1 Introduction

The theoretical computer science community of the last century has continuously been stumped by a seemingly simple conjecture: namely, $P \neq NP$. While most complexity theorists believe that it is true [Gas02], efforts to find a proof of this conjecture have all fallen short. However, this is not for lack of trying: during this time, evidence has emerged that separating $P$ and $NP$ is “hard” in a formal sense. In this work, we will attempt to give a summary of important works in this area. For the sake of discussion, we will assume for this work that indeed $P \neq NP$ and ask the question “why is it so difficult to prove such a thing?”

We start by giving a brief overview of the history of some promising techniques to proving lower bound conjectures such as $P \neq NP$. One of the first known techniques for showing lower bounds on complexity classes was diagonalization. This was used by Hartmanis and Stearns [HS65] in 1965 to prove the time hierarchy theorem, which states, roughly, that we can solve more problems if we have more time. At the time, this sort of simple diagonalization technique was more or less the only known method of proving lower bounds on complexity classes. In 1975, though, Baker, Gill, and Solovay [BGS75] showed a result now known as the relativization barrier that implied that this simple technique alone cannot show that $P \neq NP$.

Thus, progress on this conjecture was halted for some amount of time, and complexity theorists sought out other possible methods of attacking lower bound problems. One promising line was circuit complexity, first introduced by Shannon [Sha49]. Shannon proved, information-theoretically, that almost all problems require circuits of size $2^{\Omega(n)}$; however, he did not provide lower bounds on any particular problem; in particular, he did not give a way of showing lower bounds that are useful to attack complexity-theoretic questions such as $P \neq NP$. Since the relativization barrier does not apply to circuit lower bounds, a flurry of activity began to try to strengthen circuit lower bound techniques in the hope of proving that $NP \not\subseteq P$/poly, which would imply $P \neq NP$. However, this too proved mostly futile: in 1997, Razbarov and Rudich showed that, in some sense, it is unlikely that a natural circuit lower bound can show a separation of $P$ from any other complexity class, not just $NP$. Thus, yet again, one of the most seemingly promising lines of attack of the $P \neq NP$ conjecture was halted.

The goal of this work is to give a nearly completely self-contained presentation of the important results of Baker, Gill, and Solovay [BGS75] and Razbarov and Rudich [RR97]. We warm up with one result of the former work, which will be relatively short since we will only concern ourselves with the easier of the two results in that paper, and we will spend most of this work discussing the latter work. Along the way, we will see some common tools used in complexity theory and cryptography such as pseudorandom number generators and hybrid arguments.

2 Warmup: The Relativization Barrier

In 1975, Baker, Gill, and Solovay [BGS75] gave a proof the following theorem:

---

1The authors of [BGS75] also show that there is another language $B$ for which $P^{B} \neq NP^{B}$, which similarly demonstrates that no relativizing technique can show that $P = NP$. For brevity and to keep the focus on $P \neq NP$, we will not discuss this proof here.
Theorem 1 (Theorem 1 in BGS75). There is a language $A$ for which $P^A = NP^A$.

Before giving a proof of it, we will discuss why this seemingly-innocent result is annoying to those who seek to prove $P \neq NP$. Recall that our proof of $P \neq EXP$ used diagonalization. Notice that the argument used would also apply verbatim if we add an arbitrary oracle $A$. Thus, for any $A$ we have $P^A \neq EXP^A$. However, if we used similar techniques to show that $P \neq NP$, then it would also follow that $P^A \neq NP^A$ for all $A$, which contradicts Theorem 1.

A proof technique that is invariant to adding oracles is said to relativize. Thus, Theorem 1 implies that no proof technique that relativizes can possibly show that $P \neq NP$. For how simple it sounds, this is a surprisingly strong result—indeed, diagonalization was the primary tool used at the time to prove lower bounds, and this result shows that a simple diagonalization technique alone cannot suffice to show $P \neq NP$; in particular, our proof that $P \neq EXP$ cannot be easily adapted to show $P \neq NP$.

We now give a short proof of Theorem 1.

Proof of Theorem 1. Let $A$ be any PSPACE-complete language under polynomial time reductions\footnote{An example of a PSPACE-complete language is TQBF (“true quantified boolean formulas”), the language of arbitrarily quantified formulas that are satisfiable. This is the natural generalization of the languages $\Sigma_n\text{-SAT}$ for $n \in \mathbb{N}$. For a more detailed description of the language and a proof of its PSPACE-completeness, see the Wikipedia article on TQBF. As an interesting note, it can also be shown that many common board games, such as Gomoku (which is itself generalized Tic-Tac-Toe) [HT07], Othello [IK94] and Amazons [Hea05], are PSPACE-complete to determine who is winning from a given position.}. Then observe the following inclusions:

(a) $\text{PSPACE} \subseteq P^A$ since $A$ is PSPACE-complete;
(b) $P^A \subseteq NP^A$ since we don’t need to use the nondeterminism; and
(c) $NP^A \subseteq \text{NPSPACE}$ since we can use polynomial space to simulate any oracle calls to $A$ that we need to simulate.

However, recall by Savitch’s Theorem that $\text{PSPACE} = \text{NPSPACE}$; thus, the above inclusions are in fact equalities; in particular, we have $P^A = NP^A$, as desired. \hfill \blacksquare

3 Definitions

In the rest of the paper, we will give an overview of natural proofs, introduced by Razborov and Rudich [RR97], and attempt to give a proof of one of the main results in this paper.

3.1 Natural Properties

In this section, we will define the notion of a natural property. For the remainder of this paper, we will be discussing families of boolean circuits. For simplicity of notation, let $B = \{0, 1\}$. We will give a series of definitions leading up to a formal definition of a “natural proof”.

Definition 1. A boolean function on $n$ inputs is a function $f_n : B^n \rightarrow B$. We will denote by $F_n$ the set of all such boolean functions. A boolean family is a sequence $f = \{f_n\}_{n \in \mathbb{N}}$ where $f_n$ has $n$ inputs for all $n$.

It will often be useful to think of boolean functions $f_n$ on $n$ inputs as truth tables of length $2^n$, which can be represented by strings of length $2^n$. We will thus often use $F_n$ (the set of such boolean functions) interchangeably with $B^{2^n}$ (the set of strings of length $2^n$).

Definition 2. A combinatorial property is a sequence $C = \{C_n\}_{n \in \mathbb{N}}$ where $C_n \subseteq F_n$ for all $n$. A boolean family $f$ possesses the property $C$ if $f_n \in C_n$ for infinitely many $n$.

With these definitions, a seemingly reasonable method of proving that $P \neq NP$ may go as follows:

(a) Exhibit a “natural” combinatorial property $C$, for some notion of “natural”.
(b) Show that some NP boolean family, say 3-SAT, possesses property $C$.
(c) Show that no boolean family admitting a polynomial-size circuit family can possess property $C$.
(d) Conclude that 3-SAT \( \not\in \text{P/poly} \); thus \( \text{NP} \not\subseteq \text{P/poly} \) and so \( \text{P} \neq \text{NP} \).

Before the seminal paper of Razborov and Rudich, this sort of proof was considered a very promising route to proving that \( \text{P} \neq \text{NP} \): for one, circuit lower bounds do not relativize (so we do not run into the issue discussed in Section 3). However, this paper showed, in some sense, that this technique is very unlikely to work: for some reasonable definition of “natural”, assuming some commonly-held assumptions about pseudorandom number generators, there is no “natural combinatorial property” that both contains some NP-complete boolean family and cannot admit any polynomial-size circuits.

We will now define what we mean by “natural”. Observe that subsets \( C_n \subseteq F_n \) can also be thought of as boolean functions on \( 2^n \) inputs (namely, \( C_n(t) \) on a string \( t \in B^{2^n} = F_n \) is 1 when \( t \in C_n \)). We can thus discuss how difficult it is to compute a combinatorial property \( C \).

**Definition 3.** A combinatorial property \( C \) is *constructive* if deciding whether \( f_n \in C_n \) when given \( f_n \) as a truth table can be done by a circuit family with size \( \text{poly}(2^n) \); i.e. polynomial in the size of the truth table.

**Definition 4.** A combinatorial property \( C \) is *large* if \( |C_n|/|F_n| \geq 2^{-\Omega(n)} \). Notice, again, that \( C_n \) itself is a function with \( N = 2^n \) inputs; thus, largeness is the condition that the fraction of functions \( f_n \) possessing \( C_n \) should be non-negligible in \( N \) (i.e. \( 1/\text{poly}(N) \)).

**Definition 5.** Let \( L \) be a complexity class, which we will view as a set of boolean families. A combinatorial property \( C \) is *natural against \( L \)* if (1) it is constructive; (2) it is large, and (3) no boolean family in \( L \) possesses property \( C \).

With these definitions, we can now define a *natural proof* as one that follows steps (a)-(d) above:

**Definition 6.** A *natural proof* is one that exhibits a combinatorial property that both is natural against \( \text{P/poly} \), and contains an \( \text{NP} \) problem.

### 3.2 Pseudorandom Number Generators

The power (or perhaps lack thereof) of randomness is one of the core concepts in modern complexity theory. In particular, much recent work has been devoted to *derandomization*, the study of creating deterministic algorithms out of randomized ones. One possible route to derandomization is to show the existence of a *pseudorandom number generator*: a method of generating new random bits that cannot be distinguished from purely random bits by any efficient algorithm. In this section, we will formalize a notion of pseudorandomness which will be useful in the ensuing formalization and proof. For the rest of the paper, we will use the notation \( a \overset{\$}{\leftarrow} S \) to denote a uniformly random element \( a \) sampled from a finite set \( S \).

**Definition 7.** A *pseudorandom generator* (PRG) on \( k \) bits is a function \( G_k : B^k \to B^{2k} \). A PRG family \( G = \{G_k\}_{k \in \mathbb{N}} \) is a set of pseudorandom generators, one for each length \( k \in \mathbb{N} \). A PRG family is *efficient* if it can be computed by some polynomial size circuit family.

The above definition says nothing about how *good* a PRG family is; indeed, the map \( x \mapsto 0 \), which is not at all random, satisfies this definition. To discuss what a good PRG family is, it will be useful to introduce the notion of *computational distinguishability*.

**Definition 8.** Let \( D \) and \( D' \) be two distributions over \( B^m \). Then \( D \) and \( D' \) are \( S \)-*computationally distinguishable* if there is a circuit \( C \) of size at most \( S \) such that

\[
\left| \Pr_{x \overset{\$}{\leftarrow} D} [C(x) = 1] - \Pr_{x \overset{\$}{\leftarrow} D'} [C(x) = 1] \right| \geq \frac{1}{S}.
\]

---

3The original paper [RR97] defines \( C \) to be constructive/large if some subset \( C^* \subseteq C \) is constructive/large; for simplicity, we do not use this definition here. This change does not affect the main result. Also, for simplicity, in this work, we have define our notion of efficiency to be polynomial-size circuits. Generalizing these results by swapping polynomial size circuits with polynomial time (deterministic or randomized) Turing machines where possible is left as an exercise to the overachieving reader.
**Definition 9.** The hardness $H(G_k)$ of a PRG $G_k$ is the smallest number $S$ such that the distribution of $G_k(x)$ for $x \notin B^k$ is $S$-computationally distinguishable from a random string $y \in B^{2k}$. Similarly, the hardness $H_G$ of a PRG family $G$ is the function $H_G(k) = H(G_k)$. We will say $G_k$ is broken by a circuit of size $S$ if it has hardness at most $S$.

We are primarily interested in PRG families that are (1) efficiently computable and (2) hard. In particular, it is conjectured by many (e.g. [RR97]) that PRGs with exponential hardness exist, and there are many important consequences of their existence (or nonexistence) across complexity theory, cryptography, and many other areas of computer science. We will give an example of an important theorem relating hard PRG families to the important cryptographic notion of one-way functions.

One-way functions, which this work does not contain enough space to discuss in detail, are functions that are easy to evaluate but hard to invert: in particular, a function $f : B^* \rightarrow B^*$ is one-way if $f$ is efficiently computable, and given $y \in f(B^*)$, it is difficult to find $x$ such that $f(x) = y$. Much of the security of modern cryptography rests on the existence of such functions, but it is very difficult to prove that they exist.

**Theorem 2** (Corollary of Theorems 1 and 2 in [GGM86]). A PRG family of superpolynomial hardness exists if and only if one-way functions exist. Further, in this case, $P \neq NP$.

### 4 Natural Proofs Cannot Show $P \neq NP$

In this section, we state and prove an important result in the seminal work of Razborov and Rudich [RR97], namely that it is unlikely that one can prove that $P \neq NP$ using a natural proof as defined in Section 3.1.

**Theorem 3** (Theorem 4.1 in [RR97]). If there is a natural combinatorial property against $P/\text{poly}$, then no efficient PRG family $G = \{G_k\}$ has hardness $2^{k^{O(1)}}$.

Notice that this is much stronger than what we need to show. To show that we cannot show $P \neq NP$ using a natural proof, we only need to show that there is no natural combinatorial property against $P/\text{poly}$ that contains an NP problem. So, in fact, this theorem implies the stronger result that no natural proof can separate P from any other complexity class! We now give a proof of Theorem 3.

**Proof of Theorem 3**. Suppose for contradiction that there is some natural combinatorial property $C$ against $P/\text{poly}$. Let $G = \{G_k\}$ be an efficient PRG family, and pick an arbitrary $\varepsilon > 0$. Let $k \in \mathbb{N}$ and $n = k^\varepsilon$. We will show that $H_G(k) \leq 2^{O(n)} = 2^{O(k^\varepsilon)}$, which, since $\varepsilon$ was arbitrary, would imply that $G$ does not have hardness $2^{k^{O(1)}}$.

Let $G_0, G_1 : B^k \rightarrow B^k$ be the first $k$ bits and the last $k$ bits of $G_k$, respectively. For any $r \in \mathbb{N}$ and any string $y \in B^r$, define $G_y = G_{y_r} \circ G_{y_{r-1}} \circ \ldots \circ G_{y_1}$, where $\circ$ denotes function composition. Finally, for strings $x \in B^k$ and $y \in B^{r}$, let $f_x(y)$ be the first bit of $G_y(x)$.

We claim now that, for sufficiently large $k$, we have $C_n \cap \{f_x : x \in B^k\} = \emptyset$. To see this, suppose not. Then there is some sequence $\{x_k\}_{k=1}^\infty$ such that $f_{x_k} \in C_n$ infinitely often. But note that $\{f_{x_k}\} \in P/\text{poly}$, since each $f_{x_k}$ is a composition of polynomial-sized circuits (namely the $G$s) for each $k$. This contradicts the definition of a natural combinatorial property against $P/\text{poly}$.

Thus, in particular, note that, for sufficiently large $k$, we have

$$\Pr\left[f \notin C_n\right] - \Pr_{x \sim B^k, \; f_x \in C_n} = \left|\frac{|C_n|}{|F_n|} - 0\right| \geq 2^{-O(n)} \tag{1}$$

since $C$ is large. Further, since $C$ is efficient, by definition, deciding containment in $C_n$ can be done with a circuit of size $2^{O(n)}$. This leaves us with something that looks close to the definition of hardness for $S = 2^{O(n)}$; the only problem is that $f$ wasn’t our original pseudorandom number generator $G$.

To fix this, our general strategy be as follows. We will define a sequence of distributions $D_0, \ldots, D_N$ for some $N = 2^{O(n)}$ where $D_1$ will be the distribution of $f_x$ for $x \sim B^n$ drawn uniformly at random, and $D_N$ is a uniform random element from $F_n$. By Eqn. (1) and the triangle inequality, we will conclude that

\[\text{Floors and ceilings won’t matter here, so we will ignore them.}\]

\[\text{This may seem like black magic, and that is fine. The utility of this weird definition will hopefully become clear later on.}\]
there is some $i$ such that $D_i$ and $D_{i+1}$ are $2^{O(n)}$-computationally distinguishable. We will then show, by our construction of the $D_i$, that this distinguishability implies that $G$ is broken by circuits of size $2^{O(n)}$; i.e. $H_G(k) \leq 2^{O(n)}$, completing the proof. The technique of defining a sequence of distributions and applying the triangle inequality to recover that at least one of the consecutive pairs in the sequence is distinguishable is commonly used in proofs in cryptography, where it is known as a *hybrid argument*, and the distributions $D_i$ for $0 < i < N$ are known as *hybrids*, since they are something in between $D_0$ and $D_N$.

We first need a bit more machinery\footnote{The construction that is described here was originally given by [GGM86], and the presentation and usage of the construction in [RR97] was impossible for me to follow. With the intent of giving a self-contained presentation of this proof, in this work we give an overview of the construction of [GGM86], and attempt to reconstruct this part of the proof in [RR97] manually using hopefully clearer notation.}. For $i = 0, \ldots, 2^n - 2$ let $r_i \leftarrow B^k$. We will define a sequence of trees $T_0, \ldots, T_{2^n - 2}$ as follows. The tree $T_0$ is a complete binary tree of height $n$ (i.e. with $2^n - 1$ nodes). The nodes are numbered in breadth-first and then left-to-right order: that is, we will call the root $v_0$, and the two children of a node $v_i$ will be numbered $v_{2i+1}$ and $v_{2i+2}$. We will also label the nodes with strings as follows: The root is labeled $r_0$. Then, for any node $v_i$, with label $s$, the left child of $v_i$ is labeled $G_0(s)$ and the right child is labeled $G_1(s)$. In the tree $T_j$ for any $j > 0$, the nodes $v_i$ for $i \leq j$ have their labels replaced with $r_i$. The nodes $v_i$ for $i > j$ are then recomputed according to the algorithm specified above. For clarity, we include in Figure 1 a picture of what $T_1$ would look like if the tree had height $n = 2$. Note in particular, that $v_3$ and $v_4$ have labels $G_0(r_1)$ and $G_1(r_1)$, not $G_0(r_0)$ and $G_0(r_1)$.

Notice that each tree $T_j$ defines a function $g_j \in F_n$ that depends on the random values $r_i$ for $i \leq j$. In particular, define $g_j(y)$ for $y \in B^n$ to be the first bit of the label on the leaf node reached by starting at the root node of $T_j$ and, for each bit $i = 1, \ldots, n$, following the left child if $y_i = 0$, or the right child if $y_i = 1$. For example, in tree $T_0$, one will arrive at the leaf node labeled with $(G_{y_n} \circ G_{y_{n-1}} \circ \cdots \circ G_{y_1}) (r_0) = G_y(r_0)$, so $g_1(y)$ is just the first bit of $G_y(r_0)$, which by definition is $f_{r_0}(y)$. This allows us to define our distributions. For $j = 0, \ldots, 2^n - 2$, let $D_j$ be the distribution of functions $g_j$ constructed as above, over the random choices of $r_i \leftarrow B^k$ for $i \leq j$. Observe that $g_0 = f_{r_0}$, and that $g_{2^n - 2}$ is a uniformly random function. We will ignore the odd-numbered $T_j$'s; it will suffice to consider the even-numbered ones. Using the triangle inequality, Eqn. \footnote{The construction that is described here was originally given by [GGM86], and the presentation and usage of the construction in [RR97] was impossible for me to follow. With the intent of giving a self-contained presentation of this proof, in this work we give an overview of the construction of [GGM86], and attempt to reconstruct this part of the proof in [RR97] manually using hopefully clearer notation.} gives

\[
\sum_{j=0}^{2^n-2-2} \left| \Pr[g_{2j+2} \in C_n] - \Pr[g_{2j} \in C_n] \right| \geq \sum_{j=0}^{2^n-2-2} \Pr[g_{2j+2} \in C_n] - \Pr[g_{2n} \in C_n] \geq \Pr[g_{2n-2} \in C_n] - \Pr[g_0 \in C_n] \geq 2^{-O(n)}
\]
which, combining with Eqn. (2), gives

\[
\Pr[g_{2j+2} \in C_n] - \Pr[g_{2j} \in C_n] \geq 2^{-O(n)} = 2^{-O(n)}
\]

that is, \( D_{2j} \) and \( D_{2j+2} \) are \( 2^{O(n)} \)-computationally distinguished by the circuit family for the constructive property \( C \). Now, consider the following algorithm, which we will call \( A \), and which we claim breaks \( G_k \).

1. on input \( y \in B^{2k} \):
2. write \( y = (y_0, y_1) \) for \( y_0, y_1 \in B^k \)
3. construct the tree \( T_{2j} \), except we replace the labels on nodes \( v_{2j+1} \) and \( v_{2j+2} \)
4. with \( y_0 \) and \( y_1 \) respectively (and recompute their children as usual). call this tree \( T^* \).
5. construct the function \( g^* \) represented by this tree
6. return 1 if \( g^* \in C_n \), else 0.

Since \( g \in C_n \) can be decided by a \( 2^{O(n)} \) size circuit and the rest of this computation is clearly efficient in the size of the tree (which is \( 2^{O(n)} \)), this algorithm can be computed by a polynomial size circuit. Further, notice that

(a) if \( y \) is a uniformly random, then constructing \( g^* \) is equivalent to constructing \( g_{2j+2} \) in the tree \( T_{2j+2} \), \( y_0 \) and \( y_1 \) are identified with \( r_{2j+1} \) and \( r_{2j+2} \) respectively.
(b) if \( x \sim B^k \) and \( y = G_k(x) \), then constructing \( g^* \) is equivalent to constructing the function \( g_{2j} \). In tree \( T_{2j} \), the label \( r_j \) on \( v_j \) should technically be replaced with \( x \), which does not change the outcome—\( g^* \) is still the same function regardless of what the label on \( v_j \) is, since it only depends on the leaves. Identifying \( x \) with \( r_j \) is safe because they are both uniformly random from \( B^k \).

Thus, we have

\[
\Pr_{x \sim B^k} [A(G_k(x)) = 1] = \Pr_{y \sim B^{2k}} [g_{2j} \in C_n]
\]

\[
\Pr_{y \sim B^k} [A(y) = 1] = \Pr_{y \sim B^{2k}} [g_{2j+2} \in C_n]
\]

which, combining with Eqn. (2), gives

\[
\left| \Pr_{x \sim B^k} [A(G_k(x)) = 1] - \Pr_{y \sim B^{2k}} [A(y) = 1] \right| \geq 2^{-O(n)};
\]

that is, \( A \) breaks the PRG \( G \), completing the proof.

5 Conclusion

In this work, we have developed some tools to formally analyze why proving the conjecture \( P \neq NP \) is so difficult. In particular, we looked at the relativization barrier \( [BGS75] \) and the natural proof barrier \( [RR97] \).

There is more work along these lines that we do not have the space to discuss in this work. In particular, Shamir’s 1990 proof that \( IP = \text{PSPACE} \) \( [Sha90] \) developed a novel technique known as arithmetization, which again created some hope that \( P \) vs \( NP \) would be resolved soon using this new technique that neither relativized nor was a natural proof. However, more recently, Aaronson and Widgerson \( [AW09] \) showed yet another impossibility result that implies, informally, that arithmetization also cannot suffice to show \( P \neq NP \).

What tools, then, will resolve \( P \) vs \( NP \), and how long will this cycle of hopeful techniques being squashed by impossibility results continue? We can only guess, and we will never know for sure until someone actually resolves this monster of a problem.

References


