

Claremont Colleges

## Scholarship @ Claremont

---

HMC Senior Theses

HMC Student Scholarship


---

2021

### Going Meta on the Minimum Circuit Size Problem: How Hard Is It to Show How Hard Showing Hardness Is?

Zoë Bell

Follow this and additional works at: [https://scholarship.claremont.edu/hmc\\_theses](https://scholarship.claremont.edu/hmc_theses)

 Part of the [Theory and Algorithms Commons](#)

---

#### Recommended Citation

Bell, Zoë, "Going Meta on the Minimum Circuit Size Problem: How Hard Is It to Show How Hard Showing Hardness Is?" (2021). *HMC Senior Theses*. 250.

[https://scholarship.claremont.edu/hmc\\_theses/250](https://scholarship.claremont.edu/hmc_theses/250)

This Open Access Senior Thesis is brought to you for free and open access by the HMC Student Scholarship at Scholarship @ Claremont. It has been accepted for inclusion in HMC Senior Theses by an authorized administrator of Scholarship @ Claremont. For more information, please contact [scholarship@cuc.claremont.edu](mailto:scholarship@cuc.claremont.edu).

Going Meta on the  
Minimum Circuit Size Problem:  
How Hard Is It to Show How Hard  
Showing Hardness Is?

**Zoë Ruha Bell**

Nicholas Pippenger, Advisor

Ran Libeskind-Hadas, Reader



**Department of Mathematics**

December, 2020

Copyright © 2020 Zoë Ruha Bell.

The author grants Harvey Mudd College and the Claremont Colleges Library the nonexclusive right to make this work available for noncommercial, educational purposes, provided that this copyright statement appears on the reproduced materials and notice is given that the copying is by permission of the author. To disseminate otherwise or to republish requires written permission from the author.

# Abstract

The Minimum Circuit Size Problem (MCSP) is a problem with a long history in computational complexity theory which has recently experienced a resurgence in attention. MCSP takes as input the description of a Boolean function  $f$  as a truth table as well as a size parameter  $s$ , and outputs whether there is a circuit that computes  $f$  of size  $\leq s$ . It is of great interest whether MCSP is NP-complete, but there have been shown to be many technical obstacles to proving that it is. Most of these results come in the following form: If MCSP is NP-complete under a certain type of reduction, then we get a breakthrough in complexity theory that seems well beyond current techniques. These results indicate that it is unlikely we will be able to show MCSP is NP-complete under these kinds of reductions anytime soon.

I seek to add to this line of work, in particular focusing on an approximation version of MCSP which is central to some of its connections to other areas of complexity theory, as well as some other variants on the problem. Let  $f$  indicate an  $n$ -ary Boolean function that thus has a truth table of size  $2^n$ . I have used the approach of Saks and Santhanam (2020) to prove that if on input  $f$  approximating MCSP within a factor superpolynomial in  $n$  is NP-complete under general polynomial-time Turing reductions, then  $E \not\subseteq P/\text{poly}$  (a dramatic circuit lower bound). This provides a barrier to Hirahara (2018)'s suggested program of using the NP-completeness of a  $2^{(1-\epsilon)n}$ -approximation version of MCSP to show that if NP is hard in the worst case ( $P \neq NP$ ), it is also hard on average (i.e., to rule out Heuristica). However, using randomized reductions to do so remains potentially tractable.

I also extend the results of Saks and Santhanam (2020) to what I define as  $\Sigma_k$ -MCSP and Q-MCSP, getting stronger circuit lower bounds, namely  $E \not\subseteq \Sigma_k P/\text{poly}$  and  $E \not\subseteq PH/\text{poly}$ , just from their NP-hardness. Since  $\Sigma_k$ -MCSP and Q-MCSP seem to be harder problems than MCSP, at first glance one might think it would be easier to show that  $\Sigma_k$ -MCSP or Q-MCSP is NP-hard, but my results demonstrate that the opposite is true.



# Contents

<b>Abstract</b>	<b>iii</b>
<b>Acknowledgments</b>	<b>xi</b>
<b>1 Introduction</b>	<b>1</b>
<b>2 Wait, What is Computational Complexity?</b>	<b>5</b>
2.1 Origins . . . . .	5
2.2 Formal framework . . . . .	7
2.3 Introducing the Minimum Circuit Size Problem (MCSP) . . . . .	27
<b>3 Why Do We Care About MCSP?</b>	<b>31</b>
3.1 Impagliazzo’s Five Worlds . . . . .	31
3.2 MCSP and Pessiland . . . . .	32
3.3 MCSP and Heuristica . . . . .	36
<b>4 Why Don’t We Know Whether MCSP is NP-complete?</b>	<b>39</b>
4.1 Previous Results for MCSP . . . . .	40
4.2 Previous Results for Gap-MCSP . . . . .	51
4.3 New Results for Gap-MCSP, $\Sigma_k$ -MCSP, & Q-MCSP . . . . .	52
<b>5 Conclusion</b>	<b>69</b>
5.1 Takeaways So Far . . . . .	69
5.2 Future Work . . . . .	71
<b>Bibliography</b>	<b>73</b>



# List of Figures

2.1	This finite automaton decides the language PARITY over the alphabet $\{0, 1\}$ . Note that the triangle indicates $q_0$ is the start state, the double circle indicates $q_1$ is an accepting state, and the arrows define the transition function. The program JFLAP was used to produce this image. . . . .	9
2.2	This depiction of a Turing machine in action was sourced from <a href="https://commons.wikimedia.org/wiki/File:Turing_machine.png">https://commons.wikimedia.org/wiki/File:Turing_machine.png</a> . .	10
2.3	This comparison of deterministic and nondeterministic computation is by Vectorization: OmenBreeze - Own work based on: Difference between deterministic and Nondeterministic.png by Eleschinski 2000, CC BY-SA 3.0, <a href="https://commons.wikimedia.org/w/index.php?curid=84727658">https://commons.wikimedia.org/w/index.php?curid=84727658</a> . . . . .	14
2.4	In this depiction of a many-one reduction from $\mathcal{L}'$ to $\mathcal{L}$ , $\mathcal{A}$ is a function algorithm under a particular computational resource bound. . . . .	18
2.5	This differentiation of the hardness of problems in NP depending on whether $P = NP$ is by Behnam Esfahbod, CC BY-SA 3.0, <a href="https://commons.wikimedia.org/w/index.php?curid=3532181">https://commons.wikimedia.org/w/index.php?curid=3532181</a> . . . .	19
2.6	This depiction of an oracle Turing machine is by Fschwarzentruber - Own work, CC BY-SA 4.0, <a href="https://commons.wikimedia.org/w/index.php?curid=51793814">https://commons.wikimedia.org/w/index.php?curid=51793814</a> . . . . .	20
2.7	This depiction of the relationships between randomized complexity classes and some other common complexity classes is by Bilorv - Own work, CC0, <a href="https://commons.wikimedia.org/w/index.php?curid=91765136">https://commons.wikimedia.org/w/index.php?curid=91765136</a> . . . . .	25





# List of Tables

- 5.1 A summary of the additions of this thesis to the line of work showing that is it hard to show the hardness of MCSP and various variants on the problem. \*For certain functions  $\sigma$ . . . 70



# Acknowledgments

Thank you to my advisor, Professor Pippenger, for expertly guiding me throughout this thesis journey and being excited to take a chance on an unfamiliar topic, as well as for introducing me to complexity theory in the first place and leading me to discover my passion for it. Additionally, many thanks to my reader, Professor Ran, for providing excellent feedback, as well as for introducing me to theoretical computer science more broadly and its exciting epistemological framework.



# Chapter 1

## Introduction

How hard are computational problems to solve on average, not just in the worst case? What is the complexity of figuring out the circuit complexity of Boolean functions? Can we have truly secure cryptography? These may initially seem to be very different questions, but recent work on the Minimum Circuit Size Problem has shown that the resolution of any one of them may give us the answers to all three.

The Minimum Circuit Size Problem (MCSP) is a problem with a long history in computational complexity theory which has recently experienced a resurgence in interest. MCSP asks the question: given the truth table of a Boolean function  $f$  and size parameter  $s$ , is there a circuit of size no greater than  $s$  which computes  $f$ ? (The size of a circuit is simply the number of gates in it.) Thus, in a certain sense, the complexity of MCSP captures how hard it is to determine the circuit complexity of Boolean functions.

MCSP was initially a subject of interest in the USSR starting in the 1950's as a potential problem that had no better algorithm than brute force search, before the framework of complexity theory as we know it today was introduced. In the early '70's, both Leonid Levin and Stephen Cook independently published papers (on the opposite sides of the Iron Curtain) introducing the concept of NP-completeness, which is central to complexity theory. Levin has said he delayed publishing his seminal paper because he wanted to show that MCSP is NP-complete as well (see Allender et al. (2011)). NP is a class of problems that (complexity theorists think) includes problems which are hard to solve. Roughly speaking, an NP-complete problem is in NP and can be used to efficiently solve any problem in NP via what's called a reduction, which converts answers to one problem into an answer to the other. Thus an NP-complete problem encapsulates the hardest problems in

NP. It is fortuitous that Levin went ahead to introduce this concept without the additional result, as whether MCSP is NP-complete is still open to this day.

Why do we even care whether MCSP is NP-complete? In addition to this history that goes back to before the idea of NP-completeness was introduced, in recent years MCSP has been shown to have connections to many areas, such as circuit complexity, Kolmogorov complexity, average-case complexity, proof complexity, pseudorandomness, cryptography, learning, and more. We will examine the increasingly central role MCSP seems to play in several of these areas using the framework of Impagliazzo's five worlds.

Two special properties of MCSP lead to many of these connections. First, it is a meta-computational question. The study of the complexity of MCSP is a study of the meta-mathematics of complexity, while complexity theory is already a meta project. In determining the complexity of MCSP, we seek to determine the complexity of determining the circuit complexity of Boolean functions. Thus, studying MCSP can be seen as a next step on the historical path that took us from trying to fulfill Hilbert's program, to computability, and then to complexity theory. This meta nature is not only philosophically interesting, but also leads MCSP to behave in some strange and interesting ways. Second, MCSP can distinguish order from chaos, in the following sense. While the Boolean functions we are practically interested in utilizing necessarily have small circuits, it is a well-known result that a randomly selected Boolean function will have high circuit complexity with high probability. MCSP thus allows us to distinguish random Boolean functions from pseudorandom ones (which attempt to use small circuits to imitate truly random functions), a property that suggests its connections to cryptography and other areas.

These philosophically flavorful properties lead MCSP to have relevance in many domains, and the implications it has for them are shaped by whether MCSP and approximation variants of it are actually NP-complete or not under various kinds of reductions. Researchers in the area widely believe MCSP is indeed NP-complete under at least some general types of reductions. So, with all of this motivation, why haven't we been able to prove that MCSP is NP-complete?

In 2000, a paper by Kabanets and Cai showed that if MCSP is NP-complete under "natural"<sup>1</sup> reductions like those that had shown the NP-completeness of virtually every such problem thus far, then this would lead to dramatic

---

<sup>1</sup>What this means precisely will be discussed in more detail later.

results in complexity theory that seem well beyond current techniques (such as showing strong circuit lower bounds). This resparked interest in MCSP, and a line of work has continued to explore the implications of various kinds of reductions being used to show the NP-completeness of MCSP. These implications are often so strong and beyond current complexity-theoretic approaches that they are widely interpreted as showing that these kind of reductions are intractable for us to come up with given current techniques.

Thus, these results suggest that it is “hard” to determine the hardness of MCSP, which itself determines how hard Boolean functions are, adding another meta layer. Establishing the implications of MCSP being shown to be NP-complete under different kinds of reductions confronts the question: what mathematical barriers are there to using mathematics to understand how hard it is for computation to determine how hard it is to compute functions? In this thesis, I will seek to add to this line of work and continue to better understand under which kinds of reductions it might be more or less possible to show that MCSP or various variants on the problem are NP-complete.

A note before we continue. If you have taken a course on complexity theory, then the second chapter will serve as a refresher, concluding with a few complexity classes which may or may not be familiar and a formal introduction of the Minimum Circuit Size Problem. If you have take an introductory course like Computability and Logic, then this chapter will serve as a partial refresher on computability which also introduces the ideas from complexity theory that will be needed for future chapters. If you have taken neither, then hopefully it will provide a rough understanding of basic complexity theory which will allow you to to get the big idea of what this thesis is about and why it is interesting. It is my intention that in all three of these cases, the second chapter will allow the third to be generally understandable, at least enough to get a sense for the motivation behind studying MCSP. While all of the necessary concepts will have been introduced, the fourth chapter will be most approachable for someone already familiar with complexity theory, so if this is not the case for you I recommend not getting bogged down in the details. The concluding chapter will provide a brief summary and discuss potential future directions.





## Chapter 2

# Wait, What is Computational Complexity?

### 2.1 Origins

In order to proceed in this thesis, we first have to understand the overarching framework it operates within: computational complexity theory. Computational complexity theory is a relatively young subfield at the intersection of computer science and mathematics. One way we can trace its historical origins is back to Hilbert's program and the blows it received in the mid-1900's. Much of this history can be found in Doxiadis and Papadimitriou (2009).

Going into the 20th century, mathematics was in turmoil. For over two thousand years, Euclid's *Elements* had set the standard for proof, but now it was coming under increasing scrutiny for assuming that its axioms were obviously true and relying too much on intuition to bridge logical gaps. Many new areas of mathematics were challenging this old understanding of axioms, going back to the necessity of using imaginary numbers to understand even real polynomials, and continuing with non-Euclidean geometries and the characterization of infinity in set theory in the 1800's. This challenged mathematics' understanding of its relationship with certainty and ultimate capital-T Truth, so in order to shore up the place of mathematics as the Queen of the Sciences, there was an increasing emphasis on mathematics as the rigorous study of logically sound axiomatic systems. While the axioms themselves no longer had any particular relationship to Truth, any conclusions that were derived from them would be logically required.

The most ambitious version of this new vision of mathematics and

its quest for secure foundations was encapsulated in Hilbert's program, promoted by famous mathematician David Hilbert (1862-1943). Hilbert's program sought a broad formal system for all mathematics that could prove its own consistency (the system entails no logical contradictions), completeness (all true statements in the system have a proof), and decidability (we can efficiently determine whether a statement follows from the axioms via some sort of algorithm). This would banish from mathematics all dependence on intuition, ensure its logical soundness and completeness, and allow proofs to be mechanistically generated. The paradigm of Hilbertian formalism is encapsulated by the conclusion of an influential talk Hilbert gave in 1930:

For the mathematician there is no Ignorabimus, and, in my opinion, not at all for natural science either. ... The true reason why [no-one] has succeeded in finding an unsolvable problem is, in my opinion, that there is *no* unsolvable problem. In contrast to the foolish Ignorabimus, our credo avers: We must know, We shall know. (Dawson (1997))

This (to some) utopian dream was shattered when Kurt Gödel (1906-1978) presented his two Incompleteness Theorems less than a year later. The mathematical community anticipated that Gödel had finally shown completeness, but they got the opposite. His First Incompleteness Theorem informally states that for any mathematical system that can describe basic fundamental objects such as the natural numbers or sets, if it is consistent there will necessarily be true statements that cannot be proven. Namely, it is clear that any system which can formally express "this statement is unprovable" must either be inconsistent or incomplete. Perhaps even more damningly, Gödel's Second Incompleteness Theorem shows if any such system could prove its own consistency it would be inconsistent, implying that we can never know whether the most fundamental axiomatic system upon which we rest mathematics is even logically consistent in the first place. After the talk in which Gödel presented these results, the mathematician John von Neumann (1903-1957) famously exclaimed "it's all over!"

But it wasn't all over—this was just the beginning. The very limitations of mathematics and computation, shown by their own methodologies, birthed the fields of computability theory and complexity theory to follow. Gödel left one component of Hilbert's program alive, the Entscheidungsproblem, or Decision Problem. Perhaps it would still be possible to use algorithms to

decide whether a statement was provable using the system's axioms, even though not all true statements would be. As Alan Turing (1912-1954) was studying the foundations of mathematics in the early 1930's, he became interested in this problem. In order to answer it, he first had to come up with a formal definition of an algorithm—which we now call the Turing machine. In his famous 1936 paper, Turing used this notion to reprove Gödel's Incompleteness Theorems in a simpler and more intuitive way and answer the Entscheidungsproblem in the negative, giving the final blow to Hilbert's program. In the same stroke, this laid the foundations for the whole field of computability, which examines what can be computed by Turing machines. It wasn't the end for von Neumann either, who himself went on to become a famous computer scientist and whose von Neumann architecture, drawing on the work of Turing, remains the basis of computer design to this day. Von Neumann even credited Turing's paper on the Entscheidungsproblem as giving the "fundamental conception" of the modern computer itself.

Following the development of computability theory and the increasing use of modern computers, it became clear that a more discerning mathematical theory of which problems are computationally tractable was needed. Something being computable isn't very useful if it will take longer than the age of the universe for the answer to actually be computed. Thus computational complexity theory was born. This field studies the resources, such as time and memory, needed to solve problems via computation, and in particular the complicated relationships between classes of problems that can be solved within different computational resource bounds, called complexity classes. (NP is an example of a particularly important complexity class.) Complexity theory thus extends and complicates computability theory's epistemological framework for what problems can be solved via mathematics and computation when we examine them using these fields' own metrics and methodologies.

## 2.2 Formal framework

Now that we have the historical background of computational complexity theory's development providing us with context and motivation, we will define some of its foundational concepts which will be used throughout the rest of this work.

### 2.2.1 Languages and models of computation

Computational complexity theory (like computability theory) is based around *decision problems*, or “yes”-“no” problems, which can be asked in sequence in order to answer more complicated questions and can be expressed as *languages* over particular *alphabets*. Recall that for a set  $S$ ,  $S^*$  (where  $*$  is the Kleene star) consists of all strings which are made up of concatenated elements of  $S$  (including the empty string).

**Definition 2.1.** *An alphabet  $\Sigma$  is a finite set, whose members are called symbols. A language  $\mathcal{L}$  over an alphabet  $\Sigma$  is a subset of  $\Sigma^*$ . Viewing  $\mathcal{L}$  as a decision problem, for each  $x \in \Sigma^*$ , if  $x \in \mathcal{L}$ , then it is called a “yes” instance, while if  $x \notin \mathcal{L}$ , it is a “no” instance.*

We will use  $\Sigma = \{0, 1\}$ , and thus utilize binary strings to encode our problems. Notice that given a certain finite number of symbols in our alphabet, a fixed number of bits can be used to encode each symbol, so restricting ourselves to binary strings will prove immaterial. (Though it is very important that we only allow finite alphabets.) Thus, for us a language will be some particular subset of binary strings.

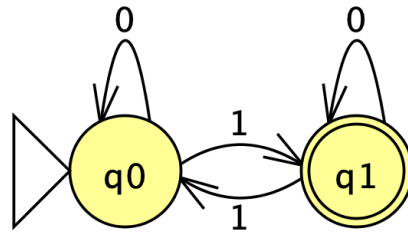
A simple example of a language is PARITY, which consists of all binary strings with an odd number of 1’s. It thus consists of all strings with a parity of 1, giving us a decision problem which asks what the parity of a binary string is. Two more complicated yet ultimately illustrative languages which we will discuss further later in this section are CIRCUIT EVALUATION and CIRCUIT SATISFIABILITY. Here we mean logical circuits with the usual AND, OR, and NOT gates which take an  $n$ -ary binary input and output either 0 or 1. CIRCUIT EVALUATION asks, given such a circuit  $C$  (in some standard encoding) and a binary input  $x$ , does  $C$  output 1 on  $x$  (i.e., does  $C[x] = 1$ )? Thus more precisely, CIRCUIT EVALUATION consists of the set of binary strings which encode a circuit  $C$  followed by a binary input  $x$  such that  $C[x] = 1$ . On the other hand, CIRCUIT SATISFIABILITY asks, given such a circuit  $C$ , does there exist *any* input  $x$  such that  $C[x] = 1$ ? While this latter question is still certainly computable—we could check all possible inputs—it seems like a potentially much harder problem, which complexity theory will allow us to formalize.

In order to do so however, we first need to establish what it means for an algorithm to decide a language—and what an algorithm even is! As we’ve alluded to, the most common general model of computation in complexity theory is the Turing machine. In order to discuss Turing machines, we first

need to introduce a much simpler model of computation, the *finite automaton*.

**Definition 2.2.** A finite automaton is a tuple  $(\Sigma, Q, q_0, F, \delta)$  consisting of an input alphabet  $\Sigma$ , a finite set of states  $Q$ , a unique starting state  $q_0$ , a subset of accepting states  $F \subseteq Q$ , and a transition function  $\delta : Q \times \Sigma \rightarrow Q$ . The language accepted by such a finite automaton is the set of strings in  $\Sigma^*$  such that starting in state  $q_0$  and following  $\delta$  based on the current state and the next unread symbol in the input string, when there are no more symbols left to read the automaton is in an accepting state  $q \in F$ .

One can thus think of a finite automaton as modelling a computer with finite memory. See the following diagram of a simple finite automaton solving PARITY.

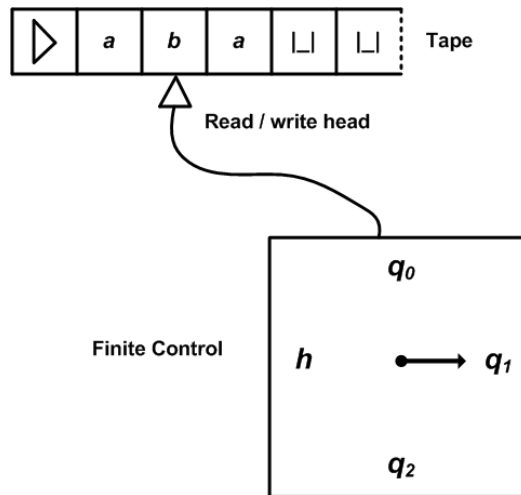


**Figure 2.1** This finite automaton decides the language PARITY over the alphabet  $\{0, 1\}$ . Note that the triangle indicates  $q_0$  is the start state, the double circle indicates  $q_1$  is an accepting state, and the arrows define the transition function. The program JFLAP was used to produce this image.

However, it is not too difficult to see that finite automata cannot solve all decision problems, even very simple ones such as  $\mathcal{L}_{01} = \{0^k 1^k \mid k \in \mathbb{N}\}$ . Consider the set  $S = \{0^k \mid k \in \mathbb{N}\}$ . If we take any two members of  $S$   $0^k$  and  $0^{k'}$  for  $k \neq k'$ , appending  $1^k$  to the end of each causes the first to become an accepted string of  $\mathcal{L}_{01}$  and the latter to become a rejected one. Thus if we have a finite automaton for  $\mathcal{L}_{01}$  which correctly deals with these two inputs,  $0^k$  and  $0^{k'}$  must have led to different states because otherwise  $0^k 1^k$  and  $0^{k'} 1^k$  would end up at the same state and either both be accepted or both be rejected. But since this holds for any pair from  $S$ , each member of  $S$ , an infinite set, must lead to a different state, contradicting the automaton being finite. (A generalization of this argument for any language with a pairwise distinguishable set of this kind is sometimes called the distinguishability

theorem.) This limitation with finite automata comes down to the fact that they have a fixed amount of memory for any input, and for languages like  $\mathcal{L}_{01}$ ,  $k$  might be arbitrarily large and thus we will have to count arbitrarily high—though always a reasonable amount given the input length.

Any computer can be represented as a (very complicated) finite automaton, but it seem ludicrous to stop here and conclude that such a simple language as  $\mathcal{L}_{01}$  isn't even computable! To address this, we can add access to an unbounded amount of memory, though only a finite amount of it will be used when processing a given single input (as long as our machine halts). Namely, we can give the finite automaton a one-way infinite tape made up of cells which it can write symbols from a working alphabet on. It will have a tapehead on a particular cell so that it can use the symbol it sees there to decide which state to transition to, what symbol to write in that cell (perhaps the same one again), and whether to move the tapehead a cell to the left, a cell to the right, or stay put. The tape will start with the input string written on it and the rest blanks, and the tapehead will start positioned at the first symbol of the input.



**Figure 2.2** This depiction of a Turing machine in action was sourced from [https://commons.wikimedia.org/wiki/File:Turing\\_machine.png](https://commons.wikimedia.org/wiki/File:Turing_machine.png).

**Definition 2.3.** A Turing machine is a tuple  $(\Sigma, \Gamma, Q, q_0, q_{\text{accept}}, q_{\text{reject}}, \delta)$  where  $\Sigma$  is the input alphabet,  $\Gamma$  is the working alphabet,  $Q$  is a finite set of states,  $q_0$  is the unique starting state,  $q_{\text{accept}}$  and  $q_{\text{reject}}$  are the unique accepting and rejecting states respectively, and  $\delta : Q \setminus \{q_{\text{accept}}, q_{\text{reject}}\} \times \Gamma \rightarrow Q \times \Gamma \times \{L, R, S\}$  is the

transition function. The language  $\mathcal{L}$  accepted by such a Turing machine  $M$  is the set of strings in  $\Sigma^*$  such that  $M$  eventually enters  $q_{\text{accept}}$  (and then halts since there is no transition out of this state) when following this procedure: Start in state  $q_0$  with the input string  $x \in \Sigma^*$  written on the tape (one symbol per cell), the rest of the tape cells blank (we require that this blank symbol to be in  $\Gamma$  but not  $\Sigma$  so that the end of the input can be distinguished), and the tapehead looking at the cell containing the first symbol of  $x$ . Then using the current state and the symbol on the cell the tapehead is looking at, based on  $\delta$ , transition to a new state, write a symbol on the current cell, and move the tapehead to either the cell to the left, to the right right, or let it stay where it is. In general, we say  $M$  semidecides  $\mathcal{L}$ . If  $M$  always eventually enters  $q_{\text{reject}}$  for every string not in  $\mathcal{L}$  (i.e., that it doesn't accept), then we say it decides  $\mathcal{L}$ .

This gives us access to an unbounded number of possible configurations by combining different states and symbols written on the tape, while finite automata only have access to a fixed number. This definition is also intuitively appealing, as it mirrors how a mathematician might figure out a problem step-by-step using a piece of paper to write things out. We can think of a Turing machine as a computer with access to as much time and memory as it desires.

However, in order to understand Turing machines from a complexity perspective as well as a computational one, we also need to consider how to measure, and then bound, the use of computational resources on a Turing machine. The two primary computational resources here are *time* and *space*, which each have relatively intuitive definitions.

**Definition 2.4.** *The time  $t(M, x)$  taken by a Turing machine  $M = (\Sigma, \Gamma, Q, q_0, q_{\text{accept}}, q_{\text{reject}}, \delta)$  on input  $x \in \Sigma^*$  is the number of transitions made by  $\delta$  on  $x$  before it enters  $q_{\text{accept}}$  or  $q_{\text{reject}}$  and halts. The space  $s(M, x)$  taken by  $M$  on  $x$  is the number of cells visited by the tapehead over the course of the computation before halting.*

A Turing machine may not halt on a given input and thus not decide any language—indeed the Halting Problem, which asks whether a given Turing machine will halt on a given input, is the notorious original undecidable problem in computability—so  $t(M, x)$  or  $s(M, x)$  can be infinite. To examine finer notions of complexity, in this work we will be interested in situations where the Turing machine will halt on all inputs, deciding some language, and will do so while respecting a bound on the time or space it uses that is based only on the length of the input  $n = |x|$ . We can think of this as bounding the Turing machine to only use an amount of time or space



which is reasonable given the length of the input—if we seek for a computer to handle an input of a certain length, it must be able to operate within reasonable amounts of time and memory compared to that length. We will denote the output of a Turing machine  $M$  on input  $x$  by  $M(x)$ , so that if  $M$  enters the accepting state when run on input  $x$ ,  $M(x) = 1$ , while if it enters the rejecting state,  $M(x) = 0$ . If  $M$  does not halt, this value is undefined, but we will not encounter this scenario.

### 2.2.2 The complexity classes P and NP

With this background, we can at last define particular *complexity classes* which will be of interest to us. In general, complexity classes are sets of languages which can be solved by algorithms (of some certain computational model) which respect some particular computational resource bound, often on time or space. We will first introduce the two most important complexity classes in complexity theory.

**Definition 2.5.** *The complexity class*

$$P = \{\mathcal{L} \subseteq \{0, 1\}^* \mid \exists k \in \mathbb{N} \text{ such that } \mathcal{L} \text{ is decided by a Turing machine } M \\ \text{where } \forall x \in \{0, 1\}^*, t(M, x) \leq |x|^k\}.$$

Using big-O notation, we can also describe P by specifying  $t(M, x) \in O(|x|^k)$  for some  $k \in \mathbb{N}$  (this is equivalent since the finite number of small  $x$  such that the bound doesn't apply can be hardcoded into  $M$  so that  $t(M, x) \leq |x|^{k+1}$  always holds). Thus P is the set of languages which can be solved in polynomial time on a Turing machine, and it is often thought of as the set of decision problems we can solve within a tractable amount of time. Once we reach problems that take more time than this (such as time  $2^{|x|}$ , or exponential time, on some inputs) that is generally no longer considered tractable. Notice that this is understood with respect to *worst-case* analysis, since a language could be solved by a Turing machine that takes polynomial time on most inputs and yet exponential time on some others, and this would not be sufficient for the language to be in P.

The other of the two most important complexity classes is closely related to P.

**Definition 2.6.** *The complexity class*

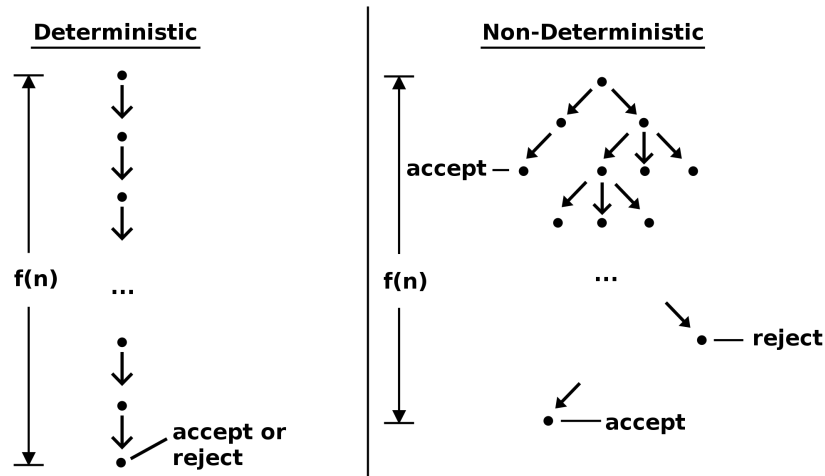
$$\text{NP} = \{ \mathcal{L} \subseteq \{0, 1\}^* \mid \exists \text{ language } \mathcal{L}' \in \text{P and constant } k \in \mathbb{N} \text{ where } \forall x \in \{0, 1\}^*, \\ x \in \mathcal{L} \iff \exists y \in \{0, 1\}^* \text{ such that } |y| \in O(|x|^k) \text{ and} \\ (x, y) \in \mathcal{L}' \}.$$

With a bit of examination, we can see that NP corresponds to the set of languages with solutions or proofs which can be *verified* in polynomial time: Notice that it makes sense to think of  $y$  as a proof that  $x \in \mathcal{L}$  with  $\mathcal{L}'$  as the verifier of the proof. First,  $\mathcal{L}'$  will not accept  $(x, y)$  for any  $y$  if  $x \notin \mathcal{L}$ , so if  $\mathcal{L}'$  accepts  $(x, y)$ , this means we must have  $x \in \mathcal{L}$ . Thus  $\mathcal{L}'$  is a sound verifier. It is also complete, since for any  $x \in \mathcal{L}$ , there will be some proof  $y$  that  $\mathcal{L}'$  will accept.

It is often helpful to consider a second characterization of NP, which also clarifies its name. To do so, first we must define *nondeterministic* Turing machines (see Figure 2.3).

**Definition 2.7.** *For a set  $S$ , let  $\mathcal{P}(S)$  be the power set of  $S$ , consisting of all subsets of  $S$  (including the empty set). A nondeterministic Turing machine is a tuple  $(\Sigma, \Gamma, Q, q_0, q_{\text{accept}}, q_{\text{reject}}, \delta)$  where  $\Sigma$  is the input alphabet,  $\Gamma$  is the working alphabet,  $Q$  is a finite set of states,  $q_0$  is the unique starting state,  $q_{\text{accept}}$  and  $q_{\text{reject}}$  are the unique accepting and rejecting states respectively, and  $\delta : Q \setminus \{q_{\text{accept}}, q_{\text{reject}}\} \times \Gamma \rightarrow \mathcal{P}(Q \times \Gamma \times \{L, R, S\})$  is the transition function. The language  $\mathcal{L}$  accepted by such a Turing machine  $M$  is the set of strings in  $\Sigma^*$  such that for some sequence of guesses (to be described)  $M$  eventually enters  $q_{\text{accept}}$  (and then halts) when following this procedure: Start in state  $q_0$  with the input string  $x$  written on the tape (one symbol per cell) and the tapehead looking at the first symbol of  $x$ . Then using the current state and the symbol on the cell the tapehead is at, based on guessing one of the state-symbol-movement possibilities in the set given by  $\delta$ , transition to a new state, write a symbol on the current cell, and move the tapehead to either the cell to the left, to the right right, or let it stay where it is. In general, we say  $M$  semidecides  $\mathcal{L}$ . If for every string not in  $\mathcal{L}$ , for all sequences of guesses  $M$  eventually enters  $q_{\text{reject}}$ , then we say it decides  $\mathcal{L}$ .*

We can think of a nondeterministic Turing machine as guessing which of several possible computational paths available to it it wants to take; if there is some sequence of guesses which causes it to reach  $q_{\text{accept}}$  then it accepts (such a path is called an accepting path), else it rejects. Note that time and space for nondeterministic Turing machines are each defined by the highest deterministic value of these along any computational path the machine can



**Figure 2.3** This comparison of deterministic and nondeterministic computation is by Vectorization: OmenBreeze - Own work based on: Difference between deterministic and Nondeterministic.png by Eleschinski 2000, CC BY-SA 3.0, <https://commons.wikimedia.org/w/index.php?curid=84727658>.

take on the given input. Thus in order to halt, a nondeterministic Turing machine must halt along every potential computational path, and in order to run in polynomial time, it must also do so along all computational paths. Now we can see that NP is even more closely related to P than was initially apparent, as we can also define it as follows.

**Definition 2.8.** *The complexity class*

$$\text{NP} = \{ \mathcal{L} \subseteq \{0, 1\}^* \mid \exists k \in \mathbb{N} \text{ s.t. } \mathcal{L} \text{ is decided by a nondeterministic Turing machine } M \text{ where } \forall x \in \{0, 1\}^*, t(M, x) \in O(|x|^k) \}.$$

We can now make sense of the full names of P and NP, which stand for (deterministic) *Polynomial time* and *Nondeterministic Polynomial time* respectively. It also is not too difficult to see that the two definitions for NP are equivalent. If we have  $\mathcal{L} \in \text{NP}$  by the first definition, then to show that it is also in NP by the second definition, simply use the nondeterminism to guess  $y$  and then run the original  $M$  on it to verify whether  $M((x, y)) = 1$ —since  $M$  is a sound and complete verifier, we will find such a  $y$  and accept if and only if the input is in  $\mathcal{L}$ . For the other direction, we can choose to interpret  $y$  as identifying a particular computational path of  $M$  and make the appropriate choices to simulate the original  $M$  along this path—if  $M$  accepts nondeterministically,

there will be some appropriate  $y$  that identifies the computational path which leads to  $q_{accept}$ .

To get a better feel for what problems are in P versus NP, we will consider a particularly pertinent language in each, namely CIRCUIT EVALUATION in P and CIRCUIT SATISFIABILITY in NP, which are representative of the hardness of each class in a certain technical sense. To define these languages a bit more formally, recall that a logical or Boolean circuit  $C$  operates on a binary input of a fixed length  $n$  using AND, OR, and NOT gates, outputting either 0 or 1 on each input; for a given  $x \in \{0, 1\}^n$ , this is denoted by  $C[x] = 0$  or  $C[x] = 1$  respectively. We can fix some reasonable binary encoding of such circuits which records the various gates used and their inputs. We will denote the encoding of a logical circuit  $C$  by  $\langle C \rangle$ . Then

**Definition 2.9.**

$$\text{CIRCUIT EVALUATION} = \{(\langle C \rangle, x) \in \{0, 1\}^* \mid C[x] = 1\}$$

where the list  $(\langle C \rangle, x)$  is encoded in some particular way in binary

and

**Definition 2.10.**

$$\text{CIRCUIT SATISFIABILITY} = \{\langle C \rangle \in \{0, 1\}^* \mid \exists x \in \{0, 1\}^n \text{ s.t. } C[x] = 1\}.$$

It is not too difficult to see that CIRCUIT EVALUATION can be solved in P by scanning over the encoding of the circuit, following through what each gate will compute on the particular input until you get the ultimate output. This then makes it easy to see that CIRCUIT SATISFIABILITY  $\in$  NP. For  $\langle C \rangle \in$  CIRCUIT SATISFIABILITY, we can give a proof that is simply the input  $x$  to the circuit which causes it to be satisfied, which we can verify using CIRCUIT EVALUATION  $\in$  P. Thinking nondeterministically, this corresponds to guessing the input which will satisfy  $\langle C \rangle$  and then using CIRCUIT EVALUATION to check our guess. If  $\langle C \rangle \in$  CIRCUIT SATISFIABILITY, at least one of our guesses will work, and if not, none of them will. However, it is not clear how to go from this easy verification procedure to actually *solving* CIRCUIT SATISFIABILITY in polynomial time. The naïve solution is simply to go through all  $2^n$  possible inputs, but for many circuits this is exponential in the size of its encoding. While one can do a bit better than this by applying some tricks, there is no known strictly subexponential algorithm for the problem. We have thus arrived at the question: does  $P = NP$ ? While complexity theorists widely believe the answer

is no, proving this is the biggest open problem in complexity theory, and has been for the last 50 years.

Before we go on, it is important to note that there are computable languages which are neither in P nor in NP. For instance, SUCCINCT CIRCUIT SATISFIABILITY, which is like CIRCUIT SATISFIABILITY except that the circuit in question is not encoded as a string, but instead using another circuit, is provably not in NP. Roughly speaking, since the input can be encoded much more succinctly than before, we no longer have enough time to verify this problem within NP. To give a more intuitive example of a language which is likely in neither, CIRCUIT UNSATISFIABILITY asks whether *all* input strings leave the given circuit unsatisfied. (Thus amongst the set  $S$  of all strings which validly encode circuits,  $\text{CIRCUIT UNSATISFIABILITY} = S \setminus \text{CIRCUIT SATISFIABILITY}$ .) While again for a circuit with  $n$  input variables we can naïvely use CIRCUIT EVALUATION to check all  $2^n$  possible inputs, taking potentially exponential time, it is not clear how to provide a short proof that this circuit outputs zero on *every* input. One can likewise do better here than checking all inputs, but it is nevertheless widely believed this problem is not even in NP.

### 2.2.3 Completeness and Reductions

Now, in order to better understand the question of whether  $P = NP$ , we will consider the following one: what does it mean that CIRCUIT EVALUATION and CIRCUIT SATISFIABILITY are “representative of the hardness” of P and NP respectively? It turns out that they are what are called *complete languages* for each class. As the following definition makes clear, the notion of a complete class rests on the definition of a certain kind of *reduction*. Intuitively speaking, a reduction can be thought of as an efficient way to use solutions to one problem to solve another (i.e. to reduce solving one problem to solving a different one).

**Definition 2.11.** A language  $\mathcal{L}$  is complete for a complexity class  $C$  under  $R$ -reductions if

- $\mathcal{L} \in C$ , and
- for every language  $\mathcal{L}' \in C$ , there is an  $R$ -reduction from  $\mathcal{L}'$  to  $\mathcal{L}$ .

Thus under the appropriate kind of reduction  $R$ , an efficient algorithm for a complete language  $\mathcal{L}$  could be used to efficiently solve *any other problem* in  $C$  as well. A complete language can be thought of as a hardest problem in  $C$

and is thus representative of the hardness of the complexity class overall. We can also call a language *hard* for a class, in which case it fulfills the second point of the definition for completeness but is not necessarily in the complexity class itself, in which case it is as hard *or harder* than any language in the class.

To understand this better, we will define the two main types of reductions, starting with the *many-one* (a.k.a. *Karp*) reduction. Many-one refers to the fact that such a reduction is a mapping from problems of one kind to another which is a function (the one) which may not be injective (the many). To formally define this, first notice that we can modify the definition of a Turing machine slightly so that it outputs a binary string instead of just accepting and rejecting, in which case we call it a *function Turing machine*. We can add an extra tape (with its own tapehead) that we can use to write the string we wish to output. This is restricted to be right-only (and thus write-only) such that once a non-blank symbol is written on a cell of the output tape, we immediately move our tapehead a cell to the right so we are ready to write the next symbol and all of the previous symbols are immutable. We also use a single  $q_{halt}$  state, instead of  $q_{accept}$  and  $q_{reject}$ , which signals when we have finished producing the output and halted. The output string produced by such a function Turing machine  $M$  on input  $x$  will be denoted by  $M(x)$ . Further, it is convenient to use a separate read-only input tape, whereupon the input will be provided and which cannot be altered, in addition to a work tape upon which we can both read and write as normal. We then only measure the amount of space used on the work tape when considering  $s(M, x)$ . This will allow us to consider function Turing machines that use sub-linear space, or less space than the input itself takes up (most commonly, logarithmic space). This is indeed possible: for instance, we could have a function Turing machine which outputs the length of the input in unary, in which case we don't need to use any space on the work tape at all. This is an example of a function problem, where instead of asking for a "yes"- "no" response to each input, we ask for a specific output in binary. We can define function complexity classes with respect to function problems by asking for all function problems which can be computed by a function Turing machine operating within certain computational resource bounds. Now,

**Definition 2.12.** Let  $t : \mathbb{N} \rightarrow \mathbb{N}$ ,  $s : \mathbb{N} \rightarrow \mathbb{N}$ . A  $t(n)$ -time many-one reduction from language  $\mathcal{L}'$  to  $\mathcal{L}$  is a function Turing machine  $M$  such that  $\forall x \in \{0, 1\}^*$ ,  $t(M, x) \leq t(|x|)$  and

$$x \in \mathcal{L}' \iff M(x) \in \mathcal{L}.$$

Similarly, a  $s(n)$ -space many-one reduction from language  $\mathcal{L}'$  to  $\mathcal{L}$  is a function Turing machine  $M$  such that  $\forall x \in \{0, 1\}^*$ ,  $s(M, x) \leq s(|x|)$  and

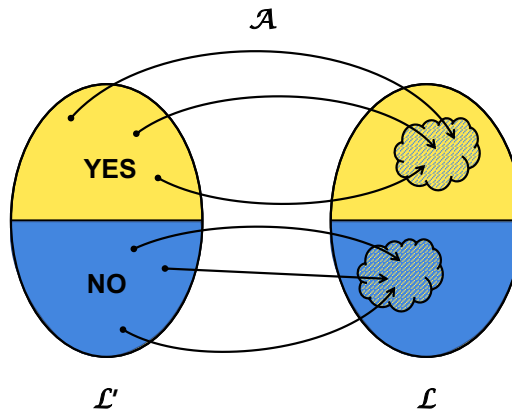
$$x \in \mathcal{L}' \iff M(x) \in \mathcal{L}.$$

Most generally, for a function complexity class  $C$ , a  $C$  many-one reduction from language  $\mathcal{L}'$  to  $\mathcal{L}$  is a function algorithm  $\mathcal{A}$  such that  $\mathcal{A}$  demonstrates that the function problem it computes is in  $C$  and  $\forall x \in \{0, 1\}^*$ ,

$$x \in \mathcal{L}' \iff \mathcal{A}(x) \in \mathcal{L}.$$

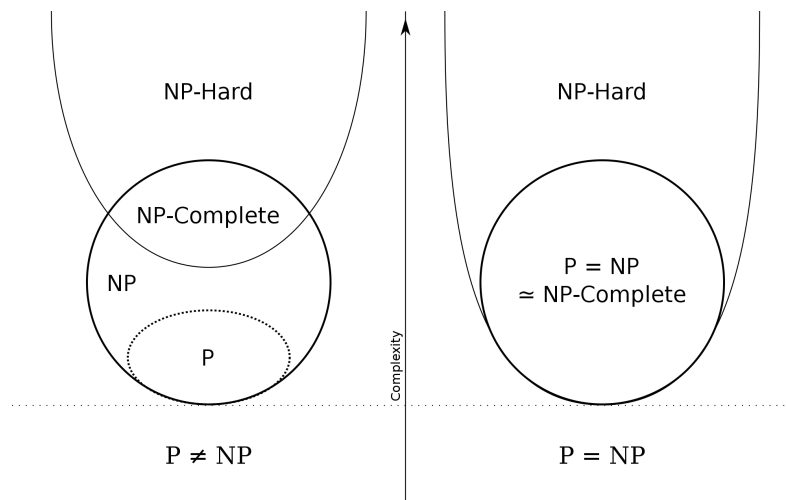
This algorithm is usually a function Turing machine, but can also be a different model of computation, such as a multi-output circuit family.

Circuit families as models of computation will be discussed later. In essence, a many-one reduction turns “yes” instances of one problem into “yes” instances of the other, and likewise with “no” instances. To get the general idea of a many-one reduction, consider the following figure.



**Figure 2.4** In this depiction of a many-one reduction from  $\mathcal{L}'$  to  $\mathcal{L}$ ,  $\mathcal{A}$  is a function algorithm under a particular computational resource bound.

To give a simple example of a many-one reduction, we will introduce a very well-known language which is closely related to one we have seen before, called SATISFIABILITY, or SAT for short. Recall that a Boolean expression on variables  $x_1, x_2, \dots, x_n$  combines these variables using the logical operators  $\wedge$ ,  $\vee$ , and  $\neg$ ; for example,  $x_1 \wedge \neg(x_2 \vee x_3)$ . SAT is the set of strings encoding Boolean expressions which evaluate to true for some setting of their  $x_i$  variables, and thus, for instance, includes  $\langle x_1 \wedge \neg(x_2 \vee x_3) \rangle$



**Figure 2.5** This differentiation of the hardness of problems in NP depending on whether  $P = NP$  is by Behnam Esfahbod, CC BY-SA 3.0, <https://commons.wikimedia.org/w/index.php?curid=3532181>.

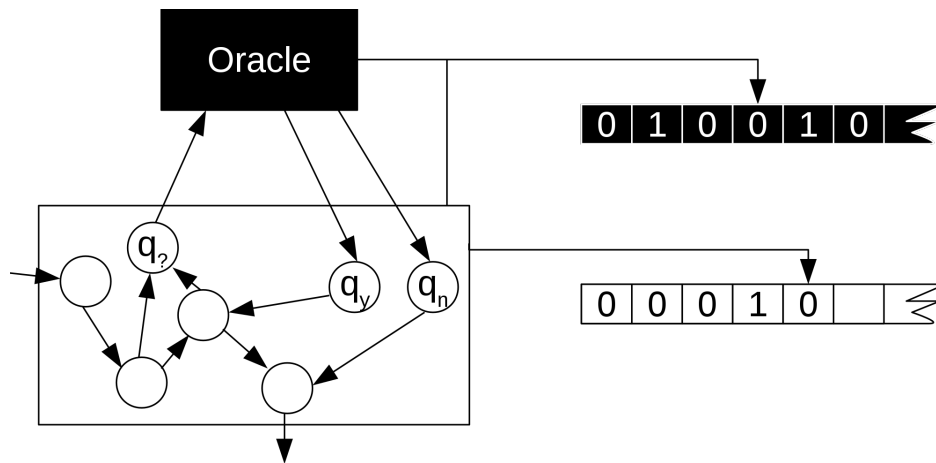
(set  $x_1 = 1$ ,  $x_2 = x_3 = 0$ ). SAT is NP since we can guess a satisfying assignment and then verify it in polynomial time. We can reduce SAT to CIRCUIT SATISFIABILITY, or CSAT for short, by converting the given Boolean expression into a Boolean circuit with AND, OR, and NOT gates in the obvious way, which can easily be done in polynomial time. SAT is actually the canonical NP-complete problem under polynomial-time many-one reductions, going back to the initial definition of these ideas by Levin and Cook. Indeed, the proposition that SAT is NP-complete is called the Cook-Levin theorem. Thus by providing a polynomial-time many-one reduction from SAT to CSAT, with the knowledge that SAT is NP-complete under polynomial-time many-one reductions, we have shown that CSAT is also NP-complete under polynomial-time many-one reductions (any problem in NP can be converted into a SAT problem which can be converted into a CSAT problem). This demonstrates the power of having at least one complete problem for a class identified, because now we can show another problem in the class is also complete by reducing an already-known complete problem to it.

We can see the importance of different kinds of reductions by further considering their connections to completeness. Notice that any nontrivial language is complete for P under polynomial-time many-one reductions because you have time to just solve the problem and output a fixed “yes”/“no”



instance. Thus it is only interesting to consider completeness for P under more restricted kinds of reductions. For instance, CIRCUIT EVALUATION is complete for P under logarithmic-space many-one reductions, and indeed CIRCUIT SATISFIABILITY is NP-complete under this restricted kind of reduction as well. On the other hand, if any language complete for NP under the more general polynomial-time many-one reductions was shown to be in P, then this would prove  $P = NP$ ! (For any problem in NP, we could convert it into the complete problem in polynomial time and then solve that problem in polynomial time, getting an answer to our original problem in polynomial time overall.) This changes the problem of P versus NP from requiring us to potentially examine whether *every* language in NP is in P to just having to determine whether *one* NP-complete language is in P or not. See Figure 2.5 for a depiction of the two different scenarios at hand. Note that if  $P \neq NP$ , then there are problems in NP which are neither in P nor NP-complete. These are called NP-intermediate, and would be contained in the gap between P and NP-complete problems on left side of the diagram.

A more general type of reduction is the *Turing* (a.k.a *Cook*) reduction. To understand this kind of reduction, we need to define oracle Turing machines. Such a Turing machine is depicted below.



**Figure 2.6** This depiction of an oracle Turing machine is by Fschwarzentruber - Own work, CC BY-SA 4.0, <https://commons.wikimedia.org/w/index.php?curid=51793814>.

An *oracle Turing machine* with access to a language  $\mathcal{L}$ , called the oracle, has three extra designated states,  $q_{query}$ ,  $q_{yes}$ , and  $q_{no}$ , as well as an additional query tape. It can write something on the query tape and then enter  $q_{query}$ ,

upon which it will automatically enter either the state  $q_{yes}$  or  $q_{no}$  depending on whether the string on the query tape is in  $\mathcal{L}$  or not, respectively. Thus it gets to automatically access whether a given string it writes out is in  $\mathcal{L}$  in a single step, without needing to do any computation to determine this as it would otherwise need to do. Note that depending on the oracle  $\mathcal{L}$  provided to a given oracle Turing machine, that machine may very well decide different languages. We can also define oracle complexity classes as follows. For a complexity class  $C$  (defined with respect to a computational resource bound on Turing machines) and language  $\mathcal{L}$ , the oracle complexity class  $C^{\mathcal{L}}$  is the set of languages which can be decided by an oracle Turing machine with access to  $\mathcal{L}$  which respects the same computational resource bound required by  $C$ . Then

**Definition 2.13.** *Let  $C$  be a complexity class. A  $C$  Turing reduction from language  $\mathcal{L}'$  to  $\mathcal{L}$  is an oracle Turing machine  $M$  that decides  $\mathcal{L}'$  when given an oracle for  $\mathcal{L}$  and demonstrates that  $\mathcal{L}'$  is in  $C^{\mathcal{L}}$ . In other words,  $\mathcal{L}'$   $C$ -Turing reduces to  $\mathcal{L}$  if  $\mathcal{L}' \in C^{\mathcal{L}}$ .*

Note that for two complexity classes  $C_1$  and  $C_2$ ,  $C_1^{C_2}$  denotes languages recognizable by a  $C_1$ -bounded oracle Turing machine with access to some particular oracle in  $C_2$ .

Now, notice that we can think of a many-one reduction as a very restricted kind of Turing reduction where we only ask the oracle one question at the very end and must answer the same way it does. However, it is still the case that if any language complete for NP under polynomial-time Turing reductions was shown to be in P, this would prove  $P = NP$ . In this case, we could simulate the oracle Turing machine of the reduction for a given language in NP by actually using our polynomial-time algorithm for  $\mathcal{L}$  to compute the answer to each query, keeping the simulation polynomial-time overall and thus solving the language in P.

This examination of reductions also further clarifies why we are so interested in P versus NP. Not only is the question of whether all problems with small proofs can also be solved efficiently philosophically interesting, it is also practically significant, as many important problems we would very much like to be able to solve efficiently are NP-complete under polynomial-time Turing reductions, and are thus solvable in P if and only if  $P = NP$  in general. Beyond SAT and CSAT, another famous example is the Traveling Salesperson Problem, and there are many, many others. Even just examining the question of P versus NP to this extent, it becomes clear why reductions and completeness are of central importance to complexity theory. They will

also be central to the specific topic of this thesis.

### 2.2.4 A few more complexity classes of note

Before we move on, we will need to know about a few more complexity classes. First, we will introduce the  $\Sigma_k\text{P}$  classes and the *Polynomial Hierarchy*, or PH.

**Definition 2.14.** Define  $\Sigma_0\text{P} = \text{P}$ . For  $k > 0$ , let  $\Sigma_k\text{P} = \text{NP}^{\Sigma_{k-1}\text{P}}$ . Then

$$\text{PH} = \bigcup_{k \geq 0} \Sigma_k\text{P}.$$

The first few levels of this hierarchy are thus  $\Sigma_0\text{P} = \text{P}$ ,  $\Sigma_1\text{P} = \text{NP}^{\text{P}} = \text{NP}$ ,  $\Sigma_2\text{P} = \text{NP}^{\text{NP}}$ ,  $\Sigma_3\text{P} = \text{NP}^{\text{NP}^{\text{NP}}}$ , etc. We can also understand these complexity classes using the idea of *alternation*. It will be useful for us to note that equivalently,

**Definition 2.15.** *The complexity class*

$$\begin{aligned} \Sigma_k\text{P} = \{ \mathcal{L} \subseteq \{0, 1\}^* \mid \exists \text{ language } \mathcal{L}' \in \text{P} \ \& \ \text{constant } k \in \mathbb{N} \ \text{where } \forall x \in \{0, 1\}^*, \\ x \in \mathcal{L} \iff \exists y_1 \in \{0, 1\}^* \ \forall y_2 \in \{0, 1\}^* \ \exists y_3 \in \{0, 1\}^* \\ \dots y_k \in \{0, 1\}^* \ \text{s.t. each } |y_i| \in O(|x|^k) \\ \text{and } (x, y_1, y_2, y_3, \dots, y_k) \in \mathcal{L}' \}. \end{aligned}$$

For  $k = 1$ , this definition matches our original definition of  $\text{NP} = \Sigma_1\text{P}$ , so we can see that the two definitions are clearly equivalent in that case. That this equivalence continues can be shown using induction. The Polynomial Hierarchy can thus be seen as a further generalization of the ideas that brought us from P to NP. To get further intuition about these classes, note that for  $k \geq 1$   $\text{QSAT}_k$  is complete for  $\Sigma_k\text{P}$ :

**Definition 2.16.**

$$\begin{aligned} \text{QSAT}_k = \{ \phi(x_1, x_2, x_3, \dots, x_k) \mid \phi \text{ is a Boolean formula, } x_i \in \{0, 1\}^*, \text{ and} \\ \exists x_1 \forall x_2 \exists x_3 \dots x_k \text{ such that } \phi(x_1, x_2, x_3, \dots, x_k) = 1 \}. \end{aligned}$$

We can think of  $\text{QSAT}_k$  as a game between two players, the  $\exists$  player and the  $\forall$  player, who are presented with a certain Boolean function  $\phi(x_1, x_2, x_3, \dots, x_k)$ . The  $\exists$  player goes first and sets  $x_1$ , which the  $\forall$  player can examine before going second and setting  $x_2$ , upon which they continue

to alternate turns setting the  $x_i$  in order until all  $k$  are set. The  $\exists$  player wins if  $\phi$  is satisfied at the end of the game, and the  $\forall$  player wins if  $\phi$  is left unsatisfied. Notice that the  $\exists$  player has a winning strategy exactly when  $\phi \in \text{QSAT}_k$ .

Additionally, a couple of randomized complexity classes will be useful for us. These can be defined using nondeterministic Turing machines. First is RP, which stands for Randomized Polynomial time.

**Definition 2.17.** *The complexity class*

$$\begin{aligned} \text{RP} = \{ \mathcal{L} \subseteq \{0, 1\}^* \mid \exists k \in \mathbb{N} \text{ such that there is a nondeterministic Turing} \\ \text{machine } M \text{ where } \forall x \in \{0, 1\}^*, t(M, x) \in O(|x|^k), \\ \geq \frac{1}{2} \text{ of } M(x)\text{'s computational paths accept if } x \in \mathcal{L}, \text{ and} \\ \text{all of } M(x)\text{'s computational paths reject if } x \notin \mathcal{L} \} \end{aligned}$$

or, equivalently,

$$\begin{aligned} \text{RP} = \{ \mathcal{L} \subseteq \{0, 1\}^* \mid \exists \text{ a language } \mathcal{L}' \in \text{P taking 2-tuple input } (x, y), \\ k \in \mathbb{N}, \text{ and polynomial } p(n) \text{ where} \\ \forall x \in \{0, 1\}^*, \text{ the fraction of } y \in \{0, 1\}^{p(|x|)} \text{ which lead} \\ \text{the input to be in } \mathcal{L}', \frac{\#\{y \in \{0, 1\}^{p(|x|)} \mid (x, y) \in \mathcal{L}'\}}{2^{p(n)}}, \\ \text{is } \geq \frac{1}{2} \text{ if } x \in \mathcal{L} \text{ and is } 0 \text{ if } x \notin \mathcal{L} \}. \end{aligned}$$

We can see that these two definitions are equivalent in a similar way as for the two definitions of NP. With a bit of examination, we can interpret RP as the set of languages which can be decided by a Turing machine with access to a source of randomness, or equivalently a polynomial-length random string, which runs in polynomial time and has a one-sided 1/2 error rate but must always reject when  $x \notin \mathcal{L}$ . Notice that an RP algorithm is also an NP algorithm, just one where we know many more than one computational paths will accept if  $x \in \mathcal{L}$ , so clearly  $\text{RP} \subseteq \text{NP}$ .

Second is the more general BPP, which stands for *Bounded-Error Probabilistic Polynomial-Time*, where we allow two-sided error so that  $\text{BPP} \supseteq \text{RP}$  and whether it is still contained within NP unknown.

**Definition 2.18.** *The complexity class*

$$\begin{aligned} \text{BPP} = \{ \mathcal{L} \subseteq \{0, 1\}^* \mid \exists k \in \mathbb{N} \text{ such that there is a nondeterministic Turing} \\ \text{machine } M \text{ where } \forall x \in \{0, 1\}^*, t(M, x) \in O(|x|^k), \\ \geq \frac{3}{4} \text{ of } M(x)\text{'s computational paths accept if } x \in \mathcal{L}, \text{ and} \\ \leq \frac{1}{4} \text{ of } M(x)\text{'s computational paths accept if } x \notin \mathcal{L} \} \end{aligned}$$

or, equivalently,

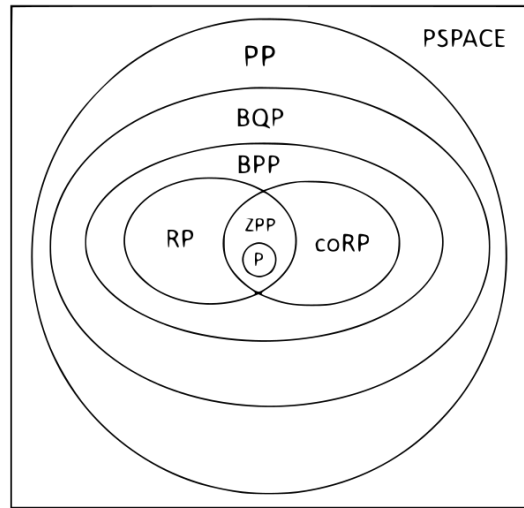
$$\begin{aligned} \text{BPP} = \{ \mathcal{L} \subseteq \{0, 1\}^* \mid \exists \text{ a language } \mathcal{L}' \in \text{P taking 2-tuple input } (x, y), \\ k \in \mathbb{N}, \text{ and polynomial } p(n) \text{ where} \\ \forall x \in \{0, 1\}^*, \text{ the fraction of } y \in \{0, 1\}^{p(|x|)} \text{ which lead} \\ \text{the input to be in } \mathcal{L}', \frac{\#\{y \in \{0, 1\}^{p(|x|)} \mid (x, y) \in \mathcal{L}'\}}{2^{p(n)}}, \\ \text{is } \geq \frac{3}{4} \text{ if } x \in \mathcal{L} \text{ and is } \leq \frac{1}{4} \text{ if } x \notin \mathcal{L} \}. \end{aligned}$$

We can again see that these two definitions are equivalent in a similar way as for the two definitions of NP. With a bit of examination, we can interpret BPP as the set of languages which can be decided by a Turing machine with access to a source of randomness, or equivalently a polynomial-length random string, which runs in polynomial time and has no more than a 1/4 error rate.<sup>1</sup> Thus BPP is often thought of as consisting of the problems that can feasibly be solved by an algorithm with access to true randomness. Another big open question in complexity theory is whether BPP can be derandomized, i.e., whether  $\text{P} = \text{BPP}$ . Unlike with NP, it is generally believed that this should be possible and P is indeed equal to BPP. However, this would require proving superpolynomial circuit lower bounds which seem far out of reach.

Another randomized complexity class which will come up is ZPP, which is short for *Zero-Error Probabilistic Polynomial-Time*. However, this class is less important for the topic of this thesis, so we will not need a precise definition. Roughly speaking, this is the class of problems which can be solved by an algorithm with access to randomness that always answers correctly and

<sup>1</sup>The constants used to define RP and BPP are somewhat arbitrary. For RP, any nonzero error rate strictly less than 1 gives an equivalent class of problems, and for BPP any nonzero error rate strictly less than 1/2 does. This can be shown by simply repeating the procedure enough times to drive the error rate back down (this is called probability amplification) while remaining in polynomial time overall.

whose *expected* runtime is polynomial in the length of the input. See Figure 2.7 for a depiction of the relationships between these randomized complexity classes and some other common complexity classes.



**Figure 2.7** This depiction of the relationships between randomized complexity classes and some other common complexity classes is by Bilorv - Own work, CC0, <https://commons.wikimedia.org/w/index.php?curid=91765136>.

Finally, it will be important to note that languages can be computed by circuits, or more precisely circuit families, giving us yet another model of computation. We will consider standard  $n$ -input Boolean circuits with fan-in one NOT gates and fan-in two AND and OR gates which output a single bit. Clearly, this bit can be interpreted as accept (1) or reject (0). However, a single circuit can only handle inputs of a fixed length  $n$ , so we will need a series of circuits, or a *circuit family*, to be able to compute a language. To this end, let  $\mathcal{L}_n$  denote  $\mathcal{L} \cap \{0, 1\}^n$ , or all of the strings in  $\mathcal{L}$  of length  $n$ . Then we will say that a circuit  $C_n$  decides  $\mathcal{L}_n$  if for each  $x \in \mathcal{L}_n$ ,  $C_n[x] = 1$ , and for each  $x \in \{0, 1\}^n \setminus \mathcal{L}_n$ ,  $C_n[x] = 0$ . Further, a circuit family  $\{C_n\}$  decides  $\mathcal{L}$  if for each  $n \in \mathbb{N}$ ,  $C_n$  decides  $\mathcal{L}_n$ . Notice that by allowing each input length to have its own circuit, we are actually giving circuit families more power than Turing machines, which we require to deal with all inputs in one. Something interesting and important to note is that circuit families can compute languages that Turing machines cannot even when given unbounded computational resources, such as the Halting Problem. (Since there are only a finite number of binary strings of a given

length, there are only a finite number of Turing machine encoding-input pairs of a given length which halt. Thus there is indeed a circuit for each length which hard-codes in all of these halting Turing machine-input pairs individually—we just have no chance of figuring out what these circuits would be.)

Now, in order to consider the complexity of languages with respect to circuits, we must define a computational resource of circuits which we can bound. Recall that the size of a circuit is the number of gates within it. It turns out that circuit size and Turing machine time are closely related, and for both we consider a polynomial amount to be feasible. For technical reasons, the class of languages which can be decided by circuit families of polynomial size is called P/poly. More precisely,

**Definition 2.19.** *The complexity class*

$$\text{P/poly} = \{ \mathcal{L} \subseteq \{0, 1\}^* \mid \exists k \in \mathbb{N} \text{ such that for all } n \in \mathbb{N}, \mathcal{L}_n \text{ is decided by a circuit } C_n \text{ of size } O(n^k) \}.$$

We can see why this name is used through an equivalent definition, namely

**Definition 2.20.** *The complexity class*

$$\begin{aligned} \text{P/poly} = \{ \mathcal{L} \subseteq \{0, 1\}^* \mid \exists k \in \mathbb{N} \text{ where for each integer } n, \\ \text{there exists an advice string } a(n) \text{ of length } O(n^k) \\ \text{s.t. } \mathcal{L} \text{ is decided by a polynomial-time Turing machine} \\ M \text{ that takes } (x, a(|x|)) \text{ instead of just } x \text{ as input} \}. \end{aligned}$$

Thus the name of this class indicates that we are adding polynomial-length advice to P, where this advice is fixed for each length of input. It is not too difficult to see why these two definitions are equivalent. If a language  $\mathcal{L}$  fulfills the first definition, then  $a(n)$  can just be the encoding of the polynomial-size circuit  $C_n$  and  $M$  can perform CIRCUIT EVALUATION on  $C_{|x|}$  with input  $|x|$  to compute the correct output. For the other direction, by slightly modifying the reduction used to show that CIRCUIT EVALUATION is P-complete, it can be seen that any language in P can be decided by a family of polynomial-size circuits. Thus for any  $\mathcal{L}$  fulfilling the second definition there is a family of polynomial-size circuits which decides it when given access to  $(x, a(|x|))$ . Indeed, since we get a different circuit  $C_n$  for each  $n$ , we can hard-code  $a(n)$  into  $C_n$  so that we have a family of polynomial-size circuits which decides  $\mathcal{L}$  when given only  $x$  as input.

Note that  $P \subseteq P/\text{poly}$  since we can just ignore the advice to compute any language in  $P$  as we normally would. Indeed,  $P/\text{poly}$  is actually strictly larger than  $P$  because it gets to have a circuit tailored to each input length, while  $P$  must use a single Turing machine for all input lengths, so  $P/\text{poly}$  contains some undecidable languages (including all unary undecidable languages). However, it is unknown whether  $NP \subseteq P/\text{poly}$ , though generally this thought not to be the case. Thus if it could be shown that indeed  $NP \not\subseteq P/\text{poly}$ , this would prove  $P \neq NP$ .

With this foundation established, we are at last ready to start discussing MCSP in earnest.

### 2.3 Introducing the Minimum Circuit Size Problem (MCSP)

We can now define the Minimum Circuit Size Problem, or MCSP, using the same standard notion of a circuit and of circuit size (sometimes the number of wires or length of particular encoding is used instead of the number of gates, though all of these size measures only differ by at most a logarithmic factor in the number of input bits, and generally which one you choose is immaterial). Also note that the *truth table* of an  $n$ -ary Boolean function  $f$  is a binary string of length  $2^n$  which lists the outputs of  $f$  to each possible input organized in lexicographic order of the inputs. For instance, the truth table of the 2-ary Boolean function  $x_1 \wedge x_2$  is 0001. Then

#### Definition 2.21.

$MCSP = \{(f, s) \in \{0, 1\}^* \mid f \text{ is an } n\text{-ary Boolean function represented by its truth table, } s \text{ is an integer, \& there exists a circuit } C \text{ of size at most } s \text{ which computes } f\}$ .

Throughout this work, we will use  $N$  to denote  $|f|$  so that  $N = 2^n$ . Notice that the length of the input is  $N + \log_2(s)$ , so a polynomial-time algorithm for MCSP must have a runtime that is polynomial in  $N + \log(s)$ . Indeed, it is well-known that all  $n$ -ary Boolean functions have circuits of size  $O(2^n/n)$ , so for  $s$  larger than this threshold, any algorithm for MCSP can automatically accept. Thus we can always consider  $s \in O(2^n)$ , in which case  $N + \log_2(s) \in O(N + \log_2(2^n)) = O(N + n) = O(N)$ , so really just  $N$  is the relevant input length for judging the performance of an algorithm for MCSP. (Additionally,  $s$  can be encoded in unary, but this also doesn't significantly change matters.)



To see why an  $n$ -ary Boolean function  $f$  will at least have a circuit of size  $O(2^n)$ , consider the following. It is not too difficult to construct what is called a conditional gate out of a constant number of ANDs, ORs, and NOTs that performs the following on input bits  $d$  (the decision bit),  $o_1$  (output option 1), and  $o_2$  (output option 2): if  $d$ , output  $o_1$ , if  $\neg d$ , output  $o_2$ . Such a conditional gate is given by  $(d \wedge o_1) \vee (\neg d \wedge o_2)$ . Now, to compute a Boolean function  $f$ , we can use a tree of conditional gates as follows. First, we have  $2^n$  leaves with the value output by each possible input to  $f$ , i.e., for each  $x \in \{0, 1\}^n$ , we have a 0/1 node of value  $f(x)$ . Now for each  $y \in \{0, 1\}^{n-1}$ , we have a conditional gate with decision bit  $d = x_n$ ,  $o_1 = f(y1)$ , and  $o_2 = f(y0)$ . Thus this layer of conditional gates allows us to pass down the correct value of  $f$  assuming the first  $n - 1$  bits are correct, but having distinguished on the basis of the final bit. The values we have produced can be represented by  $f(y-)$ , indicating that this is the value of  $f$  if  $x$  starts with  $y$  and then has the correct final bit. Call this layer  $n$ , since we conditioned on variable  $x_n$ . We repeat this with more layers of conditional gates, which take the following form: for layer  $i$ , we are given  $f(y-^{n-i})$  for each  $y \in \{0, 1\}^i$ . Then for each  $y' \in \{0, 1\}^{i-1}$ , we use a conditional gate with decision bit  $d = x_i$ ,  $o_1 = f(y'1-^{n-i})$ , and  $o_2 = f(y'0-^{n-i})$  to produce  $f(y'-^{n-(i-1)})$ . By the end, when we have conditioned on all of the input variables and have only one bit remaining, we will have selected the correct value of  $f(x)$ , which we output. Since there are  $O(2^n)$  conditional gates in this tree and  $O(1)$  AND, OR, and NOT gates being used to make each conditional gate, we have shown that any  $n$ -ary Boolean function is indeed computed by a circuit of size  $O(N)$ .

Now it is not too difficult to see that MCSP is in NP. Since we can assume  $s \in O(N)$  (otherwise accept immediately), we can guess a circuit  $C$  of size  $s$  in time polynomial in  $N$ . Then for a given possible input to the circuit, we can use CIRCUIT EVALUATION to check that  $C$  outputs the correct bit indicated by  $f$  in time polynomial in the size of the circuit  $s \in O(N)$ . There are  $2^n = N$  possible inputs to  $C$ , so we have enough time to check every one to verify that  $C$  is indeed computing  $f$  while ultimately operating within polynomial time in  $N$  overall. Thus  $\text{MCSP} \in \text{NP}$ .

Alright, but is  $\text{MCSP} \in \text{P}$ ? It turns out, probably not. Long story short, Kabanets and Cai (2000) showed that if  $\text{MCSP} \in \text{P}$  then there is no private-key cryptography (let alone public-key). There are strong candidates for private-key cryptography which have remained uncracked through decades of scrutiny, so we conclude that likely  $\text{MCSP} \notin \text{P}$ . When we have a problem in NP but probably not in P, this immediately suggests the question: is it NP-complete?

Before we explore this possibility for MCSP, there is first the question: why is it the case that if  $\text{MCSP} \in \text{P}$  then there is no private-key cryptography? Does MCSP's connection to cryptography end here, or is there something deeper going on? Looking into these questions will lead us to see that MCSP and its possible NP-completeness increasingly seem to play a central role in connecting a subset of what are called Impagliazzo's five worlds, and thus unifying our understandings of worst-case complexity, average-case complexity, and private-key cryptography.



## Chapter 3

# Why Do We Care About MCSP?

### 3.1 Impagliazzo's Five Worlds

The well-known complexity theorist Russell Impagliazzo introduced a conceptual framework for considering the relationship between worst-case complexity, average-case complexity, and cryptography in Impagliazzo (1995). He proposed five aptly named complexity-theoretic worlds we could be in:

- *Algorithmica*, where  $P = NP$ ,
- *Heuristica*, where  $P \neq NP$  but  $NP$  is easy on average for any samplable distribution,
- *Pessiland*, where  $NP$  is hard on average and there is no secure private-key cryptography,
- *Minicrypt*, where there is secure private- but not public-key cryptography, and
- *Cryptomania*, where there is public-key cryptography. <sup>1</sup>

These five worlds are based around whether  $P = NP$ ,  $NP$  is hard on average, and private-key or public-key cryptography exists, and include all of the possible combinations of these because of a chain of implications that can be

---

<sup>1</sup>Note that the formal technical definitions of each of these worlds can vary based whether we allow “tractible” algorithms access to randomness and how we precisely interpret “easy/hard on average.” These worlds are a useful conceptual framework with some looseness based on context.

relatively easily seen going one way. If there is public-key cryptography, then there is also private-key cryptography. If there is private-key cryptography, then NP must be hard on average because otherwise we could break private-key cryptography. If NP is hard on average, it certainly must be hard in the worst case. Thus we can narrow down the world we are in to one of these five.

Can we narrow in even further onto our actual world? To rule out the potential existence of one of these worlds, we must prove implications in the other direction. Can we prove  $P \neq NP$  with no premise, eliminating Algorithmica? Does  $P \neq NP$  imply that NP is hard on average, eliminating Heuristica? Does NP being hard on average mean that there must be private-key cryptography, eliminating Pessiland? Does the existence of private-key cryptography imply the existence of public-key cryptography, eliminating Minicrypt? Or is there simply no public-key cryptography, eliminating Cryptomania?

Most complexity theorists believe that we are either in Minicrypt or Cryptomania, and thus should be able to rule out the other worlds. While it is unlikely we will eliminate Algorithmica anytime soon, there was recent progress that unexpectedly suggested that MCSP could be used to eliminate both Heuristica and Pessiland if certain variants of it were shown to be NP-complete. Even more recently, there was a huge breakthrough that eliminated (at least one version of) Pessiland by utilizing a problem closely related to MCSP. MCSP's connection to private-key cryptography was hinted at at the end of the previous chapter, so let's start there.

## 3.2 MCSP and Pessiland

First, why does the existence of private-key cryptography imply  $MCSP \notin P$ ? What exactly is *private-key cryptography* anyway? The basic idea is that if a small key is privately shared between two parties beforehand, they will be able to use an unbreakable code to communicate securely over an open channel. (In public-key cryptography, a private key needn't be securely shared ahead of time in order to accomplish this.) Combining the results of Impagliazzo and Luby (1989) and Håstad et al. (1999), private-key cryptography exists if and only if *one-way functions* do, which are simpler to define formally. Following the definitions of Impagliazzo and Luby (1989) and Allender and Hirahara (2019),

**Definition 3.1.** A function  $f : \mathbb{N} \rightarrow \mathbb{N}$  is *one-way* if

- $f$  can be computed in polynomial time, and
- for any algorithm  $\mathcal{A}$  computed by polynomial-size circuits, the inverting probability

$$\Pr \{f(\mathcal{A}(f(\mathbf{x}))) = f(\mathbf{x})\} = \frac{1}{n^{\omega(1)}},$$

where  $\mathbf{x}$  is drawn uniformly at random from  $\{0, 1\}^n$ .

Essentially, one-way functions are those which are easy to compute, but difficult to invert, which is suggestive of their fundamental importance to cryptography. These two papers also show that the existence of one-way functions is also equivalent to the existence of *pseudorandom generators (PRGs)*. Following the definition of Razborov and Rudich (1997),

**Definition 3.2.** Let  $G_k : \{0, 1\}^k \rightarrow \{0, 1\}^{2k}$ , and define its hardness  $H(G_k)$  to be the smallest  $s$  such that there exists a circuit  $C$  which is of size  $\leq s$  and has

$$|\Pr\{C[G_k(\mathbf{x})] = 1\} - \Pr\{C[\mathbf{y}] = 1\}| \geq \frac{1}{s}$$

where  $\mathbf{x}$  and  $\mathbf{y}$  are drawn uniformly at random from  $\{0, 1\}^k$  and  $\{0, 1\}^{2k}$  respectively. We will call  $G_k$  a strong pseudorandom generator, or PRG, if  $H(G_k) \geq 2^{k^{\Omega(1)}}$ .

Thus, roughly speaking, a strong PRG can extend a random binary string into a longer one which cannot be distinguished from true randomness by any circuit of subexponential size. To understand how PRGs and thus one-way functions and private-key cryptography connect to MCSP, we have to go back to the famous “natural proofs” framework of Razborov and Rudich (1997).

One program to prove that  $P \neq NP$  approaches the problem from the direction of circuit lower bounds. If  $NP \not\subseteq P/poly$ , then since of course  $P \subseteq P/poly$  (just ignore the advice),  $P \neq NP$ . While initially there was progress on this front, by the late '90's it had stalled out. The framework of natural proofs in this paper showed why. A  $P/poly$ -natural property against  $P/poly$  is a set of Boolean functions that is (1) constructive (membership can be determined in  $P/poly$ ), (2) large (the members constitute a significant fraction of Boolean functions), and (3) useful (the members are not in  $P/poly$ ). A proof is  $P/poly$ -natural against  $P/poly$  if its proof contains (explicitly or implicitly) a  $P/poly$ -natural property against  $P/poly$ .

Razborov and Rudich showed that as far as they could ascertain, all nonmonotone circuit lower bounds to date were  $P/poly$ -natural against

various circuit classes. However, their paper proves that if a superpolynomial circuit lower bound was shown for NP via a proof P/poly-natural against P/poly, then all PRGs in P/poly would be easy to break (and thus one-way functions would not exist), so that we don't even have private-key cryptography. Since it is widely believed that secure cryptography exists, this has been taken to mean that altogether new techniques for proving circuit lower bounds will be necessary for this program to be successful.

However, while this framework provided a barrier in this area of complexity theory, it also suggested a new direction with MCSP which resparked interest in the problem. Kabanets and Cai (2000) pointed out that Razborov and Rudich (1997) implicitly shows

**Theorem 3.1.** *If  $MCSP \in P/poly$ , then there are no strong PRGs in P/poly.*

*Proof.* Notice that in this case the set of Boolean functions that require circuits of size  $\Omega(2^n/n)$  is P/poly natural against P/poly. This set is constructive since we can use  $MCSP \in P/poly$  with  $s = \Omega(2^n/n)$  to determine membership, it is large since most Boolean functions require large circuits, and it is useful since clearly no function in this set has polynomial-size circuits.  $\square$

This result also holds if MCSP is can be approximated sufficiently well in P/poly, namely within a factor of  $n^c$  for  $c \in \mathbb{N}$ , meaning that if a function has a circuit of size  $\leq s$  the approximation algorithm must accept, but it only has to reject if the function requires circuits of size  $> n^c s$  (otherwise it answer however it likes). In this case, using  $s = O(2^n/n^{c+1})$ , the set of functions which require circuits of size  $O(2^n/n)$  plus some which require size at least  $O(2^n/n^{c+1})$  is still constructive, at least as large as before, and useful since all members require superpolynomial circuits. Indeed, this holds for any approximation factor  $a(n)$  such that  $2^n/(a(n) \cdot n)$  is still superpolynomial in  $n$ . This gives strong motivation that MCSP nor certain approximation versions of it are in P/poly, let alone P.

It also suggests that we should examine the connection between MCSP and cryptographic objects further. Perhaps not only do one-way functions not exist if MCSP is easy—perhaps if MCSP is hard, one-way functions *must* exist. In order to understand this relationship further, we must introduce average-case complexity.

Hirahara and Santhanam (2017) use the connection to natural proofs to show that in fact the existence of one-way functions implies that MCSP is not only worst-case hard, but is zero-error average-case hard for any fixed size parameter  $s \in 2^{\Omega(n)}$ . Roughly speaking, a zero-error average-case algorithm

for a language outputs “yes,” “no,” or “?” for each input, is always correct when it outputs “yes” or “no”, and doesn’t output “?” too often, so it figures out the correct answer a significant portion of the time but refrains from making a proclamation either way otherwise. Thus the following definition of *zero-error average-case hardness* makes sense.

**Definition 3.3.** *A language  $\mathcal{L}$  is zero-error average-case hard if for any family of polynomial-size circuits  $\{C_n\}$  which outputs “yes,” “no,” or “?” for each input, for all large enough  $n$  either  $C_n$  outputs “yes” or “no” incorrectly on some input or outputs “?” for more than  $1 - 1/n^{O(1)}$  of the inputs.*

Notice that this is considering “average-case” over the uniform distribution, which for MCSP means we must use a zero-error notion instead of a bounded-error notion of average-case performance. This is because when drawing from the uniform distribution, with high probability we will get a function with high circuit complexity, so for small sizes an algorithm could safely answer “no” on all inputs and still have only a small probability of error, while for large enough sizes it can safely answer “yes.” Thus MCSP is trivially easy in the bounded-error sense. By requiring zero-error instead, this tactic can no longer be used. Now, notice that if MCSP isn’t zero-error average-case hard for some fixed size parameter  $s \in 2^{\Omega(n)}$ , while we can’t use it to necessarily identify *all* functions which require circuits of size  $> s$ , we still identify enough of them that our natural property is large.

In order to connect the zero-error average-case hardness to the existence of one-way functions in the other direction, Santhanam (2020) introduces the Universality Conjecture. This conjecture hypothesizes that there is a universal construction of what are called succinct pseudorandom distributions against arbitrary polynomial-size adversaries. This is not unreasonable, since there are indeed universal constructions of a variety of other cryptographic objects such as one-way functions, PRGs, and hitting set generators (HSGs). Further, if one-way functions exist, then we have pseudorandom function generators which can serve as universal succinct PRGs.

In any case, Santhanam (2020) shows that under the Universality Conjecture, one-way functions exist if and only if MCSP is zero-error average-case hard when the size parameter is set to  $2^{\epsilon n}$  for some  $\epsilon > 0$ , amongst other equivalences. Thus if the Universality Conjecture holds and this version of MCSP is shown to be NP-complete, the average-case hardness of NP would imply the average-case hardness of this version of MCSP, which would imply one-way functions and thus private-key cryptography exist, eliminating Pessiland.



Ren and Santhanam (2021) also construct one-way functions from the hardness of MCSP, but this time from a particular notion of MCSP being *exponentially* hard on average. However, this allows them to eliminate the use of a conjecture like the Universality Conjecture, the first such result. Further, MCSP being exponentially hard on average is well-motivated by viewing it as what is called a Weak Peregbor Hypothesis, implying that the best algorithm for MCSP is essentially brute-force search. The idea that solving MCSP potentially requires brute-force search is what motivated interest in the problem in the USSR going back to the 1950's. This work constructs a one-way function from MCSP in a very straightforward and appealing way. Namely, the one-way function takes as input a circuit  $C$  and outputs the size of  $C$  and its truth table. Thus if MCSP is sufficiently hard, this function must be one-way because inverting it requires solving MCSP (and even more, solving the search version of MCSP).

Very recently, Liu and Pass (2021) showed an NP-complete language whose mild average-case hardness implies the existence of one-way functions (building on their other recent breakthrough work Liu and Pass (2020)). The language they use is a minimization problem that is very closely related to MCSP, which instead of circuit size considers a certain kind of what is called Kolmogorov complexity.<sup>2</sup> Further, they show that this problem is mild average-case hard *if and only if* one-way functions exist. (They also note a concurrently developed, as-yet-unpublished result showing that the average-case hardness of a different NP-complete problem, likewise based on Kolmogorov complexity, also implies the existence of one-way functions, though this approach does not achieve the *if and only if* component.) Their result eliminates (at least one variant of) Pessiland. Since Pessiland is clearly the least desirable of Impagliazzo's five worlds, this is a major accomplishment.

### 3.3 MCSP and Heuristica

We can also connect the NP-completeness of variants of MCSP to the possibility of eliminating Heuristica, or a world in which NP is hard in the worst

---

<sup>2</sup>Defining Kolmogorov complexity and the relationships of minimization problems based on variants of this notion with each other and with MCSP is beyond the scope of this thesis. However, these meta-complexity problems share many of the interesting properties of MCSP including some of its other connections to Impagliazzo's Five Worlds, and can sometimes be more approachable to work with because Kolmogorov complexity is better understood than circuit complexity in many of ways.

case but not on average, which remains open. This is important since a common critique of complexity theory is that it typically focuses on worst-case complexity, when in many applications being able to solve problems efficiently on average would also be very advantageous. For instance, the simplex algorithm for linear programming is an example of a very useful algorithm which takes exponential time in the worst case, but behaves quite well on average.

Hirahara (2018) presents the first non-black-box worst-case to average-case reduction from a problem believed to be outside of what's called co-NP, namely from an approximation version of MCSP to average-case MCSP. To do so, they use techniques relating Kolmogorov randomness, true randomness, and pseudorandomness building on Allender et al. (2006), as well as connections between natural properties and learning circuits. Specifically, they show that approximating MCSP within a factor of  $2^{(1-\epsilon)n}$  is in BPP for some fixed  $\epsilon > 0$  if, and only if, for any  $c \in \mathbb{N}$ , MCSP has a zero-error average-case BPP algorithm such that “?” is output no more than  $1/N^c$  of the time when drawing the truth table uniformly at random from  $\{0, 1\}^N$  and with the size parameter is set to  $2^{\epsilon n}$  for some  $0 < \epsilon < 1$  (i.e., MCSP with this size parameter is in AvgBPP).

Previous results indicated that it was very unlikely that it would be possible to reduce worst-case hardness for a problem outside co-NP (as all NP-complete problems seem to be) to average-case hardness for a NP problem over a certain input distribution by using what are called black-box techniques. The most straightforward way to exclude the existence of Heuristica is via a reduction from the worst-case hardness of a NP-complete problem to the average-case hardness of a distributional NP problem, so this paper makes progress overcoming a significant obstacle. Notice that if approximating MCSP within a factor of  $2^{(1-\epsilon)n}$  was shown to be NP-complete under BPP-Turing reductions, then in conjunction with this result (and the assumption that  $P = BPP$ , which is widely believed) Heuristica would be eliminated.<sup>3</sup>

Additionally, Hirahara (2020) shows equivalence between the average-case complexity of the polynomial hierarchy (PH); the worst-case complexity of an approximation version of the PH-variant on MCSP,  $MCSP^{PH}$ ; and the existence of PH-computable hitting set generators, another cryptographic object. This again provides motivation for how the hardness of MCSP variants

<sup>3</sup>Alternatively, Algorithmica and Heuristica are sometimes sometimes defined with respect to BPP instead of P in the first place, as these definitions can be considered “morally equivalent.” See Impagliazzo (1995) for a discussion of this.

has important implications, and that worst-case complexity, average-case complexity, and the existence of cryptographic objects could all be connected through MCSP, leading the paper to conclude that “meta-complexity is indispensable for studying average-case complexity” (14).

Again reminiscent of Peregory Hypotheses, another line of attack on ruling out Heuristica is basing the average-case hardness of NP on something stronger than  $P \neq NP$  (or  $NP \not\subseteq BPP$ ), like that NP cannot be solved in deterministic  $2^{O(n/\log n)}$  time. Hirahara (2021) shows that if this is the case, then, roughly speaking, there are distributional NP problems which cannot be decided by algorithms which have an expected polynomial runtime such that this runtime on a given instance can be computed in polynomial time (i.e.  $\text{DistNP} \not\subseteq \text{Avg}_P$ ). While being able to compute a runtime upper bound for such algorithms would be quite useful in practice, this result would get closer to the desired ruling out of Heuristica if it didn’t restrict to algorithms with polynomial-time computable runtimes (then the result of the implication would be  $\text{DistNP} \not\subseteq \text{Avg}_P$ ), as this additional requirement places these algorithms somewhere between worst-case polynomial time and average-case polynomial time. (The paper also presents some evidence that  $\text{Avg}_P$  and  $\text{Avg}_P$  are ultimately not too dissimilar.) Hirahara (2021) establishes their result by utilizing a minimum Kolmogorov complexity problem. They also identify MCSP variants as candidate problems to show NP cannot be solved in deterministic  $2^{O(n/\log n)}$  time.<sup>4</sup> As a corollary to their result, they show that if any C-MCSP cannot be solved in deterministic  $2^{O(n/\log n)}$  time, then  $\text{DistNP} \not\subseteq \text{Avg}_P$ . This result thus utilizes minimization problems related to MCSP to eliminate a restricted version of Heuristica where NP cannot be solved in deterministic  $2^{O(n/\log n)}$  time but  $\text{DistNP} \subseteq \text{Avg}_P$ .

Between having played a part in ruling out Pessiland and potentially ruling out Heuristica, it’s clear how impactful proving MCSP and its variants to be NP-complete would be. Additionally, complexity theorists have been thinking about this problem starting with Leonid Levin back in the ’70’s. With all of this time and motivation, why has no one been able to prove that MCSP is NP-complete? It turns out there are technical barriers in this direction. We will discuss this in the next chapter.

---

<sup>4</sup>Note that for any circuit complexity class  $C \supseteq \text{DNF}$ , there are no known algorithms for C-MCSP which run in time  $2^{o(n)}$ , and for some C there is evidence against this being possible. As previously noted for MCSP itself, brute-force search is essentially the best strategy known. Further, some of these C-MCSP are known to be NP-complete—see the beginning the next chapter. Thus this is the first result to show that NP is average-case hard in some way if an NP-complete problem is worst-case hard in a plausible way.

## Chapter 4

# Why Don't We Know Whether MCSP is NP-complete?

Of course, one possibility is that MCSP is not NP-complete after all. However, many people working in this area believe it is, and there is a line of work showing that problems similar to MCSP are indeed NP-complete, such as DNF-MCSP (Allender et al. (2008)), (DNF  $\circ$  XOR)-MCSP (Hirahara et al. (2018)), Oracle-MCSP (Ilango (2019)), Multi-Output MCSP (Ilango et al. (2020)), and Constant-Depth-Formula-MCSP (Ilango (2020)). Why haven't we been able to do the same for MCSP itself? Are there formal barriers we can find to showing that MCSP is NP-complete?

Well, this chapter title is a little misleading. It turns out that we *do* know that MCSP is *not* NP-complete under certain "local" many-one reductions! We must always keep in mind that completeness is necessarily defined in relation to a particular kind of reduction. As the previous chapter made clear, we're interested in completeness under reductions as generous as BPP-Turing reductions.

There is a line of work exploring why it is hard to show that MCSP is NP-complete under different kinds of reductions, going back to the paper that reinvigorated interest in the problem, Kabanets and Cai (2000). These results come mainly in the following form: if MCSP is NP-complete under a certain kind of reduction, then we get a dramatic result in complexity theory that seems well beyond our current technical reach. This is interpreted as showing that it is very unlikely we will show MCSP to be NP-complete under this kind of reduction anytime soon. Most of these results will simply be stated with a brief proof idea. We will give the full proof for a couple of

results that are particularly pertinent precursors to the additions of this thesis. Note that these are in roughly chronological order of appearance, despite some seemingly mixed-up dates due to publishing delays for certain papers.

## 4.1 Previous Results for MCSP

### 4.1.1 Kabanets and Cai (2000)

This paper resparked interest in MCSP. Kabanets and Cai define the idea of a *natural many-one reduction*.

**Definition 4.1.** *A natural many-one reduction is a polynomial-time many-one reduction  $R$  such that for all  $x \in \{0, 1\}^*$ ,*

- $|R(x)|$  and the value of any numerical parameters output by it depend only on  $|x|$ , and
- $|x|$  and  $|R(x)|$  are polynomially related.

While this definition may appear somewhat arbitrary, the authors note that at the time of their writing "all 'natural' NP-complete problems that we are aware of are complete under natural reductions; this includes the Minimum Size DNF Problem" formally presented in Allender et al. (2008).

Note that  $E$  is the set of languages decided by deterministic Turing machines that run in linear exponential time, or time  $2^{O(n)}$ . Kabanets and Cai show that if MCSP is NP-complete via a natural reduction from SAT, then amongst other results we would prove  $E \not\subseteq P/\text{poly}$ , a dramatic circuit lower bound that seems far beyond current techniques. This is widely interpreted as showing that it is unlikely that we will be able to prove MCSP is NP-complete in this way. The proof of this is relatively straightforward and is illustrative of the essence behind many results in this line of work.

**Theorem 4.1.** *If there is a natural many-one reduction from SAT to MCSP, then  $E \not\subseteq P/\text{poly}$ .*

*Proof.* We will proceed by cases. First, suppose that  $NP \subseteq QP$ , where  $QP$  is the set of languages decided by deterministic Turing machines that run in quasi-polynomial time, or time  $n^{\text{polylog}(n)}$ . In this case, clearly  $PH \subseteq QP$  as

well. It can be shown that for some  $k \in \mathbb{N}$ ,  $\text{QP}^{\Sigma_k \text{P}}$  contains circuits outside of  $\text{P/poly}$ ,<sup>1</sup> so certainly in this case  $\text{E} \not\subseteq \text{P/poly}$  either.

Now, suppose  $\text{NP} \not\subseteq \text{QP}$ . Let  $R$  be the Turing machine computing the natural reduction from SAT to MCSP. Notice, by the definition of natural, that the size of any truth table output by  $R$  on an input of length  $n$  must be polynomial in  $n$ , so the Boolean function it defines will be on  $\Theta(\log(n))$  variables.

Also notice that there is a single size parameter  $s_n$  output by all inputs of this length. If for all but finitely many  $n$ ,  $s_n \leq (\log(n))^c$  for some constant  $c$ , then we can simply use brute-force search to check all  $\text{polylog}(n)^{\text{polylog}(n)}$  possible circuits on  $s_n \leq (\log(n))^c$  gates. (There are no more than  $(\log(n))^c$  gates, a constant number of kinds of gate each can be, and no more than  $\text{polylog}(n)$  previously computed values, for  $\text{polylog}(n)^{\text{polylog}(n)}$  possible circuits.) For each circuit, run CIRCUIT EVALUATION on it in  $\text{polylog}(n)$  time for all  $2^{O(\log(n))} = \text{poly}(n)$  possible inputs to check whether it indeed computes the desired function. Notice that this has taken no more than  $n^{\text{polylog}(n)}$  time overall. Thus in this case we can decide any language in NP by reducing it to SAT and then MCSP in polynomial time and deciding these instances of MCSP in  $n^{\text{polylog}(n)}$  time, in which case  $\text{NP} \subseteq \text{QP}$ , a contradiction.

Therefore, for all  $c$ ,  $s_n > (\log(n))^c$  for infinitely many  $n$ . Now, consider an easily-constructible trivial family of unsatisfiable formulas. On an input of

<sup>1</sup>We can adapt the argument of Kannan (1982). First, query the oracle with the length of the input. To compute this oracle, guess a circuit  $C$  on  $n$  variables of size  $n^{3\log(n)}$ . Then query the oracle, which will decide the language consisting of circuits which compute functions such that no circuit of size  $n^{\log(n)}$  computes the same function. This language will be computed by querying an oracle which computes the language consisting of circuits which compute functions such that there *is* a circuit of size  $n^{\log(n)}$  which computes the same function, and then answering opposite to the oracle. This next language will be computed by guessing a circuit of size  $n^{\log(n)}$  and querying an oracle with both circuits, and then answering opposite to the oracle. This oracle will decide the language consisting of pairs of circuits which compute different functions by guessing an input which makes them disagree and using CIRCUIT EVALUATION to check. Notice that we have used four levels to determine whether there is a circuit of size  $n^{\log(n)}$  which computes the same function as  $C$ . If there is, we reject. If not, now we also want to know whether there is a lexicographically earlier circuit of size  $n^{3\log(n)}$  that has no circuit of half the size computing the same function, so that we uniquely identify such a circuit. To check this, we query an oracle which decides this question by guessing a lexicographically earlier circuit and then goes through this whole process again to verify whether the function it computes has a half as small circuit, adding an extra level. If there is such a lexicographically earlier circuit, we reject. Else, use CIRCUIT EVALUATION to compute the value of  $C$  on the original input and return the same. Thus we have computed a language that can only be decided by a family of circuits of quasipolynomial size, so  $\text{QP}^{\Sigma_5 \text{P}} \not\subseteq \text{P/poly}$ .

length  $k$ , we will construct one of these formulas of size  $n = 2^{\Theta(k)}$ , and then run  $R$  on it to get an MCSP instance. Recall that the Boolean functions output by the reduction are on  $\Theta(\log(n)) = \Theta(k)$  variables. Thus since the answer to all such MCSP instances is no and for all  $c$ ,  $s_n > (\log(n))^c = \Theta(k^c)$  for infinitely many  $n$ , the language defined by this series of Boolean functions must not have polynomial-size circuits. Since we were able to compute this language in  $2^{O(k)}$  time,  $E \not\subseteq P/\text{poly}$ .

More precisely, since on input  $n$  we receive a truth table on  $v(n) = \Theta(\log(n))$  variables, not on exactly  $\log(n)$ , we need to take a little more care. Since  $v(n) = \Theta(\log(n))$ , by definition, for large enough  $n$ ,  $c_1 \log(n) \leq v(n) \leq c_2 \log(n)$  for some constants  $c_1, c_2$ . Thus if we want  $v(n) = k$  so that the original input is the correct length to serve as an input to the truth table we find, this can only occur for  $n$  such that  $2^{k/c_2} \leq n \leq 2^{k/c_1}$ . Since we're operating in  $E$ , we have enough time to check all such  $n$ . We may not always find an  $n$  such that ultimately  $v(n) = k$ , but we will infinitely often, since for each  $n = 1, 2, 3, \dots$  you will get some value for  $v(n)$  which we will find when  $k = v(n)$ , and these  $v(n)$  values can't repeat infinitely often as  $v(n) = \Theta(\log(n))$ . If we do find an  $n$  such that  $v(n) = k$ , we output the same thing as the truth table we've found would on our original input of length  $k$ . Further, as for all  $c$ ,  $s_n > (\log(n))^c$  for infinitely many  $n$ , since all  $n$  (except for finitely many too small ones or values which lead to repeated  $v$  outputs) lead to a  $v(n)$  which will be utilized when  $k = v(n) = \Theta(\log(n))$ , the language defined by this series of Boolean functions will indeed not have polynomial-size circuits. Since we have still taken  $2^{O(k)}$  time,  $E \not\subseteq P/\text{poly}$ .  $\square$

This proof illustrates how the NP-completeness of MCSP under certain kinds of reductions has particular repercussions because MCSP is a meta-computational problem—the answer it gives can have direct implications for the complexity of the language being computed itself, leading to surprising results.

Additionally, we can easily extend this result of Kabanets and Cai (2000) to certain kinds of randomized reductions. Consider that a RP many-one reduction will take a “yes” instance to a “yes” instance with probability at least one half and will always take a “no” instance to a “no” instance. We will let RET stand for Randomized linear Exponential Time in analogy to RP (RE already stands for the Recursively Enumerable languages).

**Corollary 4.1.** *If there is a natural RP many-one reduction from SAT to MCSP, then  $\text{RET} \not\subseteq P/\text{poly}$ .*

*Proof.* To adapt the first case of the previous proof, let RQP stand for Randomized Quasi-Polynomial time in analogy to RP. Notice that if  $\text{NP} \subseteq \text{RQP}$ , then so is PH because  $\text{RQP}^{\text{RQP}} \subseteq \text{RQP}$  using probability amplification. This implies that  $\text{QP}^{\Sigma_k^P} \subseteq \text{RQP}$ . Thus since for some  $k \in \mathbb{N}$ ,  $\text{QP}^{\Sigma_k^P} \not\subseteq \text{P/poly}$ ,  $\text{RQP} \not\subseteq \text{P/poly}$ , so certainly  $\text{RET} \not\subseteq \text{P/poly}$  in this case.

If  $\text{NP} \not\subseteq \text{RQP}$ , for all  $c$   $s_n > (\log(n))^c$  for infinitely many  $n$  because otherwise the natural RP many-one reduction would get us an RQP-algorithm for any problem in NP using the same approach as in the original proof. Again, consider an easily-constructible trivial family of unsatisfiable formulas. On an input of length  $k$ , we will construct one of these formulas of size  $n = 2^{\Theta(k)}$ , and then run the reduction  $R$  on it using a randomness string of all 0's to get an MCSP instance. Recall that the Boolean functions output by the reduction are on  $\Theta(\log(n)) = \Theta(k)$  variables. Thus since the answer to all such MCSP instances is no, so the RP many-one reduction must output a “no”-instance using any randomness string, and for all  $c$ ,  $s_n > (\log(n))^c = \Theta(k^c)$  for infinitely many  $n$ , the language defined by this series of Boolean functions must not have polynomial-size circuits. Since we were able to compute this language in  $2^{O(k)}$  time,  $\text{E} \not\subseteq \text{P/poly}$ , so certainly  $\text{RET} \not\subseteq \text{P/poly}$ .

Therefore in either case we get  $\text{RET} \not\subseteq \text{P/poly}$ , as desired.  $\square$

Now, notice that it was very crucial for the reasoning we used that our reduction had no chance of error on a “no” instance. In the case that  $\text{NP} \not\subseteq \text{RQP}$ , this led us to the stronger conclusion that  $\text{E} \not\subseteq \text{P/poly}$ , but we had to loosen this to  $\text{RET} \not\subseteq \text{P/poly}$  to accommodate the result we were able to get from the other case. This suggests that there might be room to improve this result, but if we allow a probability of error for “no” instances to generalize to many-one BPP reductions, we get a significant difficulty. Our approach shows that  $\text{E} \not\subseteq \text{P/poly}$  by, when given an input  $x$ , using the reduction on a “no” SAT instance to generate a truth table of a hard function on  $|x|$  variables and then outputting 0 or 1 based on the entry corresponding to  $x$  in the truth table. If the reduction involves randomness on such instances, it will have a high probability of outputting a “no” instance, but it may output entirely different “no” instances along each path. Thus if we output 0 or 1 based on the generated truth table, we have no guarantee that there will be a significant majority of 0 or 1 being output because even though there is a significant majority of hard truth tables all being generated, they may all be different hard truth tables. Ultimately, there is just no coherent singular hard function being followed amongst the majority of paths.

However, this suggests another kind of restricted reduction that may



work. What if we allow randomness but require that at least  $3/4$  of the paths output the *same* “yes” or “no” instance respectively? Indeed, this is exactly what is required by *Bellagio*, or *pseudo-deterministic*, algorithms for search problems (here we’re either “searching” for a “yes” or “no” instance of MCSP depending on whether the input was a “yes” or “no” instance). This kind of algorithm was first introduced by Gat and Goldwasser (2011), and a recent paper that surveys the resulting line of work on pseudo-deterministic algorithms and uses a definition more in line with the one we will use is Goldwasser et al. (2020). Namely, an algorithm with access to randomness is *pseudo-deterministic* if for each input  $x$ , there is a unique output  $o(x)$  which when the algorithm is run on  $x$  will be output with probability  $\geq 3/4$ . We will use the term *pseudo-deterministic BPP many-one reduction* to describe a BPP many-one reduction which on a given input produces a unique output with probability  $\geq 3/4$  (thus if the input is a “yes” instance, the unique output must be a “yes” instance, and likewise for “no” instances). Thus we can also get the following corollary, with Bounded-Error Probabilistic E, or BPE, being the linear exponential-time analogy of BPP.

**Corollary 4.2.** *If there is a natural pseudo-deterministic BPP many-one reduction from SAT to MCSP, then  $\text{BPE} \not\subseteq \text{P/poly}$ .*

*Proof.* To adapt the first case of the previous proof, let BPQP stand for Bounded-Error Probabilistic Quasi-Polynomial time in analogy to BPP. Notice that if  $\text{NP} \subseteq \text{BPQP}$ , then so is PH because  $\text{BPQP}^{\text{BPQP}} \subseteq \text{BPQP}$  using probability amplification. This implies that  $\text{QP}^{\Sigma_k \text{P}} \subseteq \text{BPQP}$ . Thus since for some  $k \in \mathbb{N}$ ,  $\text{QP}^{\Sigma_k \text{P}} \not\subseteq \text{P/poly}$ ,  $\text{BPQP} \not\subseteq \text{P/poly}$ , so certainly  $\text{BPE} \not\subseteq \text{P/poly}$  in this case.

If  $\text{NP} \not\subseteq \text{BPQP}$ , for all  $c$   $s_n > (\log(n))^c$  for infinitely many  $n$  because otherwise the natural BPP many-one reduction would get us an BPQP-algorithm for any problem in NP using the same approach as in the original proof. Again, consider an easily-constructible trivial family of unsatisfiable formulas. On an input of length  $k$ , we will construct one of these formulas of size  $n = 2^{\Theta(k)}$ , and then run the reduction  $R$  on it to get a unique MCSP instance with probability  $\geq 3/4$ . Recall that the Boolean functions output by the reduction are on  $\Theta(\log(n)) = \Theta(k)$  variables. Thus since the answer to all such MCSP instances is no, so the unique instance is a “no” instance, and for all  $c$ ,  $s_n > (\log(n))^c = \Theta(k^c)$  for infinitely many  $n$ , the language defined by this series of unique Boolean functions must not have polynomial-size circuits. Notice that if we output the appropriate entry of the generated truth table, with probability  $\geq 3/4$  we are outputting the appropriate entry

of the unique function's truth table, so the procedure laid out does indeed compute the language defined by this series of unique Boolean functions in the BP sense. Since we were able to BP-compute this language in  $2^{O(k)}$  time,  $\text{BPE} \not\subseteq \text{P/poly}$ .

Therefore in either case we get  $\text{BPE} \not\subseteq \text{P/poly}$ , as desired.  $\square$

Note that we could generalize this a little and allow the reduction to not respect the pseudo-deterministic restriction for "yes" instances, as the proof only utilizes this property for "no" instances.

#### 4.1.2 Murray and Williams (2017)

This paper shows that MCSP is unconditionally not NP-hard under "local," sufficiently efficient polynomial-time many-one reductions where sublinear time is spent to compute a single output bit on its own, or even randomized versions of these reductions. Further, even PARITY cannot be so reduced to MCSP. This may be surprising because most known NP-complete problems can be shown to be NP-hard under these kinds of local reductions, and indeed even more restricted ones where only poly-logarithmic time is spent to compute a single output bit. These kinds of reductions can be described as using local structure to make "gadgets," so this shows that sort of approach will not work for MCSP.

It is especially interesting that this extends to randomized reductions because these generally seem to be the most promising avenue to showing that MCSP is NP-complete. Intuitively speaking, this is because using randomness we can produce functions with high circuit complexity, while we don't have many circuit lower bounds for explicit functions we could deterministically produce. Indeed, the unconditional barrier against local reductions is proven using circuit lower bounds against PARITY, so paired with the circuit lower bounds which result from MCSP being NP-complete under certain reductions, our inability to determine the NP-hardness of MCSP under stronger kinds of reductions seems to be tied up in our lack of understanding of circuit lower bounds.

Murray and Williams also prove that if MCSP was shown to be NP-hard under only slightly more powerful reductions (namely logtime-uniform  $\text{AC}^0$  many-one reductions, where  $\text{AC}^0$  is a relatively weak circuit class that does not contain PARITY), then we would prove  $\text{NP} \not\subseteq \text{P/poly}$  (and thus  $\text{P} \neq \text{NP}$ ) as well as strong circuit size lower bounds against E that would imply  $\text{P} = \text{BPP}$ , thereby derandomizing BPP. While these results are believed to be true,

they seem to be dramatically beyond current techniques.

One approach used for several of these results is “naturalizing” reductions from PARITY by strategically padding the input instance with zeros so that all the bits examined to determine the size parameter will be zero, while the parity of the instance remains the same. This naturalized reduction allows the authors to bound the size parameter of the output MCSP instance because in order to address no-instances, it must be small enough for Boolean functions without circuits that large to exist. This is then used to construct circuits for PARITY which contradict known lower bounds.

Additionally, they show that if MCSP is NP-complete under logarithmic-space many-one reductions, then  $PSPACE \neq ZPP$  (PSPACE consists of the languages which can be computed in polynomial space on a deterministic Turing machine). Further, if MCSP is NP-complete under *general* many-one polynomial-time reductions, then  $EXP \not\subseteq NP \cap P/poly$  (EXP consists of the languages which can be computed in exponential time  $2^{n^{O(1)}}$  on a deterministic Turing machine) which implies  $EXP \neq ZPP$ , another result that seems far beyond our reach. These results are proven relatively straightforwardly by leveraging the reductions to get a complexity class equivalence that contradicts what's called the nondeterministic time hierarchy.

With this last result, we seem to have run out of possibilities for deterministic many-one reductions, since we need them to be polynomial-time at the very least. Perhaps Turing reductions could get around some of these inconvenient implications?

### 4.1.3 Hitchcock and Pavan (2015)

This paper shows two main results. First, it proves that if MCSP was shown to be NP-hard via general polynomial-time many-one reductions, then a strong circuit lower bound amplification result will hold (i.e., minor lower bounds will immediately imply much stronger ones). Indeed, both MCSP being in P or being NP-hard implies this kind of result. It is interesting to note that circuit lower bounds result both from MCSP being in P or it being NP-hard, so perhaps they could be shown to be true without resolving the complexity of MCSP if we could show they are implied by MCSP being NP-intermediate as well, though this seems more difficult to leverage.

Second, if MCSP is shown to be NP-hard via poly-logarithmic-space Turing reductions or truth-table reductions, then  $EXP \neq NP \cap SIZE(2^{n^\epsilon})$  for some  $\epsilon > 0$  (where  $SIZE(2^{n^\epsilon})$  denotes exponential-sized circuits for a specific exponent  $n^\epsilon$ ) and thus  $EXP \neq ZPP$ . Truth table reductions are

also known as nonadaptive reductions, and are polynomial-time Turing reductions which must decide what queries to make to the oracle with only access to the input, before learning the answer given to any previous query. It's been shown that a hypothesized NP-intermediate problem, Graph Isomorphism, reduces to MCSP under randomized adaptive reductions, so either of those characteristics (or both) could be the key for showing that MCSP is NP-complete.

Their overall approach is to show that under certain assumptions, if MCSP is NP-complete then it's complete under a certain kind of "parametric honest" reduction. Parametric honest reductions are those where on an input of length  $n$ , the numerical parameter of the output is bounded below by  $n^c$  for some  $c > 0$ . Similarly to earlier papers, once the size parameter of the MCSP instance output is bounded, that can be leveraged to get interesting results.

#### 4.1.4 Hirahara and Watanabe (2016)

This paper proves results in two main directions. First, it defines "oracle independent" reductions and shows several results related to using these to show that MCSP is NP-complete. Second, it generalizes some previous results. Like Hitchcock and Pavan (2015), they extend the  $EXP \neq ZPP$  implication of Murray and Williams (2017) from polynomial-time many-one reductions to the more general truth-table or nonadaptive polynomial-time Turing reductions.

Returning to the first point, oracle independent reductions to MCSP are those that apply equally well to  $MCSP^A$  for any oracle  $A$ . This version of MCSP asks whether there is circuit of a certain size which computes a particular function when that circuit has access oracle gates which compute  $A$  for it in one query. This characterization encompassed all reductions to MCSP known by the authors at the time, as these usually rely on relativizing properties of MCSP. Namely, most use that the circuit complexity of a random Boolean function is large with high probability. Then since minimum circuit size only decreases when given access to an oracle but the oracle circuit complexity of a random Boolean function still remains large with high probability, the reduction works just as well for  $MCSP^A$ .

However, this paper shows unconditionally that no language outside of  $P$  reduces to MCSP under oracle independent polynomial-time Turing reductions. The big idea of the proof is that for each such polynomial-time reduction  $R$ , we can adversarially choose an oracle  $A_R$  such that  $R$  never

queries truth tables with high circuit complexity (so querying the oracle doesn't help us learn anything we couldn't have figured out on our own in P). Thus, if  $P \neq NP$ , we cannot use oracle independent reductions to show that MCSP is NP-hard. Interestingly, as randomized reductions remain a potentially accessible avenue to showing that MCSP is NP-complete, this result also extends to show *randomized* oracle independent polynomial-time reductions which make only at most one oracle call are not sufficient to show that MCSP is NP-complete unless the polynomial hierarchy collapses (i.e.,  $PH = \Sigma_k P$  for some  $k \in \mathbb{N}$ ).

#### 4.1.5 Allender et al. (2017)

This paper shows a variety of results having to do with  $MCSP^A$  for various oracles  $A$ . Generally, the results in this paper are of the following form: if  $MCSP^A$  for a certain oracle  $A$  is hard for a certain complexity class, then some complexity class separation or equivalence follows. (Similar results also hold for time-bounded Kolmogorov complexity, since it has a close relationship with circuit complexity.) Something interesting to note is that the relationship between  $MCSP^A$  and  $MCSP^B$  for oracles  $A \neq B$  is generally unknown and may not follow intuitively from the complexity of languages  $A$  versus  $B$ . Indeed, this paper was able to show certain results under assumptions having to do with one oracle that they weren't able to show using another stronger oracle, which one might intuitively think would be a stronger assumption. Also, they note that for  $C \supseteq PSPACE$ , generally  $MCSP^C$  is complete for  $C/poly$  under  $P/poly$  reductions (though it is rarely known to be for uniform versions of such reductions).

One highlighted result shows that while  $MCSP^{QBF}$  is complete for PSPACE under ZPP-Turing reductions (QBF is a PSPACE-complete language), it is not so complete under logarithmic-space reductions, and it is not hard for the circuit complexity class  $TC^0$  under uniform  $AC^0$  reductions. They also show that if MCSP is hard for P under  $AC^0$  reductions, then  $P \neq NP$  as we expect, while if  $MCSP^{QBF}$  is hard for P under much more general logarithmic-space reductions, then  $EXP = PSPACE$ , which is not generally believed to be true. They conjecture that a similar result to the latter may be possible for MCSP itself, but this gap can't be easily bridged because there is not known to be an efficient reduction from MCSP to  $MCSP^{QBF}$ .

The general technique they use to prove these results is first assuming the opposing statement for contradiction, showing that under that assumption the reduction can be adapted to succinct versions of the input problem

and  $\text{MCSP}^A$ , that the existence of the reduction itself implies that small circuits exist for the Boolean function output by the reduction, that this in turn implies that these instances are easy to solve, contradicting that succinct versions of problems are generally “exponentially” harder than the original versions. This basically leverages that MCSP itself does not become exponentially harder when made succinct.

#### 4.1.6 Saks and Santhanam (2020)

This paper defines “natural” and “parametric honest” *Turing* reductions, and establishes that if MCSP was shown to be NP-hard under either of these types of reductions, then this would prove the seemingly-out-of-reach result  $\text{E} \not\subseteq \text{P/poly}$ .

**Definition 4.2.** *Parametric honest Turing reductions to MCSP are those where each query to the MCSP oracle has a size parameter of value at least  $n^c$  for some constant  $c > 0$ .*

This is analogous to the definition for many-one reductions from Hitchcock and Pavan (2015).

**Definition 4.3.** *Natural Turing reductions to MCSP are defined as those where each query to the MCSP oracle has a size parameter whose value depends only on the size of the input.*

This is likewise analogous to the definition in Kabanets and Cai (2000), though it is a somewhat weaker requirement when applied to many-one reductions since there are no restrictions of the length of the query overall, just the value of the size parameter.

The main strategy used to prove these results is simulating the oracle Turing machine for SAT by either simply saying yes to every query or brute-forcing small enough queries. If the simulation is correct, then we get a circuit lower bounds from NP being easy. If the simulation is incorrect, we can use the result of another paper to produce input instances of SAT that it fails on, which must have queries that should have been said no to. Using the aforementioned restricted versions of polynomial-time Turing reductions, we can make sure these “no”-queries have sufficiently high circuit complexity to get our circuit lower bound again. This is very simple for parametric honest Turing reductions, and involves some more complicated finagling with natural Turing reductions.

As a new result presented at the end of this chapter builds on the method of this paper, we will present the proof for parametric honest Turing

reductions to make this more concrete. First, we rely on Lemma 3.1 from Gutfreund et al. (2007) (which, as noted in Remark 3.6, holds whether or not  $P = NP$ ).

**Lemma 4.1.** *Let  $BSAT$  be an algorithm that fails to solve SAT and runs in time  $n^a$  for a constant  $a$ . Then there is a deterministic algorithm  $R$  such that on input  $(n, a, \langle BSAT \rangle)$  for  $n \in \mathbb{N}$ ,  $R$  takes polynomial time to output at most three formulas of length either  $n$  or a fixed polynomial in  $n^a$ . For infinitely many  $n$ , at least one of these output formulas is an instance  $BSAT$  fails to solve SAT on.*

This allows us to actually efficiently find instances that a purported polynomial-time algorithm for SAT fails on. Now we can straightforwardly prove our desired result.

**Theorem 4.2.** *If MCSP is complete for NP under parametric honest polynomial-time Turing reductions, then  $E \not\subseteq P/poly$ .*

*Proof.* Let  $\mathcal{M}$  be the oracle Turing machine reducing SAT to MCSP. Consider the following simple simulation  $M$  of this machine: whenever  $M$  would have queried the oracle, simply assume the answer to the query is “yes.”

There are two possibilities. Either  $M$  simulates  $\mathcal{M}$  correctly on all but finitely many inputs, in which case  $P = NP$  and this implies  $E \not\subseteq P/poly$ ,<sup>2</sup> or  $M$  simulates  $\mathcal{M}$  incorrectly on infinitely many inputs.

If  $M$  simulates  $\mathcal{M}$  incorrectly on infinitely many inputs, then on each such input  $\mathcal{M}$  must query the oracle with  $(f, s)$  such that the answer to this query is “no.” Since the reduction is parametric honest,  $s \geq n^c$ . Notice that as  $\mathcal{M}$  runs in polynomial time,  $|f| \in O(n^{c'})$  for some constant  $c'$ , and thus  $f$  is on  $k = O(\log(n))$  variables. Therefore  $f$  requires superpolynomial circuits.

Now we use the lemma from Gutfreund et al. (2007) to actually find these instances. Utilizing  $R$  from this lemma, for infinitely many  $n$ , if we run

<sup>2</sup>Assume for contradiction that  $P = NP$  and  $E \subseteq P/poly$ . Now consider an algorithm  $\mathcal{A}$  which takes as input the description of a Turing machine  $\langle M \rangle$  and input  $x$  and simulates  $M$  on input  $x$  for  $2^n$  of its own steps. If at that point  $M$  hasn't halted,  $\mathcal{A}$  rejects. If it has halted,  $\mathcal{A}$  answers the same way as  $M$ . The language  $\mathcal{A}$  decides is thus clearly in  $E \subseteq P/poly$ , so it can be decided by circuits of size bounded by  $n^k$  some fixed  $k \in \mathbb{N}$ . Notice that  $\mathcal{A}$  can simulate any language in  $P$  for all large enough  $n$ , which implies  $P$  can be decided by circuits of size  $n^k$  too. However, for any fixed  $k$ ,  $\Sigma_4P$  contains a language which requires circuits of size greater than  $n^k$ —this follows a very similar argument as the previous footnote, where now we can guess a circuit of larger yet still polynomial size, such as size  $n^{3k}$ , and check there is no circuit of size  $n^k$  which computes the same function. Kannan (1982) gives this argument. But if  $P = NP$ ,  $\Sigma_4P = P$ , so we get a contradiction. Therefore  $P = NP$  implies  $E \not\subseteq P/poly$ .

$R$  on  $(n, a, \langle M \rangle)$ , then in polynomial time we will get at most three formulas of length polynomial in  $n$  such that  $M$  fails on at least one. Then we can run  $M$  on each of these formulas, keeping track of all of the queries made. Finally, concatenating these queries (at least one of which we know to have a true answer of “no”), we have found a truth table of size  $\text{poly}(n)$ , and thus on  $O(\log(n))$  variables, which requires superpolynomial-size circuits. Therefore since this process took time polynomial in  $n$ , and thus linear exponential time in  $O(\log(n))$ ,  $E \not\subseteq P/\text{poly}$  in this case as well.  $\square$

## 4.2 Previous Results for Gap-MCSP

The previous chapter additionally showed us that the question of whether approximation versions of MCSP are NP-complete is relevant for eliminating Heuristica. There are a couple results showing obstacles in this direction as well.

### 4.2.1 Hirahara and Watanabe (2016)

We have already seen some results from this paper, but it also shows that if an approximation version of MCSP was proven to be NP-complete under simply polynomial-time Turing reductions, then we also get  $\text{EXP} \neq \text{ZPP}$ . Roughly speaking, this version of MCSP, called  $\text{Gap}^k\text{MCSP}$ , is a promise problem asking for an approximation of the logarithm of circuit complexity within a factor of  $k$ .

**Definition 4.4.** *Let  $f$  be a Boolean function on  $n$  variables represented as a truth table and  $s \in \mathbb{N}$ . Then for  $k \in \mathbb{N}$ , we define the YES and NO instances of  $\text{Gap}^k\text{MCSP}$  as follows:*

*YES:  $(f, s)$  such that  $f$  can be computed by a circuit of size  $s_f$  such that*

$$\log_2(s_f) \leq s$$

*NO:  $(f, s)$  such that  $f$  requires circuits of size  $s_f$  such that  $\log_2(s_f) > ks$ .*

Notice that this is equivalent to having “yes” instances where  $s_f \leq s'$  and “no” instances where  $s_f > s'^k$  (if we ask using parameter  $s' = 2^s$ ).

Their proofs are based on close connections between circuit complexity and time-bounded Kolmogorov complexity. These approximation versions of MCSP are known to be hard for Statistical Zero Knowledge (SZK) under BPP-Turing reductions, which suggests that perhaps these kinds of randomized reductions are more feasible.



### 4.2.2 Allender and Hirahara (2019)

This paper shows that if auxiliary-input one-way functions (which are weaker than normal one-way functions) exist, then approximating MCSP (or MKTP) within a factor of  $2^{(1-o(1))n}$  is NP-intermediate under P/poly Turing reductions, in the sense that in this case this problem is provably not in P/poly and is also not NP-hard under P/poly Turing reductions. These appear to be the first natural NP-intermediate problems under the assumption that one-way functions exist. Also note that if  $\text{SZK} \not\subseteq \text{P/poly}$ , then auxiliary-input one-way functions exist, further supporting that they likely do. To show this result, the authors utilize that MCSP would be “strongly downward self-reducible” if these kinds of reductions existed. This method can also be applied to show that if  $\text{NP} \not\subseteq \text{P/poly}$ , approximating largest-clique within the same factor is NP-intermediate. This paper thus provides some caution against assuming that MCSP will be NP-hard to approximate within larger approximation factors, though this is close to the loosest approximation factor you could have without the problem being trivial.

They also have some hardness results for a time-bounded-Kolmogorov-complexity variant of MCSP, MKTP, that are “local”—a surprise—by leveraging non-uniformity. Additionally, they present a reduction that gets around oracle independence under a plausible assumption. Finally, they show that to reduce Gap-MSCP problems (where the gap here is for a fixed constant  $\epsilon$  instead of  $o(1)$ ) amongst each other, there must be large stretch, or mapping of short inputs to long ones.

## 4.3 New Results for Gap-MCSP, $\Sigma_k$ -MCSP, & Q-MCSP

### 4.3.1 Gap-MCSP

Following the definition by Hirahara (2018) of an approximation version of MINKT, we will define an approximation version of MCSP by the following promise problem:

**Definition 4.5.** *Let  $\sigma : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$  be a function such that for all  $N, s \in \mathbb{N}$ ,  $\sigma(N, s) \geq s$ . Let  $f$  be a Boolean function on  $n$  variables represented as a truth table and  $s \in \mathbb{N}$ . Then we define the YES and NO instances of  $\text{Gap}_\sigma\text{MCSP}$  as follows:*

*YES:  $(f, s)$  such that  $f$  can be computed by a circuit of size  $\leq s$*

*NO:  $(f, s)$  such that  $f$  requires circuits of size  $> \sigma(|f|, s)$ .*

Notice that if  $\sigma(N, s) = s$ , this is simply MCSP.

Now we seek to use the approach of Saks and Santhanam (2020) to prove the following.

**Theorem 4.3.** *Let  $\sigma : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$  be a function such that  $\sigma$  is superpolynomial in  $s$ , or  $\sigma(N, s) \in \omega(s^k)$  for all  $k \in \mathbb{N}$ . If  $\text{Gap}_\sigma\text{MCSP}$  is complete for NP under polynomial-time Turing reductions, then  $\text{E} \not\subseteq \text{P/poly}$ .*

*Proof.* Let  $\mathcal{M}$  be the oracle machine reducing SAT to  $\text{Gap}_\sigma\text{MCSP}$ . Consider the following simulation  $M$  of this machine on an input  $x$  with  $n = |x|$ . Whenever  $M$  would have queried the oracle with  $(f, s)$ , if  $s \leq \log(n)^{1/2}$ ,  $M$  uses brute-force search to determine the correct answer to this query in time  $\text{poly}(n)2^{O(s \log(s))} = \text{poly}(n)n^{O(1)}$ , or polynomial time overall. (Notice if there are at most  $s$  gates, encoding which kind of gate each is and which previous gates provide each input takes at most  $O(\log(s))$  bits, and we can easily verify whether each string encodes a valid circuit. Then we just have to use CIRCUIT EVALUATION to check each possible input against what is output by the circuit we're trying in  $\text{poly}(n)$  time.) Else if  $s > \log(n)^{1/2}$ ,  $M$  assumes the answer to the query is "yes."

Now we have two possibilities. Either  $M$  simulates  $\mathcal{M}$  correctly on all but finitely many inputs, in which case  $\text{P} = \text{NP}$  and this implies  $\text{E} \not\subseteq \text{P/poly}$ , or  $M$  simulates  $\mathcal{M}$  incorrectly on infinitely many inputs. If  $M$  simulates  $\mathcal{M}$  incorrectly on infinitely many inputs, then on each such input  $\mathcal{M}$  must query the oracle with  $(f, s)$  such that  $s > \log(n)^{1/2}$  and the answer to this query cannot be "yes." Since our oracle is the promise problem  $\text{Gap}_\sigma\text{MCSP}$ , it is bound to answer "no" only if  $f$  requires circuits of size  $> \sigma(|f|, s)$ . Since  $\sigma$  is superpolynomial in  $s$ ,  $f$  thus must require circuits superpolynomial in  $\log(n)^{1/2}$ , or simply superpolynomial in  $\log(n)$ . Since  $\mathcal{M}$  runs in polynomial time,  $|f| \in O(n^c)$  for some constant  $c$ , and is thus on  $k = O(\log(n))$  variables. Therefore  $f$  requires superpolynomial circuits. Notice this means that if we concatenate all of the queries  $\mathcal{M}$  makes to the oracle on this input to create a truth table of size polynomial in  $n$  (maybe padded with some 0's at the end to that its length will be a power of 2), this will represent a function that also requires superpolynomial circuits.

Now we use the lemma from Gutfreund et al. (2007) to actually find these instances. Utilizing  $R$  from this lemma, for infinitely many  $n$ , if we run  $R$  on  $(n, a, \langle M \rangle)$ , then in polynomial time we will get at most three formulas of length polynomial in  $n$  such that  $M$  fails on at least one. Then we can run  $M$  on each of these formulas, keeping track of all of the queries made. Finally, concatenating these queries (at least one of which we know to have a true answer of "no"), we have found a truth table of size  $\text{poly}(n)$ , and

thus on  $O(\log(n))$  variables, which requires superpolynomial-size circuits. Therefore since this process took time polynomial in  $n$ , and thus linear exponential time in  $O(\log(n))$ ,  $E \not\subseteq P/\text{poly}$  in this case as well.  $\square$

**Corollary 4.3.** *This implication clearly also holds for any  $\sigma(N, s)$  that is superpolynomial in  $\log(n)$  when  $s > \log(n)^{1/2}$  and  $N = \text{poly}(n)$ . Specifically, it holds for  $\sigma(N, s) = \tau(N) \cdot s$  such that  $\tau$  is superpolynomial in  $\log(N)$ .*

Therefore if, when given a truth table of size  $2^n$ , approximating MCSP within a factor superpolynomial in  $\log(2^n) = n$  is shown to be NP-hard via polynomial-time Turing reductions,  $E \not\subseteq P/\text{poly}$ . This provides an obstacle to Hirahara (2018)'s program for eliminating Heuristica by showing that approximating MCSP within a factor of  $2^{(1-\epsilon)n}$  is NP-hard for some constant  $\epsilon > 0$ . However, this may still be possible to do without proving circuit lower bounds that seem well beyond current techniques if BPP-Turing reductions are used.

### 4.3.2 $\Sigma_k$ -MCSP and Q-MCSP

As in Murray and Williams (2017), we define nondeterministic circuits as follows. A nondeterministic circuit  $C$  of size  $s$  is a normal circuit of size  $s$  with its input split into two parts, a primary input  $x$  which has length  $n$ , and an auxiliary input  $y$  which has length  $\leq s$  (notice that this is as many auxiliary bits as we could potentially take into account with  $s$  gates given a dependence on at least one bit of  $x$ ). Then  $C$  is considered to output 1 on input  $x$  if there exists a  $y$  such that  $C[x, y] = 1$ . Then like Murray and Williams (2017) we can define NMCSPP similarly to MCSP but with nondeterministic circuits instead of regular ones. They prove the result that if NMCSPP is MA-hard (MA is a randomized complexity class contained in  $\Sigma_2P$ ) for polynomial-time many-one reductions, then  $\text{EXP} \not\subseteq P/\text{poly}$ . It is not too difficult to see that this extends to polynomial-time Turing reductions, not just many-one reductions. One can even straightforwardly use their technique to show that if NMCSPP is MA-hard for BPP Turing reductions, then  $\text{BPEXP} \not\subseteq P/\text{poly}$  (where BPEXP is Bounded-Error EXP).

We can extend this notion to consider  $\Sigma_k$ -circuits of size  $s$ , which have a primary input  $x$  of length  $n$  and auxiliary inputs  $y_1, y_2, \dots, y_k$  such that the  $y_i$  have length  $\leq s$  when concatenated altogether. A  $\Sigma_kP$  circuit  $C$  outputs 1 on input  $x$  if there exists a  $y_1$  such that for all  $y_2$  there exists a  $y_3$  etc.  $\dots$  such that  $C[x, y_1, y_2, \dots, y_k] = 1$ . Then we can extend the definition of  $P/\text{poly}$  to these new kinds of circuits.

**Definition 4.6.** *The complexity class*

$$\Sigma_k\text{P}/\text{poly} = \{\mathcal{L} \subseteq \{0, 1\}^* \mid \exists c \in \mathbb{N} \text{ such that for all } n \in \mathbb{N}, \mathcal{L}_n \text{ is decided by a } \Sigma_k\text{-circuit } C_n \text{ of size } O(n^c)\}.$$

Note that again this definition is equivalent to adding polynomial-length advice to  $\Sigma_k\text{P}$ . This can be seen in a similar way as for  $\text{P}/\text{poly}$  using the fact that  $\text{QSAT}_k$  (or more precisely the circuit satisfiability version) is complete for  $\Sigma_k\text{P}$ .<sup>3</sup>

We likewise consider  $\Sigma_k$ -MCSP defined in the obvious way with regards to  $\Sigma_k$ -circuits.

**Definition 4.7.**

$$\Sigma_k\text{-MCSP} = \{(f, s) \in \{0, 1\}^* \mid f \text{ is an } n\text{-ary Boolean function represented by its truth table, } s \text{ is an integer, and there exists a } \Sigma_k\text{-circuit } C \text{ which computes } f \text{ of size } \leq s\}.$$

Notice that  $\Sigma_0$ -MCSP is the same as MCSP and  $\Sigma_1$ -MCSP is NMCSPP. Further, notice that  $\Sigma_k$ -MCSP can easily be solved in  $\Sigma_{k+1}\text{P} = \text{NP}^{\Sigma_k\text{P}}$  by guessing a circuit and then using  $\Sigma_k$ -CSAT for the  $\Sigma_k\text{P}$  oracle in order to check whether the circuit computes the desired function on each input.

We can also provide evidence against  $\Sigma_k$ -MCSP being solvable in any  $\Sigma_\ell\text{P}$  for  $\ell \leq k$  by using the natural proofs framework of Razborov and Rudich (1997). First note that

**Lemma 4.2.** *If  $\Sigma_k$ -MCSP  $\in \Sigma_\ell\text{P}/\text{poly}$ , then there is a  $\Sigma_\ell\text{P}/\text{poly}$  natural property against  $\Sigma_k\text{P}/\text{poly}$ .*

*Proof.* Notice that in this case the set of Boolean functions that require  $\Sigma_k$ -circuits of size  $O(2^n/n)$  is  $\Sigma_\ell\text{P}/\text{poly}$  natural against  $\Sigma_k\text{P}/\text{poly}$ . This set is constructive since we can use  $\text{MCSP} \in \Sigma_\ell\text{P}/\text{poly}$  with  $s = O(2^n/n)$  to determine membership, it is large since most Boolean functions still require large  $\Sigma_k$ -circuits, and it is useful since clearly no function in this set has polynomial-size  $\Sigma_k$ -circuits.  $\square$

<sup>3</sup>See Balcázar et al. (1987) for an argument that the set of languages decided by polynomial-size families of quantified circuits is equivalent  $\text{PSPACE}/\text{poly}$ , which can be slightly modified to show this.

In order to leverage the existence of this property, we need a definition of a pseudorandom function generator and its hardness, and additionally a generalization of this to pseudorandom function generators secure against  $\Sigma_\ell$ -circuits. Inspired by Razborov and Rudich (1997), we use

**Definition 4.8.** A pseudorandom function generator (PRFG) is a family of functions  $f_{i,\epsilon} : \{0, 1\}^i \rightarrow F_n$  associated with each integer  $i$  and  $\epsilon \in \{1/2^m : m \in \mathbb{Z}\}$  (or any other set of positive integers that gets arbitrarily close to 0), where  $n = \lceil i^\epsilon \rceil$  and  $F_n$  is the set of all  $n$ -ary Boolean functions, so that for  $x \in \{0, 1\}^i$  and  $y \in \{0, 1\}^n$ ,  $f_{i,\epsilon}(x)(y)$  is the output of the Boolean function  $f_{i,\epsilon}(x)$  on input  $y$ . The PRFG  $f_{i,\epsilon}$  is considered to be within a complexity class  $C$  if the family of Boolean functions  $f_{i,\epsilon}(x)$  for each  $i$  can be computed within  $C$ . The hardness of the PRFG when using randomness length  $i$ ,  $H(f_i)$ , is the minimum  $s$  for which there exists a circuit  $C$  of size  $\leq s$  such that for some  $\epsilon$

$$|P(C(\mathbf{f}_n) = 1) - P(C(f_{i,\epsilon}(\mathbf{x})))| \geq \frac{1}{s}$$

where  $\mathbf{f}_n$  is drawn uniformly at random from  $F_n$  and  $\mathbf{x}$  is drawn uniformly at random from  $\{0, 1\}^i$ . The PRFG is considered strong if  $H(f_i) \geq 2^{i^{\Omega(1)}}$ .

More generally, the  $\Sigma_\ell$ -hardness of the PRFG when using randomness length  $i$ ,  $H_\ell(f_i)$ , is the minimum  $s$  for which there exists a  $\Sigma_\ell$ -circuit  $C$  of size  $\leq s$  such that for some  $\epsilon$

$$|P(C(\mathbf{f}_n) = 1) - P(C(f_{i,\epsilon}(\mathbf{x})))| \geq \frac{1}{s}$$

where  $\mathbf{f}_n$  is drawn uniformly at random from  $F_n$  and  $\mathbf{x}$  is drawn uniformly at random from  $\{0, 1\}^i$ . The PRFG is considered  $\Sigma_\ell$ -strong if  $H_\ell(f_i) \geq 2^{i^{\Omega(1)}}$ .

While this definition might seem somewhat arbitrary at first glance, Razborov and Rudich (1997) show how such a PRFG can be built from a pseudorandom generator (PRG) which instead of producing pseudorandom functions in various numbers of variables, simply generates pseudorandom strings which are twice as long as the random input string. When the original PRG is in P/poly, the PRFG will be as well, while if it's originally higher up the polynomial hierarchy, the PRFG may end up in PSPACE/poly as the original PRG is applied  $n$  times (if it was only applied a constant number of times it would remain in PH/poly). There are strong candidates for strong PRGs in P/poly, and as discussed in the last chapter their existence is equivalent to the existence of one-way functions.

**Theorem 4.4.** Let  $\ell$  be an integer no larger than  $k$ . If  $\Sigma_k\text{-MCSP} \in \Sigma_\ell\text{P/poly}$ , then there are no  $\Sigma_\ell$ -strong PRFGs in  $\Sigma_k\text{P/poly}$ .

*Proof.* Let  $\{f_{i,\epsilon}\}$  be a PRFG in  $\Sigma_k\text{P/poly}$ . By Lemma 4.2, since  $\Sigma_k\text{-MCSP} \in \Sigma_\ell\text{P/poly}$ , there is a  $\Sigma_\ell\text{P/poly}$  natural property against  $\Sigma_k\text{P/poly}$  decided by some family of circuits  $\{C_n\}$  in  $\Sigma_\ell\text{P/poly}$ . Now, notice that since  $\{f_{i,\epsilon}\}$  is in  $\Sigma_k\text{P/poly}$ , by the usefulness of the natural property against  $\Sigma_k\text{P/poly}$ , for large enough  $i$ ,  $C_n(f_{i,\epsilon}(x)) = 0$  for any fixed  $x \in \{0, 1\}$ . Further, by largeness, the number of Boolean functions  $f_n \in F_n$  such that  $C_n(f_n) = 1$  is at least  $2^{-O(n)} \cdot |F_n|$ . Therefore for large enough  $i$

$$|P(C_n(\mathbf{f}_n) = 1) - P(C_n(f_{i,\epsilon}(\mathbf{x})))| \geq 2^{-O(n)} - 0 = 2^{-O(n)}.$$

Finally, since by constructivity  $\{C_n\}$  is in  $\Sigma_\ell\text{P/poly}$ , this test is computable by  $\Sigma_\ell$ -circuits of size polynomial in the size of the relevant truth tables, or  $\Sigma_\ell$ -circuits of size  $2^{O(n)}$ . Thus

$$H_\ell(f_i) \leq 2^{O(n)} \leq 2^{O(i^\epsilon)},$$

so since this holds for any arbitrary  $\epsilon > 0$ , we must have  $H_\ell(f_i) \leq 2^{i^{o(1)}}$ . Therefore the PRFG  $f$  is not  $\Sigma_\ell$ -strong.  $\square$

**Corollary 4.4.** *If  $\Sigma_k\text{-MCSP} \in \text{P/poly}$ , then there are no strong PRFGs in  $\Sigma_k\text{P/poly}$ , so there are certainly no strong PRFGs in  $\text{P/poly}$ , so there are also no strong PRGs, one-way functions, or private-key cryptography.*

We can also define what I'll call Q-MCSP (for Quantified MCSP) as a natural extension of the  $\Sigma_k\text{-MCSP}$  idea where we ask if there are small-enough  $\Sigma_k$ -circuits for the function for *any*  $k$ , i.e. are there small-enough quantified circuits with *any number of alternations*.

**Definition 4.9.**

Q-MCSP =  $\{(f, s) \in \{0, 1\}^* \mid f \text{ is an } n\text{-ary Boolean function represented by its truth table, } s \text{ is an integer, and there exists an integer } k \text{ so that there is a } \Sigma_k\text{-circuit } C \text{ which computes } f \text{ of size } \leq s\}$ .

Q-MCSP can be easily solved in PSPACE (where we can do an arbitrary number of alternations that can depend on the input length, instead of being restricted to a constant number). We can then use similar arguments to show the following.

**Theorem 4.5.** *For any complexity class  $C$ , if Q-MCSP  $\in C$ , then there are no  $C$ -strong PRFGs in PSPACE.*

**Corollary 4.5.** *If  $Q\text{-MCSP} \in P/\text{poly}$ , then there are no strong PRFGs in PSPACE, so there are certainly no strong PRFGs in  $P/\text{poly}$ , so there are also no strong PRGs, one-way functions, or private-key cryptography.*

Thus there is strong evidence that  $\Sigma_k\text{-MCSP}$  and  $Q\text{-MCSP}$  are at least not in  $P/\text{poly}$ , and probably live higher up in the Polynomial Hierarchy or in PSPACE respectively. Like with regular MCSP, these results also hold for approximation versions of these problems with gap  $\tau(n)$  such that  $2^n/(\tau(n) \cdot n)$  is still superpolynomial in  $n$ , so these approximation versions are also likely hard.

### 4.3.3 Gap- $\Sigma_k\text{-MCSP}$ and Gap- $Q\text{-MCSP}$

By considering the approximation version of minimization for these more powerful kinds of circuits and applying the approach of Saks and Santhanam (2020) again, we can get stronger circuit lower bounds from the NP-hardness of Gap- $\Sigma_k\text{-MCSP}$  and Gap- $Q\text{-MCSP}$  under polynomial-time Turing reductions. This is particularly interesting since it seems like it would potentially be *easier* to show the NP-hardness of these problems since they appear even harder than MCSP (as they naturally live higher in the polynomial hierarchy), and yet we get a stronger result. We define Gap- $\Sigma_k\text{-MCSP}$  analogously to Gap-MCSP and proceed with a very similar proof.

**Theorem 4.6.** *Let  $\sigma : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$  be a function such that  $\sigma$  is superpolynomial in  $s$ , or  $\sigma(N, s) \in \omega(s^k)$  for all  $k \in \mathbb{N}$ . If  $\text{Gap}_{\sigma}\Sigma_k\text{-MCSP}$  is hard for NP under polynomial-time Turing reductions, then  $E \not\subseteq \Sigma_k P/\text{poly}$ .*

*Proof.* Let  $\mathcal{M}$  be the oracle machine reducing SAT to Gap- $\sigma\Sigma_k\text{-MCSP}$ . Consider the following simulation  $M$  of this machine on an input  $x$  with  $n = |x|$ . Whenever  $\mathcal{M}$  would have queried the oracle with  $(f, s)$ , if  $s \leq \log(n)^{1/2}$ ,  $M$  uses brute-force search to determine the correct answer to this query. First, similarly to before, we use time  $\text{poly}(n)2^{O(s \log(s))} = \text{poly}(n)n^{O(1)}$  to guess all possible  $\Sigma_k P$ -circuits (notice that there are at most about  $(k-1)\log(s)$  extra bits, which tell us how to partition the  $s$  auxiliary bits into the  $k$   $y_i$ 's, to guess compared to before). Then for each circuit  $C$ , to check if it computes  $f$  correctly, for each input  $x'$  to  $f$  we must consider all  $2^{\log(s)} = n^{1/2}$  possible values for the  $y_i$  and for each of them run CIRCUIT EVALUATION in polynomial time to determine  $C(x', y_1, y_2, \dots, y_k)$ , maintaining a polynomial runtime overall. Else if  $s > \log(n)^{1/2}$ ,  $M$  assumes the answer to the query is “yes.”

Now we have two possibilities. Either  $M$  simulates  $\mathcal{M}$  correctly on all but finitely many inputs, in which case  $P = NP$  and this implies  $E \not\subseteq \Sigma_k P / \text{poly}$ ,<sup>4</sup> or  $M$  simulates  $\mathcal{M}$  incorrectly on infinitely many inputs. If  $M$  simulates  $\mathcal{M}$  incorrectly on infinitely many inputs, then on each such input  $\mathcal{M}$  must query the oracle with  $(f, s)$  such that  $s > \log(n)^{1/2}$  and the answer to this query cannot be “yes.” Since our oracle is the promise problem  $\text{Gap}_{\sigma} \Sigma_k$ -MCSP, it is bound to answer “no” only if  $f$  requires  $\Sigma_k P$ -circuits of size  $> \sigma(|f|, s)$ . Since  $\sigma$  is superpolynomial in  $s$ ,  $f$  thus must require  $\Sigma_k$ -circuits superpolynomial in  $\log(n)^{1/2}$ , or simply superpolynomial in  $\log(n)$ . Since  $\mathcal{M}$  runs in polynomial time,  $|f| \in O(n^c)$  for some constant  $c$ , and is thus on  $k = O(\log(n))$  variables. Therefore  $f$  requires superpolynomial  $\Sigma_k$ -circuits. Notice this means that if we concatenate all of the queries  $\mathcal{M}$  makes to the oracle on this input to create a truth table of size polynomial in  $n$  (maybe padded with some 0’s at the end to that its length will be a power of 2), this will represent a function that also requires superpolynomial  $\Sigma_k$ -circuits.

Now we use the lemma from Gutfreund et al. (2007) to actually find these instances. Utilizing  $R$  from this lemma, for infinitely many  $n$ , if we run  $R$  on  $(n, a, \langle M \rangle)$ , then in polynomial time we will get at most three formulas of length polynomial in  $n$  such that  $M$  fails on at least one. Then we can run  $M$  on each of these formulas, keeping track of all of the queries made. Finally, concatenating these queries (at least one of which we know to have a true answer of “no”), we have found a truth table of size  $\text{poly}(n)$ , and thus on  $O(\log(n))$  variables, which requires superpolynomial-size  $\Sigma_k$ -circuits. Therefore since this process took time polynomial in  $n$ , and thus linear exponential time in  $O(\log(n))$ ,  $E \not\subseteq \Sigma_k P / \text{poly}$  in this case as well.  $\square$

**Corollary 4.6.** *This implication clearly also holds for any  $\sigma(N, s)$  that is super-*

<sup>4</sup>To show this, we follow a similar argument as for  $P = NP \Rightarrow E \not\subseteq P / \text{poly}$ . Assume for contradiction that  $P = NP$  and  $E \subseteq \Sigma_k P / \text{poly}$ . Now consider an algorithm  $\mathcal{A}$  which takes as input the description of a Turing machine  $\langle M \rangle$  and input  $x$  and simulates  $M$  on input  $x$  for  $2^{|x|}$  of its own steps. If at that point  $M$  hasn’t halted,  $\mathcal{A}$  rejects. If it has halted,  $\mathcal{A}$  answers the same way as  $M$ . The language  $\mathcal{A}$  decides is thus clearly in  $E \subseteq \Sigma_k P / \text{poly}$ , so it can be decided by a  $\Sigma_k P$ -circuit of size bounded by  $n^c$  some fixed  $c \in \mathbb{N}$ . Notice that  $\mathcal{A}$  can simulate any language in  $P$  for all large enough  $n$ , which implies  $P$  can be decided by  $\Sigma_k$ -circuits of size  $n^c$  too. However, for any fixed  $c$ , there exists a  $k'$  such that  $\Sigma_{k'} P$  contains a language which requires circuits of size greater than  $n^c$ . We again follow the approach of Kannan (1982) to show this, the only difference being that whenever we need to check what the circuit evaluates to on a given input, we use  $k$  extra levels to compute  $\Sigma_k$ -CSAT with that input plugged into the circuit, so that  $k' = 3k + 4$  will suffice. But if  $P = NP$ ,  $\Sigma_{k'} P = P$ , so we get a contradiction. Therefore  $P = NP$  implies  $E \not\subseteq \Sigma_k P / \text{poly}$ . Alternatively, just note that if  $P = NP$ , then since  $\Sigma_k P = P$ ,  $E \not\subseteq P / \text{poly} = \Sigma_k P / \text{poly}$ .



polynomial in  $\log(n)$  when  $s > \log(n)^{1/2}$  and  $N = \text{poly}(n)$ . Specifically, it holds for  $\sigma(N, s) = \tau(N) \cdot s$  such that  $\tau$  is superpolynomial in  $\log(N)$ .

Thus if, when given a truth table of size  $2^n$ , approximating  $\Sigma_k$ -MCSP within a factor superpolynomial in  $\log(2^n) = n$  is shown to be NP-hard via polynomial-time Turing reductions,  $E \not\subseteq \Sigma_k P/\text{poly}$ , while for regular MCSP we only get the implication that  $E \not\subseteq P/\text{poly}$ .

Using nearly identical arguments to above, one can show a similar result for Q-MCSP.

**Theorem 4.7.** *Let  $\sigma : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$  be a function such that  $\sigma(N, s)$  is superpolynomial in  $\log(n)$  when  $s > \log(n)^{1/2}$  and  $N = \text{poly}(n)$ . (Specifically, this holds for  $\sigma(N, s)$  which are superpolynomial in  $s$  or  $\sigma(N, s) = \tau(N) \cdot s$  such that  $\tau$  is superpolynomial in  $\log(N)$ .) If  $\text{Gap}_\sigma \text{Q-MCSP}$  is hard for NP under polynomial-time Turing reductions, then  $E \not\subseteq \text{PH}/\text{poly}$ .*

*Proof.* We only need to patch up this proof in two places. First, note that we can still use brute-force search to determine the correct answer to  $\text{Gap}_\sigma \text{Q-MCSP}$  queries with  $s(n) \leq \log(n)^{1/2}$  in polynomial time. Recall that in order for a Q-circuits to have size no more than  $s(n)$ , the length of the concatenated auxiliary inputs must also be bounded by  $s(n)$ , allowing at most  $s(n)$  alternations (if each  $y_i$  is only one bit). Thus brute-forcing a  $\text{Gap}_\sigma \text{Q-MCSP}$  instance is like doing so for a  $\text{Gap}_\sigma \Sigma_k \text{-MCSP}$  but with  $k = s(n)$ , which adds  $s \log(s)$  extra bits to guess and thus still leads to running time  $\text{poly}(n)2^{O(s \log(s))} = \text{poly}(n)n^{O(1)}$  to guess all possible Q-circuits. Therefore this process still takes polynomial time overall. Second,  $P = \text{NP}$  implies  $E \not\subseteq \text{PH}/\text{poly}$ .<sup>5</sup> With these two points noted, the rest of the proof goes through to establish the desired result.  $\square$

Indeed, we could get the stronger implication that  $E \not\subseteq \text{PSPACE}/\text{poly}$  (since  $\text{PSPACE}/\text{poly}$  can be defined as the set of languages with polynomial-size quantified circuits where the number of alternations can vary with  $n$ <sup>6</sup>) except that it is not clear how to show that  $P = \text{NP} \Rightarrow E \not\subseteq \text{PSPACE}/\text{poly}$  for the first case.

<sup>5</sup>This is due to essentially the same argument as the previous footnote since now  $\mathcal{A}$  is in  $E \subseteq \text{PH}/\text{poly}$ , so  $\mathcal{A}$  must specifically be in some  $\Sigma_k P/\text{poly}$  and the rest of the argument follows. Alternatively, just note that if  $P = \text{NP}$ , then since  $\text{PH} = P$ ,  $E \not\subseteq P/\text{poly} = \text{PH}/\text{poly}$ .

<sup>6</sup>See Balcázar et al. (1987).

#### 4.3.4 Parametric Honest, Natural, and Many-One Reductions for $\Sigma_k$ -MCSP and Q-MCSP

The result of Saks and Santhanam (2020) for parametric honest Turing reductions goes through pretty straightforwardly for  $\Sigma_k$ -MCSP.

**Theorem 4.8.** *If  $\Sigma_k$ -MCSP is NP-hard under parametric honest Turing reductions, then  $E \not\subseteq \Sigma_k P/\text{poly}$ .*

*Proof.* Let  $\mathcal{M}$  be the oracle machine reducing SAT to  $\Sigma_k$ -MCSP with parametric honest constant  $\epsilon > 0$ . Consider the following simulation  $M$  of this machine on an input  $x$  with  $n = |x|$ . Whenever  $\mathcal{M}$  would have queried the oracle with  $(f, s)$ ,  $M$  assumes the answer to the query is “yes.”

Now we have two possibilities. Either  $M$  simulates  $\mathcal{M}$  correctly on all but finitely many inputs, in which case  $P = NP$  and this implies  $E \not\subseteq \Sigma_k P/\text{poly}$ , or  $M$  simulates  $\mathcal{M}$  incorrectly on infinitely many inputs. If  $M$  simulates  $\mathcal{M}$  incorrectly on infinitely many inputs, then on each such input  $\mathcal{M}$  must query the oracle with  $(f, s)$  such that  $s \geq n^\epsilon$  and the answer to this query is “no.” Since  $\mathcal{M}$  runs in polynomial time,  $|f| \in O(n^c)$  for some constant  $c$ , and is thus on  $k = O(\log(n))$  variables. Therefore  $f$  requires exponential  $\Sigma_k$ -circuits. Notice this means that if we concatenate all of the queries  $\mathcal{M}$  makes to the oracle on this input to create a truth table of size polynomial in  $n$  (maybe padded with some 0’s at the end to that its length will be a power of 2), this will represent a function that also requires exponential and thus superpolynomial  $\Sigma_k$ -circuits.

Now we use the lemma from Gutfreund et al. (2007) to actually find these instances. Utilizing  $R$  from this lemma, for infinitely many  $n$ , if we run  $R$  on  $(n, a, \langle M \rangle)$ , then in polynomial time we will get at most three formulas of length polynomial in  $n$  such that  $M$  fails on at least one. Then we can run  $M$  on each of these formulas, keeping track of all of the queries made. Finally, concatenating these queries (at least one of which we know to have a true answer of “no”), we have found a truth table of size  $\text{poly}(n)$ , and thus on  $O(\log(n))$  variables, which requires exponential-size  $\Sigma_k$ -circuits. Therefore since this process took time polynomial in  $n$ , and thus linear exponential time in  $O(\log(n))$ ,  $E \not\subseteq \Sigma_k P/\text{poly}$  in this case as well.  $\square$

Likewise, using essentially the same arguments we can show

**Theorem 4.9.** *If Q-MCSP is NP-hard under parametric honest Turing reductions, then  $E \not\subseteq PH/\text{poly}$ .*

Saks and Santhanam (2020)'s argument for natural Turing reductions is ultimately along these same lines but has some additional complications. By fixing up a few lemmas for the  $\Sigma_k$ -MCSP variant, we can get this result to go through as well. First note that

**Definition 4.10.** A time-constructible function  $T : \mathbb{N} \rightarrow \mathbb{N}$  is one for which a function Turing machine can compute  $T(n)$  on input  $0^n$  in  $O(T(n))$  time.

**Definition 4.11.** For  $g : \mathbb{N} \rightarrow \mathbb{N}$ , a language  $\mathcal{L}$  is  $g(n)$ -robustly often in a complexity class  $\mathcal{C}$  if there is a language  $\mathcal{L}' \in \mathcal{C}$  where for any constant  $k$ , there is an integer  $m$  for which  $\mathcal{L}$  and  $\mathcal{L}'$  agree on each input  $x$  such that  $n = |x|$  satisfies  $m \leq n \leq g(m)^k$  (i.e.  $\mathcal{L}$  and  $\mathcal{L}'$  agree with each other on every input for increasingly long stretches). When  $g(n) = n$ , we will say  $\mathcal{L}$  is just robustly often in a complexity class  $\mathcal{C}$ .

We need the following five lemmas to prove our desired result.

**Lemma 4.3.** Let  $T : \mathbb{N} \rightarrow \mathbb{N}$  be a time-constructible function such that  $T(n) \geq n$  for all  $n$ . If SAT is  $T(\text{poly}(n))$ -robustly often in  $\text{DTIME}(T)$ , then for a given  $k$ ,  $\Sigma_k\text{P}$  is robustly often in  $\text{DTIME}(T(\text{poly}(\cdots T(\text{poly}(n)) \cdots)))$  where  $T(\text{poly}(n))$  has been applied  $k$  times.

*Proof.* Like Proposition 3 in Saks and Santhanam (2020), this is a parameterized version of Theorem 12 in Fortnow and Santhanam (2017), now for an arbitrary  $k$  instead of 2 specifically.  $\square$

**Lemma 4.4.** If  $E \subseteq \Sigma_k\text{P}/\text{poly}$ , then  $E = \Sigma_{k+2}\text{P}$ .

*Proof.* This follows essentially the same proof as Theorem 6.6 in Karp and Lipton (1980), which they credit to Meyer (while it is stated for  $\text{EXPTIME}$ , in this case  $\text{EXPTIME} = E$ ). They present the special case where  $k = 0$ , but this is easily extended. In short, in order to eliminate the advice, two additional layers of alternation are added.  $\square$

**Lemma 4.5.** Let  $T : \mathbb{N} \rightarrow \mathbb{N}$  be a time-constructible function such that  $T(n) \geq n$  and when  $T(\text{poly}(n))$  is applied  $k + 2$  times, the result is in  $o(2^n)$ . If SAT is  $T(\text{poly}(n))$ -robustly often in time  $T(n)$ , then  $E \not\subseteq \Sigma_k\text{P}/\text{poly}$ .

*Proof.* This proof proceeds similarly to that of Lemma 9 in Saks and Santhanam (2020). Suppose that SAT is  $T(\text{poly}(n))$ -robustly often in  $\text{DTIME}(T)$ . Using Lemma 4.3, this implies that  $\Sigma_{k+2}\text{P}$  is robustly often in  $\text{DTIME}(T(\text{poly}(\cdots T(\text{poly}(n)) \cdots)))$  where  $T(\text{poly}(n))$  has been applied  $k + 2$  times. Thus  $\Sigma_{k+2}\text{P}$  is robustly often in  $\text{DTIME}(T')$  for a time-constructible

function  $T'$  in  $o(2^n)$ . Now, assume for contradiction that  $E \subseteq \Sigma_k P/\text{poly}$ . Then by Lemma 4.4, this implies  $E = \Sigma_{k+2} P$ . Putting these two pieces together, we have that  $E$  is robustly often in  $\text{DTIME}(T')$  for a time-constructible function  $T'$  in  $o(2^n)$ . But this directly contradicts Proposition 4 from Saks and Santhanam (2020).  $\square$

**Lemma 4.6.** *If the size parameter of a  $\Sigma_k$ -MCSP instance is bounded above by  $s(n)$ , then  $\Sigma_k$ -MCSP can be solved in time  $\text{poly}(n)2^{O(s(n)^2)}$ .*

*Proof.* We can simply use brute-force search to determine the correct answer. First, we use time  $\text{poly}(n)2^{O(s \log(s))}$  to guess all possible  $\Sigma_k P$ -circuits (notice that there are at most about  $(k-1) \log(s)$  extra bits, which tell us how to partition the  $s$  auxiliary bits into the  $k$   $y_i$ 's, to guess compared to before). Then for each circuit  $C$ , to check if it computes  $f$  correctly, for each input  $x$  to  $f$  we must consider all  $2^{\log(s)}$  possible values for the  $y_i$  and for each of them run CIRCUIT EVALUATION in polynomial time to determine  $C(x', y_1, y_2, \dots, y_k)$ . Notice this takes time  $\text{poly}(n)2^{O(s \log(s))} \cdot \text{poly}(n)2^{\log(s)} = \text{poly}(n)2^{O(s \log(s))}$ , which is better than we need for the lemma.  $\square$

**Lemma 4.7.** *Let  $r(n) = n^{\log(n)}$ ,  $s_1(n) = \log(n)^{\log \log n}$ , and  $T(n) = \text{poly}(r(n))2^{O(s_1(r(n))^2)}$ . Then for all  $k$ ,  $T(\text{poly}(n))$  applied to itself  $k$  times is in  $o(2^n)$ .*

*Proof.* Let  $R$  be the set of functions bounded by a function of the form  $f(n) = 2^{\log(n)^{O(\log \log(n))}}$ . Notice that all functions in  $R$  are in  $2^{o(n)}$  and thus certainly in  $o(2^n)$ . It can be shown that  $R$  is closed under products and composition, so in order to obtain the desired result it suffices to show that  $T(\text{poly}(n))$  is in  $R$ . Indeed, since  $\text{poly}(n) \in R$  we just need to show that  $T(n) \in R$ . Since  $\text{poly}(r(n)) = \text{poly}(n^{\log(n)}) = \text{poly}(2^{\log(n)^2}) \in R$ , all that is left to show is that  $s_1(r(n))^2 = \log(n)^{O(\log \log(n))}$ . This is easy to see since  $r(n) = 2^{\log(n)^2} \Rightarrow \log(r(n)) = \log(n)^2$  and  $s_1(n) = \log(n)^{\log \log n}$ .  $\square$

Plugging these lemmas into Saks and Santhanam (2020)'s proof, we establish the following:

**Theorem 4.10.** *If  $\Sigma_k$ -MCSP is NP-hard under natural Turing reductions, then  $E \not\subseteq \Sigma_k P/\text{poly}$ .*

*Proof.* We will give the outline of this proof and how the above lemmas fit into it. See Saks and Santhanam (2020) for the full details.

Let  $\mathcal{M}$  be the oracle machine naturally reducing SAT to  $\Sigma_k$ -MCSP with  $s : \mathbb{N} \rightarrow \mathbb{N}$  being the function such that all inputs of length  $n$  only use queries with size parameter  $s(n)$ . Let  $r(n) = n^{\log(n)}$ ,  $s_1(n) = \log(n)^{\log \log n}$ , and  $s_2(n) = \log(n)^{\log \log \log n}$ . To deal with this more complicated situation, we will consider two simulations of  $\mathcal{M}$ ,  $M_1$  and  $M_2$ .

On an input formula  $x$  with  $n = |x|$ ,  $M_1$  proceeds as follows. First, it computes *padded* versions of  $x$  such that  $x_i$  is an equivalent formula to  $x$  of length  $i$  for  $n \leq i \leq r(n)$  (one can choose an encoding of Boolean formulas for which this is easily doable). Then  $M_1$  tries running  $\mathcal{M}$  on each  $x_i$ , checking if the size parameter of the first query is bounded above by  $s_1(i)$ . If so, it brute-force solves this and all future  $\Sigma_k$ -MCSP queries each in time  $\text{poly}(i)2^{O(s_1(i)^2)}$  by following Lemma 4.6, thereby correctly deciding  $x$  in time  $\text{poly}(r(n))2^{O(s_1(r(n))^2)}$  overall. Else if the size parameter is too big,  $M_1$  tries the next  $x_i$ . If none of the  $x_i$  work,  $M_1$  gives up and just simulates  $\mathcal{M}$  on  $x$  using brute-force search on each query regardless of the size parameter.

Based on this, it can be shown that either SAT can be decided in time  $\text{poly}(r(n))2^{O(s_1(r(n))^2)}$ , or there is an infinite sequence of disjoint intervals  $I_j = [n_j, r(n_j))$  of input lengths with large associated size parameters, specifically such that  $s(n) > s_1(n)$  for  $n \in I_j$ . Now, assume that the first case holds. Note that  $T(n) = \text{poly}(r(n))2^{O(s_1(r(n))^2)}$  is a time-constructible function such that  $T(n) \geq n$  and, by Lemma 4.7,  $T(\text{poly}(n))$  applied to itself  $k+2$  times is in  $o(2^n)$ . Further, since in this case SAT is in time  $T(n)$ , it is also in  $T(\text{poly}(n))$ -robustly often in  $T(n)$ , so by Lemma 4.5,  $E \not\subseteq \Sigma_k\text{P}/\text{poly}$ .

Therefore going forward we will assume the second case holds. Now we will define the second simulation,  $M_2$ . On an input formula  $x$  with  $n = |x|$ ,  $M_2$  follows the standard search-to-decision reduction for SAT to try to find a satisfying assignment for  $x$ . This reduction proceeds as follows. If  $x \notin \text{SAT}$  we immediately reject, else while  $x$  has unset variables, we set one of them,  $x_i$ , to 0 and determine if the resulting formula is satisfiable or not. If so, we add  $x_i = 0$  to our satisfying assignment, else we add  $x_i = 1$  to our satisfying assignment, and either way we recurse on  $x$  with  $x_i$  set appropriately. As long as we correctly determined whether each formula was satisfiable along the way, by the end we will have a satisfying assignment for  $x$  (if such exists). We thus check whether we ended up with a satisfying assignment, accept if so, and reject if not. We will slightly modify this procedure for  $M_2$  by padding each formula we seek to determine is in SAT or not so that they are all of length  $|x|$  and not any shorter. Then whenever we have such a formula, to determine whether it is in SAT we simulate running  $\mathcal{M}$  on it. When a

$\Sigma_k$ -MCSP query is encountered, if its size parameter is at most  $s_2(n)$  then  $M_2$  uses Lemma 4.6 to brute-force the answer in time  $\text{poly}(n)2^{O(s_2(n)^2)}$ , and else it assumes the answer is “yes.” This completes the characterization of  $M_2$ .

Now, notice that  $M_2$  always halts within time  $\text{poly}(n)2^{O(s_2(n)^2)}$ . However, unlike  $M_1$ , it may be incorrect on some inputs if it encounters a query with size parameter  $s(n) > s_2(n)$  for which the answer should have been “no.” Since the search-to-decision reduction ensures that whenever  $M_2$  accepts it must be correct, this must involve inputs it rejects but should have accepted.

Instead of considering whether or not  $M_2$  solves SAT correctly, we consider whether or not there is a set  $S \subseteq \mathbb{N}$  such that for any constant  $k$ , there is an integer  $m$  for which  $M_2$  solves SAT correctly on each input  $x$  such that  $n = |x|$  satisfies  $m \leq n \leq 2^{ks_2(m)^3}$ . If so, then SAT is  $2^{s_2(n)^3}$ -robustly often in time  $\text{poly}(n)2^{O(s_2(n)^2)}$ , or time  $2^{s_2(n)^3}$  to be generous. Note that  $T(n) = 2^{s_2(n)^3}$  is a time-constructible function such that  $T(n) \geq n$  and that by the reasoning of Lemma 4.7,  $T(\text{poly}(n))$  applied to itself  $k + 2$  times is in  $o(2^n)$ . Thus, since in this case SAT is  $T(\text{poly}(n))$ -robustly often in  $T(n)$ , by Lemma 4.5,  $E \not\subseteq \Sigma_k\text{P}/\text{poly}$ .

So, now we assume that there is no such set  $S$ . Then by combining this with the existence of the  $I_j$ , it can be shown that there is an infinite sequence of input lengths  $m_i$  such that (1)  $M_2$  is incorrect on some input  $x$  of length  $m_i$  such that on input  $x$   $M$  makes a query with size parameter at least  $s_1(m_i)$  and (2) for all input lengths  $m$  such that  $m_i \leq m \leq 2^{s_2(m_i)^3}$ ,  $s(m) \geq s_1(m)$ . We can then use this to modify the argument of the lemma from Gutfreund et al. (2007) to determine an algorithm that for each  $i$  outputs a truth table  $f_i$  in time  $\text{poly}(|f_i|)2^{O(s_2(|f_i|)^3)}$  which is not computed by  $\Sigma_k$ -circuits of size  $s_1(|f_i|)$ . This truth table is naturally on  $\log(|f_i|)$  variables, but since  $s_2(|f_i|)^3 \geq \log(|f_i|)$ , we can consider it a truth table on  $s_2(|f_i|)^3$  variables where the inputs after the first  $\log(|f_i|)$  are ignored in determining the output. Then we have computed  $f_i$  in linear exponential time in the number of input variables. Since  $f_i$  requires  $\Sigma_k$ -circuits of size  $s_1(|f_i|)$ , which is superpolynomial in  $s_2(|f_i|)^3$ , in this case we have also shown that  $E \not\subseteq \Sigma_k\text{P}/\text{poly}$ . □

We can also adapt the lemmas used to establish this result to show

**Theorem 4.11.** *If Q-MCSP is NP-hard under natural Turing reductions, then  $E \not\subseteq \text{PH}/\text{poly}$ .*

In particular, all we need are versions of Lemmas 4.4, 4.5, and 4.6.

**Lemma 4.8.** *If  $E \subseteq \text{PH/poly}$ , then  $E = \Sigma_{k'}\text{P}$  for some  $k'$ .*

*Proof.* In this case the E-complete language used in Meyer's proof of Theorem 6.6 in Karp and Lipton (1980) is in  $\text{PH/poly}$ , so it must specifically be in  $\Sigma_k\text{P}$  for some  $k$  and thus actually  $E \subseteq \Sigma_k\text{P}$ . Now by applying Lemma 4.4, we get the desired result.  $\square$

**Lemma 4.9.** *Let  $T : \mathbb{N} \rightarrow \mathbb{N}$  be a time-constructible function such that  $T(n) \geq n$  and when  $T(\text{poly}(n))$  is applied a constant number of times, the result is in  $o(2^n)$ . If SAT is  $T(\text{poly}(n))$ -robustly often in time  $T(n)$ , then  $E \not\subseteq \text{PH/poly}$ .*

*Proof.* Assume for contradiction that  $E \subseteq \text{PH/poly}$ . Then by Lemma 4.8, this implies  $E = \Sigma_{k'}\text{P}$  for some constant  $k'$ .

Now, since we're assuming that SAT is  $T(\text{poly}(n))$ -robustly often in  $\text{DTIME}(T)$ , by Lemma 4.3, this implies that  $\Sigma_{k'}\text{P}$  is robustly often in  $\text{DTIME}(T(\text{poly}(\cdots T(\text{poly}(n)) \cdots)))$  where  $T(\text{poly}(n))$  has been applied  $k'$  times. Thus  $\Sigma_{k'}\text{P}$  is robustly often in  $\text{DTIME}(T')$  for a time-constructible function  $T'$  in  $o(2^n)$ .

Putting these two pieces together, we have that  $E$  is robustly often in  $\text{DTIME}(T')$  for a time-constructible function  $T'$  in  $o(2^n)$ . But this directly contradicts Proposition 4 from Saks and Santhanam (2020).  $\square$

**Lemma 4.10.** *If the size parameter of a Q-MCSP instance is bounded above by  $s(n)$ , then Q-MCSP can be solved in time  $\text{poly}(n)2^{O(s(n)^2)}$ .*

*Proof.* As noted when discussing the gap variant, we only need to guess at most about  $(s - 1) \log(s)$  extra bits, keeping us within  $\text{poly}(n)2^{O(s \log(s))}$  time like before.  $\square$

With these three lemmas, the same arguments go through to establish the theorem.

We can also give a result for just general polynomial time many-one reductions by extending Theorem 1.6 of Murray and Williams (2017), which as discussed earlier shows that if MCSP is NP-hard under polynomial-time many-one reductions then  $\text{EXP} \not\subseteq \text{NP} \cap \text{P/poly}$  (which implies  $\text{EXP} \neq \text{ZPP}$ ), to  $\Sigma_k\text{-MCSP}$ . We show

**Theorem 4.12.** *If  $\Sigma_k\text{-MCSP}$  is NP-hard under polynomial-time many-one reductions (or is even only hard for the sparse languages in NP), then  $\text{EXP} \neq \text{NP} \cap \Sigma_k\text{P/poly}$ . (Note that even  $\text{EXP} \neq \text{NP} \cap \text{P/poly}$  implies that  $\text{EXP} \neq \text{ZPP}$ .)*

*Proof.* Suppose for contradiction that  $\text{EXP} = \text{NP} \cap \Sigma_k\text{P}/\text{poly}$ . Then  $\text{EXP} \subseteq \Sigma_k\text{P}/\text{poly}$ , so with the hardness assumption, by Lemma 4.11 to follow,  $\text{NEXP} = \text{EXP}$ . Thus  $\text{NEXP} = \text{EXP} = \text{NP} \cap \Sigma_k\text{P}/\text{poly} \subseteq \text{NP}$  so  $\text{NP} = \text{NEXP}$ , contradicting the nondeterministic time hierarchy. Therefore  $\text{EXP} \neq \text{NP} \cap \Sigma_k\text{P}/\text{poly}$ .  $\square$

Now, this essential lemma is

**Lemma 4.11.** *If every sparse language in NP has a polynomial-time many-one reduction to  $\Sigma_k$ -MCSP, then  $\text{EXP} \subseteq \Sigma_k\text{P}/\text{poly}$  implies  $\text{EXP} = \text{NEXP}$ .*

*Proof.* This follows the exact same proof as the one given in Murray and Williams (2017) for Theorem 4.1 with each reference to P/poly replaced with  $\Sigma_k\text{P}/\text{poly}$  and the use of Lemma 4.6 in place of traditional circuit brute-forcing.  $\square$

We can also similarly show this lemma for Q-MCSP, but this time don't run into a barrier with getting a full  $\text{E} \not\subseteq \text{PSPACE}/\text{poly}$  result (because we don't need to use an implication like  $\text{P} = \text{NP} \Rightarrow \text{E} \not\subseteq \text{PSPACE}/\text{poly}$ ), establishing

**Theorem 4.13.** *If Q-MCSP is NP-hard under polynomial-time many-one reductions (or is even only hard for the sparse languages in NP), then  $\text{EXP} \neq \text{NP} \cap \text{PSPACE}/\text{poly}$ .*

Therefore we observe the general pattern that even though  $\Sigma_k$ -MCSP and Q-MCSP intuitively appear to be clearly harder than MCSP, if they were shown to be NP-hard under many kinds of reductions with barriers for MCSP we would get even stronger, more out-of-reach results. This happens in particular with implications that involve circuit classes, again reinforcing the intuition that our inability to understand the hardness of MCSP and its variants is due to our lack of understanding of lower bounds for circuits of the appropriate type. Thus since we understand  $\Sigma_k$ - and Q-circuits even less than regular ones, we can make sense of why there are even more barriers to showing the hardness of the  $\Sigma_k$ -MCSP and Q-MCSP variants compared to regular MCSP.





# Chapter 5

## Conclusion

### 5.1 Takeaways So Far

The study of the complexity of MCSP is a study of the meta-mathematics of complexity, while complexity theory is already a meta project. Thus, studying MCSP can be seen as a next step on the path that took us from trying to fulfill Hilbert's program, to computability, and then to complexity theory. Establishing the implications of MCSP being shown to NP-complete under different kinds of reductions confronts the fundamental question: what mathematical barriers are there to using mathematics to understand how hard it is for computation to determine how hard it is to compute functions?

Not only is this kind of question philosophically interesting in its own right, but it also underpins the implications MCSP has for Impagliazzo's five worlds, which are of great practical importance. On the one hand, if we were in Heuristica, this would have significant positive implications because of how many useful problems are in NP which we would like to be able to solve on average. Unfortunately, this would also imply that secure cryptography doesn't exist, undermining private open-channel communication. We would especially like to rule out (more variants of) Pessiland, where we lose out on both good things. Understanding the hardness of MCSP is thus crucial as it shows promise in ruling out both of these possibilities from Impagliazzo's five worlds, so that either  $P = NP$  or we have some form of secure cryptography.

Despite the great interest in resolving these questions, unfortunately there is a long line of work which shows that it is hard to show the hardness

If ...	is NP-complete under ...	then ...
MCSP	natural RP many-one reductions	$\text{RET} \not\subseteq \text{P/poly}$
MCSP	natural pseudo-deterministic BPP many-one reductions	$\text{BPE} \not\subseteq \text{P/poly}$
$\Sigma_k$ -MCSP	poly-time many-one reductions	$\text{EXP} \neq$ $\text{NP} \cap \Sigma_k \text{P/poly}$
$\Sigma_k$ -MCSP	natural or parametric-honest poly-time Turing reductions	$\text{E} \not\subseteq \Sigma_k \text{P/poly}$
Q-MCSP	poly-time many-one reductions	$\text{EXP} \neq$ $\text{NP} \cap \text{PSPACE/poly}$
Q-MCSP	natural or parametric-honest poly-time Turing reductions	$\text{E} \not\subseteq \text{PH/poly}$
$\text{Gap}_\sigma$ MCSP*	poly-time Turing reductions	$\text{E} \not\subseteq \text{P/poly}$
$\text{Gap}_\sigma \Sigma_k$ -MCSP*	poly-time Turing reductions	$\text{E} \not\subseteq \Sigma_k \text{P/poly}$
$\text{Gap}_\sigma$ Q-MCSP*	poly-time Turing reductions	$\text{E} \not\subseteq \text{PH/poly}$

**Table 5.1** A summary of the additions of this thesis to the line of work showing that it is hard to show the hardness of MCSP and various variants on the problem. \*For certain functions  $\sigma$ .

of MCSP and related problems, which we have added to in this thesis. See Table 5.1 for a summary of these additions.

Some open directions which remain include investigating  $\Sigma_k$ -MCSP and Q-MCSP further and exploring results for pseudo-deterministic and fully randomized reductions.

## 5.2 Future Work

### 5.2.1 Further Exploring $\Sigma_k$ -MCSP and Q-MCSP

From the results I have presented, it is natural to ask whether all of the theorems involving Q-MCSP can be extended to imply  $E \not\subseteq PSPACE/poly$  as one would naturally expect instead of just  $E \not\subseteq PH/poly$ . Additionally, it would be interesting to further explore the  $\Sigma_k$ -MCSP and Q-MCSP variants and see if, like MCSP, they can be connected to a variety of interesting areas in complexity and theoretical computer science generally.

### 5.2.2 Randomized Reductions

It is interesting to note that there is a version of the lemma of Gutfreund et al. (2007) for BPP algorithms, prompting the question of whether the results of Saks and Santhanam (2020) and the extensions of their technique I have presented in this thesis could be further extended to BPP Turing reductions. However, we run into the same issue as we did when trying to extend the result of Kabanets and Cai (2000) to natural BPP many-one reductions. Namely, we end up finding many *different* hard truth tables when using different random strings so that we are not able to compute a unique hard function.

Could restricting to some kind of pseudo-deterministic reduction save us again? First we have to extend our definition of what this means to Turing reductions instead of just many-one. A natural extension that is useful for our context is that a *pseudo-deterministic BPP Turing reduction* is a BPP Turing reduction which on a given input uses the same set of canonical queries to produce the correct answer with probability  $\geq 3/4$ . With this new kind of reduction, the array of Saks and Santhanam (2020) results *almost* goes through easily (with the modification that the implication is now  $BPE \not\subseteq P/poly$ ,  $BPE \not\subseteq \Sigma_k P/poly$ , or  $BPE \not\subseteq PH/poly$  instead of  $E \not\subseteq P/poly$ ,  $E \not\subseteq \Sigma_k P/poly$ , or  $E \not\subseteq PH/poly$  respectively).

The holdup is that the search algorithm  $R$  in the randomized version of the lemma of Gutfreund et al. (2007) is not (at least not obviously) pseudo-deterministic—it is not guaranteed to produce the *same* set of instances the purported SAT algorithm fails on with high probability. If this was shown to be the case, perhaps only for some natural kind of purported SAT algorithm which is produced by our simulation of the reduction at hand, then the results would go through. Indeed, consider if we have a purported BPP algorithm for SAT called BSAT so that

$\text{BSAT}(x, u)$  indicates the output of BSAT on input formula  $x(v_1, v_2, \dots, v_n)$  with randomness  $u$ . Then by examining the approach of Gutfreund et al. (2007) it is not difficult to see that if for any partial assignment  $\alpha_i \in \{0, 1\}$ ,  $\text{BSAT}(x(v_1, v_2, \dots, v_n), u) = \text{BSAT}(x(v_1, v_2, \dots, v_n), u')$  implies  $\text{BSAT}(x(\alpha_1, \alpha_2, \dots, \alpha_i, v_{i+1}, v_{i+2}, \dots, v_n), u) = \text{BSAT}(x(\alpha_1, \alpha_2, \dots, \alpha_i, v_{i+1}, v_{i+2}, \dots, v_n), u')$  (i.e. the output of BSAT being the same under two different randomness strings is preserved under partial assignment), then their  $R$  will behave pseudo-deterministically. However, this does not seem like a natural property for BSAT to possess and it is not clear how a certain special kind of reduction would lead the simulation we make based off of it to have this property.

Thus we are left with the interesting further question to explore: can  $R$  in the randomized version of the lemma of Gutfreund et al. (2007) be made pseudo-deterministic? Can we come up with some other strategy to pseudo-deterministically find the instances an incorrect algorithm for SAT fails on? If we could answer these questions in the affirmative, we would find even more barriers towards proving the NP-hardness of MCSP and the variants on it that we have discussed. Of particular interest, we would be able to show that not even general pseudo-deterministic BPP Turing reductions can be used to show the hardness of Gap-MCSP (for sufficient gap) without proving  $\text{BPE} \not\subseteq \text{P/poly}$ , providing a further roadblock for using this approach to eliminate Heuristica. On the other hand, is there reason to believe that no such pseudo-deterministic version of  $R$  should exist? Or, working the other side of the problem, might there be some way to show MCSP is NP-hard using pseudo-deterministic BPP Turing reductions? Can the NP-hardness results for certain variants of MCSP which were mentioned at the beginning of Chapter 4 be accomplished with pseudo-deterministic BPP Turing reductions instead of fully randomized ones (when such were needed)? (Those which use randomized reductions include Ilango (2019), Ilango et al. (2020), and Ilango (2020).)

Exploring these questions would help further elucidate what kinds of randomized reductions would be necessary to show the NP-hardness of MCSP without also getting clearly out-of-reach results, furthering our understanding of MCSP and the many connections it has to diverse areas of theoretical computer science.

# Bibliography

Allender, Eric, Harry Buhrman, Michal Koucký, Dieter van Melkebeek, and Detlef Ronneburger. 2006. Power from random strings. *SIAM J Comput* 35(6):1467–1493. doi:10.1137/050628994. URL <https://doi.org/10.1137/050628994>.

Allender, Eric, Lisa Hellerstein, Paul McCabe, Toniann Pitassi, and Michael E. Saks. 2008. Minimizing disjunctive normal form formulas and  $AC^0$  circuits given a truth table. *SIAM J Comput* 38(1):63–84. doi:10.1137/060664537. URL <https://doi.org/10.1137/060664537>.

Allender, Eric, and Shuichi Hirahara. 2019. New insights on the (non-)hardness of circuit minimization and related problems. *ACM Trans Comput Theory* 11(4):27:1–27:27. doi:10.1145/3349616. URL <https://doi.org/10.1145/3349616>.

Allender, Eric, Dhiraj Holden, and Valentine Kabanets. 2017. The minimum oracle circuit size problem. *Comput Complex* 26(2):469–496. URL <https://doi.org/10.1007/s00037-016-0124-0>.

Allender, Eric, Michal Koucký, Detlef Ronneburger, and Sambuddha Roy. 2011. The pervasive reach of resource-bounded kolmogorov complexity in computational complexity theory. *J Comput Syst Sci* 77(1):14–40. doi:10.1016/j.jcss.2010.06.004. URL <https://doi.org/10.1016/j.jcss.2010.06.004>.

Balcázar, José L., Josep Díaz, and Joaquim Gabarró. 1987. On characterizations of the class PSPACE/poly. *Theor Comput Sci* 52:251–267. doi:10.1016/0304-3975(87)90111-3. URL [https://doi.org/10.1016/0304-3975\(87\)90111-3](https://doi.org/10.1016/0304-3975(87)90111-3).

Dawson, John W. 1997. *Logical Dilemmas: The Life and Work of Kurt Gödel*. Wellesley, MA: A. K. Peters.

- Doxiadis, Apostolos, and Christos Papadimitriou. 2009. *Logicomix: An Epic Search for Truth*. New York: Bloomsbury USA.
- Fortnow, Lance, and Rahul Santhanam. 2017. Robust simulations and significant separations. *Inf Comput* 256:149–159. doi:10.1016/j.ic.2017.07.002. URL <https://doi.org/10.1016/j.ic.2017.07.002>.
- Gat, Eran, and Shafi Goldwasser. 2011. Probabilistic search algorithms with unique answers and their cryptographic applications. *Electron Colloquium Comput Complex* 18:136. URL <http://ecc.hpi-web.de/report/2011/136>.
- Goldwasser, Shafi, Ofer Grossman, Sidhanth Mohanty, and David P. Woodruff. 2020. Pseudo-deterministic streaming. In *11th Innovations in Theoretical Computer Science Conference, ITCS 2020, January 12-14, 2020, Seattle, Washington, USA*, ed. Thomas Vidick, *LIPICs*, vol. 151, 79:1–79:25. Schloss Dagstuhl - Leibniz-Zentrum für Informatik. doi:10.4230/LIPICs.ITCS.2020.79. URL <https://doi.org/10.4230/LIPICs.ITCS.2020.79>.
- Gutfreund, Dan, Ronen Shaltiel, and Amnon Ta-Shma. 2007. If NP languages are hard on the worst-case, then it is easy to find their hard instances. *Comput Complex* 16(4):412–441. doi:10.1007/s00037-007-0235-8. URL <https://doi.org/10.1007/s00037-007-0235-8>.
- Håstad, Johan, Russell Impagliazzo, Leonid A. Levin, and Michael Luby. 1999. A pseudorandom generator from any one-way function. *SIAM J Comput* 28(4):1364–1396. doi:10.1137/S0097539793244708. URL <https://doi.org/10.1137/S0097539793244708>.
- Hirahara, Shuichi. 2018. Non-black-box worst-case to average-case reductions within NP. In *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*, ed. Mikkel Thorup, 247–258. IEEE Computer Society. URL <https://doi.org/10.1109/FOCS.2018.00032>.
- . 2020. Characterizing average-case complexity of PH by worst-case meta-complexity. *Electron Colloquium Comput Complex* 27:143. URL <https://ecc.weizmann.ac.il/report/2020/143/>.
- . 2021. Average-case hardness of NP from exponential worst-case hardness assumptions. *Electron Colloquium Comput Complex* 28:58. URL <https://ecc.weizmann.ac.il/report/2021/058>.

Hirahara, Shuichi, Igor Carboni Oliveira, and Rahul Santhanam. 2018. NP-hardness of minimum circuit size problem for OR-AND-MOD circuits. In *33rd Computational Complexity Conference, CCC 2018, June 22-24, 2018, San Diego, CA, USA*, ed. Rocco A. Servedio, *LIPICs*, vol. 102, 5:1–5:31. Schloss Dagstuhl - Leibniz-Zentrum für Informatik. doi:10.4230/LIPICs.CCC.2018.5. URL <https://doi.org/10.4230/LIPICs.CCC.2018.5>.

Hirahara, Shuichi, and Rahul Santhanam. 2017. On the average-case complexity of MCSP and its variants. In *32nd Computational Complexity Conference, CCC 2017, July 6-9, 2017, Riga, Latvia*, ed. Ryan O'Donnell, *LIPICs*, vol. 79, 7:1–7:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik. doi:10.4230/LIPICs.CCC.2017.7. URL <https://doi.org/10.4230/LIPICs.CCC.2017.7>.

Hirahara, Shuichi, and Osamu Watanabe. 2016. Limits of minimum circuit size problem as oracle. In *31st Conference on Computational Complexity, CCC 2016, May 29 to June 1, 2016, Tokyo, Japan*, ed. Ran Raz, *LIPICs*, vol. 50, 18:1–18:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik. URL <https://doi.org/10.4230/LIPICs.CCC.2016.18>.

Hitchcock, John M., and Aduri Pavan. 2015. On the NP-completeness of the minimum circuit size problem. In *35th IARCS Annual Conference on Foundation of Software Technology and Theoretical Computer Science, FSTTCS 2015, December 16-18, 2015, Bangalore, India*, eds. Prahladh Harsha and G. Ramalingam, *LIPICs*, vol. 45, 236–245. Schloss Dagstuhl - Leibniz-Zentrum für Informatik. URL <https://doi.org/10.4230/LIPICs.FSTTCS.2015.236>.

Ilango, Rahul. 2019.  $AC^0$  lower bounds and NP-hardness for variants of MCSP. *Electron Colloquium Comput Complex* 26:21. URL <https://eccc.weizmann.ac.il/report/2019/021>.

———. 2020. Constant depth formula and partial function versions of MCSP are hard. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Online Conference, November 16-19, 2020*, 1–52. IEEE Computer Society. URL <https://www.rahulilango.com/papers/FOCS2020.pdf>.

Ilango, Rahul, Bruno Loff, and Igor C. Oliveira. 2020. NP-Hardness of Circuit Minimization for Multi-Output Functions. In *35th Computational Complexity Conference (CCC 2020)*, ed. Shubhangi Saraf, *Leibniz International Proceedings in Informatics (LIPIcs)*, vol. 169, 22:1–22:36. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum für Informatik. doi:10.4230/LIPICs.CCC.2020.22. URL <https://drops.dagstuhl.de/opus/volltexte/2020/12574>.



Impagliazzo, Russell. 1995. A personal view of average-case complexity. In *Proceedings of the Tenth Annual Structure in Complexity Theory Conference, Minneapolis, Minnesota, USA, June 19-22, 1995*, 134–147. IEEE Computer Society. URL <https://doi.org/10.1109/SCT.1995.514853>.

Impagliazzo, Russell, and Michael Luby. 1989. One-way functions are essential for complexity based cryptography (extended abstract). In *30th Annual Symposium on Foundations of Computer Science, Research Triangle Park, North Carolina, USA, 30 October - 1 November 1989*, 230–235. IEEE Computer Society. doi:10.1109/SFCS.1989.63483. URL <https://doi.org/10.1109/SFCS.1989.63483>.

Kabanets, Valentine, and Jin-yi Cai. 2000. Circuit minimization problem. In *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing, May 21-23, 2000, Portland, OR, USA*, eds. F. Frances Yao and Eugene M. Luks, 73–79. ACM. URL <https://doi.org/10.1145/335305.335314>.

Kannan, Ravi. 1982. Circuit-size lower bounds and non-reducibility to sparse sets. *Inf Control* 55(1-3):40–56. doi:10.1016/S0019-9958(82)90382-5. URL [https://doi.org/10.1016/S0019-9958\(82\)90382-5](https://doi.org/10.1016/S0019-9958(82)90382-5).

Karp, Richard M., and Richard J. Lipton. 1980. Some connections between nonuniform and uniform complexity classes. In *Proceedings of the 12th Annual ACM Symposium on Theory of Computing, April 28-30, 1980, Los Angeles, California, USA*, eds. Raymond E. Miller, Seymour Ginsburg, Walter A. Burkhard, and Richard J. Lipton, 302–309. ACM. doi:10.1145/800141.804678. URL <https://doi.org/10.1145/800141.804678>.

Liu, Yanyi, and Rafael Pass. 2020. On one-way functions and Kolmogorov complexity. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, 1243–1254. IEEE. doi:10.1109/FOCS46700.2020.00118. URL <https://doi.org/10.1109/FOCS46700.2020.00118>.

———. 2021. On one-way functions from NP-complete problems. *Electron Colloquium Comput Complex* 28:59. URL <https://ecc.weizmann.ac.il/report/2021/059>.

Murray, Cody D., and R. Ryan Williams. 2017. On the (non) NP-hardness of computing circuit complexity. *Theory Comput* 13(1):1–22. URL <https://doi.org/10.4086/toc.2017.v013a004>.

Razborov, Alexander A., and Steven Rudich. 1997. Natural proofs. *J Comput Syst Sci* 55(1):24–35. URL <https://doi.org/10.1006/jcss.1997.1494>.

Ren, Hanlin, and Rahul Santhanam. 2021. Hardness of KT characterizes parallel cryptography. *Electron Colloquium Comput Complex* 28:57. URL <https://eccc.weizmann.ac.il/report/2021/057>.

Saks, Michael, and Rahul Santhanam. 2020. Circuit lower bounds from NP-hardness of MCSP under Turing reductions. In *35th Computational Complexity Conference, CCC 2020, July 28-31, 2020, Saarbrücken, Germany (Virtual Conference)*, ed. Shubhangi Saraf, *LIPICs*, vol. 169, 26:1–26:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik. URL <https://doi.org/10.4230/LIPICs.CCC.2020.26>.

Santhanam, Rahul. 2020. Pseudorandomness and the minimum circuit size problem. In *11th Innovations in Theoretical Computer Science Conference, ITCS 2020, January 12-14, 2020, Seattle, Washington, USA*, ed. Thomas Vidick, *LIPICs*, vol. 151, 68:1–68:26. Schloss Dagstuhl - Leibniz-Zentrum für Informatik. URL <https://doi.org/10.4230/LIPICs.ITCS.2020.68>.