



DEGREE PROJECT IN THE FIELD OF TECHNOLOGY
ENGINEERING PHYSICS
AND THE MAIN FIELD OF STUDY
COMPUTER SCIENCE AND ENGINEERING,
SECOND CYCLE, 30 CREDITS
STOCKHOLM, SWEDEN 2018

Hardness of showing hardness of the minimum circuit size problem

EMANUEL GEDIN

Hardness of showing hardness of the minimum circuit size problem

EMANUEL GEDIN

Date: July 2018
Supervisor: Per Austrin
Examiner: Johan Håstad

Abstract

The problem of finding the smallest size of a circuit that computes a given boolean function, usually referred to as the minimum circuit size problem (MCSP), has been studied for many years but it is still unknown whether or not the problem is NP-hard. With this in mind we study properties of potential reductions to this problem. The reductions in focus are local natural reductions which has been common in other well-known proofs of NP-hardness. We present a generalized method that shows the existence of an algorithm solving any problem which has a local natural reduction to MCSP. In particular we show that if the decision problem of distinguishing satisfiable 3-SAT instances from those where at most $7/8 + o(1)$ of the clauses can be satisfied has a reduction to MCSP where any arbitrary bit of the output can be computed in $\mathcal{O}(n^{1-\epsilon})$ time for any $\epsilon > 0$ then k -SAT can be solved by a circuit of depth 3 and size $2^{o(n)}$.

Sammanfattning

Problemet att finna den minsta storleken på en krets som beräknar en given boolesk funktion, ofta kallat minimum circuit size problem (MCSP), har studerats i många år men det är fortfarande okänt om problemet är NP-svårt eller inte. Med detta i åtanke studerar vi egenskaper hos potentiella reduktioner till det här problemet. Vi fokuserar på naturliga lokala reduktioner som är vanliga i många bevis av NP-svårighet. Vi presenterar en method som visar att det finns en algoritm för att lösa varje problem som har en lokal naturlig reduktion till MCSP. Vi visar att om beslutsproblemet att skilja satisfierbara instanser av 3-SAT från de där som mest $7/8 + o(1)$ av klausulerna går att satisfiera har en reduktion till MCSP där en godtycklig bit av utdata kan beräknas i $\mathcal{O}(n^{1-\epsilon})$ tid för varje $\epsilon > 0$ då kan k -SAT lösas av en krets med djup 3 och storlek $2^{o(n)}$.

Acknowledgements

Thanks to professor Hiroshi Imai, assistant professor Hidefumi Hiraishi, and Shuichi Hirahara at the The University of Tokyo for guidance in selecting the topic of this thesis as well as useful feedback throughout the research process.

Contents

1	Introduction	1
1.1	Circuit Complexity	1
1.1.1	Minimum Circuit Size Problem	2
1.2	Conditional Hardness	3
1.3	Fine-grained Reductions	4
1.4	Our Results	5
1.5	Overview	5
2	Preliminaries	7
2.1	Decision Problems and Languages	7
2.2	Reductions	7
2.3	Minimum Circuit Size Problem	8
2.4	Boolean Satisfiability Problem	8
2.4.1	Exponential Time Hypothesis	9
2.4.2	Complexity in Terms of the Number of Variables .	10
2.5	Orthogonal Vectors Problem	10
2.5.1	SETH-hardness	10
3	Reductions to MCSP	12
3.1	Circuit Complexity	14
3.2	Reductions from SAT to MCSP	15
3.2.1	Circuit Complexity	17
3.2.2	Non-natural Reductions	17
3.2.3	Gap-3-SAT	20
3.2.4	Combining the Results	23
3.3	Reductions from OV to MCSP	24
3.3.1	Circuit Complexity	25

4 Discussion	26
4.1 Dependence on Encoding	26
4.2 Conclusion	27
4.3 Open Problems	27
Appendices	31
A Local Reductions from SAT	32
A.1 Independent Set	32
A.1.1 Reducing 3-SAT to Independent Set	33
A.1.2 Locality	33
A.2 Vertex Cover	34
A.2.1 Reducing 3-SAT to Vertex Cover	34
A.2.2 Locality	35
A.3 Hitting Set	36
A.3.1 Reducing 3-SAT to Hitting Set	36
A.3.2 Locality	36

Chapter 1

Introduction

1.1 Circuit Complexity

The computational complexity of problems can be studied under many different models of computation. A boolean circuit is a model closely related to digital electronic circuits. A circuit is a directed acyclic graph where every node represents an input or some logic gate, along with a single node representing the output of the circuit. A gate has one or more inputs and a single output. The maximum number of inputs a gate may have is referred to as its fan-in, which may be bounded or unbounded. Circuit models can behave in many different ways depending of the gates that are used and their fan-in. Common gates are AND, OR and NOT, but sometimes other logic gates such as NAND and XOR are used.

Any decision problem in computer science can be modelled as a function from bit strings to a single bit (either TRUE or FALSE). If we fix the length of the input bit string to n bits a decision problem is a function $f : \{0, 1\}^n \rightarrow \{0, 1\}$. Such a function can be computed by some boolean circuit with n inputs.

The circuit complexity of a problem considers the size and depth of circuits that can compute the function. The size of a circuit is a measurement relating to the number of gates or the number of edges in the circuit, but the precise definition may vary between different models. The depth of a circuit refers to the length of the longest path from an input to the output. It is sometimes meaningful to study the size of circuits with the condition that the depth is at most some constant value. In these cases it makes sense to give gates unlimited fan-in since

constant fan-in makes it impossible to have a large number of inputs influence an output without having the depth depend on the number of inputs. In cases where the depth is not of concern it is common to use limited fan-in.

Circuit complexity can easily be related to generic time complexity since the problem of computing the output of a circuit of polynomial size is in P. The computation is easily done by evaluating the outputs of all gates in topological order. There is also a complexity class known as P/poly which can be defined as the set of all problems that can be solved by polynomial size circuits.

1.1.1 Minimum Circuit Size Problem

Given a truth table T and an integer s , the minimum circuit size problem, often denoted MCSP, is to determine if there exists a circuit of size at most s which computes the boolean function specified by T . The size of the input is $\mathcal{O}(2^m)$ where m is the number of variables in the input of the boolean function. It is easy to see that MCSP is in NP since we can guess a circuit and then evaluate it for every input in polynomial time and check if the outputs match T .

The specifics of this problem depend on the circuit model. One common model which is also the one we will use in this thesis includes AND and OR gates with fan-in two along with NOT gates with fan-in one. The size of a circuit is defined as the number of AND and OR gates.

MCSP is a meta problem in the sense that the problem itself is about computational complexity. Because of this fact the problem has some interesting properties. Efficient solutions to MCSP and the existence of some reductions to MCSP would have implications concerning the hardness of other well-known problems.

It is not known if MCSP is NP-hard, nor is it known if it is in P. Kabanets and Cai [1] argued that MCSP is unlikely to be in P by showing that if MCSP is in P there is no strong pseudorandom generator in P/poly. Showing the existence or non-existence of strong pseudorandom generators in P/poly is an open problem. Furthermore, this would imply the existence of an algorithm that factors Blum integers¹ well on average better than currently known factoring algorithms. Ka-

¹Blum integers are integers on the form $n = pq$ where p and q are primes on the form $4t + 3$. Blum integers have been significant in the field of cryptography.

banets and Cai [1] also showed that if MCSP is NP-hard under a natural reduction² from SAT then $E \not\subseteq P/\text{poly}$. This is because a natural reduction showing NP-hardness of MCSP would imply the existence of hard instances of MCSP. More specifically there must exist a boolean function in E with superpolynomial circuit size. While this is not reason to believe that MCSP is not NP-hard it indicates that if it is NP-hard it is likely to be difficult to prove with current techniques.

Even though MCSP is often believed to be intractable there are certain types of reductions for which MCSP does not appear hard. Murray and Williams [2] showed that even PARITY has no local reductions to MCSP in $\text{TIME}(n^\delta)$ for $\delta < 1/2$. Local reductions consider the time to compute any arbitrary bit of the output of the reductions instead of the time to compute the entire output. One would expect that it is easy to reduce an easy problem to a hard problem. PARITY is in P and very straight forward to solve in linear time which makes this result surprising. For comparison, several well known NP-hard problems are NP-hard under $\mathcal{O}(\log n)$ -local reductions from SAT, see Appendix A for examples.

There are some variations of MCSP for which NP-hardness has been proved. Masek [3] showed that computing the minimum number of terms in a DNF formula that corresponds to a given truth table is NP-hard. Hirahara et al. [4] showed that MCSP restricted to circuits on the form OR-AND-MOD₂ is NP-hard.

1.2 Conditional Hardness

In computer science it is often difficult to give unconditional specific statements about the hardness of problems. Therefore many such statements are true only if some condition is fulfilled. A common condition is $P \neq NP$ which was introduced by Cook [5] together with a proof that the boolean satisfiability problem (SAT) is NP-Complete. This can be used to show that any statement which implies a polynomial algorithm for SAT (or some other NP-Complete problem) must be false unless $P = NP$. By using stronger conditions than that it is sometimes possible to make even stronger statements about computational complexity. One such statement about SAT is the *Exponential Time Hypothesis* (ETH) formulated by Impagliazzo and Paturi [6]. The hypothesis

²Natural reductions are defined in Chapter 2.

states that solving 3-SAT requires exponential time. This can be extended to an even stronger hypothesis known as the *Strong Exponential Time Hypothesis* (SETH) which states that the required time for solving k -SAT is $\mathcal{O}(2^{s_k n})$ where $\lim_{k \rightarrow \infty} s_k = 1$.

Even though showing NP-hardness of MCSP seems out of reach at this point it may be possible to show some hardness results for MCSP under ETH or SETH.

1.3 Fine-grained Reductions

In the field of complexity theory problems in P are often referred to as easy problems. Still the time complexity required to solve problems in P can vary greatly and a problem in P might very well be intractable in practical applications.

There are problems in P which have been known for many years and still have seen little to no improvements in efficient algorithms for solving them. In recent years several papers have focused on classifying the hardness of problems within P and showing for which problems major improvements are unlikely to be found. These bounds are often shown by reducing some well known problem such as CNF-SAT, 3SUM, or All-Pairs Shortest Paths. These reductions are fine-grained in the sense that they are not just polynomial reductions but also distinguish between different polynomial running times[7].

Using SETH and fine-grained reductions from SAT a set of problems referred to as SETH-hard problems has emerged. These are problems where a significant improvement in the complexity of their respective algorithms would imply that SETH is false. In 2004 Williams [8] showed that the *Orthogonal Vectors Problem* cannot be solved in strictly subquadratic time unless SETH is false. Backurs and Indyk [9] showed the same result for *Edit Distance*. Bringmann [10] showed that the same is also true for *Fréchet Distance*.

MCSP is a little bit different since there are no known polynomial time algorithms for it. Given the apparent difficulty of showing NP-hardness, it could be possible to give some polynomial lower bound for the complexity of MCSP using fine-grained reductions.

1.4 Our Results

We introduce a method based on the works of Murray and Williams [2] which makes a statement on the complexity of any language L which has local natural reductions to MCSP. The main idea is the interesting property that if a reduction R to MCSP can produce the i th bit of the truth table in time t for any arbitrary i then R computes the function defined by said truth table in time t . This knowledge can provide us with an upper bound on the size of a circuit that computes the function defined by the truth table given by R . With this upper bound it may be relatively efficient to try all possible circuits and thus we get an algorithm solving any language that can be reduced to MCSP. Finally we combine this with assumptions such as ETH and SETH to get particular results about reductions from SAT. To make the result even stronger for the case of SAT we show that the existence of any local reduction implies a natural local reduction with only a factor $\text{polylog}(n)$ worse time complexity. We arrive at the result that if ETH is true it is impossible to show NP-hardness of MCSP with a reduction from k -SAT for which any arbitrary bit of the output can be computed in $o(\frac{n}{\text{polylog}(n)})$ time. We extend this to show that if there exists a reduction k -SAT to MCSP where any arbitrary bit of the output can be computed in $o(\frac{n}{\text{polylog}(n)})$ time then k -SAT can be solved by a circuit of depth 3 and size $2^{o(n)}$.

Building on results from Håstad [11] and Moshkovitz and Raz [12] we look at the decision problem of distinguishing satisfiable 3-SAT instances from those where at most $7/8 + o(1)$ of the clauses can be satisfied. This problem has been shown to be NP-hard under non-local reductions. Because of this non-local separation from k -SAT one might hope that this problem can have local reductions to MCSP. However, we prove that even in this case reductions where any arbitrary bit of the output can be computed in $\mathcal{O}(n^{1-\epsilon})$ time for any $\epsilon > 0$ imply that k -SAT can be solved by a circuit of depth 3 and size $2^{o(n)}$.

1.5 Overview

In this thesis we will show several non-reducibility results to MCSP. In Chapter 2 we introduce definitions and problems that will be used. In Chapter 3 a generalized theorem about reductions to MCSP is pre-

sented and then applied to a few different problems. The main focus is SAT under the assumption that the Exponential Time Hypothesis is true. We also apply the theorem to Orthogonal Vectors. A polynomial time reduction from Orthogonal Vectors to MCSP would imply a polynomial lower bound for MCSP under the assumption that the Strong Exponential Time Hypothesis is true. In Chapter 4 we discuss implications and limitations of the results presented in Chapter 3 along with a few ideas for further research.

Chapter 2

Preliminaries

In this section we shall introduce definitions, models and notations used throughout the rest of the thesis.

2.1 Decision Problems and Languages

All the problems we study in this thesis are decision problems. These are problems where every instance is answered by either TRUE or FALSE. This can be formalized as languages over the alphabet $\{0, 1\}$. The decision problem defined by a language L is to determine if some input bit string x is a member of L .

In this thesis we will use slightly different languages. We separate some parameters from the bit string that we will refer to as *size parameters*. These are parameters that describe the size of some structures in the input or the desired output. This could for example be the length of the input, the number of vertices in a graph or the number of dimensions in a vector room. The words in a language are tuples of size parameters and a bit string.

2.2 Reductions

Definition 2.1. A reduction R from a language A to a language B is a function such that $(S, a) \in A \iff R(S, a) \in B$, where S denotes the size parameters and a is a bit string.

Definition 2.2. Let $(S', b) = R(S, a)$. R is natural if for every (S, a) , S' and $|b|$ only depend on the S and $|a|$. Furthermore, R is $\mathcal{O}(t(S))$ -natural if

$|b| \in \mathcal{O}(t(S))$.

Definition 2.3. Let $(S', b) = R(S, a)$. R is $\mathcal{O}(t(S))$ -local if for every (S, a) any arbitrary bit of b can be computed in $\mathcal{O}(t(S))$ time.

Remark. We may say that a reduction is $o(t(s))$ -natural or $o(t(s))$ -local which simply means \mathcal{O} is replaced with o .

In this thesis we will let reductions have oracle access to size parameters. Since all the results are about non-reducibility this assumption will only make the results stronger. A reduction that has to read size parameters encoded in the input would be at least as efficient if it did not have to do that.

2.3 Minimum Circuit Size Problem

Minimum Circuit Size Problem (MCSP)

Input: Truth table T of a function with m inputs, integer s .

Output: TRUE if there exists a circuit of size at most s computing the boolean function specified by T . FALSE otherwise.

Size parameters: m, s

Input encoding: $M = 2^m$ bits representing the truth table.

The circuit model allows AND and OR gates with fan-in two as well as NOT gates with fan-in one. The size of a circuit is defined as the number of AND and OR gates.

2.4 Boolean Satisfiability Problem

The boolean satisfiability problem, commonly known as SAT, is to decide whether or not there exists an interpretation which satisfies a given boolean formula. A common variant of this problem is CNF-SAT where the formula is always given in conjunctive normal form (informally an AND of ORs). CNF-sat where every clause contains k literals is referred to as k -SAT.

k -SAT	
Input:	Boolean formula on n variables in conjunctive normal form with d distinct clauses and k literals in every clause.
Output:	TRUE if there is an interpretation satisfying the formula. FALSE otherwise.
Size parameters:	n, d
Input encoding:	A list of kd literals representing the clauses. Each literal can be encoded by $\mathcal{O}(\log n)$ bits.

By distinct clauses we mean that each valid clause is a unique set of k of the $2n$ literals. A clause may not contain the same variable more than once. Sometimes the definition of k -SAT does not require that every clause has exactly k literals but rather at most k literals. Any instance of the problem with at most k variables can be extended to an equivalent instance with exactly k by introducing dummy variables. In order to simplify the encoding of the input we will without loss of generality assume that every clause has exactly k literals.

2.4.1 Exponential Time Hypothesis

Let

$$s_k = \inf\{\delta \mid k\text{-SAT can be solved in } \mathcal{O}(2^{\delta n}) \text{ time}\}.$$

It is known that $s_2 = 0$ since 2-SAT can be solved in polynomial time. It is also clear that the sequence is monotonic ($s_2 \leq s_3 \leq \dots$).

Hypothesis 2.1. Exponential Time Hypothesis (ETH)

$$s_3 > 0$$

Hypothesis 2.2. Strong Exponential Time Hypothesis (SETH)

$$s_\infty = \lim_{k \rightarrow \infty} s_k = 1$$

Note that $\text{SETH} \implies \text{ETH} \implies \text{P} \neq \text{NP}$ and both ETH and SETH are stronger statements on the complexity of SAT than $\text{P} \neq \text{NP}$.

2.4.2 Complexity in Terms of the Number of Variables

It is the norm to consider the complexity of k -SAT in terms of n and not in terms of the number of clauses. According to the Sparsification Lemma [13] for every $\epsilon > 0$, every k -CNF formula with n variables can be written as a disjunction of $2^{\epsilon n}$ k -CNF formulas, all of which contain each variable at most a constant number of times. That is, in the disjunction of k -CNF formulas the number of clauses in each formula is at most linear in the number of variables.

2.5 Orthogonal Vectors Problem

Define the inner product between vectors $u, v \in \{0, 1\}^d$ as

$$u \cdot v = \sum_{i=1}^d u_i \cdot v_i$$

where u_i, v_i denotes the i th coordinate of u, v respectively.

Orthogonal Vectors (OV)

Input: $U, V \subset \{0, 1\}^d$ with $|U| = |V| = n$

Output: TRUE if $\exists u \in U, \exists v \in V$ such that $u \cdot v = 0$.
FALSE otherwise.

Size parameters: n, d

Input encoding: $2nd$ bits representing the vectors.

2.5.1 SETH-hardness

Williams [8] showed that OV cannot be solved in strictly subquadratic time unless SETH is false. The proof is quite straight forward and therefore we will include it here as well.

Theorem 2.1. *SETH $\implies \nexists \epsilon > 0$ such that OV can be solved in $\mathcal{O}(n^{2-\epsilon} f(d))$*

Proof. Let us make a fine-grained reduction from k -SAT to OV. Assume we have an instance of k -SAT for some k with n variables x_1, x_2, \dots, x_n and d clauses. WLOG assume that n is even (if n is odd add a dummy variable which is not in any clause). Divide the variables into two sets of equal size.

$$S_1 = x_1, \dots, x_{n/2}, S_2 = x_{n/2+1}, \dots, x_n$$

For each assignment of the variables in S_1 make a vector $u \in \{0, 1\}^d$.

$$u_i = \begin{cases} 0, & \text{if the partial assignment satisfies the } i\text{:th clause} \\ 1, & \text{otherwise} \end{cases}$$

Let U be the set of all such vectors u (one for each of the $2^{n/2}$ assignments). Do the same for S_2 to create another set of vectors V .

For $u \in U, v \in V$

$u \cdot v = 0 \iff$ the combination of the partial assignments corresponding to u and v satisfy the formula.

Assume for contradiction $\exists \epsilon > 0$ such that OV can be solved in $\mathcal{O}(n^{2-\epsilon} f(d))$. Then we can solve k-SAT in $\mathcal{O}((2^{n/2})^{2-\epsilon} f(d)) = \mathcal{O}(2^{(1-\epsilon/2)n} f(d))$ which contradicts SETH.

□

Chapter 3

Reductions to MCSP

In this section we introduce a theorem based on the theorem about reductions from PARITY to MCSP by Murray and Williams [2]. The theorem we present considers a reduction from any language L to MCSP instead of only PARITY in particular.

In order to simplify the main theorem we first define *rejectability* of a language. The rejectability of a language L is a measurement of the hardness of locally generating inputs that are not in L .

Definition 3.1. *A language L is $\mathcal{O}(q(S))$ -rejectable if for every set of sufficiently large size parameters S there exists x such that $(S, x) \notin L$ and there exists an algorithm A that can produce some such x . Furthermore, given an index i the algorithm can output the i th bit of x in $\mathcal{O}(q(S))$ time.*

Remark. *By sufficiently large we mean that for each size parameter there is some constant lower bound such that if all parameters are larger than their respective bounds A is successful. The definition will be used in the context of time complexity of reductions. For this reason we will assume that the size parameters are always sufficiently large. We can handle the small constant size cases separately in constant time.*

With this definition we provide the main theorem.

Theorem 3.1. *Let L be a $\mathcal{O}(q(S))$ -rejectable language. Assume there exists a $\mathcal{O}(t(S))$ -local natural reduction R from L to MCSP which gives an instance of MCSP with a truth table of size $M(S) = 2^m$, where m is the number of input variables to the circuit. Then there exists an algorithm for L in time*

$$2^{\mathcal{O}(t(S)q(S) \log(t(S)q(S) + \log M(S)))} \cdot \mathcal{O}(M(S) \cdot t(S)q(S)).$$

The idea for proving this theorem is to create a deterministic algorithm for any MCSP instance produced by R . The algorithm works by trying all possible circuits and checking if they correspond to the truth table. In order to argue about the complexity of the algorithm we will introduce two lemmas. The first lemma gives an upper bound for the size parameter s of MCSP. Using this upper bound we then give an upper bound on the number of circuits of size s that could potentially compute the function defined by the truth table.

Lemma 3.1. *The size parameter s of MCSP given by R is $\mathcal{O}(t(S)q(S))$.*

Proof. Let x be the bit string given by A in Definition 3.1. Consider the function $f(z) = R((S, x), z)$, that is f outputs the z th bit of the truth table given by R with input (S, x) . It takes $\mathcal{O}(t(S)q(S))$ time to compute f since R takes $t(S)$ time and it takes $\mathcal{O}(q(S))$ time to access any arbitrary bit of x . This implies that f can be represented by a circuit of size $\mathcal{O}(t(S)q(S))$. Furthermore, f is a function corresponding to the truth table given by $R(S, x)$. Since $(S, x) \notin L$ we must also have $R(S, x) \notin \text{MCPS}$. Therefore the size parameter s must be smaller than the size of any circuit computing f . In particular $s \in \mathcal{O}(t(S)q(S))$. Since R is a natural reduction the size parameter is the same for all bit strings of length n . \square

Before we get to the deterministic algorithm, let us introduce a non-deterministic algorithm for L given by R . For a given input (S, x) guess a circuit C with m inputs and size s , where m and s are given by R . For all bit strings z of m bits check if $C(z) = R((S, x), z)$. Return YES if this is true for all z , otherwise return NO.

Note that if there is a circuit with size strictly less than s computing some function we can turn it into a circuit of size s computing the same function by adding gates that do not change the output. Therefore it is enough to guess circuits of size precisely s .

Formally $(S, x) \in L$ if and only if

$$\begin{aligned} \exists C \text{ of size } s \text{ with } m \text{ inputs} \\ \forall z \in 0, 1^m \\ C(z) = R((S, x), z). \end{aligned} \tag{3.1}$$

(3.1) can be turned into a deterministic algorithm by trying all possible circuits C and all possible bit strings z . To prove Theorem 3.1 we have to determine the time complexity of this algorithm. The tricky part is the outer loop that must iterate over all possible circuits.

Lemma 3.2. *The number of possible circuits in (3.1) is*

$$2^{\mathcal{O}(s \log(s+m))}$$

Proof. Recall that the number of NOT gates does not influence the size of the circuit. Let us attempt to count the number of possible gates that do affect the size of the circuit. Each gate can be either AND or OR. Each of the two inputs can either be an input variable, the negation of an input variable, the output of another gate, or the negation of the output of another gate. Therefore a single gate can be constructed in at most $2(2s + 2m)^2$ different ways. There are s gates and therefore the number of different circuits which can be constructed in this manner is

$$(2(2s + 2m)^2)^s = 2^{s(2+2\log(s+m)+2\log 2)} \in 2^{\mathcal{O}(s \log(s+m))}$$

□

The deterministic algorithm can be used to prove the main theorem.

Proof of Theorem 3.1. All we have to do is calculate the time complexity of the algorithm. We already know the number of circuits we have to try from Lemma 3.2. For each circuit we try $M = 2^m$ different inputs and evaluate it. Recall that the size of each circuit is $s \in \mathcal{O}(t(S)q(S))$ and therefore evaluating each circuit takes $\mathcal{O}(t(S)q(S))$ time. Combine these results and substitute m with $\log M(S)$ to find that the algorithm takes time

$$2^{\mathcal{O}(t(S)q(S) \log(t(S)q(S) + \log M(S)))} \cdot \mathcal{O}(M(S) \cdot t(S)q(S)).$$

□

3.1 Circuit Complexity

A slightly stronger statement than Theorem 3.1 can be made by showing that the deterministic algorithm can be computed by circuits of depth 3. Unlike the circuits in MCSP we will allow unbounded fan-in. This is because we restrict the depth to a constant which means that any circuit with constant fan-in gates would be unable to consider the entire input if the input is big. With unlimited fan-in we also redefine the size as the number of wires. The reason for this is that the

time complexity of evaluating the output of a unbounded fan-in circuit is not necessarily linear in the number of gates but it is linear in the number of wires.

Theorem 3.2. *Let L be a $\mathcal{O}(q(S))$ -rejectable language. Assume there exists a $\mathcal{O}(t(S))$ -local natural reduction R from L to MCSP which gives an instance of MCSP with a truth table of size $M(S) = 2^m$, where m is the number of input variables to the circuit. Then there exists a circuit solving L with depth 3 and size*

$$2^{\mathcal{O}(t(S)q(S) \log(t(S)q(S) + \log M(S)))} \cdot \mathcal{O}(M(S) \cdot 2^{t(S)q(S)}).$$

Proof. $C(z) = R((S, x), z)$ can be formulated as propositional formula of size $t(S)q(S)$. Such a formula can be converted to an equivalent formula on conjunctive normal form by using the double negative law, De Morgans's laws, and the distributive law. The conversion process will in the worst case result in a formula of exponential size related to the original formula. We can guarantee that the new formula is at worst exponential in size since any boolean function on n variables can be represented by a CNF of exponential size in the number of variables. This tells us that there is a formula on conjunctive normal form of size $\mathcal{O}(2^{t(S)q(S)})$ that is equivalent to $C(z) = R((S, x), z)$. As a circuit this is an AND of ORs with $\mathcal{O}(2^{t(S)q(S)})$ wires.

The entire algorithm is an OR over all possible circuits, for each circuit an AND over all z , and for every z an AND of ORs to check if $C(z) = R((S, x), z)$. This is a depth 4 circuit but the two middle layers are both AND gates. An AND of ANDs is the same as one big AND so we can easily merge these layers. The result is a depth 3 circuit. Converting the last step to conjunctive normal form changed the factor $t(S)q(S)$ to $\mathcal{O}(2^{t(S)q(S)})$ and therefore the size of the circuit is

$$2^{\mathcal{O}(t(S)q(S) \log(t(S)q(S) + \log M(S)))} \cdot \mathcal{O}(M(S) \cdot 2^{t(S)q(S)}).$$

□

3.2 Reductions from SAT to MCSP

In this section we will apply Theorem 3.1 on k -SAT to make an argument about which reductions from k -SAT to MCSP are impossible if ETH is true.

Theorem 3.3. *If ETH is true then there are no $o(\frac{n}{\log^2 n})$ -local $2^{o(n)}$ -natural reductions from k -SAT to MCSP for $k \geq 3$.*

We start by looking at the rejectability of k -SAT.

Lemma 3.3. *k -SAT is $\mathcal{O}(\log n)$ -rejectable.*

Proof. Let us show that there exists an algorithm A that can generate x such that $(S, x) \notin k$ -SAT when the number of clauses is at least 2^k . We let the first 2^k clauses be all possible clauses made from the first k variables. This clearly makes the formula unsatisfiable. For the rest of the clauses we just keep going through all the other sets of k variables in order. For each set we go through all the 2^k clauses. Repeat until we have d clauses as given by S . We can get any bit of any clause in $\mathcal{O}(\log n)$ time (arithmetic and comparisons of $\mathcal{O}(\log n)$ -bit integers). \square

Let us assume that there exists a $\mathcal{O}(t(n))$ local natural reduction R from k -SAT to MCSP. According to Theorem 3.1 R can be used to create an algorithm for k -SAT. We use the existence of such an algorithm to prove the main theorem.

Proof of Theorem 3.3. Use Theorem 3.1 with $q(n) \in \mathcal{O}(\log n)$ as was shown in Lemma 3.3. The result is that there exists an algorithm solving k -SAT in time

$$2^{\mathcal{O}(t(n) \log(n) \log(t(n) \log(n) + \log M(n)))} \cdot \mathcal{O}(M(n) \cdot t(n) \log n).$$

Assume that $t(n) \in o(\frac{n}{\log^2 n})$ and $M(n) \in 2^{o(n)}$. We simplify the expression for the running time in steps starting with the exponent of the first factor.

$$\log(t(n) \log(n) + \log M(n)) \in \log(o(n)) \subset \mathcal{O}(\log n)$$

This can be used to give a bound for the entire exponent.

$$t(n) \log(n) \log(t(n) \log(n) + \log M(n)) \in \mathcal{O}(t(n) \log^2 n) \subset o(n)$$

The first factor is bounded by

$$2^{\mathcal{O}(t(n) \log(n) \log(t(n) \log(n) + \log M(n)))} \in 2^{o(n)}.$$

Let us look at the second factor.

$$\mathcal{O}(M(n) \cdot t(n) \log n) \in 2^{o(n)} \cdot \mathcal{O}(\log n)$$

By putting it all together we find that we can solve k -SAT in

$$2^{\mathcal{O}(t(n) \log(n) \log(t(n) \log(n) + \log M(n)))} \cdot \mathcal{O}(M(n) \cdot t(n) \log n) \in 2^{o(n)} \cdot 2^{o(n)} \cdot o\left(\frac{n}{\log n}\right).$$

If ETH is true this is false for $k \geq 3$.

□

3.2.1 Circuit Complexity

Using Theorem 3.2 it is possible to show the same results is true even for a weaker assumption than ETH.

Theorem 3.4. *Let*

$$r_k = \inf\{\delta \mid k\text{-SAT can be solved by a depth 3 circuit with size } \mathcal{O}(2^{\delta n})\}.$$

If $r_k > 0$ then there are no $o\left(\frac{n}{\log^2 n}\right)$ -local $2^{o(n)}$ -natural reductions from k -SAT to MCSP.

Proof. The proof follows the same steps as the proof of Theorem 3.3 but from Theorem 3.2 the last factor is $\mathcal{O}(2^{t(n)q(n)}) \subset 2^{o\left(\frac{n}{\log n}\right)}$ instead of $\mathcal{O}(t(n)q(n)) \subset o\left(\frac{n}{\log n}\right)$. This shows that $t(n) \in o\left(\frac{n}{\log^2 n}\right)$ and $M(n) \in 2^{o(n)}$ implies that there exists a circuit solving k -SAT with depth 3 and size

$$2^{o(n)} \cdot 2^{o(n)} \cdot 2^{o\left(\frac{n}{\log n}\right)} = 2^{o(n)}$$

which implies $r_k = 0$.

□

Remark. *Theorem 3.4 is stronger than Theorem 3.3 since for every k , $s_k > 0 \implies r_k > 0$, where s_k is the integer sequence from the definition of ETH.*

3.2.2 Non-natural Reductions

In this section we will study non-natural reductions from k -SAT to MCSP. We no longer assume that we have oracle access to size parameters but rather size parameters will be encoded in binary in the input strings. Our goal is to show that it is possible to take any reduction

and by modifying the k -SAT input we can turn it into a natural reduction. Furthermore we can do so in such a way that we do not have to generate the full modified input string and can therefore preserve locality of the reductions. This leads to a theorem that applies to local reductions that are not necessarily natural.

Theorem 3.5. *If ETH is true it is impossible to show NP-hardness of MCSP under $o(\frac{n}{\log^4 n})$ -local reductions from k -SAT.*

One important detail to keep in mind here is that such a reduction from k -SAT has to take polynomial time (for the entire output, not just a single bit). This is because if the reduction is superpolynomial a polynomial time algorithm for MCSP would not imply a polynomial time algorithm for k -SAT, which means we have not shown NP-hardness. In particular, the size of the truth table, M is polynomial in the size of the input to k -SAT. That is, $M \in \mathcal{O}(\text{poly}(n))$.

We divide the proof of the theorem into three lemmas. Assume R is a polynomial time reduction from k -SAT to MCSP that is $\mathcal{O}(t(s))$ -local, where $t(s) \in \mathcal{O}(\text{poly}(n))$

Lemma 3.4. *Any truth table given by R specifies a function that can be computed by a circuit of size $\mathcal{O}(\text{poly}(n))$.*

Proof. Given any truth table we can make a trivial circuit computing the specified function. For every TRUE row in the table make an AND of all the literals in that row. Finally make an OR of all such ANDs. Each row has m literals. Because we have fan-in two, an AND of m literals requires $m - 1$ AND gates. There are at most M TRUE rows. The final OR will require at most $M - 1$ OR gates. This is a total of $(m - 1) \cdot M \cdot (M - 1) \in \mathcal{O}(M^2 \log M)$ gates. Because $M \in \mathcal{O}(\text{poly}(n))$ we also know that the number of gates is $\mathcal{O}(\text{poly}(n))$. \square

The idea is that we can assume $s \in \mathcal{O}(\text{poly}(n))$. If R was computing an s larger than that we already know that the answer is TRUE and we can ignore the rest of the reduction and simply output any trivial TRUE instance of MCSP.

Corollary 3.1. *M and s can be encoded in $\mathcal{O}(\log n)$ bits.*

Lemma 3.5. *When R computes M and s it reads at most $\mathcal{O}(t(n) \log n)$ bits from the k -SAT input.*

Proof. R is $t(n)$ -local which means that every bit of M and s takes $\mathcal{O}(t(n))$ time to compute. Therefore there is only time to read at most $\mathcal{O}(t(n))$ bits for every bit of s and M . Because M and s are encoded in $\mathcal{O}(\log n)$ bits each that is a total of at most $\mathcal{O}(t(n) \log n)$ bits read when computing them. \square

What we want to do next is modify the input such that for any given n and d , R always reads the same values when computing M and s no matter what the original input was. This has to be done in such a way that the satisfiability is not changed when the input is modified. We do this by adding polynomially many dummy variables and new clauses. We can make the clauses in a systematic way, for example by making each clause out of k unique dummy variables with no negations. Then the clauses are clearly satisfiable and they are easy to generate.

Lemma 3.6. *For every n, d and $\mathcal{O}(t(n))$ -local polynomial time reduction R from k -SAT to MCSP there is a natural $\mathcal{O}(t(n) \log^2 n)$ -local natural reduction R' from k -SAT to MCSP.*

Proof. All we have to do is construct R' such that M and s are only functions of n and d . Let x be any instance of k -SAT and let d denote the number of clauses. Let $n' \in \mathcal{O}(t(n) \log n)$. Create a new instance of k -SAT where we add kn' dummy variables. Let $c_1, c_2, \dots, c_{n'}$ denote the clauses made up of each group of k consecutive dummy variables with no negations. Because there are no negations there is obviously a satisfying interpretation for these clauses.

Next, let us define the input for the new instance of k -SAT. Simulate R on an input with $n + kn'$ variables and $d + n'$ clauses. Specifically simulate the process of computing M and s . Let i_1, i_2, \dots denote the indices of clauses that R reads in this process (there are $\mathcal{O}(t(n) \log n)$ such indices according to Lemma 3.5). When R reads from the i_j th clause proceed as if that clause was c_j . In this simulation record all indices to make the list of tuples $I = [(i_1, c_1), (i_2, c_2) \dots]$ and sort it in ascending order of i_j . Construct the instance x' of k -SAT by starting with x . Go through I in order of increasing i_j and insert clause c_j at index i_j (pushing all later clauses to a higher index). Finally insert any remaining dummy clauses at the end of the input until all $c_{n'}$ clauses have been added to the input.

Let R' behave identical to R but instead of reading from x we simulate reading from x' . When reading from the i th clause we binary

search to find if i is one of the indices listed in I .

- If i is listed in I we read from the corresponding dummy clause listed in I .
- If i is not listed in I , let l be the number of indices listed in I smaller than i . If $i - l \leq d$ read the $(i - l)$ th clause of x . Otherwise read $c_{n' - (n - i)}$.

This procedure ensures that when R' computes M and s it is only a function of n and d .

Let us find the time it takes for R' to compute any arbitrary bit of its output. Recall that $t(n) \in \mathcal{O}(\text{poly}(n))$. Generating I takes $\mathcal{O}(t(n)\log^2 n)$ time because of sorting. Binary searching a list of $\mathcal{O}(t(n)\log n)$ items where we compare $\mathcal{O}(\log n)$ bit integers takes a total of $\mathcal{O}(\log^2 n)$ time. Generating c_i for some given i takes $\mathcal{O}(\log n)$ time. Therefore outputting any arbitrary bit with R' takes an additional $\mathcal{O}(t(n)\log^2 n)$ for computing I and an additional $\mathcal{O}(\log^2 n)$ time per bit read more than R . Therefore R' is $\mathcal{O}(t(n)\log^2 n)$ -local. \square

Finally we use the reduction R' to prove the main theorem.

Proof of Theorem 3.5. According to Theorem 3.3 there are no $o(\frac{n}{\log^2 n})$ -local natural reductions from k -SAT to MCSP unless ETH is false. R' in Lemma 3.6 contradicts this if $t(s) \in o(\frac{n}{\log^4 n})$. Therefore R cannot be $o(\frac{n}{\log^4 n})$ -local. \square

Using Theorem 3.4 an even stronger statement can be made.

Corollary 3.2. *If there exists a $o(\frac{n}{\log^4 n})$ -local reduction from k -SAT to MCSP then k -SAT can be solved by a circuit of depth 3 and size $2^{o(n)}$.*

3.2.3 Gap-3-SAT

3-SAT can be turned into an optimization problem by asking what is the maximum number of clauses that can be satisfied in a formula. It has been shown that it is NP-hard to approximate the solution to this problem within a factor of $7/8 + o(1)$ [11][12]. Using this knowledge we can make an NP-hard decision problem we will call Gap-3-SAT.

Gap-3-SAT

Input:	Boolean formula on n variables on conjunctive normal form with d clauses and 3 literals in every clause. <i>It is guaranteed that either all clauses can be satisfied or at most $7/8 + o(1)$ of the clauses can be satisfied.</i>
Output:	TRUE if there is an interpretation satisfying the formula. FALSE otherwise.
Size parameters:	n, d
Input encoding:	A list of $3d$ literals representing the clauses. Each literal is encoded by $\mathcal{O}(\log n)$ bits.

Note that 3-SAT is at least as hard as Gap-3-SAT since any algorithm solving 3-SAT would also solve Gap-3-SAT, but the opposite is not necessarily true. We will show that the results we showed for k -SAT also apply to Gap-3-SAT.

To understand why this is interesting consider a reduction in two steps starting at 3-SAT going to Gap-3-SAT and then to MCSP. We have already shown that the entire reduction cannot be particularly local. Showing hardness of Gap-3-SAT goes through the PCP theorem which is also not local. For our two step reduction one might hope that the fact that the first step is not local would allow the second step to be local. However, it turns out that reductions from Gap-3-SAT to MCSP cannot be much more local than those from k -SAT.

Theorem 3.6. *If ETH is true there are no $\mathcal{O}(n^{1-\epsilon})$ -local $2^{o(n)}$ -natural reductions from Gap-3-SAT to MCSP for $\epsilon > 0$.*

Just like before we begin the proof by looking at the rejectability.

Lemma 3.7. *Gap-3-SAT is $\mathcal{O}(\log n)$ -rejectable.*

Proof. We will show that if $d \geq 8$ we can generate instances of Gap-3-SAT where at most $\frac{7}{8}d + 7$ out of the d clauses can be satisfied. Write $d = 8u + v$ where $0 \leq v < 8$. For the $u + 1$ first sets of 3 elements from $[1, n]$, make all 8 possible clauses

$$c_{1,1}, c_{1,2} \dots c_{1,8}, c_{2,8}, \dots c_{u,8}, c_{u+1,1} \dots c_{u+1,8}$$

Let the instance consist of the d first of these clauses. Let us show that every interpretation satisfies at most $\frac{7}{8}d + 7$ of them. Assume we have picked some interpretation of the variables and look at the first $8u$ clauses. Every group of 8 clauses contains every possible clause for some choice of 3 variables. That means that no matter which interpretation we picked at least one clause in every such group is not satisfied. Therefore we must have at least u unsatisfied clauses. It follows that the number of satisfied clauses are at most

$$d - u = 7u + v < \frac{7}{8}d + v \leq \frac{7}{8}d + 7.$$

Since the clauses are in order we can generate $c_{i,j}$ in $\mathcal{O}(\log n)$ time for any i, j . □

Because this problem has the same rejectability as k -SAT we get similar results using Theorem 3.1.

Corollary 3.3. *Assume there exists a $\mathcal{O}(t(n))$ -local natural reduction R from Gap-3-SAT to MCSP. Then there exists an algorithm for Gap-3-SAT that runs in time*

$$2^{\mathcal{O}(t(n) \log(n) \log(t(n) \log(n) + \log M(n)))} \cdot \mathcal{O}(M(n) \cdot t(n) \log n).$$

Corollary 3.4. *Assume there exists a $\mathcal{O}(t(S))$ -local natural reduction R from Gap-3-SAT to MCSP. Then there exists a circuit solving Gap-3-SAT with depth 3 and size*

$$2^{\mathcal{O}(t(n) \log(n) \log(t(n) \log(n) + \log M(n)))} \cdot \mathcal{O}(M(n) \cdot 2^{t(n)q(n)}).$$

Moshkovitz and Raz [12] showed that approximating 3-SAT within a factor of $7/8 + o(1)$ is NP-hard under almost linear reductions from 3-SAT. Specifically an instance of 3-SAT with n variables can be reduced to an instance of Gap-3-SAT with $n^{1+o(1)}$ variables. Therefore any algorithm for Gap-3-SAT in $\mathcal{O}(T(n))$ time implies an algorithm for 3-SAT in $\mathcal{O}(T(n^{1+o(1)}))$ time. With this knowledge we have all we need to prove Theorem 3.6.

Proof of Theorem 3.6. Assume there exists a $\mathcal{O}(n^{1-\epsilon})$ -local $2^{o(n)}$ -natural reduction from Gap-3-SAT to MCSP for some $\epsilon > 0$. Then according to Corollary 3.3 there exists an algorithm for Gap-3-SAT in time

$$2^{\mathcal{O}(n^{1-\epsilon} \log(n) \log(n^{1-\epsilon} \log(n) + \log 2^{o(n)}))} \cdot \mathcal{O}(2^{o(n)} \cdot n^{1-\epsilon} \log n).$$

and then there exists an algorithm for 3-SAT in time

$$2^{\mathcal{O}(n^{(1-\epsilon)(1+o(1))} \log(n) \log(n^{(1-\epsilon)(1+o(1))} \log(n) + \log 2^{o(n)}))} \cdot \mathcal{O}(2^{o(n)} \cdot n^{(1-\epsilon)(1+o(1))} \log n).$$

which contradicts ETH. □

Similar to k -SAT we get a slightly stronger statement with an assumption about depth 3 circuits instead of ETH.

Corollary 3.5. *If there exists a $\mathcal{O}(n^{1-\epsilon})$ -local $2^{o(n)}$ -natural reduction from Gap-3-SAT to MCSP for any $\epsilon > 0$ then k -SAT can be solved by circuits of depth 3 and size $2^{o(n)}$.*

3.2.4 Combining the Results

In this section we combine the results from Section 3.2.2 and Section 3.2.3. That is, we show that the results about Gap-3-SAT apply even for non-natural reductions.

Theorem 3.7. *If there exists a $\mathcal{O}(n^{1-\epsilon})$ -local reduction from Gap-3-SAT to MCSP for any $\epsilon > 0$ then k -SAT can be solved by circuits of depth 3 and size $2^{o(n)}$.*

Proof. In Section 3.2.2 we showed that we can simulate our reduction on an input of k -SAT where we have added $\mathcal{O}(t(s) \log n)$ dummy clauses in order to make a natural reduction from any non-natural reduction. We can do the same thing for Gap-3-SAT but we have to show that an input where we add $\mathcal{O}(t(s) \log n)$ dummy clauses is still a valid input. If the input was a satisfiable formula adding the dummy clauses does not change that fact and the input is still valid. Assume we have an input where at most $(7/8 + o(1))d$ clauses can be satisfied. In the modified input we can satisfy at most $(7/8 + o(1))d + \mathcal{O}(t(s) \log n)$ clauses. We can assume that $d \geq n/k$, otherwise we have variables that are not being used in the formula which may be removed. Now insert $t(n) \in \mathcal{O}(n^{1-\epsilon})$ in the expression. We have

$$\mathcal{O}(t(s) \log n) \subseteq \mathcal{O}(n^{1-\epsilon} \log n) \subseteq o(n) \subseteq o(d).$$

In the modified input we can satisfy at most $(7/8 + o(1))d + o(d)$ clauses. The $o(d)$ term can be absorbed in $o(1) \cdot d$ which means the modified input is a valid input for Gap-3-SAT.

We now know that any $\mathcal{O}(t(n))$ -local reduction from Gap-3-SAT to MCSP implies a $\mathcal{O}(t(n) \log^2 n)$ -local natural reduction from Gap-3-SAT to MCSP. Combine that with Corollary 3.5 to find that a $\mathcal{O}(n^{1-\epsilon})$ -local reduction from Gap-3-SAT to MCSP implies a circuit of depth 3 and size $2^{o(n)}$ that solves k -SAT. □

3.3 Reductions from OV to MCSP

A polynomial time reduction from OV to MCSP would imply that if SETH is true there is a polynomial lower bound on the time complexity of solving MCSP. There are still no well known superlinear bounds for MCSP and therefore such a result would reveal new information concerning its complexity. In this section we show that some naive reduction attempts from OV to MCSP do not work.

Theorem 3.8. *If SETH is true then for every $\epsilon > 0$ there are no $\mathcal{O}(\frac{\log n}{(\log \log n)^2})$ -local $\mathcal{O}(n^{2-\epsilon})$ -natural reductions from OV to MCSP.*

Lemma 3.8. *OV is $\mathcal{O}(1)$ -rejectable.*

Proof. Let $x = 1^{2^{nd}}$, that is every vector is 1 in every position. Clearly there is no pair of vectors which are orthogonal. We can make an algorithm which for every index i outputs 1 which takes $\mathcal{O}(1)$ time. □

Let us assume that there exists a $\mathcal{O}(\frac{\log n}{(\log \log n)^2})$ local natural reduction R from OV to MCSP. According to Theorem 3.1 R can be used to create an algorithm for OV.

Proof of Theorem 3.8. Insert $t(n) \in \mathcal{O}(\frac{\log n}{(\log \log n)^2})$, $q(n) \in \mathcal{O}(1)$ (shown in 3.3) and $M \in \mathcal{O}(n^{2-\epsilon})$ into the expression given by 3.1. Start by finding a bound for the exponent in the first factor.

$$2^{\mathcal{O}(t(n) \log(t(n) + \log M))} \in 2^{\mathcal{O}(\frac{\log n}{\log \log n})}$$

We get an algorithm solving OV in time

$$2^{\mathcal{O}(\frac{\log n}{\log \log n})} \cdot \mathcal{O}\left(\frac{\log n}{(\log \log n)^2} \cdot n^{2-\epsilon}\right).$$

which contradicts SETH. □

3.3.1 Circuit Complexity

Similar to what we did with SAT we can use Theorem 3.2 to show the same results is true even for a weaker assumption than SETH.

Theorem 3.9. *If there is no $\epsilon > 0$ such that OV can be solved by a circuit with depth 3 and size $\mathcal{O}(n^{2-\epsilon})$ then for every $\epsilon > 0$ there are no $\mathcal{O}(\log \log n)$ -local $\mathcal{O}(n^{2-\epsilon})$ -natural reductions from OV to MCSP.*

Proof. The proof follows the same steps as the proof of Theorem 3.8 but we use Theorem 3.2. Insert $t(n) \in \mathcal{O}(\frac{\log n}{(\log \log n)^2})$, $q(n) \in \mathcal{O}(1)$ and $M \in \mathcal{O}(n^{2-\epsilon})$ into the expression given by Theorem 3.2. We get a circuit solving OV with depth 3 and size

$$\mathcal{O}(\frac{\log n}{\log \log n}) \cdot \mathcal{O}(\frac{\log n}{(\log \log n)^2} \cdot n^{2-\epsilon}) \cdot 2^{\mathcal{O}(\frac{\log n}{(\log \log n)^2})}$$

which contradicts our assumption. □

Remark. *Theorem 3.9 is stronger than Theorem 3.8 since SETH implies that there are no circuits with strongly subquadratic size solving OV.*

Chapter 4

Discussion

4.1 Dependence on Encoding

One unfortunate aspect of studying natural local reductions is that results will depend on the input encoding of the problems. For some problems there is an obvious encoding that is essentially always used. Both MCSP and PARITY fall in this category. Since the inputs are bit strings there is little to gain by using any other encoding than the bit strings themselves. The input to OV is a list of bit strings which also implies an obvious encoding to use. For k -SAT it is not nearly as obvious.

A quick glance at the results of this thesis show that the non-reducibility results get weaker as $q(S)$ gets larger. In particular, if $q(S)$ is a fast growing function the results prove nothing and no natural local reductions can be ruled out as impossible. With this in mind it might seem tempting to look for encodings where $q(S)$ is as large as possible. It is important to recall that the $q(S)$ -rejectability is a measure of how hard it is to construct inputs that are not in a given language. A large $q(S)$ shows that such inputs are hard to generate. It seems counter-intuitive that an encoding for which it is hard to generate such inputs would make reductions from the language easier to find. On the opposite, when making reductions one would likely prefer an encoding that is concise and easy to work with. It may very well be the case that this contradiction is a “flaw” of the proof technique and not an indication that overly complicated encodings lead to faster reductions.

One alternative encoding of k -SAT is to consider a sorted list of all possible clauses. Then we encode the input as a bit string of length

$\mathcal{O}(n^k)$ with one bit per possible clause. A bit is 1 if the corresponding clause is included in the formula and 0 otherwise. In this encoding the instance containing all possible clauses is a string of ones and k -SAT is therefore $\mathcal{O}(1)$ -rejectable.

4.2 Conclusion

Many papers on MCSP seem to indicate that showing either NP-hardness or finding a polynomial solution are hard problems and this paper is no exception. While the types of reductions we consider here may seem oddly specific it is worth noting that many reductions are in fact both natural and local. A few examples of such well known reductions from SAT to other NP-hard problems can be found in Appendix A. All these examples use gadgets in the reductions. A gadget is a representation of some structure in A as some structure in B . In the examples reducing k -SAT to graph problems we see that the graph is made of subgraphs relating to either a single variable or a single clause. There are also more abstract structures one can use, for example reducing partial assignments of variables to vectors as seen in the fine-grained reduction from k -SAT to OV. Other SETH-hard problems such as Edit Distance and Fréchet Distance were also shown to be SETH-hard under gadget reductions[9][10]. Gadget reductions are usually both natural and local. The results in this paper indicate that looking for simple gadget reductions from k -SAT to MCSP is unlikely to be fruitful. The results concerning OV indicate that attempts to simply reduce every vector to some part of the truth table in order to give a polynomial lower bound for MCSP are unlikely to be successful.

4.3 Open Problems

Perhaps the most natural question to ask is whether or not there are any non-local reduction techniques that could prove useful in showing NP-hardness or polynomial lower bounds for MCSP. There might exist such a non-local reduction with which it is possible to prove that MCSP is NP-hard.

Another possibility is to try to make progress in the opposite direction by showing that there are no $t(n)$ -local reductions for even larger $t(n)$ than was shown in this thesis. By using the technique in this the-

sis it might be possibly to gain some logarithmic factors by modifying encodings and improving the rejectability algorithms. That being said, the technique seems to become useless around $t(n) \in \mathcal{O}(n)$ and anything beyond that will likely require some different technique.

Bibliography

- [1] Valentine Kabanets and Jin-Yi Cai. “Circuit minimization problem”. In: *Proceedings of the thirty-second annual ACM symposium on Theory of computing*. ACM. 2000, pp. 73–79.
- [2] Cody D Murray and R Ryan Williams. “On the (non) NP-hardness of computing circuit complexity”. In: *LIPICs-Leibniz International Proceedings in Informatics*. Vol. 33. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik. 2015.
- [3] William J Masek. “Some NP-complete set covering problems”. In: *Unpublished manuscript* (1979).
- [4] Rahul Santhanam Shuichi Hirahara Igor Carboni Oliveira. “NP-hardness of Minimum Circuit Size Problem for OR-AND-MOD Circuits”. In: *Electronic colloquium on computational complexity (ECCC)* (2018).
- [5] Stephen A Cook. “The complexity of theorem-proving procedures”. In: *Proceedings of the third annual ACM symposium on Theory of computing*. ACM. 1971, pp. 151–158.
- [6] Russell Impagliazzo and Ramamohan Paturi. “On the complexity of k-SAT”. In: *Journal of Computer and System Sciences* 62.2 (2001), pp. 367–375.
- [7] Virginia Vassilevska Williams. “Hardness of easy problems: Basing hardness on popular conjectures such as the strong exponential time hypothesis”. In: *Proc. International Symposium on Parameterized and Exact Computation*. 2015, pp. 16–28.
- [8] Ryan Williams. “A new algorithm for optimal constraint satisfaction and its implications”. In: *International Colloquium on Automata, Languages, and Programming*. Springer. 2004, pp. 1227–1237.

- [9] Arturs Backurs and Piotr Indyk. “Edit distance cannot be computed in strongly subquadratic time (unless SETH is false)”. In: *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*. ACM. 2015, pp. 51–58.
- [10] Karl Bringmann. “Why walking the dog takes time: Frechet distance has no strongly subquadratic algorithms unless SETH fails”. In: *Foundations of Computer Science (FOCS), 2014 IEEE 55th Annual Symposium on*. IEEE. 2014, pp. 661–670.
- [11] Johan Håstad. “Some optimal inapproximability results”. In: *Journal of the ACM (JACM)* 48.4 (2001), pp. 798–859.
- [12] Dana Moshkovitz and Ran Raz. “Two-query PCP with subconstant error”. In: *Journal of the ACM (JACM)* 57.5 (2010), p. 29.
- [13] Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. “Which problems have strongly exponential complexity?” In: *Journal of Computer and System Sciences* 63.4 (2001), pp. 512–530.

Appendices

Appendix A

Local Reductions from SAT

In order to understand how to relate to the results concerning non-existence of some local reductions to MCSP it is important to have some frame of reference. It is natural to ask whether or not one should be surprised that these reductions seem impossible. For this purpose it is beneficial to look at well known NP-hard problems and study under which reductions we can show NP-hardness. We will find that there are in fact several well known NP-hard problems for which there exist $\mathcal{O}(\log n)$ -local natural reductions from 3-SAT. This further strengthens the belief that showing NP-hardness of MCSP is a difficult problem with currently known techniques.

A.1 Independent Set

Independent Set

Input:	Graph $G = (V, E)$ where V, E denotes the set of vertices and edges respectively, and an integer k .
Output:	TRUE if there exists a set $V' \subseteq V$ such that $ V' = k$ and for every $u, v \in V', (u, v) \notin E$. FALSE otherwise.
Size parameters:	$ V , k$
Input encoding:	$ V ^2$ bits encoding the adjacency matrix of G .

A.1.1 Reducing 3-SAT to Independent Set

Assume we have an instance of 3-SAT with n variables and d clauses C_1, C_2, \dots, C_d each with exactly 3 literals. For every clause create 3 vertices labeled according with the literals in the clause. Connect the nodes with edges. For every pair of vertices, connect them with edges if their labels correspond to some variable and its inverse. Let $k = d$.

We omit the formal proof that this is a reduction but provide an outline of the idea behind the proof. The way the graph is constructed we can pick at most one vertex for every clause. Since $k = d$ the only way to find an independent set of size k is to pick exactly one vertex from every clause. Edges between variables and their inverses ensure that we can never include both labels. An independent set will therefore correspond to an assignment of some subset of the n variables. If the size of the independent set has size k the assignment must satisfy all clauses.

A.1.2 Locality

Theorem A.1. *The reduction from 3-SAT to Independent Set is a $\mathcal{O}(\log n)$ -local natural reduction.*

Proof. Let us start with showing that the reduction is natural. The reduction gives $|V| = 3d \in \mathcal{O}(n^3)$ which is decided by and polynomially related to the size of the 3-SAT instance. $k = d$ which is also decided by the size of the 3-SAT instance.

Let us show that the reduction is $\mathcal{O}(\log n)$ local. Let $V = \{v_1, v_2, \dots, v_{3d}\}$. A bit in the adjacency matrix is 1 if one of two conditions apply.

1. For every $0 \leq l \leq d - 1$, vertices v_{3l+1}, v_{3l+2} and v_{3l+3} are pairwise connected with edges. That is, for every $1 \leq i, j \leq 3d$, there is an edge (v_i, v_j) if $\lfloor \frac{i-1}{3} \rfloor = \lfloor \frac{j-1}{3} \rfloor$ which takes $\mathcal{O}(\log n)$ time to check.
2. For every $1 \leq i, j \leq 3d$, there is an edge (v_i, v_j) if their labels correspond to some variable and its inverse. We can find the label of any vertex in $\mathcal{O}(\log n)$ time by checking the corresponding literal in the 3-SAT instance. The labels are $\mathcal{O}(\log n)$ bits long.

This proves that any bit in the adjacency matrix can be computed in $\mathcal{O}(\log n)$ time. \square

A.2 Vertex Cover

Vertex Cover

Input: Graph $G = (V, E)$ where V, E denotes the set of vertices and edges respectively, and an integer k .

Output: TRUE if there exists a set $V' \subseteq V$ such that $|V'| \leq k$ and for every $(u, v) \in E, u \in V'$ or $v \in V'$. FALSE otherwise.

Size parameters: $|V|, k$

Input encoding: $|V|^2$ bits encoding the adjacency matrix of G .

A.2.1 Reducing 3-SAT to Vertex Cover

Assume we have an instance of 3-SAT with n variables x_1, x_2, \dots, x_n and d clauses C_1, C_2, \dots, C_d each with exactly 3 literals. For every variable x_i we create a *variable gadget* consisting of 2 vertices labelled x_i, \bar{x}_i . These vertices are connected with an edge. For every clause we create a *clause gadget* consisting of 3 vertices labelled with the respective literals in the clause. The 3 vertices are pairwise connected with edges. Every vertex in a clause gadget is also connected to the vertex with the same label in one of the variable gadgets. Let $k = n + 2d$.

We omit the formal proof that this is a reduction but provide an outline of the idea behind the proof. We notice that a vertex cover must include at least one vertex per variable gadget and two vertices per clause gadget. In order to not exceed $k = n + 2d$ vertices we have to pick exactly one vertex per variable gadget and exactly two vertices per clause gadget. If the 3-SAT is satisfiable, a vertex cover can be achieved by picking the vertex with the label corresponding to the assignment. For each clause we will at worst have two vertices with labels that are not in the assignment. We pick these for the cover. If there are less than two vertices with labels not in the assignment we can pick the second or both vertices in any way we want.

A.2.2 Locality

Theorem A.2. *The reduction from 3-SAT to Vertex Cover is a $\mathcal{O}(\log n)$ -local natural reduction.*

Proof. Let us start with showing that the reduction is natural. The reduction gives $|V| = 2n + 3d \in \mathcal{O}(n^3)$ which is decided by and polynomially related to the size of the 3-SAT instance. $k = n + 2d$ which is also decided by the size of the 3-SAT instance.

Let us show that the reduction is $\mathcal{O}(\log n)$ local. Let $V = \{v_1, v_2, \dots, v_{2n+3d}\}$. A bit in the adjacency matrix is 1 if one of three conditions apply.

1. For $0 \leq l \leq n - 1$, there is an edge (v_{2l+1}, v_{2l+2}) . These are the variable gadgets. This part of the adjacency matrix is determined only by n . When we output a bit of the adjacency matrix we can check if it is in this $n \times n$ submatrix in $\mathcal{O}(\log n)$ time.
2. For every $0 \leq l \leq m - 1$, vertices $v_{2n+3l+1}, v_{2n+3l+2}$ and $v_{2n+3l+3}$ are pairwise connected with edges. These are the clause gadgets. That is, for every $2n + 1 \leq i, j \leq 2n + 3m$, there is an edge (v_i, v_j) if $\lfloor \frac{i-2n-1}{3} \rfloor = \lfloor \frac{j-2n-1}{3} \rfloor$ which takes $\mathcal{O}(\log n)$ time to check.
3. For every $1 \leq i \leq 2n$ and $2n + 1 \leq j \leq 2n + 3m$, there is an edge (v_i, v_j) if v_i and v_j share the same label. The label of v_i can be deduced from i . We can find the label of v_j in $\mathcal{O}(\log n)$ time by checking the corresponding literal in the 3-SAT instance. The labels are $\mathcal{O}(\log n)$ bits long.

This proves that any bit in the adjacency matrix can be computed in $\mathcal{O}(\log n)$ time. \square

A.3 Hitting Set

Hitting Set

Input: Integers n', m', k' . m' sets $S_1, S_2, \dots, S_{m'} \subseteq [1, n]$.
Output: TRUE if there exists a set $S \subseteq [1, n]$ such that $|S| \leq k$ and $S \cap S_i \neq \emptyset$. FALSE otherwise.
Size parameters: n', m', k'
Input encoding: For each set we encode its size followed by at most n' integers describing the members of the set. Both the size and the members are $\mathcal{O}(\log n')$ bit integers.

A.3.1 Reducing 3-SAT to Hitting Set

Assume we have an instance of 3-SAT with n variables x_1, x_2, \dots, x_n and d clauses C_1, C_2, \dots, C_d each with exactly 3 literals. Let $n' = 2n$ and $m' = n + d$. Map the literals to integers, $x_1 \mapsto 1, \bar{x}_1 \mapsto 2, x_2 \mapsto 3, \bar{x}_2 \mapsto 4, \dots, x_n \mapsto n' - 1, \bar{x}_n \mapsto n'$. Create the sets $\{1, 2\}, \{3, 4\}, \dots, \{n' - 1, n'\}$. For every clause create a set of 3 integers according to the mapping for each literal in the clause. Let $k' = n$. We omit the formal proof that this is a reduction but provide an outline of the idea behind the proof. The n disjoint sets with two members correspond to the n variables in the 3-SAT instance. Any set of size $k = n$ that hits all of them corresponds to an assignment of the variables. The sets with 3 members are hit iff at least one of the literals in the corresponding clause is in the assignment.

A.3.2 Locality

Theorem A.3. *The reduction from 3-SAT to Hitting Set is a $\mathcal{O}(\log n)$ -local natural reduction.*

Proof. Let us start with showing that the reduction is natural. The reduction gives $n' = 2n, m' = n + d$, and $k' = n$, all which is decided by and polynomially related to the size of the 3-SAT instance.

Let us show that the reduction is $\mathcal{O}(\log n)$ local. When we look up a bit of the output there are two cases. Either the bit is part of one of

the first n sets or one of the last d sets. We can check which case it is in $\mathcal{O}(\log n)$ time.

- Case 1: first n sets. These sets are determined by only n . We have to check which set corresponds to the index of the bit and generate the two integers in the set. All integers are $\mathcal{O}(\log n)$ bits so we need $\mathcal{O}(\log n)$ time.
- Case 2: last d sets. These sets correspond to clauses so all we have to do is find the corresponding clause and the corresponding literal in the clause. We are comparing $\mathcal{O}(\log n)$ bit integers so we need $\mathcal{O}(\log n)$ time.

This proves that any bit in the output can be computed in $\mathcal{O}(\log n)$ time. \square

