The Minimum Formula Size Problem is (ETH) Hard

Rahul Ilango ✉
Massachusetts Institute of Technology, USA

Abstract

A longstanding open question is whether the Minimum Circuit Size Problem (MCSP) is NP-complete. In fact, even determining whether MCSP has a search-to-decision reduction has been open for over twenty years.

We show that, under the Exponential Time Hypothesis, the Minimum (De Morgan) Formula Size Problem, MFSP, is not in P. Building on this, we show that MFSP has a *polynomial-time* (exact) search-to-decision reduction, a result that does not relativize. Our main lemma relates the formula complexity of a partial function with the formula complexity of an associated total function and is proved using the "leaf weighting" technique of Buchfuhrer and Umans.

2012 ACM Subject Classification Theory of computation → Circuit complexity; Theory of computation → Problems, reductions and completeness

Keywords and phrases Minimum Formula Size Problem, Minimum Circuit Size Problem, Boolean Formulas, Meta-Complexity

Funding During parts of this work, the author was supported by an Akamai Presidential Fellowship and by NSF Grants CCF-1741615 and CCF-1909429.

Acknowledgements We thank Eric Allender, Abhishek Bhrushundi, Aditi Dudeja, Igor Oliveira, Hanlin Ren, Rahul Santhanam, and Ryan Williams for useful discussions on this work. In particular, Rahul Santhanam first pointed out to us the question of whether one could show that MFSP is not in P under ETH.

1 Introduction

The Minimum Circuit Size Problem (MCSP) asks one to determine whether a given function f (represented by its truth table) has a circuit of a given size s. While it is easy to see that this problem is in NP[1], it is a longstanding open question whether MCSP is NP-complete. Indeed, research on the intractability of MCSP dates back at least to the 1950s (see Traktenbrot [39] for a historical survey), and Levin actually delayed publishing his results on the theory of NP-completeness, in part because he hoped to show MCSP is NP-complete [24].

Interest in MCSP is further motivated by a growing number of intriguing connections between MCSP and areas such as pseudorandomness [30, 38], cryptography [25, 38], learning theory [7], circuit complexity [35, 22], proof complexity [34], and average-case complexity [12][2].

1.1 How Hard is MCSP?

The main motivation of our paper is to further understand the hardness of MCSP and its variants. Empirically, MCSP seems to be an intractable problem. Indeed, we do not know

[1] All functions $f : \{0,1\}^n \to \{0,1\}$ have a circuit C of size $2^n n$ (the naive DNF), so we can assume $s \leq 2^n n$. Thus, one can non-deterministically guess a circuit of size at most s and check if it computes f by evaluating it on all 2^n inputs. The truth table of f has length 2^n, so verifying that C computes f can be done in polynomial-time.

[2] These citations are not meant to be exhaustive. See Allender's survey [1] for a more complete overview.

any algorithm for solving MCSP better than brute-force search. On the other hand, so far the intractability of MCSP (i.e. MCSP $\notin$ P) can only be based on *cryptographic assumptions*. For example, Kabanets and Cai [22] show that MCSP $\notin$ P if one-way functions exist, and Allender and Das [2] prove that MCSP is hard for the cryptographically important class Statistical Zero Knowledge (SZK)[3].

Ideally, one would be able to prove MCSP is NP-complete. Besides characterizing the complexity of a natural optimization problem, this would also extend many of the intriguing properties known for MCSP to NP. For example, Hirahara [12] shows that if a certain approximation to MCSP is NP-complete, then NP has a worst-case to average-case reduction, resolving a longstanding open question in average-case complexity.

Despite this significant motivation, a proof that MCSP is NP-complete (or, conversely, evidence that it is unlikely to be NP-complete) remains elusive. Researchers have, however, discovered significant *technical barriers* to showing that MCSP is NP-hard [22, 30, 15, 37, 14]. These technical barriers do not suggest that MCSP is or is not likely NP-hard, they just imply that proving NP-hardness would be difficult. For example, Murray and Williams [30] show that if MCSP is NP-hard under polynomial-time many-one reductions, then EXP $\neq$ ZPP. Note that EXP $\neq$ ZPP is a consequence we believe but seems difficult to show. At heart, these technical barriers boil down to the following intuition[4]: any proof of hardness for MCSP implicitly shows how to "efficiently generate" intractable NO instances for MCSP, but intractable NO instances of MCSP correspond to functions that require large circuits, and we do not know how to produce explicit hard functions.

This raises a natural question: is the difficulty of showing that MCSP is, say, NP-complete primarily because of a lack of techniques (in particular, a lack of circuit lower bounds) or is it because it is actually just false? Could it be that MCSP is a hard problem, but just not an NP-hard problem? In support of the latter possibility, Allender and Hirahara [4] show that, for *very large* approximation factors, approximating MCSP is *not* NP-complete under a cryptographic assumption.

Motivated by the question of whether we should expect MCSP to be NP-complete, Kabanets and Cai [22] proposed an intermediate task over two decades ago: Does MCSP have a polynomial-time search-to-decision reduction? All NP-complete problems have a search-to-decision reduction (because SAT has one), so finding a search-to-decision reduction is a necessary step to showing that MCSP is NP-hard.

Over two decades since Kabanets and Cai raised the question, the problem is still wide open. Carmosino, Impagliazzo, Kabanets and Kolokolova [7] proved an "approximate" search-to-decision reduction (where instead of outputting an optimal circuit, you output an approximately optimal circuit that computes the function on most of its inputs), and Hirahara [12] and Santhanam [38] prove further extensions of this result. But it is still open to refute the possibility that the decision version of MCSP can be solved in, say, linear-time and the search version requires exponential-time. Recently, Ren and Santhanam [36] gave a partial explanation for this: there is a relativized world where the search problem requires nearly exponential time but the decision version is in linear-time. As a result, proving a polynomial-time search-to-decision reduction for MCSP requires non-relativizing techniques.

Given the above discussion, one may feel somewhat pessimistic about the possibility of proving NP-hardness of MCSP, at least without a major breakthrough. Luckily, the

---

[3] We view this as a cryptographic hardness result since SZK $\not\subseteq$ BPP implies auxiliary input one-way functions exist [33].

[4] Actually, [14] does not follow this intuition.

aforementioned technical barriers against proving NP-hardness disappear if one relaxes the notion of reduction from, say, polynomial-time reductions to, say, subexponential-time reductions. In particular, we already know functions that require linear-sized circuits (e.g. any function that depends on all of its inputs), and the brute-force algorithm for solving MCSP on functions with linear-sized[5] circuits requires superpolynomial-time. As a result, one could hope to show that MCSP is not in P under, say, the Exponential Time Hypothesis (ETH) by utilizing existing lower bound techniques. Indeed, we conjecture that this is the case.

▶ **Conjecture 1.** *If ETH is true, then solving* MCSP *on functions* $f : \{0,1\}^n \to \{0,1\}$ *with a size threshold* $s \leq O(n)$ *is not in* P.

Ilango [18] recently showed the above conjecture is true for Partial-MCSP, the *partial function* version of MCSP, where instead of being provided with a truth table T of a total function (i.e. $T \in \{0,1\}^{2^n}$), T is the truth table of a partial function ($T \in \{0,1,\star\}^{2^n}$). Therefore, "all one needs to do" to prove Conjecture 1 is give a reduction from Partial-MCSP to MCSP. Intriguingly, such a reduction is known for some restricted circuits classes such as DNFs [9, 13].[6] One of our main results is giving such a reduction for *formulas*.

1.2 The Minimum Formula Size Problem

Our results focus on the formula version of MCSP, the *Minimum Formula Size Problem* (MFSP). Formally, MFSP is the problem of

- **Given:** the truth table $T \in \{0,1\}^{2^n}$ of a function $f : \{0,1\}^n \to \{0,1\}$ and a positive integer s
- **Determine:** if there is a De Morgan formula with at most s leaves that computes f.

Our convention is to let $N = 2^n$ denote the length of the truth table and let n denote the number of inputs to f.

We remark that we have defined MFSP using the model of *De Morgan formulas* (i.e. formulas with AND, OR, and NOT gates). This is the usual notion of formulas, however, we note that this choice will be important for our results. We discuss this in more detail when we present our results in Section 1.3.

Our understanding of MFSP is in a similar state as our understanding of MCSP. We know that neither problem can be in P under various cryptographic assumptions, like the intractability of factoring Blum integers [5, 22, 35]. We also know that both MCSP and MFSP are not in $\mathsf{AC}^0[\oplus]$ [11]. Finally, we know that the partial function versions of both MFSP and MCSP are not in P under ETH [18]. Note: the discussion in [18] focuses on the partial function version of MCSP, however, [18] notes that proof also shows that the result in the case of formulas (in fact even in the case of *read-once* formulas!). This theorem will be crucial for our results.

There are some results, however, that are not known to hold for both MCSP and MFSP. While Allender and Das [2] prove MCSP is be hard for SZK, SZK-hardness is still open for MFSP. Also, [17] shows a better than brute-force search-to-decision reduction for MFSP (it runs in time $2^{.67N}$), while such a reduction is not known for MCSP.

[5] By linear, we mean linear in the number of inputs, not the length of the truth table.
[6] We cite [9] here, but this observation for DNFs was first made by Gimpel, and Gimpel's result is described in [9].

1.3 Results and Discussion

Our main technical contribution is showing a relationship between the formula complexity of an arbitrary partial function γ and the formula complexity of a related total function $\mathsf{Extend}[\gamma, s]$. In order to state this more formally, we introduce some notation. Let $\gamma : \{0,1\}^n \to \{0,1,\star\}$ be a partial function. Let $g^\star : \{0,1\}^n \to \{0,1\}$ be the total function that outputs 1 on input x if and only if $\gamma(x) = \star$. Finally, let $\mathsf{L}(\gamma)$ denote the size of the smallest formula that computes a total function that agrees with γ.

Our main technical lemma is as follows.

▶ **Lemma 2** (Informal version of Lemma 9)**.** *Let $\gamma : \{0,1\}^n \to \{0,1,\star\}$. If $s \geq \min\{\mathsf{L}(\gamma), \mathsf{L}(g^\star)\}$, then*

$$\mathsf{L}(\mathsf{Extend}[\gamma, s]) = \mathsf{L}(\gamma) + \mathsf{L}(g^\star) + 2s,$$

where $\mathsf{Extend}[\gamma, s] : \{0,1\}^n \times \{0,1\}^{2s} \to \{0,1\}$ is some total function that depends on γ.

Note that $\mathsf{Extend}[\gamma, s]$ and $g^\star$ are both total functions. As a result, one could use Lemma 2 to compute the formula complexity of a partial function $\gamma : \{0,1\}^n \to \{0,1,\star\}$ in time $2^{O(s+n)}$, using an oracle to MFSP, if one could somehow correctly guess a value of $s \geq \min\{\mathsf{L}(\gamma), \mathsf{L}(g^\star)\}$. We remark that the quantity $\min\{\mathsf{L}(\gamma), \mathsf{L}(g^\star)\}$ seems somewhat strange, but it arises naturally in the proof and that the structure of this upper bound turns out to be crucial for our results.

One of the main ideas we use to prove Lemma 2 is the "leaf weighting" technique of Buchfuhrer and Umans [6], who use the technique to show that the $\Sigma_2\mathsf{P}$ version of MFSP (i.e. the problem of given a formula φ and a size threshold s, determining if there exists a formula of size at most s computing the same function as φ) is hard for $\Sigma_2\mathsf{P}$. The leaf weighting technique allows one to give a weight k to an input variable z by replacing it with an OR of k new inputs $z_1, \ldots, z_k$. Buchfuhrer and Umans show that this construction effectively forces that any formula that wants to read z to pay for k leaves instead of just one. In our construction of $\mathsf{Extend}[\gamma, s]$, certain inputs are given weight s and if $s \geq \min\{\mathsf{L}(\gamma), \mathsf{L}(g^\star)\}$, we show that those inputs can appear at most once in any optimal formula for $\mathsf{Extend}[\gamma, s]$. This constrains any optimal formula enough to prove the lemma.

We note that using the leaf weighting technique is *extremely expensive*, as even adding just one extra variable (i.e. increasing the weight of a variable by one) *doubles* the total size of the truth table. As a result, in order to get efficient reductions (in our case, one subexponential-time reduction and one polynomial-time reduction) one needs to be rather thrifty in applying leaf weighting. Indeed, it is particularly surprising that leaf weighting is useful at all (let alone crucially important) in our proof that there is a polynomial-time search-to-decision reduction for MFSP.

We now discuss how we use Lemma 2 to prove our two main results. As we mentioned previously, [18] showed that the partial function version of MCSP is hard under ETH. As noted in [18], the proof also shows that the partial function version of MFSP is hard under ETH. In fact this hardness result applies when one is promised that $\mathsf{L}(\gamma) \leq 10n$.

▶ **Theorem 3** ([18])**.** *Under ETH, no deterministic algorithm can compute whether $\mathsf{L}(\gamma) = n$ in time $N^{o(\log \log N)}$, even under the promise that $\mathsf{L}(\gamma) \leq 10n$.*

On the other hand, we can use Lemma 2 to compute $\mathsf{L}(\gamma)$ in time $2^{O(n)} = \mathsf{poly}(N)$ with an oracle to MFSP if we are promised that $\mathsf{L}(\gamma) = O(n)$. Note that, in this case, we are applying Lemma 2 with the $\mathsf{L}(\gamma)$ upper bound on the quantity $\min\{\mathsf{L}(\gamma), \mathsf{L}(g^\star)\}$.

Thus, we get the following theorem.

▶ **Theorem 4** (Also Corollary 18). *Assume ETH is true. Then no deterministic algorithm solves* MFSP *in time* $N^{o(\log\log N)}$.

In fact the above hardness result holds even assuming the size parameter s is restricted to be at most $O(n^{10})$.[7] In comparison, the brute force algorithm runs in time at most $N^{O(\log^{10}\log n)}$ on these type of instances, so our (conditional) lower bound is optimal up to a constant power in the exponent. This is interesting from the perspective of recent hardness magnification results [8, 31, 32, 29, 26], which show that weak lower bounds against MCSP-like problems with small size thresholds imply strong lower bounds. For example, implicit in McKay, Murray, and Williams [29] is the result that if MFSP with size parameter restricted to being $s \leq O(n^{10})$ does not have a circuit of size $n \cdot \mathsf{poly}\log n$, then NP does not have polynomial-sized circuits. Our result proves a partial converse. If MFSP with size parameter restricted to being $s \leq O(n^{10})$ has a circuit of size $N^{o(\log\log N)}$, then the non-uniform version of ETH is false.

Our second main result is a *polynomial-time* search-to-decision reduction for MFSP.

▶ **Theorem 5.** *Given an oracle to* MFSP *and the truth table of a Boolean function* f*, one can find an optimal formula for* f *in deterministic polynomial-time.*

At a high-level, our algorithm is very intuitive. It works by combining the ideas used in the "better than brute-force" search-to-decision reduction in [17] with Lemma 2. In particular, given any "non-trivial" total function $f : \{0,1\}^n \to \{0,1\}$, we show how, using an oracle to MFSP, one can efficiently find functions $g, h : \{0,1\}^n \to \{0,1\}$ such that there is an optimal formula for f where the top two subformulas feeding into the output gate compute g and h respectively. We call such a (g, h) pair an *optimal decomposition of* f. By repeatedly finding optimal decompositions, one can recursively build an optimal formula for f.

Our method for finding an optimal decomposition (g, h) is to construct a partial function $\mathsf{OrSelect}[f, \gamma, \zeta]$ such that if $\gamma, \zeta : \{0,1\}^n \to \{0,1,\star\}$ are partial functions satisfying a certain condition, then the formula complexity of $\mathsf{OrSelect}[f, \gamma, \zeta]$ is small if and only if there is an optimal decomposition (g, h) such that g agrees with γ and h agrees with ζ. Since Lemma 2 gives us a way of calculating the formula complexity of the partial function $\mathsf{OrSelect}[f, \gamma, \zeta]$ using an oracle to MFSP, this allows us to find g and h by building them up bit-by-bit from γ and ζ. While this approach is quite intuitive, it runs into a serious problem: applying Lemma 2 to convert partial function queries into total function queries is very expensive. In general, it uses exponential-time. In order to implement this procedure in polynomial-time, it is critical to take a more fine-grained approach: using the upper bound of $\mathsf{L}(g^\star)$ on the quantity $\min\{\mathsf{L}(\gamma), \mathsf{L}(g^\star)\}$ in the statement of Lemma 2 and carefully working to maintain that $\mathsf{L}(g^\star)$ always stays small (at most $O(n)$).

Intriguingly, this result does not relativize. Santhanam and Ren [36] show that there is an oracle relative to which MCSP can be solved in deterministic linear time, but the search version of MCSP requires deterministic time $2^{\Omega(N/\log N)}$. Their proof also shows the same oracle separation for MFSP. As a result, we can conclude that our search-to-decision reduction *does not relativize.* This relativization barrier means that any polynomial-time search-to-decision reduction for MFSP must make significant use of properties of the underlying gate set. Our reduction does this, as our techniques rely heavily on the underlying gate set being AND, OR, and NOT. Indeed, it is not even clear whether our results extend to the $\mathbb{B}_2$ basis

[7] The reason why we get $O(n^{10})$ here instead of $O(n)$ is because we can only bound $\mathsf{L}(g^\star) = O(n^{10})$ on the instances produced in Theorem 3.

consisting of all Boolean functions from two bits to one![8]

1.4 Further Related Work

While it is beyond the scope of this work to give an exhaustive review of the literature on MCSP, there are several related works that we have not yet discussed that are worth mentioning.

One can view our hardness result for MFSP as fitting in a general line of work of showing hardness of MCSP for restricted circuit classes. Masek [28] shows that the DNF version of MCSP is NP-hard, and a long line of work [9, 40, 3, 10, 23] culminates in near-optimal hardness of approximation for the DNF version. Hirahara, Oliveira, and Santhanam [13] extend Masek's result to show NP-hardness for the version of MCSP for DNF circuits with parity gates at the bottom. Finally, [18] shows that the constant depth formula version of MCSP is NP-hard (under quasipolynomial-time randomized reductions).

In the realm of general circuits, the conditional version of MCSP and the multi-output version of MCSP are also both known to be NP-hard under randomized polynomial-time reductions [16, 19].

2 Preliminaries

If n is a positive integer, let $[n]$ denote the set $\{1, \ldots, n\}$. Let $\mathsf{OR}_n : \{0,1\}^n \to \{0,1\}$ denote the OR function on n inputs.

Partial, Total, and Monotone Boolean Functions. A (total) Boolean function is a function $g : \{0,1\}^n \to \{0,1\}$. A partial Boolean function is a function $\gamma : \{0,1\}^n \to \{0,1,\star\}$. We say $g : \{0,1\}^n \to \{0,1\}$ *agrees* with $\gamma : \{0,1\}^n \to \{0,1,\star\}$ if $g(x) = \gamma(x)$ for all x satisfying $\gamma(x) \in \{0,1\}$. We generally use Greek letters for partial functions and Roman letters for total functions (though there are some exceptions).

A monotone Boolean function is a Boolean function $f : \{0,1\}^n \to \{0,1\}$ with the following property: for all $X, Y \subseteq [n]$ with $X \subseteq Y$ we have that $f(x) \leq f(y)$ (where $x, y \in \{0,1\}^n$ are the Boolean strings whose i'th bits are one if and only if $i \in X$ or $i \in Y$ respectively). Note that the OR function and the AND function are monotone. Also note that compositions of monotone functions are also monotone. (In particular, this means that any Boolean circuit or formula that uses only AND and OR gates — and does not use NOT gates — must compute a monotone Boolean function).

De Morgan Formulas. In this paper, we will consider the model of De Morgan formulas, i.e. rooted binary trees whose internal nodes are labeled by AND and OR gates and whose leaves are labeled by elements from the set $\{0, 1, x_1, \ldots, x_n, \neg x_1, \ldots, \neg x_n\}$.

The *size*, denoted $|\varphi|$, of a formula φ is the total number of leaves in the underlying tree of φ that are not labeled by either 0 or 1.[9] The *formula complexity* of a (total) Boolean function f, denoted $\mathsf{L}(f)$, is the minimum size of any formula computing f. Similarly, if $\gamma : \{0,1\}^n \to \{0,1,\star\}$ is a partial Boolean function, let $\mathsf{L}(\gamma)$ be the minimum value of $\mathsf{L}(g)$ over all $g : \{0,1\}^n \to \{0,1\}$ that agree with γ.

[8] Our suspicion is that one can extend our results to the $\mathbb{B}_2$ basis, but this seems to require, at the very least, significantly more case analysis.

[9] The reason we can ignore leaves labelled by constants is that a gate elimination argument shows that such leaves can always be removed whenever φ computes a non-constant function.

We will make use of the fact that integer comparison can be implemented by linear-sized formulas.

▶ **Proposition 6** (Small formulas for integer comparison)**.** *Fix an arbitrary integer a and an input length $n \geq 1$. Let $GEq_a : \{0,1\}^n \to \{0,1\}$ be the function given by $GEq_a(x) = 1$ if and only if $x \geq a$ when x is interpreted as an integer in binary. Then $\mathsf{L}(GEq_a(x)) \leq n$.*

Proof. If $a \geq 2^n$ or $a \leq 0$, then GEq_a is the constant zero function or constant one function respectively, so for the remainder of the proof we assume that $0 < a < 2^n$.

We work by induction on n. If $n = 1$, then $0 < a < 2$, so $a = 1$, so $GEq_a(x) = x$, and the proposition clearly holds.

Now suppose $n > 1$. Since $0 < a < 2^n$, let $y \in \{0,1\}^n$ be the n-bit binary representation of a. Let $x_1, \ldots, x_n$ and $y_1, \ldots, y_n$ denote the bits of x and y respectively where x_1 and y_1 denotes the highest order bit. Let $x', y' \in \{0,1\}^{n-1}$ be given by $x' = x_2 \ldots x_n$ and $y' = y_2 \ldots y_n$ respectively.

If $y_1 = 1$, then

$$x \geq a \iff (x_1) \wedge (x' \geq y').$$

if $y_1 = 0$, then

$$x \geq a \iff (x_1) \vee (x' \geq y').$$

In either case, we get by induction that $\mathsf{L}(GEq_a) \leq 1 + n - 1 = n$.

◀

The Minimum Formula Size Problem and its Variants. The *Minimum Formula Size Problem*, MFSP, is defined as follows:

- **Given:** the truth table of a function $f : \{0,1\}^n \to \{0,1\}$ and an integer size parameter s.
- **Determine:** whether $\mathsf{L}(f) \leq s$.

The search variant of this problem is Search-MFSP:

- **Given:** the truth table of a function $f : \{0,1\}^n \to \{0,1\}$.
- **Output:** an optimal (De Morgan) formula for computing f.

Similarly, the partial function version Partial-MFSP is defined as:

- **Given:** the (2^n)-length truth table of a partial function $\gamma : \{0,1\}^n \to \{0,1,\star\}$ and an integer size parameter s.
- **Determine:** whether $\mathsf{L}(\gamma) \leq s$.

Throughout this paper, we adopt the convention that n denotes the number of inputs of a function $f : \{0,1\}^n \to \{0,1\}$ and $N = 2^n$ denotes the length of the truth table.

The Exponential Time Hypothesis. The Exponential Time Hypothesis (abbreviated ETH), first formulated by Impagliazzo, Paturi, and Zane [20, 21], has been extremely useful for proving conditional lower bounds on various problems (see [27] for a survey). Since the hypothesis itself is somewhat technical to define, we refer to the above papers for a formal definition. However, roughly speaking, it is a slight strengthening of the statement that 3-SAT cannot be solved deterministically in $2^{o(n)}$ time.

3 Our Main Lemma: Connecting Partial and Total Functions

To begin, we introduce some notation we will use throughout this section. Let $\gamma : \{0,1\}^n \to \{0,1,\star\}$ be a partial function. Let $g : \{0,1\}^n \to \{0,1\}$ be a total function that agrees with γ. For our purposes, any choice of g that agrees with γ will do, but to be concrete we set

$$g(x) = \begin{cases} 0 & \text{if } \gamma(x) = 0, \\ 1 & \text{otherwise.} \end{cases}$$

We also define the total function $g^\star : \{0,1\}^n \to \{0,1\}$ by

$$g^\star(x) = \begin{cases} 1 & \text{if } \gamma(x) = \star, \\ 0 & \text{otherwise.} \end{cases}$$

In other words, $g^\star$ is the indicator function for the event that $\gamma(x)$ equals $\star$.

Our main tool for relating the formula complexity of partial and total functions will be the following definition.

▶ **Definition 7.** *The function* $\mathsf{Extend}[\gamma, s] : \{0,1\}^n \times \{0,1\}^s \times \{0,1\}^s \to \{0,1\}$ *is given by*

$$\mathsf{Extend}[\gamma, s](x, y, z) = [(g(x) \vee \mathsf{OR}_s(y)) \wedge \mathsf{OR}_s(z)] \vee g^\star(x).$$

Observe that in the definition of $\mathsf{Extend}[\gamma, s]$ that g can be replaced with any function that agrees with γ and the resulting function $\mathsf{Extend}[\gamma, s]$ will be the same.[10] As a result, we get the following upper bound on the complexity of $\mathsf{Extend}[\gamma, s]$ essentially by construction (note that $\mathsf{L}(\mathsf{OR}_s) \leq s$).

▶ **Proposition 8.**

$$\mathsf{L}(\mathsf{Extend}[\gamma, s]) \leq \mathsf{L}(\gamma) + 2s + \mathsf{L}(g^\star).$$

Our main lemma will show that if s is sufficiently large and γ is non-constant, then this upper bound is actually tight!

▶ **Lemma 9** (Main Lemma). *Assume no constant function*[11] *agrees with* γ *and that* $s \geq \min\{\mathsf{L}(\gamma), \mathsf{L}(g^\star)\}$. *Then*

$$\mathsf{L}(\mathsf{Extend}[\gamma, s]) = \mathsf{L}(\gamma) + \mathsf{L}(g^\star) + 2s.$$

Before we prove our main lemma, we prove a special case[12] where $s = 1$ and where we restrict to formulas that only have one y and z leaf each. It will turn out that the general case reduces to this case.

▶ **Lemma 10.** *Assume no constant function agrees with* γ. *Then any formula* φ *that computes* $\mathsf{Extend}[\gamma, 1]$ *and contains exactly one* y *and exactly one* z *leaf has size at least*

$$\mathsf{L}(\gamma) + \mathsf{L}(g^\star) + 2$$

---

[10] This is because the $\vee g^\star(x)$ term ensures that the function outputs 1 whenever γ is undefined.

[11] Note that this non-constant hypothesis is necessary since if γ is the constant 1 function, then $\mathsf{Extend}[\gamma, s](x, y, z) = \mathsf{OR}_s(z)$ and $\mathsf{L}(\mathsf{Extend}[\gamma, s]) = s$.

[12] Technically, our "special case" is actually incomparable to the main lemma.

Proof. The main idea is that since y and z only occur once in φ, the *only* way that φ can compute $\mathsf{Extend}[\gamma, 1]$ is to mimic the method in our upper bound.

In more detail, because φ only reads y and z once, we can decompose the formula φ into three parts that (informally) correspond to

- a part that reads x and y but not z
- a part that reads x and z but not y, and
- a part that outputs a value based on x and the output of the other two parts.

Formally, there exist formulas ψ_y, ψ_z, and Φ and a gate $\blacktriangledown \in \{\wedge, \vee\}$ such that

$$\varphi(x, y, z) = \Phi(\psi_y(x, y) \blacktriangledown \psi_z(x, z), x),$$

and the following properties hold

- $\psi_y(x, y)$ takes inputs x and y and only has a single y leaf,
- $\psi_z(x, z)$ takes inputs x and z and only has a single z leaf,
- $\Phi(w, x)$ takes inputs w and x and has exactly one w leaf (we need to introduce the new variable w to mimic the input that corresponds to the output wire of $\psi_y(x, y) \blacktriangledown \psi_z(x, z)$ in φ), and
- $|\varphi| = |\psi_y| + |\psi_z| + |\Phi| - 1$ (the minus one comes from the extra w leaf).

We will show that $\Phi(\psi_z(x, 0), x)$ computes $g^\star$ and that $\psi_y(x, 0)$ computes a function that agrees with γ. Consequently, we get that[13]

$$|\varphi| = |\psi_y| + |\psi_z| + |\Phi| - 1 \geq \mathsf{L}(\gamma) + \mathsf{L}(g^\star) + 3 - 1 \geq \mathsf{L}(\gamma) + \mathsf{L}(g^\star) + 2,$$

as desired.

It remains to show that $\Phi(\psi_z(x, 0), x)$ computes $g^\star$ and that $\psi_y(x, 0)$ computes a function that agrees with γ. First, we observe that the formulas ψ_y, ψ_z, and Φ only read the w, y and z leaves positively (i.e. they do not use the negated version of any of these variables).

▷ Claim 11. Every w, y or z leaf in ψ_y, ψ_z, or Φ is read positively.

Proof. The only w-leaf appears in Φ and it is read positively by construction (all negations in Φ are at the leaf level and in φ the wire corresponding to w in Φ has at least one y leaf and one z leaf feeding into it).

Next, let $x^0 \in \{0, 1\}^n$ be such that $\gamma(x^0) = 0$. Then we have that

$$y \wedge z = \mathsf{Extend}[\gamma, 1](x^0, y, z) = \varphi(x^0, y, z).$$

Since $y \wedge z$ is a function that is monotone in y and z and there is exactly one y-leaf and one z-leaf in φ and the gate set $\{\wedge, \vee\}$ is monotone, it follows that y and z must only be read positively in Φ. ◁

Using this monotonicity, we gain some structural information of how Φ, ψ_y, and ψ_z act when x is fixed to certain values.

▷ Claim 12. For any fixed $x' \in \{0, 1\}^n$,

- $\Phi(w, x')$ is either a constant function or equals w,
- $\psi_y(x', y)$ is either a constant function or equals y, and
- $\psi_z(x', z)$ is either a constant function or equals z.

[13] the plus three in the middle inequality come from counting the number of y, z, and w leaves

Proof. Note that when x' is fixed that $\Phi(w, x'), \psi_y(x', y)$, and $\psi_z(x', z)$ are all Boolean functions on one bit that are monotone (they are monotone by Claim 11 since w, y, and z are all only read positively). The only monotone Boolean functions on one bit are the constant functions and the identity function. ◁

We can then use this structural information to determine how Φ, ψ_y, and ψ_z act on inputs x where $\gamma(x) \in \{0, 1\}$ and determine that $\blacktriangledown = \wedge$.

▷ Claim 13. $\blacktriangledown = \wedge$. Also, if $\gamma(x') \in \{0, 1\}$, then
- $\Phi(w, x') = w$,
- $\psi_z(x', z) = z$,
- $\psi_y(x', y) = \gamma(x') \vee y$.

Proof. Fix any x' such that $\gamma(x') \in \{0, 1\}$. Then

$$(\gamma(x') \vee y) \wedge z = \mathsf{Extend}[\gamma, 1](x', y, z) = \Phi(\psi_y(x', y) \blacktriangledown \psi_z(x', z), x').$$

Observe that this implies that neither $\Phi(w, x')$ nor $\psi_z(x', z)$ can be constant functions, so Claim 12 implies $\Phi(w, x') = w$ and $\psi_z(x', z) = z$. Thus,

$$(\gamma(x') \vee y) \wedge z = \psi_y(x', y) \blacktriangledown z.$$

Observe that implies $\blacktriangledown \neq \vee$, so $\blacktriangledown = \wedge$. Consequently, we must have that $\psi_y(x', y) = \gamma(x') \vee y$. ◁

Claim 13 shows that $\psi_y(x, 0)$ agrees with γ. Thus, it just remains to show that $\Phi(\psi_z(x, 0), x)$ computes $g^\star$.

If $g^\star(x) = 0$, then $\gamma(x) \in \{0, 1\}$, so by Claim 13

$$\Phi(\psi_z(x, 0), x) = \psi_z(x, 0) = 0.$$

On the other hand, if $g^\star(x) = 1$, then

$$1 = \mathsf{Extend}[\gamma, 1](x, 0, 0) = \Phi(\psi_y(x, 0) \wedge \psi_z(x, 0), x) \leq \Phi(\psi_z(x, 0), x)$$

where the last inequality uses that Φ is monotone in w and that

$$\psi_y(x, 0) \wedge \psi_z(x, 0) \leq \psi_z(x, 0).$$ ◀

We now prove our main lemma.

▶ **Lemma 9** (Main Lemma). *Assume no constant function*[14] *agrees with* γ *and that* $s \geq \min\{\mathsf{L}(\gamma), \mathsf{L}(g^\star)\}$. *Then*

$$\mathsf{L}(\mathsf{Extend}[\gamma, s]) = \mathsf{L}(\gamma) + \mathsf{L}(g^\star) + 2s.$$

Proof. The idea is to use Buchfuhrer and Uman's leaf weighting technique to reduce to Lemma 10.

For contradiction suppose φ is a formula for $\mathsf{Extend}[\gamma, s]$ with less than $\mathsf{L}(\gamma) + \mathsf{L}(g^\star) + 2s$ leaves. We begin by proving some simple lower bounds on the number of leaves in φ. Observe that $\mathsf{Extend}[\gamma, s](x, 0^s, 0^s) = g^\star(x)$ and $\mathsf{Extend}[\gamma, s](x, 0^s, 1^s) = g(x)$. Consequently, we have

[14] Note that this non-constant hypothesis is necessary since if γ is the constant 1 function, then $\mathsf{Extend}[\gamma, s](x, y, z) = \mathsf{OR}_s(z)$ and $\mathsf{L}(\mathsf{Extend}[\gamma, s]) = s$.

that φ has at least $\max\{\mathsf{L}(\gamma), \mathsf{L}(g^\star)\}$ many x-leaves. We also know that φ must have at least s y-leaves and at least s z-leaves (in fact at least one leaf labeled by each y_i and z_i input for each $i \in [s]$) since $\mathsf{Extend}[\gamma, s](x^0, y, z) = \mathsf{OR}_s(y) \wedge \mathsf{OR}_s(z)$ where x^0 is such that $\gamma(x^0) = 0$.

On the other hand, we can show that there is at least one y_i input and at least one z_j input that are each only read once.

▷ Claim 14. There exist $i, j \in [s]$ such that there is exactly one y_i leaf and exactly one z_j leaf in φ.

Proof. We only prove the existence of i. (The proof for j is similar.) The number of y leaves in φ is at most $|\varphi|$ minus the number of x-leaves in φ minus the number of z-leaves in φ. Using the bounds above, we therefore get that the total number of y-leaves is at most

$$\mathsf{L}(\gamma) + \mathsf{L}(g^\star) + 2s - 1 - \max\{\mathsf{L}(\gamma), \mathsf{L}(g^\star)\} - s \leq 2s - 1$$

where the last equality uses that $s \geq \min\{\mathsf{L}(\gamma), \mathsf{L}(g^\star)\}$. Thus, by the pigeonhole principle there exists an $i \in [s]$ such that y_i has at most one leaf in φ, and we already established that there must be at least one y_i leaf. ◁

Consider the formula φ' on $(n+2)$-inputs obtained by taking φ and setting all y and z inputs except y_i and z_j to be equal to 0. Then φ' is a formula for computing $\mathsf{Extend}[\gamma, 1]$ of size at most $|\varphi| - 2s + 2$ that reads y_i and z_i exactly once. By Lemma 10, we get that

$$|\varphi| - 2s + 2 \geq \mathsf{L}(\gamma) + \mathsf{L}(g^\star) + 2,$$

so we get the contradiction that

$$|\varphi| \geq L(\gamma) + \mathsf{L}(g^\star) + 2s.$$ ◀

4 Hardness for MFSP

Ilango [18] showed that the partial function version of MFSP is intractable under ETH.

▶ **Theorem 15** (Ilango [18]). *Assume ETH holds. Then no algorithm solves* Partial-MFSP *in deterministic time* $N^{o(\log \log N)}$.

In fact, [18] proves something stronger. Let $g^\star : \{0,1\}^n \to \{0,1\}$ be the function where $g^\star(x) = 1$ if and only if $\gamma(x) = \star$.

▶ **Theorem 16** (Ilango [18]). *Assume ETH holds. Then no algorithm running in deterministic time $N^{o(\log \log N)}$ can solve the following promise problem: given a partial function γ satisfying $\mathsf{L}(\gamma) \leq 10n$ and $\mathsf{L}(g^\star) \leq O(n^{10})$, determine if $\mathsf{L}(\gamma) \leq n$.*

Lemma 9 allows us to give a reduction from this promise problem to MFSP.

▶ **Theorem 17.** *Given access to an oracle computing* MFSP*, in time* $\mathsf{poly}(N)$ *one can solve the following promise problem: Given a partial function γ satisfying $\mathsf{L}(\gamma) \leq 10 \log N$, determine if $\mathsf{L}(\gamma) \leq \log N$. Moreover, the queries to the oracle have size parameter at most* $\max\{\mathsf{L}(\gamma), \mathsf{L}(g^\star)\} + 1$.

Proof. Let $s = \log N$ and $s' = 10 \log N$. The algorithm R for the reduction is very simple. Given a function $\gamma : \{0,1\}^n \to \{0,1\}$, compute

$$\Delta = \mathsf{L}(\mathsf{Extend}[\gamma, s']) - \mathsf{L}(g^\star) - 2s'.$$

If $\Delta \leq s$, then output YES. Otherwise output NO. This completes the description of the reduction.

Since $\mathsf{Extend}[\gamma, s(N)]$ takes $2s + \log N = O(\log N)$ inputs and is efficiently constructable given γ, it is easy to see that this algorithm runs in time $\mathsf{poly}(N)$ using the oracle to MFSP.

It remains to argue for correctness. If γ satisfies the promise (i.e., $\mathsf{L}(\gamma) \leq 10 \log N$), then Proposition 8 and Lemma 9 imply that

$$\mathsf{L}(\mathsf{Extend}[\gamma, s']) = \mathsf{L}(\gamma) + \mathsf{L}(g^\star) + 2s',$$

and therefore that

$$\Delta = \mathsf{L}(\gamma),$$

as desired. ◀

Combining Theorem 16 and Theorem 17, we get the following lower bound on MFSP assuming ETH.

▶ **Corollary 18.** *Assume ETH holds. Then no deterministic algorithm solves* MFSP *in time* $N^{o(\log \log N)}$*, even when* s *is restricted to be at most* $O(n^{10})$*.*

5 Solving Search-MFSP using Partial-MFSP

Our approach in this section builds on the ideas in [17], which shows a better than brute-force search-to-decision reduction for MFSP.

We begin by introducing some notation. Let $f, g, h : \{0,1\}^n \to \{0,1\}$. Say $(g, \blacktriangledown, h)$ is an *optimal decomposition of* f if

- $f = g \blacktriangledown h$,
- $\mathsf{L}(f) = \mathsf{L}(g) + \mathsf{L}(h)$, and
- g and h are non-constant functions[15].

All non-trivial functions have an optimal decomposition.

▶ **Proposition 19.** *Let* $f : \{0,1\}^n \to \{0,1\}$*. If* $\mathsf{L}(f) > 1$*, then* f *has an optimal decomposition.*

Proof. Let φ be an optimal formula for f. Without loss of generality, we can assume φ has no constant leaves (since $\mathsf{L}(f) > 1$). Also since $\mathsf{L}(f) > 1$, φ has at least two leaves. Therefore, we can decompose $\varphi = \varphi_g \blacktriangledown \varphi_h$ for some subformulas φ_g and φ_h of φ and some $\blacktriangledown \in \{\wedge, \vee\}$. Let g and h be the functions computed by φ_g and φ_h respectively. Since φ is optimal and has no constant leaves, neither g nor h is a constant function.

It remains to check that $\mathsf{L}(f) = \mathsf{L}(g) + \mathsf{L}(h)$. By construction, we have that

$$|\varphi| = |\varphi_g| + |\varphi_h|.$$

By the optimality of φ, we know that $|\varphi_f| = \mathsf{L}(f)$. Furthermore, we also know that $|\varphi_g| = \mathsf{L}(g)$ and $|\varphi_h| = \mathsf{L}(h)$. This is because if any smaller formulas for computing g and h existed, then they could be used to replace φ_g or φ_h in φ, resulting in the contradiction of a smaller formula for computing f. Thus, putting these bounds together, we get $\mathsf{L}(f) = \mathsf{L}(g) + \mathsf{L}(h)$. Therefore, $(g, \blacktriangledown, h)$ is an optimal decomposition of f. ◀

[15] This condition is added to avoid "trivial" decompositions like decomposing f as $(f, \vee, 0)$.

We say an optimal decomposition $(g, \blacktriangledown, h)$ of f is an *optimal OR decomposition of f* if $\blacktriangledown = \vee$. It is easy to see that if one could find optimal OR decompositions, one could solve Search-MFSP.

▶ **Proposition 20.** *Given access to an oracle $\mathcal{O}$ that outputs an optimal OR decomposition for a function (if such a decomposition exists), one can solve* Search-MFSP *in polynomial-time.*

Proof. Note that by De Morgan's law, $(g, \wedge, h)$ is an optimal decomposition of f if and only if $(\neg g, \vee, \neg h)$ is an optimal decomposition of $\neg f$. Therefore, by running $\mathcal{O}$ on both f and $\neg f$, we can find an optimal decomposition of f (if any optimal decomposition exists).

Consider the following recursive algorithm $\mathcal{A}$ for solving Search-MFSP. Given a function f, first check if $\mathsf{L}(f) \leq 1$ (via brute force). If so, then output an optimal formula for f via brute force. Otherwise, use the oracle $\mathcal{O}$ to find an optimal decomposition $(g, \blacktriangledown, h)$ for f. Finally, output the formula $\phi_g \blacktriangledown \phi_h$ where $\phi_g = \mathcal{A}(g)$ and $\phi_h = \mathcal{A}(h)$. This completes the description for $\mathcal{A}$.

It is easy to see that this algorithm runs in polynomial-time and solves Search-MFSP. ◀

For the next definition and the remainder of this paper, it will be useful to extend the OR function and the AND function to operate on inputs in $\{0, 1, \star\}$, in the natural way. Formally, if $a, b \in \{0, 1, \star\}$, then we let

$$a \vee b = \begin{cases} 1 & \text{if } 1 \in \{a, b\}, \\ 0 & \text{if } \{0\} = \{a, b\}, \\ \star & \text{otherwise,} \end{cases}$$

and

$$a \wedge b = \begin{cases} 0 & \text{if } 0 \in \{a, b\}, \\ 1 & \text{if } \{1\} = \{a, b\}, \\ \star & \text{otherwise.} \end{cases}$$

We now make our main definition for the section. This definition is a generalization of the Select function in [17] and is carefully chosen so that Proposition 22 and Lemma 23 hold.

▶ **Definition 21.** *Let $f : \{0, 1\}^n \to \{0, 1\}$ and $\gamma, \zeta : \{0, 1\}^n \to \{0, 1, \star\}$. Then* $\mathsf{OrSelect}[f, \gamma, \zeta]$: $\{0, 1\} \times \{0, 1\}^n \times \{0, 1\} \times \{0, 1\} \to \{0, 1, \star\}$ *is given by*

$$\mathsf{OrSelect}[f, \gamma, \zeta](w, x, y, z) = \begin{cases} f(x) & \text{if } (w, y, z) = (0, 1, 1), \\ ((\gamma(x) \vee w) \wedge y) \vee (\zeta(x) \wedge z) & \text{otherwise.} \end{cases}$$

We show that the complexity of $\mathsf{OrSelect}[f, g, h]$ is related to optimal OR decompositions.

▶ **Proposition 22.** *If $\mathsf{L}(f) > 1$ and $(g, \vee, h)$ is an optimal decomposition for f, then*

$$\mathsf{L}(\mathsf{OrSelect}[f, g, h]) = \mathsf{L}(f) + 3.$$

Proof. Let $\phi = \phi_g \vee \phi_h$ be an optimal formula for f where ϕ_g computes g and ϕ_h computes h.

Then the formula

$$((\phi_g(x) \vee w) \wedge y) \vee (\phi_h(x) \wedge z)$$

computes $\mathsf{OrSelect}[f, g, h]$. This gives us the desired upper bound.

For the lower bound, we know that any formula for $\mathsf{OrSelect}[f, g, h]$ must have at least $\mathsf{L}(f)$ many x-leaves since $\mathsf{OrSelect}[f, g, h]$ computes f when $(w, y, z) = (0, 1, 1)$. On the other hand, since g and h are both non-constant[16], $\mathsf{OrSelect}[f, g, h]$ depends on each of w, y, and z, so we need at least one w-leaf, one y-leaf, and one z-leaf. In total this gives $\mathsf{L}(f) + 3$ leaves. ◀

Our key insight is that one can prove a partial converse to Proposition 22.

▶ **Lemma 23.** *Let $f : \{0,1\}^n \to \{0,1\}$ and $\gamma, \zeta : \{0,1\}^n \to \{0,1,\star\}$. Assume there is a $\tilde{x} \in \{0,1\}^n$ such that $f(\tilde{x}) = \zeta(\tilde{x}) = 1$ and $\gamma(\tilde{x}) = 0$. If*

$$\mathsf{L}(\mathsf{OrSelect}[f, \gamma, \zeta]) \leq \mathsf{L}(f) + 3,$$

then there exist total functions $g, h : \{0,1\}^n \to \{0,1\}$ agreeing with γ and ζ respectively such that $(g, \vee, h)$ is an optimal decomposition for f.

Proof. Suppose φ is a formula for computing $\mathsf{OrSelect}[f, \gamma, \zeta]$ with at most $\mathsf{L}(f) + 3$ leaves.

We will show that φ can be decomposed as

$$\varphi(w, x, y, z) = \psi_{w,y}(w, x, y) \vee \psi_z(x, z)$$

for some formulas $\psi_{w,y}$ and ψ_z satisfying:

- $\psi_{w,y}$ has exactly one w-leaf and exactly one y-leaf,
- ψ_z has exactly one z-leaf, and
- $|\varphi| = |\psi_{w,y}| + |\psi_z|$.

Assuming we could decompose φ this way, we show how to prove the lemma. Recall the definition

$$\mathsf{OrSelect}[f, \gamma, \zeta](w, x, y, z) = \begin{cases} f(x) & \text{if } (w, y, z) = (0, 1, 1), \\ ((\gamma(x) \vee w) \wedge y) \vee (\zeta(x) \wedge z) & \text{otherwise} \end{cases}.$$

For the formula $\psi_{w,y}(w, x, y) \vee \psi_z(x, z)$ to compute $\mathsf{OrSelect}[f, \gamma, \zeta]$, observe that it must be the case that $\psi_{w,y}(w, x, 0) = 0$ and $\psi_z(x, 0) = 0$ since

$$0 = \mathsf{OrSelect}[f, \gamma, \zeta](w, x, 0, 0) = \psi_{w,y}(w, x, 0) \vee \psi_z(x, 0).$$

Consequently, we get that

$$\gamma(x) = \mathsf{OrSelect}[f, \gamma, \zeta](0, x, 1, 0) = \psi_{w,y}(0, x, 1) \vee \psi_z(x, 0) = \psi_{w,y}(0, x, 1),$$

so $\psi_{w,y}(0, x, 1)$ agrees with γ. Similarly,

$$\zeta(x) = \mathsf{OrSelect}[f, \gamma, \zeta](w, x, 0, 1) = \psi_{w,y}(w, x, 0) \vee \psi_z(x, 1) = \psi_z(x, 1),$$

so $\psi_z(x, 1)$ agrees with ζ. Finally, we know that

$$\psi_{w,y}(0, x, 1) \vee \psi_z(x, 1) = \mathsf{OrSelect}[f, \gamma, \zeta](0, x, 1, 1) = f(x).$$

Summarizing, we know the function g computed by $\psi_{w,y}(0, x, 1)$ agrees with γ, the function h computed by $\psi_z(x, 1)$ agrees with ζ, and that $g \vee h = f$. This proves the lemma.

[16] To see why this non-constant hypothesis is necessary, observe that if g were, for example, the constant one function, then the output of $\mathsf{OrSelect}[f, g, h]$ would not depend on w.

It remains to show that we can decompose φ as above. Without loss of generality we can assume φ has no constant leaves. Let $\blacktriangledown \in \{\wedge, \vee\}$ be the top/output gate of φ. Then we can write $\varphi(w,x,y,z) = \varphi_L(w,x,y,z) \blacktriangledown \varphi_R(w,x,y,z)$ where φ_L and φ_R are subformulas of φ with disjoint leaves.

By a similar argument as in the proof of Proposition 22, we know that φ has exactly $\mathsf{L}(f)$ x-leaves and one w, y, and z leaf each. As a result, we know that w, y, z are each only read by exactly one of φ_L and φ_R. Let $L \subseteq \{w, y, z\}$ be the subset of w, y, and z leaves that is read by φ_L, and let $R \subseteq \{w, y, z\}$ be the subset of w, y, z leaves that is read by φ_R. Observe that $L \cup R = \{w, y, z\}$ and $L \cap R = \emptyset$. Without loss of generality, assume that $|L| > |R|$ and therefore that $|R| \leq 1$.

First, we show that $|R| = 1$.

▷ Claim 24.

$|R| \neq 0$.

Proof. For contradiction, suppose that $|R| = 0$. Then the function computed by $\varphi_R(w,x,y,z)$ only depends on input x and not inputs w, y, and z. Since φ is an optimal formula for $\mathsf{OrSelect}[f,\gamma,\zeta]$ and φ has no constant leaves, it follows that $\varphi_R(w,x,y,z)$ does not compute a constant function. Therefore there exist $x^0 \in \{0,1\}^n$ and $x^1 \in \{0,1\}^n$ such that $\varphi_R(w,x^0,y,z) = 0$ and $\varphi_R(w,x^1,y,z) = 1$ for all $w,y,z \in \{0,1\}$. Now, either $\blacktriangledown = \wedge$ or $\blacktriangledown = \vee$. If $\blacktriangledown = \wedge$, then

$$1 = \mathsf{OrSelect}[f,\gamma,\zeta](w,x^0,y,z) = \varphi_L(w,x^0,y,z) \wedge \varphi_R(w,x^0,y,z) = 0$$

if $(w,y,z) = (1,1,1)$, which is a contradiction. If $\blacktriangledown = \vee$, then

$$0 = \mathsf{OrSelect}[f,\gamma,\zeta](w,x^1,y,z) = \varphi_L(w,x^1,y,z) \vee \varphi_R(w,x^1,y,z) = 1$$

if $(w,y,z) = (0,0,0)$, which is a contradiction. ◁

Recall, $\tilde{x} \in \{0,1\}^n$ is such that $f(\tilde{x}) = \zeta(\tilde{x}) = 1$ and $\gamma(\tilde{x}) = 0$. Thus,

$$\varphi_L(w,\tilde{x},y,z) \blacktriangledown \varphi_R(w,\tilde{x},y,z) = \mathsf{OrSelect}[f,\gamma,\zeta](w,\tilde{x},y,z) = (w \wedge y) \vee z.$$

Now, since φ_L only reads two inputs from the set $\{w,y,z\}$ and since φ_R only reads one input from the set $\{w,y,z\}$, we know that $\varphi_L(w,\tilde{x},y,z)$ computes a function that depends on both inputs in L and φ_R computes a function that depends on the input in R.

On the other hand, since $\mathsf{OrSelect}[f,\gamma,\zeta]$ is function that is monotone in each of w, y, and z and φ reads inputs w, y, and z each exactly once, we know that φ reads each input w, y, and z exactly once positively. Consequently, we get that φ_L computes a function that is monotone with respect to the variables in L, and that φ_R computes a function that is monotone with respect to the variables in R. Therefore, $\varphi_R(w,\tilde{x},y,z)$ is a monotone function on one bit (the input in R) that depends on the input in R. The only monotone Boolean function on one bit that depends on its input is the identity function, so $\varphi_R(w,\tilde{x},y,z)$ just outputs r, where $R = \{r\}$.

Similarly, $\varphi_L(w,\tilde{x},y,z)$ is a monotone function on two bits (the two inputs in L) that depends on both inputs in L. Consequently, we know that $\varphi_L(w,\tilde{x},y,z)$ computes the function $\ell_1 \blacktriangledown_L \ell_2$ for some $\blacktriangledown_L \in \{\wedge, \vee\}$, where $L = \{\ell_1, \ell_2\}$. This is because the only monotone Boolean functions on two bits that depend on both inputs are the AND function and the OR function.

At this point, we have determined that

$$(w \wedge y) \vee z = \varphi_L(w, \tilde{x}, y, z) \blacktriangledown \varphi_R(w, \tilde{x}, y, z) = (\ell_1 \blacktriangledown_L \ell_2) \blacktriangledown r,$$

where $\blacktriangledown_L, \blacktriangledown \in \{\wedge, \vee\}$ and $\{\ell_1, \ell_2, r\} = \{w, y, z\}$. Observe that the only way for this to occur is if $R = \{z\}$, $\blacktriangledown = \vee$, $\{\ell_1, \ell_2\} = L = \{w, y\}$, which is what we wanted to show. ◀

▶ **Theorem 25.** *There is a deterministic polynomial time algorithm that given an oracle* Partial-MFSP *and a function f with $\mathsf{L}(f) > 1$ finds an optimal OR decomposition of f, if one exists.*

Proof. At a high-level the idea is as follows. Recall, the search to decision reduction for SAT that works by building a satisfying assignment "bit by bit." Lemma 23 gives us a way to find an optimal OR decomposition in a similar "bit by bit" way by, roughly, starting with completely undefined γ and ζ and then filling them in one bit at a time, while making sure $\mathsf{L}(\mathsf{OrSelect}[f, \gamma, \zeta]) \leq \mathsf{L}(f) + 3$ and hence that we always fill in a bit that is in an optimal OR decomposition.

In more detail, the algorithm $\mathcal{A}$ on input $f : \{0,1\}^n \to \{0,1\}$ works as follows. For each $\tilde{x} \in \{0,1\}^n$ where $f(\tilde{x}) = 1$ run the following subroutine:

1. Set $\gamma : \{0,1\}^n \to \{0,1,\star\}$ and $\zeta : \{0,1\}^n \to \{0,1,\star\}$ to be completely undefined (i.e. output $\star$) everywhere except for at $\tilde{x}$, where $\gamma(\tilde{x}) = 0$ and $\zeta(\tilde{x}) = 1$.
2. If $\mathsf{L}(\mathsf{OrSelect}[f, \gamma, \zeta]) > \mathsf{L}(f) + 3$, then skip the remainder of this subroutine and go to the next value of $\tilde{x}$. Otherwise, continue to (3).
3. While γ is not a total function:
   a. Let $x \in \{0,1\}^n$ be the lexicographically first x such that $\gamma(x) = \star$.
   b. Set $\gamma(x)$ to some Boolean value $b \in \{0,1\}$ such that $\mathsf{L}(\mathsf{OrSelect}[f, \gamma, \zeta]) \leq \mathsf{L}(f) + 3$.
4. While ζ is not a total function:
   a. Let $x \in \{0,1\}^n$ be the lexicographically first x such that $\zeta(x) = \star$.
   b. Set $\zeta(x)$ to some Boolean value $b \in \{0,1\}$ such that $\mathsf{L}(\mathsf{OrSelect}[f, \gamma, \zeta]) \leq \mathsf{L}(f) + 3$.
5. Output the OR decomposition $(\gamma, \vee, \zeta)$ of f.

Finally, output $\perp$ if we have iterated through all such $\tilde{x}$ and not output an answer. This completes the description of the algorithm $\mathcal{A}$. It is easy to see that this algorithm runs in polynomial time. It remains to argue for correctness.

We first argue that if $\mathcal{A}$ does not output $\perp$, then it outputs a (valid) OR decomposition of f. This is because if $\mathcal{A}$ does not output $\perp$, then the algorithm must output total functions γ and ζ such that $\mathsf{L}(\mathsf{OrSelect}[f, \gamma, \zeta]) \leq \mathsf{L}(f) + 3$ and such that $f(\tilde{x}) = \zeta(\tilde{x}) = 1$ and $\gamma(\tilde{x}) = 0$. Lemma 23 then implies that $(\gamma, \vee, \zeta)$ is an optimal OR decomposition for f.

It remains to show that if f has an optimal OR decomposition then, the algorithm does not output $\perp$. Let $(g, \vee, h)$ be an optimal OR decomposition of f. Since $f = g \vee h$ and $g \neq h$ (if $g = h$, this would contradict optimality), there must exist a $\tilde{x}$ such that $1 = f(\tilde{x}) = h(\tilde{x})$ and $0 = g(\tilde{x})$.

Consider the above subroutine on this $\tilde{x}$. At step (2) in the subroutine, we have that

$$\mathsf{L}(\mathsf{OrSelect}[f, \gamma, \zeta]) \leq \mathsf{L}(\mathsf{OrSelect}[f, g, h]) \leq \mathsf{L}(f) + 3$$

where the middle inequality comes from g and h agreeing with γ and ζ respectively, and where the last inequality comes from by Proposition 22. Thus, the subroutine will reach (3). At this point, the only way the subroutine would not output an OR decomposition is if step (3b) or (4b) failed. We show that step (3b) can never fail (the proof for (4b) is similar). Step (3b) will fail if there does not exist a $b \in \{0,1\}$ to set $\gamma(x)$ to such that $\mathsf{L}(\mathsf{OrSelect}[f,\gamma,\zeta]) \le \mathsf{L}(f)+3$. This is not possible because Lemma 23 implies that there must be an optimal decomposition $(g',\vee,h')$ such that g agrees with γ and ζ agrees with h'. Combining this with Proposition 22's upper bound that $\mathsf{L}(\mathsf{OrSelect}[f,g',h']) \le \mathsf{L}(f)+3$, we get that setting $b = g'(x)$ will work. ◀

6 A Search to Decision Reduction for MFSP

In this section, we show that MFSP has a polynomial-time search-to-decision reduction. The key to proving this will be showing that the algorithm in Theorem 25 actually only makes queries to the Partial-MFSP oracle of a certain type: partial functions where the locations of the $\star$-values have low circuit complexity.

▶ **Lemma 26.** *Let $Q : \{0,1\}^n \to \{0,1,\star\}$ be a (partial) function that the algorithm in Theorem 25 generates as a query to its* Partial-MFSP *oracle. Let $Q^\star : \{0,1\}^n \to \{0,1\}$ be the function where $Q^\star(x) = 1$ if and only if $Q(x) = \star$. Then $\mathsf{L}(Q^\star) \le cn$ for some universal integer constant $c \ge 1$.*

Proof. Using the notation in the proof of Theorem 25, the algorithm only makes two types of queries to the oracle. It queries $\mathsf{L}(f)$, or it queries $\mathsf{L}(\mathsf{OrSelect}[f,\gamma,\zeta])$. f is a total function, so the lemma vacuously holds in that case. So now suppose $Q = \mathsf{OrSelect}[f,\gamma,\zeta]$.

Observe that the following invariant is held throughout each subroutine: there are integers $i_\gamma \in \{0,\dots,N+1\}$ and $i_\zeta \in \{0,\dots,N+1\}$ such that for all x

- $\gamma(x) = \star$ if and only if $x \ne \tilde{x}$ and $x \ge i_\gamma$, and
- $\zeta(x) = \star$ if and only if $x \ne \tilde{x}$ and $x \ge i_\zeta$.

This invariant clearly holds at the start of the subroutine (since the only non-star value is at $\tilde{x}$), and this invariant is maintained because the values set in γ and ζ are always the lexicographically first undefined value.

Consequently, using the linear formula upper bound on integer comparison in Proposition 6 and the fact that one can check whether whether $x \ne \tilde{x}$ with a linear-sized formula, we get linear formula size upper bounds on the functions $\gamma^\star, \zeta^\star : \{0,1\}^n \to \{0,1\}$ given by $\gamma^\star(x) = 1$ if and only if $\gamma(x) = \star$ and $\zeta^\star(x) = 1$ if and only if $\zeta(x) = \star$. In other words, $\mathsf{L}(\gamma^\star) \le O(n)$ and $\mathsf{L}(\zeta^\star) \le O(n)$.

Now we bound the complexity of the function $Q^\star$ that indicates whether

$$\mathsf{OrSelect}[f,\gamma,\zeta](w,x,y,z) = \star.$$

Recall,

$$\mathsf{OrSelect}[f,\gamma,\zeta](w,x,y,z) = \begin{cases} f(x) & \text{if } (w,y,z) = (0,1,1), \\ ((\gamma(x) \vee w) \wedge y) \vee (\zeta(x) \wedge z) & \text{otherwise.} \end{cases}$$

Observe that all of the following hold:

- if $(w,y,z) = (0,1,1)$, then $Q^\star(w,x,y,z) = 0$ because $\mathsf{OrSelect}[f,\gamma,\zeta](w,x,y,z) = f(x)$,
- if $(w,y,z) = (1,1,1)$, then $Q^\star(w,x,y,z) = 0$ because $\mathsf{OrSelect}[f,\gamma,\zeta](w,x,y,z) = 1$,
- if $(y,z) = (0,0)$, then $Q^\star(w,x,y,z) = 0$ because $\mathsf{OrSelect}[f,\gamma,\zeta](w,x,y,z) = 0$,
- if $(y,z) = (0,1)$, then $Q^\star(w,x,y,z) = \zeta^\star(x)$ because $\mathsf{OrSelect}[f,\gamma,\zeta](w,x,y,z) = \zeta(x)$, and
- if $(y,z) = (1,0)$, then $Q^\star(w,x,y,z) = \gamma^\star(x) \wedge (\neg w)$ because $\mathsf{OrSelect}[f,\gamma,\zeta](w,x,y,z) = \gamma(x) \vee w$.

As a result, we can upper bound the complexity of $Q^\star$

$$\mathsf{L}(Q^\star) = O(\mathsf{L}(\gamma^\star) + \mathsf{L}(\zeta^\star) + 1) = O(n),$$

as desired. ◀

We now observe that any query Q made to a Partial-MFSP oracle where $Q^\star$ has low formula complexity can be answered using a MFSP oracle by utilizing the $\mathsf{Extend}[\cdot,\cdot]$ function.

▶ **Proposition 27.** *Let $Q : \{0,1\}^n \to \{0,1,\star\}$, $Q^\star : \{0,1\}^n \to \{0,1\}$, and c be as in the statement of Lemma 26. Then*

$$\mathsf{L}(Q) = \mathsf{L}(\mathsf{Extend}[Q, cn]) - \mathsf{L}(Q^\star) - 2cn$$

Proof. Apply Lemma 9 and use the upper bound $\mathsf{L}(Q^\star) \le cn$ proved in Lemma 26. ◀

This gives us a way to compute the complexity of a partial function Q using only total functions.

▶ **Theorem 28.** *There is a deterministic polynomial time algorithm that given the truth table of a function $f : \{0,1\}^n \to \{0,1\}$, finds an optimal formula for f.*

Proof. By Proposition 20, it suffices to show how to find an optimal OR decomposition of a given function (if one exists). Theorem 25 shows how to find optimal OR decompositions using an oracle to Partial-MFSP, and Proposition 27 shows that the queries Q made to the Partial-MFSP oracle can be efficiently replaced by queries to a MFSP oracle on $\mathsf{Extend}[Q, cn]$ and $Q^\star$. Note that the number of inputs to $\mathsf{Extend}[Q, cn]$ is $n+2cn = O(n)$. As a result, both the truth table of $Q^\star$ and the truth table of $\mathsf{Extend}[Q, cn]$ can be computed in polynomial-time given Q. This proves the theorem. ◀

References

1. Eric Allender. The new complexity landscape around circuit minimization. In *Language and Automata Theory and Applications - 14th International Conference, LATA 2020, Milan, Italy, March 4-6, 2020, Proceedings*, volume 12038 of *Lecture Notes in Computer Science*, pages 3–16. Springer, 2020.
2. Eric Allender and Bireswar Das. Zero knowledge and circuit minimization. In Erzsébet Csuhaj-Varjú, Martin Dietzfelbinger, and Zoltán Ésik, editors, *Mathematical Foundations of Computer Science 2014*, pages 25–32, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.
3. Eric Allender, Lisa Hellerstein, Paul McCabe, Toniann Pitassi, and Michael E. Saks. Minimizing DNF formulas and AC0d circuits given a truth table. In *21st Annual IEEE Conference on Computational Complexity (CCC)*, pages 237–251, 2006.
4. Eric Allender and Shuichi Hirahara. New insights on the (non-)hardness of circuit minimization and related problems. *ACM Trans. Comput. Theory*, 11(4):27:1–27:27, 2019. doi:10.1145/3349616.
5. Eric Allender, Michal Koucký, Detlef Ronneburger, and Sambuddha Roy. The pervasive reach of resource-bounded kolmogorov complexity in computational complexity theory. *Journal of Computer and System Sciences*, 77(1):14 – 40, 2011.
6. David Buchfuhrer and Christopher Umans. The complexity of boolean formula minimization. *J. Comput. Syst. Sci.*, 77(1):142–153, 2011.
7. Marco L. Carmosino, Russell Impagliazzo, Valentine Kabanets, and Antonina Kolokolova. Learning algorithms from natural proofs. In *Proceedings of the 31st Conference on Computational Complexity*, 2016.

8 Lijie Chen, Shuichi Hirahara, Igor Carboni Oliveira, Ján Pich, Ninad Rajgopal, and Rahul Santhanam. Beyond natural proofs: Hardness magnification and locality. In Thomas Vidick, editor, *11th Innovations in Theoretical Computer Science Conference, ITCS 2020, January 12-14, 2020, Seattle, Washington, USA*, volume 151 of *LIPIcs*, pages 70:1–70:48. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. `doi:10.4230/LIPIcs.ITCS.2020.70`.

9 Sebastian Lukas Arne Czort. The complexity of minimizing disjunctive normal form formulas. Master's thesis, University of Aarhus, 1999.

10 Vitaly Feldman. Hardness of approximate two-level logic minimization and PAC learning with membership queries. In *38th Annual ACM Symposium on Theory of Computing (STOC)*, pages 363–372, 2006.

11 Alexander Golovnev, Rahul Ilango, Russell Impagliazzo, Valentine Kabanets, Antonina Kolokolova, and Avishay Tal. $AC^0[p]$ lower bounds against MCSP via the coin problem. In *ICALP*, volume 132 of *LIPICS*, pages 66:1–66:15, 2019.

12 Shuichi Hirahara. Non-black-box worst-case to average-case reductions within NP. In *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS*, pages 247–258, 2018.

13 Shuichi Hirahara, Igor Carboni Oliveira, and Rahul Santhanam. NP-hardness of minimum circuit size problem for OR-AND-MOD circuits. In *33rd Computational Complexity Conference (CCC)*, volume 102, pages 5:1–5:31, 2018.

14 Shuichi Hirahara and Osamu Watanabe. Limits of minimum circuit size problem as oracle. In Ran Raz, editor, *31st Conference on Computational Complexity, CCC 2016, May 29 to June 1, 2016, Tokyo, Japan*, volume 50 of *LIPIcs*, pages 18:1–18:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. `doi:10.4230/LIPIcs.CCC.2016.18`.

15 John M. Hitchcock and A. Pavan. On the NP-completeness of the minimum circuit size problem. In Prahladh Harsha and G. Ramalingam, editors, *35th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2015)*, volume 45 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 236–245, Dagstuhl, Germany, 2015. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. URL: `http://drops.dagstuhl.de/opus/volltexte/2015/5661`, `doi:10.4230/LIPIcs.FSTTCS.2015.236`.

16 Rahul Ilango. Approaching MCSP from above and below: Hardness for a conditional variant and AC0[p]. In *11th Innovations in Theoretical Computer Science Conference (ITCS)*, volume 151, pages 34:1–34:26, 2020.

17 Rahul Ilango. Connecting perebor conjectures: Towards a search to decision reduction for minimizing formulas. In Shubhangi Saraf, editor, *35th Computational Complexity Conference, CCC 2020, July 28-31, 2020, Saarbrücken, Germany (Virtual Conference)*, volume 169 of *LIPIcs*, pages 31:1–31:35. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. `doi:10.4230/LIPIcs.CCC.2020.31`.

18 Rahul Ilango. Constant depth formula and partial function versions of MCSP are hard. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 424–433. IEEE, 2020. `doi:10.1109/FOCS46700.2020.00047`.

19 Rahul Ilango, Bruno Loff, and Igor Carboni Oliveira. NP-hardness of circuit minimization for multi-output functions. In *35th Computational Complexity Conference (CCC)*, volume 169, pages 22:1–22:36, 2020.

20 Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-SAT. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001.

21 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001.

22 Valentine Kabanets and Jin-yi Cai. Circuit minimization problem. In *32nd Annual ACM Symposium on Theory of Computing (STOC)*, pages 73–79, 2000.

23 Subhash Khot and Rishi Saket. Hardness of minimizing and learning DNF expressions. In *49th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 231–240, 2008.

24 Leonid Levin. Hardness of search problems. URL: `https://www.cs.bu.edu/fac/lnd/research/hard.htm`.

25 Yanyi Liu and Rafael Pass. On one-way functions and Kolmogorov complexity. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1243–1254, 2020. doi:10.1109/FOCS46700.2020.00118.

26 Yanyi Liu and Rafael Pass. Cryptography from sublinear-time average-case hardness of time-bounded kolmogorov complexity. *Electron. Colloquium Comput. Complex.*, 28:55, 2021. URL: `https://eccc.weizmann.ac.il/report/2021/055`.

27 Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Lower bounds based on the exponential time hypothesis. *Bull. EATCS*, 105:41–72, 2011.

28 William J. Masek. Some NP-complete set covering problems. Unpublished Manuscript, 1979.

29 Dylan M. McKay, Cody D. Murray, and R. Ryan Williams. Weak lower bounds on resource-bounded compression imply strong separations of complexity classes. In Moses Charikar and Edith Cohen, editors, *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*, pages 1215–1225. ACM, 2019. doi:10.1145/3313276.3316396.

30 Cody D. Murray and R. Ryan Williams. On the (non) NP-hardness of computing circuit complexity. *Theory of Computing*, 13(1):1–22, 2017.

31 Igor Carboni Oliveira, Ján Pich, and Rahul Santhanam. Hardness magnification near state-of-the-art lower bounds. In Amir Shpilka, editor, *34th Computational Complexity Conference, CCC 2019, July 18-20, 2019, New Brunswick, NJ, USA*, volume 137 of *LIPIcs*, pages 27:1–27:29. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.CCC.2019.27.

32 Igor Carboni Oliveira and Rahul Santhanam. Hardness magnification for natural problems. In Mikkel Thorup, editor, *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*, pages 65–76. IEEE Computer Society, 2018. doi:10.1109/FOCS.2018.00016.

33 R. Ostrovsky and A. Wigderson. One-way functions are essential for non-trivial zero-knowledge. In *[1993] The 2nd Israel Symposium on Theory and Computing Systems*, pages 3–17, 1993. doi:10.1109/ISTCS.1993.253489.

34 Jan Pich and Rahul Santhanam. Why are proof complexity lower bounds hard? In *2019 IEEE 60th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1305–1324, 2019. doi:10.1109/FOCS.2019.00080.

35 Alexander A. Razborov and Steven Rudich. Natural proofs. *J. Comput. Syst. Sci.*, 55(1):24–35, 1997.

36 Hanlin Ren and Rahul Santhanam. A relativization perspective on meta-complexity. *Electron. Colloquium Comput. Complex.*, page 89, 2021. URL: `https://eccc.weizmann.ac.il/report/2021/089`.

37 Michael Saks and Rahul Santhanam. Circuit lower bounds from NP-hardness of MCSP under turing reductions. In Shubhangi Saraf, editor, *35th Computational Complexity Conference, CCC 2020, July 28-31, 2020, Saarbrücken, Germany (Virtual Conference)*, volume 169 of *LIPIcs*, pages 26:1–26:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.CCC.2020.26.

38 Rahul Santhanam. Pseudorandomness and the minimum circuit size problem. In *11th Innovations in Theoretical Computer Science Conference, ITCS*, volume 151 of *LIPIcs*, pages 68:1–68:26, 2020.

39 Boris A. Trakhtenbrot. A survey of russian approaches to perebor (brute-force searches) algorithms. *IEEE Annals of the History of Computing*, 6(4):384–400, 1984.

40 Christopher Umans, Tiziano Villa, and Alberto L. Sangiovanni-Vincentelli. Complexity of two-level logic minimization. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 25(7):1230–1246, 2006.