_1, t_2, \ldots, t_{d+1}$, allows us to find out degree d polynomial $q(t)$ and obtain $q(0)$.

Random self correction has also been used to base the average-case of complexity classes such as **PSPACE** and **EXP** on their worst-case hardness [STV01], but the most interesting such result would be for the class **NP**. Unfortunately, as we will see shortly, such reduction might not exist (or it might be very hard to find). Before going over such results, we mention another line of research that approaches the question of **NP**-hard one-way functions from a different angel.

Lattice-Based Cryptography For a set of n independent vectors $B = \{b_1, \ldots, b_n\}$ 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 the task of computing the (Euclidian) shortest vector in $L(B)$ for a given basis B. The SVP is known to be **NP**-hard (even to approximate within a $\sqrt{2}$ factor [Mic01]).

The celebrated work of Ajtai [Ajt96] showed how to generate hard instances of the SVP problem (while the generator samples the answers along the way) by only assuming that SVP is hard to *approximate in the worst-case* within some poly(n) factor. Further work [AD96, GGH96a, GGH96b, GPV07, Pei08] showed how to get stronger cryptographic primitives (e.g., public-key encryption) from the same assumption. Unfortunately the required approximation factor in these assumptions are larger than the approximation factors over which SVP is known to be **NP**-hard. Goldreich and Goldwasser [GG98] showed that in fact, approximating SVP with large-enough polynomial factors *can not* be **NP** hard unless **NP** = **coNP**.

The proof of all above results are through a black-box reduction. So, one wonders whether there is a more general phenomenon underlying all of them that make them fail in basing average-case hardness of **NP** on its worst-case hardness.

7.2 Negative Side

In this section we will go over some results suggestion that black-box worst-case to average-case reductions for **NP** might not exist. In particular, we will see that certain forms of such reductions imply the collapse of the polynomial-time hierarchy, and even if they exist in their general (black-box) form, they still imply the existence of program checkers for SAT (which is tightly related to long-standing open questions in the complexity of interactive proofs).

The first result we describe here is due to Feigenbaum and Fortnow [FF93] who showed that *non-adaptive* random self reductions for **NP** are unlikely to exist. A non-adaptive reduction prepares all of its queries at first, and then asks them all together.

Theorem 7.1 ([FF93])**.** *There is no non-adaptive random self reduction R for* SAT*, unless the polynomial-time hierarchy collapses to the third level.*

Proof. The idea is to use the reduction R and design a (non uniform) **AM** proof system (i.e., two-message public-coin proof system with a non-uniform verifier) for UNSAT which is known to imply that $\mathbf{PH} = \mathbf{\Sigma}_3$.

Suppose a prover and a verifier are given an input x of length $|x| = n$ and the prover claims that $x \notin \mathsf{SAT}$. For a sampled randomness r of the reduction R, the verifier can ask the prover to provide the answers to the queries of $R(x)$. If the prover sends back the *true* answers, the verifier gets to know the final result $R_r(x)$ which is equal to $\mathsf{SAT}(x)$ with high probability. But there is no guarantee that a malicious prover would follow the protocol. On the other hand, whenever the prover claims that an oracle query is satisfiable, the verifier can ask for a witness for this claim. Thus if the verifier knows in advance how many of the oracle queries of $R_r(x)$ are satisfiable instances, the prover can not cheat in any of the answers! If the verifier is non-uniform, it is possible for him to know the *probability* that a uniformly sampled instance of SAT is satisfiable (note that a_n only depends on the length n). The idea is to run many instances of the reduction R and run some statistical tests over their oracle answers returns by the prover based on the given advice a_n to make sure that the prover is not cheating.

If the reduction $R(x)$ asks $q = \text{poly}(n)$ many queries, the verifier can run $k = q^6$ instances of $R(x)$ in parallel and ask for the answers to all of the queries asked in all these reductions. Using Chernoff bound, the verifier would know that with high probability the total number of YES queries is $\approx a_n \cdot k \cdot q \pm q \cdot k^{2/3} = a_n \cdot q^7 \pm q^5$ (and she would safely abort the protocol if this condition does not hold). Thus the number of queries that the prover can cheat on is bounded by q^5, and so in total, there are only $q^5/q^6 = o(1)$ executions of the reduction $R(x)$ (out of the total q^6 executions) where the prover has returned a wrong answer to one of the queries of the reduction. Therefore for $1 - o(1)$ fraction of the executions of $R(x)$ the verifier gets to know the right answer (or it catches the cheating prover). □

Bogdanov and Trevisan extended the result to reductions that ask their queries with arbitrary distributions, but are still supposed to solve SAT in the worst case (with high probability) whenever the oracle solves $1 - \frac{1}{\text{poly}(n)}$ fraction of the queries of length n correctly. Such reductions are called *self correctors*.

Theorem 7.2 ([BT03])**.** *There is no non-adaptive self corrector R for* SAT*, unless the polynomial-time hierarchy collapses to the third level.*

Proof Sketch. Note that the queries of the reduction R are not necessarily asked according to any particular distribution, so we can not give the average number of YES answers of the queries of $R(x)$ as an advice to the verifier (since this number can depend on the input x).

For simplicity suppose $R(x)$ asks only queries of length n when $|x| = n$. Call a query $y \in \{0,1\}^n$ α-heavy, if the probability that $R(x)$ asks y is at least $\alpha 2^{-n}$. If the reduction $R(x)$ asks $q = \text{poly}(n)$ queries, at most a q/α fraction of $\{0,1\}^n$ could be α-heavy. So for large enough $\alpha = \text{poly}(n)$, if the oracle given to $R(x)$ answers α-heavy queries arbitrarily, the reduction is still supposed to solve SAT correctly (with high probability). The main ideas in [BT03] to take advantage of this fact are the following:

- The verifier can force the prover to tell her which queries of the reduction are heavy and which ones are light. This can be done by using protocols to prove lower-bound and upper-bound on the probability of asking a specific query [GS86, For87, AH91]. Therefore, it is sufficient to only receive correct answers to the light queries. Similarly to the proof of Theorem it is sufficient to find out the expected fraction of YES instances among the light queries of $R(x)$ (this number, in general, cannot be provided as advice, because it depends on x). To do this, we use the following trick.

- 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 planted query (unless with small probability). Therefore, if we know the fraction of YES instances for a uniformly random query (which could be known through non-uniform 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 the prover is essentially forced to answer the light query correctly. By sampling many instances of $R(x)$ and taking one (light) query from each execution and performing the above "hiding protocol" one can estimate the fraction of light queries which are a YES instance up to small additive error. Note that all these estimation protocols are done in *parallel* and still the final protocol has only a constant number of rounds, and by the result of [GS86] it can be converted into a two-message protocol.

□

Extending the result of [BT03] to reductions with more than one round of adaptivity remains an important open question.

Holenstein, Mahmoody, and Xiao [MX10], took a different path and showed that worst-case to average-case reductions in **NP** would imply "program checkers" for SAT [BK89]. A program checker C for a language L, takes as input another algorithm P (claiming to solve L) and an input x, and it outputs in $\{0, 1, \perp\}$ in a way that we have

$$\Pr[C(P,x) \neq L(x) \text{ and } P \text{ solves } L \text{ correctly}] \leq 1/3.$$

In other words, either C finds the true answer about $L(x)$ using P, or it detects a bug in P. Whether SAT is checkable or not has been open for more than three decades and is tightly related to the complexity of the prover in interactive proofs for **coNP** [BFL91]. A *black-box* program checker C uses the program P only as an oracle. The following theorem indicates that even if black-box worst-case to average reduction for **NP** is possible, it would require finding program checkers for **NP** first.

Theorem 7.3. *If there is a black-box worst-case to average-case reduction or* **NP**, *then* **NP** *has a non-uniform program checker. Furthermore, if one can construct a one-way function f through a black-box reduction to the worst-case hardness of* **NP**, *then* **NP** *has a uniform program checker. In both cases the program checker is also black-box.*

Proof Sketch. First we outline the proof for the case of worst-case to average-case reduction or **NP**. Suppose we are given a program P claiming to solve SAT and an input x which we want to know whether $x \in \mathsf{SAT}$ or not. The design of the program checker goes through the following two steps.

- We first "test" whether P is "close" to a SAT solver under the uniform distribution or not. This can be done, given $p_n = \Pr_{x \xleftarrow{\$} U_n}[x \in \mathsf{SAT}]$ as non-uniform advice, and using self-reducibility of SAT. Namely, the tester can sample many instances $x \xleftarrow{\$} U_n$, and for every x such that $P(x) = 1$, it tries to find the satisfying solution to x using P itself. The tester anticipates at least $\approx p_n$ fraction of x to be satisfiable.
- After performing the test, we have the guarantee that P solves almost all of $x \in U_n$ correctly. Now, if R is a worst-case to average-case reduction for SAT, given any input x, the algorithm C can run $R^P(x)$ by using P as an oracle that solves SAT correctly on almost all instances. The final conclusion is that: either P does not pass the test (and C outputs $\perp$), or that we solve $x \in_? \mathsf{SAT}$ correctly with high probability (by definition of R). This is exactly what we want from a program checker!

For the case of constructing **NP**-hard one-way function f, by applying the Cook-Levin reduction to the "puzzle" $y = f(U_n)$ (with "solutions" $\{x \mid f(x) = y\}$) we get a formula Y whose satisfying assignments correspond to the solutions of the puzzle y. The distribution over Y (sampled through sampling $y = f(U_n)$ and applying the Cook-Levin reduction) is still samplable, and moreover, we know that under this distribution the puzzle y should be solvable with probability one. Therefore we can follow the outline above, while there is no need for the non-uniform advice. □

8 Limits of Black-Box Techniques in Cryptography

The theory of cryptography is concerned with formally defining the notions of security and designing secure protocols which meet these definitions. Almost all natural cryptographic tasks (also called *primitives*) imply the existence of one-way functions [IL89] which in turn implies $\mathbf{P} \neq \mathbf{NP}$. Since proving $\mathbf{P} \neq \mathbf{NP}$ seems to be beyond our current techniques in complexity theory [RR97, BGS75, AW08], the current paradigm in foundations of cryptography is to prove the security of cryptographic protocols based on complexity assumptions (which are almost always "lower level" cryptographic primitives such as one-way functions). For a comprehensive discussion on cryptographic primitives and their definitions we refer the reader to the great books [Gol01, Gol04, KL07].

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

On the other hand, we have not been able to derive the existence of other cryptographic primitives (e.g., public-key encryption (PKE)), based on one-way functions, and stronger primitives (e.g., trapdoor permutations) are employed [DH76, Rab79, GM84, NY90, CS98]. Thus the researchers wondered whether such implication can be proved at all. Note that both OWFs and PKEs are widely believed to exist, and so the logical implication is likely to be true. However, it is still possible that our cryptographic tools and techniques come short in proving such statement. Indeed, most of cryptographic constructions are *black-box*. Roughly speaking, a black-box construction of a partitive $\mathcal{Q}$ from another primitive $\mathcal{P}$ consists of two (black-box) reductions: one shows how to implement $\mathcal{Q}$ using any secure implementation of $\mathcal{P}$, and the other one shows that this implementation is indeed secure. The following formalization is due to [RTV04][11].

Definition 8.1. A *black-box construction* of a primitive $\mathcal{Q}$ from another primitive $\mathcal{P}$, consists of two oracle PPTs (Q, S) as follows.

- **Implementation:** Given oracle access to any implementation P of $\mathcal{P}$, Q^P gives an implementation of $\mathcal{Q}$. For example, if $\mathcal{P}$ is one-way permutations, then the oracle P could be any length-preserving permutation over $\{0,1\}^*$.
- **Proof of Security:** For every oracle "adversary" A breaking the security of Q^P (as an implementation of $\mathcal{Q}$), $S^{A,P}$ breaks the security of P (as an implementation of $\mathcal{P}$).

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 in the logical sense.

8.1 Black-Box Separations

In seminal work [IR88] Impagliazzo and Rudich showed that black-box constructions are incapable of proving some cryptographic implications. They showed that secure key agreement (KA) protocols (and thus public-key encryption) can not be based on one-way functions (or even one-way permutations) through a black-box construction. In a KA protocol, Alice and Bob agree on a key by communicating through a public channel while it is unfeasible for any efficient adversary who has access to the public channel to find this key. The result of [IR88] relied on the fact that black-box constructions relativize. Namely, if there is a black-box construction of $\mathcal{Q}$ from $\mathcal{P}$, then it holds that: for every oracle O, if $\mathcal{Q}$ can be realized securely relative to O, then $\mathcal{P}$ could also be realized securely relative to O.

Theorem 8.2 ([IR88])**.** *There exists an oracle relative to which one-way permutations exist but no key-agreement protocol remains sesure. Therefore, no black-box construction of key-agreement protocols from one-way permutations exists.*

[11] What we call black-box construction here, is called "fully black-box" in [RTV04].

Following [IR88] many other black-box separation results were established (e.g., [Sim98, GKM+00, GMR01, Fis02, HR04, HH09]) and even led to results that prove lower-bounds on the efficiency of the implementation reduction in black-box constructions (e.g., [KST99, GT00, GGK03, HK05, LTW05, HHRS07, BMG07, BMG08]).

Proof Outline. It can be shown that a random permutation with high probability is hard to invert by any PPT. The first step in the proof of Theorem 8.2 chooses a random permutation oracle P as a subroutine of the oracle O. Note that by adding more subroutines to O, as long as these subroutines does not depend on the random permutation P, P still remains one-way relative to O.

The second (and in fact the main) step is to add subroutines to O in a way that any KA protocol can be broken efficiently. The main technical part of [IR88] is to show that any KA can be broken by a computationally unbounded adversary who asks polynomially many number of queries to P. Moreover, they show that this adversary can be implemented efficiently if $\mathbf{P} = \mathbf{NP}$. This implies that we can take $O = (P, \mathbf{PSPACE})$, (or any $O = (P, Q)$ such that $\mathbf{P} = \mathbf{NP}$ relative to Q). □

Polynomially-Secure Key Agreement. Now that we know black-box constructions are not able to build *super-polynomially* secure key-agreement protocols from one-way permutations or even random permutations, one can still ask the following question: Is it possible to obtain key-agreement protocols with some polynomial level of security from these primitives? This question was studied by Merkle [Mer78] where he showed how to achieve quadratically secure key-agreement protocols from any random permutation. In this protocol, Alice and Bob agree on a set $[n^2]$ where a random permutation P maps $[n^2]$ to itself. Alice (resp. Bob) queries P on n random random points $x_1, \ldots, x_n$ (resp. $y_1, \ldots, y_n$) and send the answers $P(x_1) = a_1, \ldots, P(x_n) = a_n$ (resp. $P(y_1) = b_1, \ldots, P(y_n) = b_n$) to the other party. With a constant probability, there is a "collision" between the sets $\{x_1, \ldots, x_n\}$ and $\{y_1, \ldots, y_n\}$ and Alice and Bob will notice this collision due to their similar images; they take this collision to be the key. But what can Eve do after observing the messages being sent across? It is easy to see that all Eve can do is to ask the queries in $[n^2]$ one by one till she hits the collision, which on average takes $\approx n^2/2$ queries to P.

A simple modification of the protocol yields a construction only based on random *functions*. Barak and Mahmoody [BMG08] showed that quadratic security for key-agreement is the best one can achieve from random functions. Biham et al. [BGI08] showed how to use exponentially hard one-way functions (instead of random functions) to achieve quadratically secure key-agreement protocols.

Even though most cryptographic constructions are black-box, not all of them fall into this framework. The most prominent examples are secure computation protocols [Yao86] and some applications of zero-knowledge proofs in cryptography. Here we elaborate a bit on the latter type by briefly describing zero-knowledge proofs and an application of it.

8.2 Zero Knowledge Proofs and Non-Black-Box Constructions

The class **NP** contains the languages whose membership can be proved efficiently by providing a witness. If deciding membership in a language $L \in \mathbf{NP}$ is computationally hard, revealing a witness w for $x \in L$ might reveal some "knowledge" that could not be obtained independently. Proving a statement such as $x \in L$ without revealing its proof w seems contradictory at first, but in a seminal work Goldwasser, Micali, and Rackoff [GMR89] introduced the notion of an *interactive proof* that carries *zero knowledge* and showed how to achieve it for nontrivial languages such as quadratic residuacity.

Definition 8.3. A language L has a *zero knowledge* (ZK) proof system if there are interactive algorithms: V the verifier and P the prover such that the following holds.

- **Completeness:** For every $x \in L$ we have $\Pr[\langle V, P\rangle(x) = \mathsf{accept}] = 1$ where by $\langle V, P\rangle(x)$ we mean the the final output of V after the interaction with P.
- **Soundness:** If $x \notin L$, for *any* prover P^*, we have $\Pr[\langle V, P^*\rangle(x) = \mathsf{accept}] \leq 1/3$.
- **Zero Knowledge:** For every potentially cheating verifier V^* there is a simulator S such that for every $x \in L$, the output of $S(x)$ is computationally indistinguishable from $\langle V^*, P\rangle(x)$.

Note that languages in **BPP** trivially have zero knowledge proofs (without interaction).

Sequential and Parallel Repetition. To decrease the soundness error the parties can repeat the protocol many times and at the end the verifier accepts if all the executions lead to accept. There are various ways the repetition can be carried out. In a *sequential repetition* the i^{th} instance of the protocol starts after the $(i-1)^{\text{th}}$ instance ends, while in a *parallel repetition* the j^{th} round of all the executions are done at the same time. It is not hard to see that k sequential or parallel repetition of a proof systems decreases the soundness error down to $\frac{1}{2^k}$. The bad news is that ZK property does not necessarily holds under the parallel repetition (see [Gol01] Proposition 4.5.9).

The work of Goldreich Micali and Wigderson [GMW86] showed that under computational assumptions, *every* provable statement can be proved in zero-knowledge.

Theorem 8.4 ([GMW86])**.** *If one-way functions exist, then any language $L \in \mathbf{NP}$ has a zero-knowledge proof system.*

Proof Outline. Using Cook-Levin reduction, it is enough to give a ZK proof system for any **NP**-complete language. We will do it for the problem of 3-colorable graphs.

Suppose for a moment that there are "digital envelopes with locks" available to the parties. The prover who knows a 3-coloring for the input graph G first permutes the 3 colors randomly and then puts the colors $c_1, \ldots, c_n$ of the vertices into envelopes and gives them to the verifier. Now the verifier chooses a random edge $e \xleftarrow{\$} \{E(G)\}$ from G, which connects some 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 $1/|E(G)|$ (which can be amplified using sequential repetition).

But what about the ZK property? If the prover is truthful, all verifier observes is a pair of random different colors at ends of an edge which can be easily simulated.

It remains to implement the digital envelopes. This primitive is also known as a "commitment scheme" and Naor showed how to construct it based on any pseudorandom generator [Nao02]. The work of [HILL99] showed how to construct pseudorandom generators from any one-way functions, and therefore one-way functions are sufficient for zero-knowledge proofs for all languages in **NP**. □

Now we describe an application of zero-knowledge proofs that uses a one-way function in a *non-black-box* way.

8.2.1 A Non-Black-Box Identification Scheme

An identification (ID) scheme is a protocol with n parties that 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 can be used by them at a later point to prove their identity. In addition, if person i proves its identity to person j, the information revealed by that process should not let the person j to forge the identity of i at any point in the future.

In [FFS87], an ID scheme was designed based on the assumption that there a one-way function f exists. 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 suffices as a proof of his identity due to the one-way property of f. If he reveals x_i as the proof, it would be convincing, but it will let others to use it later and forge his identity. But since the code of the OWF f is known to everyone, and since x_i can be thought of as the witness to the an **NP** statement $f(x_i)$, the person i can use the ZK protocol of Theorem 8.4 to prove that he knows x_i rather than revealing x_i directly. The zero-knowledge property prevents others from learning any knowledge about x_i to be used in their subsequent interactions.

Since 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, the implementation reduction in this identification scheme uses the implementation of OWF in a non-black-box way.

8.2.2 Non-Black-Box Proofs of Security

Note that in order to decrease the soundness error of the ZK protocol of Theorem 8.4 one needs to do sequential repetition which increases the number of rounds to some polynomial. Another ZK protocol for **NP**, due to Blum [Blu87], shows how to achieve constant soundness in constant number of rounds. The next natural question is whether there are constant round protocols with *negligible* soundness error. Goldreich and Krawczyk [GK96] showed that it is impossible to achieve for nontrivial languages, if **((1))** the verifier is public coin and **((2))**

the simulator S is "universal" an uses the cheating verifier V^* in a black-box way (i.e., as an orale).

Theorem 8.5 ([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 $L \in$ **BPP**.

In a breakthrough, Barak [Bar01, Bar02] showed that the simulator can indeed use the code of the cheating verifier, in order to bypass the impossibility result of Theorem 8.5. His result is based on the assumption that collision-resistant hash functions exist. The techniques introduced by Barak [Bar01, Bar02] are essentially is the only cryptographic techniques for non-black-box proofs of security.

9 References

[AB] Sanjeev Arora and Boaz Barak. *Computational Complexity, A Modern Approach*. 14, 19

[AB07] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach (Draft)*. January 2007. 3, 15

[ACR98] Andreev, Clementi, and Rolim. A new general derandomization method. *JACM: Journal of the ACM*, 45, 1998. 20

[AD96] Ajtai and Dwork. A public-key cryptosystem with worst-case/average-case equivalence. In *ECCCTR: Electronic Colloquium on Computational Complexity, technical reports*, 1996. 24

[AH91] William Aiello and Johan Hastad. Statistical zero-knowledge languages can be recognized in two rounds. *Journal of Computer and System Sciences*, 42(3):327–345, June 1991. 26

[Ajt96] Ajtai. Generating hard instances of lattice problems. In *ECCCTR: Electronic Colloquium on Computational Complexity, technical reports*, 1996. 24

[AKS02] Manindra Agrawal, Neeraj Kayal, and Nitin Saxena. PRIMES is in P. Report, Department of Computer Science and Engineering, Indian Institute of Technology Kanpur, Kanpur-208016, India, August 2002. 18

[AW08] Scott Aaronson and Avi Wigderson. Algebrization: a new barrier in complexity theory. In Cynthia Dwork, editor, *STOC*, pages 731–740. ACM, 2008. 27

[Bar01] Boaz Barak. How to go beyond the black-box simulation barrier. In *FOCS*, pages 106–115, 2001. 32

[Bar02] Boaz 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, November 16–19 2002. IEEE COMPUTER SOCIETY. 32

[BBBV97] Bennett, Bernstein, Brassard, and Vazirani. Strengths and weaknesses of quantum computing. *SICOMP: SIAM Journal on Computing*, 26, 1997. 5

[BBC+01] Beals, Buhrman, Cleve, Mosca, and de Wolf. Quantum lower bounds by polynomials. *JACM: Journal of the ACM*, 48, 2001. 4, 5

[BCG+95] Bshouty, Cleve, Gavalda, Kannan, and Tamon. Oracles and queries that are sufficient for exact learning. In *ECCCTR: Electronic Colloquium on Computational Complexity, technical reports*, 1995. 8, 9, 10

[BdW02] Buhrman and de Wolf. Complexity measures and decision tree complexity: A survey. *TCS: Theoretical Computer Science*, 288, 2002. 3

[BF86] Barany and Furedi. Computing the volume is difficult. In *STOC: ACM Symposium on Theory of Computing (STOC)*, 1986. 18

[BF90] Beaver and Feigenbaum. Hiding instances in multioracle queries. In *STACS: Annual Symposium on Theoretical Aspects of Computer Science*, 1990. 24

[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. 13

[BFL91] L. Babai, L. Fortnow, and C. Lund. Nondeterministic exponential time has two-prover interactive protocols. *Computational Complexity*, 1(1):3–40, 1991. Preliminaryversion in FOCS' 90. 26

[BFNW93] Babai, Fortnow, Nisan, and Wigderson. BPP has subexponential time simulations unless EXPTIME has publishable proofs. *CMPCMPL: Computational Complexity*, 3, 1993. 20

[BGI08] Eli Biham, Yaron J. Goren, and Yuval Ishai. Basing weak public-key cryptography on strong one-way functions. In Ran Canetti, editor, *TCC*, volume 4948 of *Lecture Notes in Computer Science*, pages 55–72. Springer, 2008. 29

[BGP00] Mihir Bellare, Oded Goldreich, and Erez Petrank. Uniform generation of NP-witnesses using an NP-oracle. *Inf. Comput*, 163(2):510–526, 2000. 10

[BGS75] Baker, Gill, and Solovay. Relativizations of the P =? NP question. *SICOMP: SIAM Journal on Computing*, 4, 1975. 14, 27

[BK89] M. Blum and Sampath Kannan. Designing programs that check their work. *21st ACM STOC*, pages 86–97, 1989. 26

[Blu87] Manuel Blum. How to prove a theorem so no one else can claim it. In *Proceedings of the International Congress of Mathematicians*, pages 1444–1451, 1987. 31

[BMG07] Barak and Mahmoody-Ghidary. Lower bounds on signatures from symmetric primitives. In *FOCS: IEEE Symposium on Foundations of Computer Science (FOCS)*, 2007. 29

[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. 29

[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. 4

[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. 25, 26

[BT06] Bogdanov and Trevisan. Average-case complexity. In *Foundations and Trends in Theoretical Computer Science, Now Publishers or World Scientific*, volume 2. 2006. 23

[Can74] Georg Ferdinand Ludwig Philipp Cantor. Über eine Eigenschaft des Inbegriffs aller reellen algebraischen Zahlen. *Journal f. reine und angew. Math*, 77:258–262, 1874. 14

[Coo71] S. A. Cook. The complexity of theorem proving procedures. In *stoc71*, pages 151–158, 1971. 6

[Coo72] S. A. Cook. A hierarchy for nondeterministic time complexity. In *stoc72*, pages 187–192, 1972. 14

[CS98] Ronald Cramer and Victor Shoup. A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In *Crypto '98*, pages 13–25, 1998. LNCS No. 1462. 28

[Dan51] George 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. 15

[DF88] Dyer and Frieze. On the complexity of computing the volume of a polyhedron. *SICOMP: SIAM Journal on Computing*, 17, 1988. 17, 18

[DFK91] Dyer, Frieze, and Kannan. A random polynomial time algorithm for approximating the volume of convex bodies. *JACM: Journal of the ACM*, 38, 1991. 18

[DH76] W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(5):644–654, 1976. 28

[FF93] Feigenbaum and Fortnow. Random-self-reducibility of complete sets. *SICOMP: SIAM Journal on Computing*, 22, 1993. 25

[FFS87] U. Feige, A. Fiat, and A. Shamir. Zero knowledge proofs of identity. In *stoc87*, pages 210–217, 1987. 31

[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. 29

[foc00] *Proc.* 41*st FOCS*. IEEE, 2000. 36, 37

[For87] L. Fortnow. The complexity of perfect zero knowledge. In *stoc87*, pages 204–209, 1987. 26

[For00] Lance Fortnow. Diagonalization. *Bulletin of the EATCS*, 71:102–113, 2000. 13

[GG98] Goldreich and Goldwasser. On the limits of non-approximability of lattice problems. In *STOC: ACM Symposium on Theory of Computing (STOC)*, 1998. 24

[GGH96a] Goldreich, Goldwasser, and Halevi. Collision-free hashing from lattice problems. In *ECCCTR: Electronic Colloquium on Computational Complexity, technical reports*, 1996. 24

[GGH96b] O. Goldreich, S. Goldwasser, and S. Halevi. Public-key cryptosystems from lattice reduction problems. Technical Report MIT/LCS/TR-703, Massachusetts Institute of Technology, November 1996. 24

[GGK03] R. Gennaro, Y. Gertner, and J. Katz. Lower bounds on the efficiency of encryption and digital signature schemes. In *Proc.* 35*th STOC*. ACM, 2003. 29

[GGM86] Goldreich, Goldwasser, and Micali. How to construct random functions. *JACM: Journal of the ACM*, 33, 1986. 28

[GK96] Goldreich and Krawczyk. On the composition of zero-knowledge proof systems. *SICOMP: SIAM Journal on Computing*, 25, 1996. 31, 32

[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* [foc00], pages 325–338. 29

[GL89] Oded Goldreich and Leonid A. Levin. A hard-core predicate for all one-way functions. In *STOC*, pages 25–32. ACM, 1989. 8, 11

[GLS81] M. Grötschel, L. Lovasz, and A. Schrijver. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1:169–197, 1981. 17

[GM84] S. Goldwasser and S. Micali. Probabilistic encryption. *JCSS*, 28(2):270–299, April 1984. 28

[GMR89] Goldwasser, Micali, and Rackoff. The knowledge complexity of interactive proof systems. *SICOMP: SIAM Journal on Computing*, 18, 1989. 30

[GMR01] Yael Gertner, Tal Malkin, and Omer Reingold. On the impossibility of basing trapdoor functions on trapdoor predicates. In *FOCS*, pages 126–135, 2001. 29

[GMW86] O. Goldreich, S. Micali, and A. Wigderson. Proofs that yield nothing but their validity and a methodology of cryptographic design. In *focs86*, 1986. 30

[GNW95] Goldreich, Nisan, and Wigderson. On yao's XOR-lemma. In *ECCCTR: Electronic Colloquium on Computational Complexity, technical reports*, 1995. 22

[Gol01] Oded Goldreich. *Foundations of Cryptography: Basic Tools*. Cambridge University Press, 2001. Earlier version available on `http://www.wisdom.weizmann.ac.il/~oded/frag.html` . 27, 30

[Gol04] Oded Goldreich. *Foundations of Cryptography: Basic Applications*. Cambridge University Press, 2004. 27

[Gol08] Oded Goldreich. *Computational Complexity: A Conceptual Perspective*. Cambridge University Press, 1 edition, April 2008. 3, 6, 19

[GPV07] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. *Electronic Colloquium on Computational Complexity (ECCC)*, 14(133), 2007. 24

[Gro96] Lov K. Grover. A fast quantum mechanical algorithm for database search. In *STOC*, pages 212–219, 1996. 5

[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. 26

[GT00] Rosario Gennaro and Luca Trevisan. Lower bounds on the efficiency of generic cryptographic constructions. In *Proc. 41st FOCS* [foc00], pages 305–313. 29

[GW95] Goemans and Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *JACM: Journal of the ACM*, 42, 1995. 17

[HH09] Iftach Haitner and Thomas Holenstein. On the (im)possibility of key dependent encryption. In Omer Reingold, editor, *TCC*, volume 5444 of *Lecture Notes in Computer Science*, pages 202–219. Springer, 2009. 29

[HHRS07] 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. 29

[HILL99] Johan Håstad, Russell Impagliazzo, Leonid A. Levin, and Michael Luby. A pseudorandom generator from any one-way function. *SIAM Journal on Computing*, 28(4):1364–1396, 1999. Preliminary versions appeared in STOC' 89 and STOC' 90. 31

[HK05] Horvitz and Katz. Bounds on the efficiency of "black-box" commitment schemes. In *ICALP: Annual International Colloquium on Automata, Languages and Programming*, 2005. 29

[HR04] Hsiao and Reyzin. Finding collisions on a public road, or do secure hash functions need secret coins? In *CRYPTO: Proceedings of Crypto*, 2004. 29

[HS65] J. Hartmanis and R. E. Stearns. On the computational complexity of algorithms. *Trans. Amer. Math. Soc.*, 117:285–306, 1965. 14

[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. 21

[IL89] Impagliazzo and Luby. One-way functions are essential for complexity based cryptography. In *FOCS: IEEE Symposium on Foundations of Computer Science (FOCS)*, 1989. 22, 27, 28

[IR88] Impagliazzo and Rudich. Limits on the provable consequences of one-way permutations. In *CRYPTO: Proceedings of Crypto*, 1988. 28, 29

[IW97] Russell Impagliazzo and Avi Wigderson. $\mathbf{P} = \mathbf{BPP}$ if $\mathbf{E}$ requires exponential circuits: Derandomizing the XOR lemma. In *Proc. 29th STOC*, pages 220–229. ACM, 1997. 19, 20

[Jac97] Jeffrey 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. 13

[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. 18

[JVV86] Jerrum, Valiant, and Vazirani. Random generation of combinatorial structures from a uniform distribution. *TCS: Theoretical Computer Science*, 43, 1986. 10, 18

[Kab02] Valentine Kabanets. Derandomization: a brief overview. *Bulletin of the EATCS*, 76:88–103, 2002. 19

[Kha79] L. G. Khachian. A polynomial time algorithm for linear programming. *Soviet Math. Dokl.*, 20:191–194, 1979. 16

[KI04] Kabanets and Impagliazzo. Derandomizing polynomial identity tests means proving circuit lower bounds. *CMPCMPL: Computational Complexity*, 13, 2004. 21

[KL80] Karp and Lipton. Some connections between nonuniform and uniform complexity classes. In *STOC: ACM Symposium on Theory of Computing (STOC)*, 1980. 8

[KL07] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography (Chapman & Hall/Crc Cryptography and Network Security Series)*. Chapman & Hall/CRC, 2007. 27

[KM93] Kushilevitz and Mansour. Learning decision trees using the fourier spectrum. *SICOMP: SIAM Journal on Computing*, 22, 1993. 12

[KP82] Karp and Papadimitriou. On linear characterizations of combinatorial optimization problems. *SICOMP: SIAM Journal on Computing*, 11, 1982. 16, 17

[KS06] Jonathan A. Kelner and Daniel 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. 16

[KST99] Jeong Han Kim, Daniel R. Simon, and Prasad Tetali. Limits on the efficiency of one-way permutation-based hash functions. In *FOCS*, pages 535–542, 1999. 29

[KV94] Michael J. Kearns and Umesh V. Vazirani. *An Introduction to Computational Learning Theory*. MIT press, Cambridge, Massachusetts, 1994. 8

[Lev73] Levin. Universal sequential search problems. *PINFTRANS: Problems of Information Transmission (translated from Problemy Peredachi Informatsii (Russian))*, 9, 1973. 6

[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. 24

[LR85] Luby and Rackoff. How to construct pseudo-random permutations from pseudo-random functions. In *CRYPTO: Proceedings of Crypto*, 1985. 28

[LTW05] Lin, Trevisan, and Wee. On hardness amplification of one-way functions. In *Theory of Cryptography Conference (TCC), LNCS*, volume 2, 2005. 29

[M. 88] M. Grötschel and L. Lovász and A. Schrijver. *Geometric algorithms and combinatorial optimization*. Springer-Verlag, 1988. 18

[Mer78] R. C. Merkle. Secure communications over insecure channels. *Communications of the ACM*, 21:294–299, April 1978. 29

[Mic01] Micciancio. The shortest vector in a lattice is hard to approximate to within some constant. *SICOMP: SIAM Journal on Computing*, 30, 2001. 24

[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. 6

[MS73] Meyer and Stockmeyer. Word problems requiring exponential time. In *STOC: ACM Symposium on Theory of Computing (STOC)*, 1973. 6

[MX10] Mohammad Mahmoody and David Xiao. On the power of randomized reductions and the checkability of SAT. In *IEEE Conference on Computational Complexity*, pages 64–75. IEEE Computer Society, 2010. 26

[Nao02] Moni Naor. Deniable ring authentication. In *Crypto '02*, 2002. LNCS No. 2442. 31

[Nis91] Noam Nisan. CREW PRAMs and decision trees. *SIAM Journal on Computing*, 20(6):999–1007, December 1991. 4

[NS94] Noam Nisan and Mario Szegedy. On the degree of Boolean functions as real polynomials. *Computational Complexity*, 4(4):301–313, 1994. 5

[NW94] Noam Nisan and Avi Wigderson. Hardness vs randomness. *Journal of Computer and System Sciences*, 49(2):149–167, October 1994. Preliminary version in FOCS' 88. 20

[NY90] Moni Naor and Moti Yung. Public-key cryptosystems provably secure against chosen ciphertext attacks. In *In Proc. of the 22nd STOC*, pages 427–437. ACM Press, 1990. 28

[OW93] R. Ostrovsky and Avi Wigderson. Nontrivial zero-knowledge implies one-way functions. In *Proceedings of the 2nd ISTCS*, pages 3–17, 1993. 22, 28

[Pap94] C. H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994. 3, 8

[Pei08] Chris Peikert. Public-key cryptosystems from the worst-case shortest vector problem. *Electronic Colloquium on Computational Complexity (ECCC)*, 15(100), 2008. 24

[PS98] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Dover Pub., 1998. 15

[Rab79] M. O. Rabin. Digitalized signatures and public-key functions as intractable as factorization. Technical Report MIT/LCS/TR-212, Massachusetts Institute of Technology, January 1979. 28

[Rab80] M. O. Rabin. Probabilistic algorithm for testing primality. *Journal of Number Theory*, 12:128–138, 1980. 18

[Rac88] C. Rackoff. A basic theory of public and private cryptosystems. In *Proc. Advances in Cryptology – CRYPTO '88*, pages 249–255, 1988. 28

[RR97] Razborov and Rudich. Natural proofs. *JCSS: Journal of Computer and System Sciences*, 55, 1997. 27

[RTV04] Omer Reingold, Luca Trevisan, and Salil P. Vadhan. Notions of reducibility between cryptographic primitives. In Moni Naor, editor, *TCC*, volume 2951 of *Lecture Notes in Computer Science*, pages 1–20. Springer, 2004. 28

[Sch90] Schapire. The strength of weak learnability. *MACHLEARN: Machine Learning*, 5, 1990. 13

[Sch03] A. Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*, volume 24 of *Algorithms and Combinatorics*. Springer, 2003. 15

[Sha02] Ronen Shaltiel. Recent developments in extractors. *Bulletin of the European Association for Theoretical Computer Science*, 2002. Available from `http://www.wisodm.weizmann.ac.il/~ronens`. 21

[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. 29

[Sni85] Marc Snir. Lower bounds on probabilistic linear decision trees. *Theoretical Computer Science*, 38(1):69–82, May 1985. 4

[SS77] Robert Solovay and Volker Strassen. A fast monte-carlo test for primality. *SIAM J. Comput*, 6(1):84–85, 1977. 18

[ST04] Spielman and Teng. Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time. *JACM: Journal of the ACM*, 51, 2004. 16

[STV98] Madhu Sudan, Luca Trevisan, and Salil Vadhan. Pseudorandom generators without the xor lemma. ECCC Report TR98-074, 1998. http://www.eccc.uni-trier.de/eccc/. 20

[STV01] Sudan, Trevisan, and Vadhan. Pseudorandom generators without the XOR lemma. *JCSS: Journal of Computer and System Sciences*, 62, 2001. 24

[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. 20

[Tre01] Trevisan. Extractors and pseudorandom generators. *JACM: Journal of the ACM*, 48, 2001. 21, 22

[Tur36] Alan M. Turing. On computable numbers, with an application to the entscheidungsproblem. *Procedings of the London Mathematical Society*, 42(2):230–265, 1936. 14

[Uma03] Christopher Umans. Pseudo-random generators for all hardnesses. *J. Comput. Syst. Sci.*, 67(2):419–440, 2003. 20

[Val79] Valiant. The complexity of computing the permanent. *TCS: Theoretical Computer Science*, 8, 1979. 23

[Yao82] Andrew C. Yao. Theory and applications of trapdoor functions. In *Proc. 23rd FOCS*, pages 80–91. IEEE, 1982. 22

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets. In *Proc. 27th FOCS*, pages 162–167. IEEE, 1986. 29