Simple Circuit Extensions for XOR in PTIME

Marco Carmosino
MIT-IBM Watson AI Lab, Cambridge, MA, USA

Ngu Dang
Boston University, MA, USA

Tim Jackman
Boston University, MA, USA

Abstract

The Minimum Circuit Size Problem for Partial Functions (MCSP^*) is hard assuming the Exponential Time Hypothesis (ETH) (Ilango, 2020). This breakthrough result leveraged a characterization of the optimal $\{\wedge, \vee, \neg\}$ circuits for n-bit OR (OR_n) and a reduction from the partial f-Simple Extension Problem where $f = \mathsf{OR}_n$. It remains open to extend that reduction to show ETH-hardness of total MCSP. However, Ilango observed that the total f-Simple Extension Problem is easy whenever f is computed by read-once formulas (like OR_n). Therefore, extending Ilango's proof to total MCSP would require replacing OR_n with a more complex but similarly well-understood Boolean function.

This work shows that the f-Simple Extension problem remains easy when f is the next natural candidate: XOR_n. We first develop a fixed-parameter tractable algorithm for the f-Simple Extension Problem that is efficient whenever the optimal circuits for f are (1) linear in size, (2) polynomially "few" and efficiently enumerable in the truth-table size (up to isomorphism and permutation of inputs), and (3) all have constant bounded fan-out. XOR_n satisfies all three of these conditions. When $\neg$ gates count towards circuit size, optimal XOR_n circuits are binary trees of $n-1$ subcircuits computing $(\neg)\mathsf{XOR}_2$ (Kombarov, 2011). We extend this characterization when $\neg$ gates do not contribute the circuit size. Thus, the XOR-Simple Extension Problem is in polynomial time under both measures of circuit complexity.

We conclude by discussing conjectures about the complexity of the f-Simple Extension problem for each explicit function f with known and unrestricted circuit lower bounds over the DeMorgan basis. Examining the conditions under which our Simple Extension Solver is efficient, we argue that *multiplexer* functions (MUX) are the most promising candidate for ETH-hardness of a Simple Extension Problem, towards proving ETH-hardness of total MCSP.

2012 ACM Subject Classification Theory of computation → Circuit complexity; Theory of computation → Fixed parameter tractability

Keywords and phrases Minimum Circuit Size Problem, Circuit Lower Bounds, Exponential Time Hypothesis

Digital Object Identifier 10.4230/LIPIcs.STACS.2026.23

Related Version *Full Version*: `https://arxiv.org/abs/2511.16903` [5]

Acknowledgements We would like to thank Rahul Ilango for many invaluable discussions.

1 Introduction

Circuits model the computation of Boolean functions on fixed input lengths by acyclic wires between atomic processing units – logical "gates." To measure the circuit complexity of a function f, we first fix a set of gates $\mathcal{B}$ – called a *basis*. This work studies circuits over the following basis: fan-in 2 AND, fan-in 2 OR, and fan-in 1 NOT gates. We consider two complexity size measures $\mu_{\mathcal{D}}$ and $\mu_{\mathcal{R}}$, which count *only* the binary gates and the *total* number of gates in a circuit respectively. We will refer to $\mathcal{B}$ equipped with these two complexity measures as $\mathcal{D}$, the DeMorgan basis, and $\mathcal{R}$, the Red'kin basis respectively.[1]

Basic questions about these models have been open for decades; we cannot even rule out the possibility that every problem in NP is decided by a sequence of linear-size circuits (see page 564 of [23]). Despite this, the ongoing search for circuit complexity lower bounds has fostered rich and surprising connections between cryptography, learning theory, and algorithm design [14, 10, 40, 6, 38, 16]. The *Minimum Circuit Size Problem* (MCSP, [24]) appears in all of these areas, asking:

> Given an n-input Boolean function f as a 2^n-bit truth table, what is the minimum s such that a circuit of size s computes f ?

The **existential** question – do functions that require "many" gates exist? – was solved in 1949: Shannon proved that almost all Boolean functions require circuits of near-trivial[2] size $\Omega(\frac{2^n}{n})$ by a simple counting argument [43]. The current best answer to the **explicit** question in the DeMorgan basis – is such a hard function in NP? – is a circuit lower bound of $5n - o(n)$, proved via *gate elimination* [22]. This is far from the popular conjecture that NP-complete problems require super-polynomial circuit size.

The **algorithmic** question – is MCSP NP-hard? – remains open after nearly fifty years, despite substantial effort [44]. Under standard cryptographic conjectures (i.e., if efficient and strong pseudorandom generators exist) MCSP is not in P [24]. But NP-hardness seems beyond current techniques: the existence of a "standard" reduction from SAT to MCSP is often enough to imply breakthrough complexity separations *or* unlikely collapses [24, 15]. For instance, a polynomial time many-one reduction from SAT to MCSP would separate EXP from ZPP [34]. To sidestep these barriers, one can try specializing MCSP to natural variants. This approach has been fruitful: DNF-MCSP, MCSP for OR-AND-MOD circuits, and MCSP for multi-output functions are all known to be NP-hard [32, 13, 21]. Alternately, working under a stronger assumption like the Exponential Time Hypothesis (ETH) can avoid barriers by allowing for super-polynomial time (non-standard) reductions. Indeed, MCSP for partial functions (MCSP*) is ETH-hard under deterministic reductions [19].[3] Furthermore, there is formal evidence that proving the correctness of a reduction from SAT to MCSP requires non-relativizing techniques [39]. In Section 1.2, we explain why proving ETH-hardness of MCSP seems much more approachable than proving NP-hardness.

Accordingly, this work investigates the feasibility of generalizing Ilango's technique for ETH-hardness of MCSP* to MCSP, whose hardness under ETH remains wide-open and of substantial interest. Underlying Ilango's proof is a decision problem about circuit complexity of Boolean *simple extensions* which we call the f-Simple Extension Problem (f-SEP).

---

[1] We will specify a basis if a statement pertains to *only* that basis. If the basis is not specified, then the statement applies to both $\mathcal{D}$ and $\mathcal{R}$.

[2] From using a lookup table.

[3] MCSP* was later proven to be NP-hard under *randomized* reductions in [12]. We explain why a similar proof is difficult to generalize for MCSP in Section 1.2.

▶ **Definition** (Simple Extension). *Let f be a Boolean function that depends on all of its n variables. A simple extension of f is either f itself or a function g on $n+m$ variables satisfying:*

1. *g depends on all of its inputs.*
2. *$CC(g)$ – the circuit-size complexity of g – is $CC(f)+m$.*
3. *There exists a setting $k \in \{0,1\}^m$, a* key*, such that for all $x \in \{0,1\}^n$, $g(x,k) = f(x)$.*

We define the f-Simple Extension decision problem for *total functions* below.[4]

▶ **Problem** (The f-Simple Extension Problem). *Let f be a sequence of Boolean functions $\{f_n\}_{n\in\mathbb{N}}$ such that each f_n depends on all of its n inputs. The f-Simple Extension Problem is defined as follows: Given $n \in \mathbb{N}$ and $tt(g)$ – the truth table of a binary function g – decide whether g is a simple extension of f_n.*

For a fixed f whose truth tables can be efficiently computed and whose exact circuit complexity is known, f-SEP reduces to a single call to an MCSP oracle, because checking whether g is a non-degenerate extension of f can be done in polynomial time via brute force (given the truth table of g). This observation gives rise to an MCSP-hardness proof framework: if one can identify an explicit function family f for which deciding the f-Simple Extension Problem is hard, then MCSP is also hard.

This framework was used in Ilango's hardness proof for MCSP* [19], i.e. hardness is shown by reducing an ETH-hard problem to deciding whether a partial function is a simple extension of $f = \mathsf{OR}$. Can one extend this idea to total MCSP and prove MCSP $\notin$ P under ETH? Keeping $f = \mathsf{OR}$, the answer is **no:** optimal circuits for OR are so well-structured that deciding whether a *total* function is a simple extension of OR is actually *easy;* see the discussion in Section 1.2.2 of [19]. Briefly, minimal OR circuits are *read-once formulas*: every input is read exactly once and each internal gate must compute OR_2 and have fan-out 1. Simple extensions of OR will also be computed by read-once formulas, and deciding whether a given Boolean function has a read-once formula is *easy* [2, 11].

Could replacing OR with some read-many f allow Ilango's technique to prove MCSP is ETH-hard? In Section 2, we revisit Ilango's proof and make explicit the role of f-Simple Extension Problem in the ETH-hardness of MCSP*. We find that the proof relies on a deep understanding of the base function's optimal circuits beyond tight size bounds. Therefore,

> *"the missing component in extending our results to* MCSP *is finding some function f whose optimal circuits we can characterize but are also sufficiently complex."*
> – Rahul Ilango, SIAM J. of Computing, 2022

We show that XOR, one of a few read-many functions whose circuits are fully characterized, **cannot** be used to show ETH-hardness for MCSP. For other candidate functions, it is more ambiguous. We discuss prospects for alternative hard functions in Section 1.3.

1.1 Our Results and Contributions

We narrow the field of candidate functions for such a hardness proof by developing a fixed-parameter tractable algorithm for the f-Simple Extension problem (Section 4). Such an algorithm is surprising, because f-Simple Extension is a meta-complexity problem about

[4] For partial function f-Simple Extension (f-SEP*), g is a *partial* function and we must determine whether any completion of g is a simple extension of f.

general circuits and our algorithm works in regimes where we *know* explicit circuit lower bounds. Often, the combinatorial facts used in lower bounds imply a hardness result for the appropriately-restricted meta-complexity problem! Nonetheless, we obtain:

▶ **Main Result.** *The f-Simple Extension Problem is in* P *whenever*
1. *$CC(f)$ – the circuit-size complexity of f – is linear,*
2. *the maximum fan-out over all optimal circuits for f is constant, and*
3. *the optimal[5] circuits for f, up to isomorphism and permutation of its n inputs, are efficiently enumerable and polynomial few with respect to the length of its truth table: 2^n.*

To apply our main result and discount a particular f, we require an exact specification of its optimal circuits. The next natural candidate – XOR, a simple function whose circuits are neither read-once nor monotone – has been well studied. Beyond enjoying exact size bounds [42, 37], XOR is one of the few functions whose structure has been studied; it is known that, in $\mathcal{R}$, *all* optimal XOR_n circuits are binary trees of $n-1$ sub-blocks that compute XOR_2 [26]. We extend this structural analysis to $\mathcal{D}$, obtaining

▶ **Main Lemma** ([26], Lemma 6)**.** *Optimal XOR_n circuits are binary trees of $(n-1)$ sub-circuits that compute $(\neg)\mathsf{XOR}_2$, in both the Red'kin and DeMorgan bases.*

Each XOR_n circuit can therefore be characterized using binary trees with n leaves, of which there are $C_{n-1} = O(2^n)$, where C_n is the n^{th} Catalan number [45]. As there are a finite number of optimal normalized $(\neg)\mathsf{XOR}_2$ circuits, combining this characterization and our main result immediately yields

▶ **Main Corollary.** *The* XOR*-Simple Extension Problem is in* P*.*

Applying Ilango's technique to MCSP will require a deeper study of circuit minimization. This is not merely because any hardness proof needs to bypass our algorithm; knowledge of circuit lower-bounds and optimal constructions for the base function is intrinsic to the reduction itself. We make this connection explicit in Section 2, identifying that

▶ **Main Observation.** *f-SEP* is ETH-hard under Levin reductions.*

That is, the reduction efficiently maps back and forth between `yes` instance witnesses and the set of highly structured optimal circuits which compute the Boolean functions output by the reduction. On the other hand, the reduction yields non-trivial circuit lower-bounds for those truth tables output by `no` instances.

Finally, in Section 1.3, we inspect each explicit function h that enjoys DeMorgan circuit lower bounds and argue how plausible it is that the optimal set of h-circuits avoids our Main Result – a *roadmap* towards ETH-hardness of total MCSP via h-SEP. All proofs are deferred to the full version due to page limits [5].

1.2 Related Work

Proving ETH-hardness of MCSP seems much more approachable than proving NP-hardness: to the best of our knowledge, all known barriers to proving NP-hardness of MCSP do not apply to proving ETH-hardness of MCSP. We survey this work on barriers (which motivates our focus on ETH-hardness) and then cover related work on optimal circuit design.

[5] In $\mathcal{D}$, we require a circuit to be *normalized* for it to be optimal. In particular, it cannot contain any double-negations. This prevents every function from having an infinite number of optimal circuits.

Barriers to NP-hardness of MCSP From Breakthrough Consequences. Resolving the NP-hardness of MCSP using standard reductions implies breakthrough complexity lower bounds that, while believed to be true, seem far beyond current techniques. Kabanets and Cai showed that a "natural" reduction from SAT to MCSP can be used to construct functions in E with super-polynomial circuit complexity [24]. Natural reductions suffice for nearly all known NP-hardness results; they generalize textbook-style gadget reductions. Later, Murray and Williams showed that a polynomial-time many-one reduction from SAT to MCSP implies EXP $\neq$ ZPP [34]. These types of barriers use the efficiency and/or structure of an assumed reduction R from SAT to MCSP to bound the circuit complexity of truth tables printed by R and obtain dramatic consequences.

We can avoid these barriers by aiming instead for ETH-hardness of MCSP, because hardness proofs under the Exponential Time Hypothesis allow for super-polynomial time reductions. Supposing that MCSP is ETH-hard under a reduction R' gives only a *sub-exponential* time bound on R', which may well be super-polynomial. Inspecting the arguments of Kabanets and Cai, natural *super-polynomial time* reductions from SAT to MCSP imply only *linear* circuit lower bounds – which are known via gate elimination! Similarly, a key step in Murray and Williams' argument fails when the many-one reduction requires super-polynomial time. Thus, [24, 34] and similar barriers do not apply to proving ETH-hardness of MCSP by reducing an ETH-hard problem to f-Simple Extension.

Relativization-Like Barriers to NP-hardness of MCSP. More barriers to NP-hardness of MCSP were established by considering either the reduction, the MCSP problem, or both relative to an oracle. For example, *oracle-independent* reductions work for MCSP^A for every oracle A; often they use MCSP only to distinguish between "high" and "low" complexity functions and arise in arguments about pseudorandomness. Hirahara and Watanabe showed that if MCSP is NP-hard under efficient deterministic oracle-independent reductions, then P = NP – so it is natural to conjecture that such reductions are impossible [15]. Later, Ren and Santhanam showed that the complexity of MCSP is not robust in relativized worlds [39]. This suggests that non-relativizing techniques will be required to prove the correctness of a reduction from SAT to MCSP.

We conjecture that aiming for ETH-hardness of MCSP via f-Simple Extension avoids these barriers by combining gate elimination with the circuit complexity of a *specific* and *explicit* function: f. Ren and Santhanam comment that it remains unclear whether relativization is a barrier to the techniques in some recent work in meta-complexity that used gate elimination and related ideas [17, 18, 19, 21] (see Section 1, pg. 2 of [39]). Indeed, many gate elimination arguments prove non-relativizing statements. For example, relativizing DeMorgan circuits to XOR_2 falsifies the lower bounds on XOR. Furthermore, the characterization of a function's optimal circuits seems unlikely to be robust to the addition of oracle gates – avoiding oracle-independence.

NP-Hardness of MCSP*. A breakthrough result of Hirahara shows that *Partial* MCSP is unconditionally NP-hard under *randomized reductions* [12]. Extending this result could yield NP-hardness of total-function MCSP. However, that approach faces a version of the oracle-independence barrier: Hirahara's reduction is in fact many-one and oracle-independent! If there is a many-one oracle-independent randomized reduction from SAT to MCSP, then the polynomial hierarchy (PH) collapses [15]. Because PH is widely conjectured to be infinite and strict, it seems that significantly new ideas would be required to extend Hirahara's argument about MCSP^* to show NP-hardness of total MCSP.

Optimal Circuit Design. Knowing the lower-bounds of some explicit Boolean functions f, a natural question to ask is: *What about the* structure *of every optimal circuit computing f?* For some functions and bases, this is easy to answer: minimal Red'kin circuits for OR_n are binary trees of $\vee$ gates. For even slightly more complex functions, structural characterization seems to require intricate and exhaustive case analysis. Some of the earliest work on this question include [41] and [4] which investigated when optimal circuits for certain 2-output Boolean functions must compute each output independently. More recently, Kombarov extended the characterization of XOR circuits to other complete bases when NOT-gates are counted [27].

1.3 Discussion and Future Directions

A Remark on Bases. Both $\mathcal{R}$ and $\mathcal{D}$ are compatible with Ilango's proof of ETH-hardness of MCSP^*. That proof relied on functions whose optimal $\{\wedge, \vee, \neg\}$-circuits are read-once monotone formulas. There it was irrelevant whether $\neg$ gates contribute to size: those circuits simply did not contain negations. However, when we move to more complex functions like XOR, negations *must* appear in the optimal circuits and as such, we must decide how to treat them.

In the search for non-linear circuit lower bounds, the choice between μ_R and μ_D is largely irrelevant: the two complexity measures the same up to a small constant factor. However, as we discuss in Section 2, Ilango's technique requires knowledge of the *structure* of optimal circuits. In this setting, there is not necessarily as strong connection between the two bases. By extending Kombarov's characterization of XOR circuits in $\mathcal{R}$ to $\mathcal{D}$, we show for that *particular* function, optimal circuits are structurally similar in both bases. But, for other functions, this may well not be the case. Negations could enable a function's optimal circuits to greatly vary under the two complexity measures. Indeed, negations can greatly increase a problems complexity: [3] showed that a Boolean function learning problem became difficult only once the number of $\neg$ gates exceeded a small threshold.

By considering both bases in this work, we limit how negations impact the complexity of f-SEP. We show that they do not greatly increase the complexity of the simple extensions themselves: in Section 4.1, we explain how simple extensions under both complexity measures are highly structured. If a future hardness reduction relies on negations, their role will be in increasing the structural complexity of the underlying base function, either by increasing the number of distinct optimal circuits, or by enabling non-constant fan-out in a base circuit.

A Roadmap for ETH-Hardness Proofs via Simple Extensions. To prove f-SEP is ETH-hard we will need a Boolean function whose optimal circuits are more complex and/or varied than XOR. Specifically, these circuits must either (1) be superlinear in size, (2) require non-constant fanout, or (3) be sufficiently numerous. Superlinear bounds seem beyond current techniques – the most fruitful of which, gate elimination, seems unlikely to be able to prove lower bounds above even $11n$ for wide classes of functions [9]. As such, it seems more sensible to identify Boolean functions which violate the latter conditions. However, structural characterization of the optimal circuits is also hard, and seems to require very tight circuit complexity bounds. Indeed, our DeMorgan basis characterization of XOR repeatedly exploited Schnorr's *exact* $3(n-1)$ bound for XOR_n [42]. This greatly narrows the prospective class of functions from the original specification: "more complex than read-once formulas."

Table 1 Explicit Functions with Circuit Lower Bounds in the DeMorgan Basis.

Function(s)	Lower Bound		Upper Bound	Ω(1) Fanout	Source(s)
XOR	$\mathbf{3(n-1)}$	$=$	$\mathbf{3(n-1)}$	NO	[42]
Sum Mod 4	$4n - O(1)$		$5n - O(1)$	NO?	[46]
Sum Mod 2^k	$4n - O(1)$		$7n - o(n)$	NO?	[46]
Multiplexer	$2(n-1)$		$2n + O(\sqrt{n})$	YES?	[36, 25]
Well-Mixed	$5n - o(n)$		poly	MAYBE?	[28, 22]
Weighted Sum of Parities	$5n - o(n)$		$5n + o(n)$	YES?	[1]

However, the known explicit functions with tight DeMorgan bounds that may violate these conditions are few and far between. Table 1 summarizes these explicit functions to assess how suitable they are for ETH-hardness of f-Simple Extension Problem.[6]

Observe that every function besides XOR in the table has a (small) gap between the circuit lower and upper bound, and the "Ω(1) Fanout" column ends with a question mark (?). This is because XOR is the *only* listed function for which we know the *exact* circuit complexity and an optimal circuit characterization. The other "Ω(1) Fanout" entries above are extrapolated by assuming that their respective DeMorgan *upper bound* constructions are optimal. For instance, Zwick conjectured that optimal circuits computing the Sum Mod 4 (MOD_4) function are "shaped like" ternary full-adder blocks [46]. If this conjecture is true, then MOD_4-Simple Extension can be solved in poly-time since such circuits satisfy the properties of our Main Lemma. Since the MOD_{2^k} functions are computed similarly, we conjecture that exactly characterizing the optimal circuits for Zwick's functions would yield efficient Simple Extension Solvers – not a proof of ETH-hardness for total MCSP.

However, we do have linear lower bounds for functions whose best known constructions have non-constant fanout: the multiplexing function (MUX) contains sub-circuits which are reused a logarithmic number of times [25]. In contrast to XOR however, the bounds for MUX are not tight. The best lower bound is $2(n-1)$, given by Paul [35].

Future Directions. The most obvious next step is to either (1) obtain total characterization of the Multiplexer or (2) extend our Simple Extension Solver to handle circuits with super-constant fanout. Neither of these tasks seems easy, but also they have not been subject to intensive research the way that super-linear circuit lower bounds and hardness of MCSP have. We hope that connecting these kinds of results to ETH-hardness of MCSP provides new perspective and motivation.

Ilango's MCSP^* result has also formed the basis for several hardness results in other models of computation such as formulas and branching programs [20, 7, 8]; could one show that the simple extension problem for formulas or branching programs is hard? These other models of computations may prove easier to work with than unrestricted circuits. They also enjoy superlinear lower bounds thereby bypassing our algorithm. Investigating the simple extension problem in these settings would provide insight into the feasibility of the approach in the circuit setting.

[6] Some of listed bounds are in $\mathcal{U}_2$, the basis consisting of every binary Boolean function besides XOR_2 and $\neg\mathsf{XOR}_2$. For non-degenerate functions besides $f(x) = \neg x$, $\mathcal{U}_2$ and $\mathcal{D}$ are equivalent in terms of size. The multiplexer lower bound of [35] is for the $\mathcal{B}_2$, the basis of all binary Boolean functions, but it also serves as the best known lower bound in $\mathcal{D}$.

We conclude this section with a discussion of the simple extension problem in general. Despite having only been studied as a tool for proving hardness of MCSP thus far, it may be of independent interest. For example, while hardness of time-bounded Kolmogorov complexity is tightly connected to the existence of one-way functions, hardness of MCSP has much weaker quantitative connections [29, 38]. The f-Simple Extension Problem is more "structured" than MCSP and easily reduces to it, so hardness assumptions about f-Simple Extension Problem are stronger. Could such assumptions imply one-way functions?

1.4 Abstract Outline

The remainder of this abstract gives a technical overview of our results and observations.

- Section 2 explores the implicit reduction to f-SEP present in the ETH-hardness proof for MCSP* in [19]. This discussion yields our Main Observation, motivating study of f-SEP and optimal circuit characterization.
- Section 3 sketches our characterization of optimal XOR circuits in the DeMorgan basis.
- Section 4 explains the ideas used by our fixed parameter tractable algorithm for f-SEP.

All proofs are deferred to the full version due to page limits [5].

2 Revisiting ETH hardness for MCSP* via Simple Extensions

We re-examine the proof that MCSP* is ETH-hard from [19]. Our aim is to better understand the reduction from $2n \times 2n$ Bipartite Permutation Independent Set (BPIS) to the Partial f Simple Extension problem (f-SEP*).

2.1 The f-Simple Extension Problem

We give formal definitions of simple extensions and its associated decision problem. We first define non-degeneracy of a Boolean function.

▶ **Definition 1.** *A function $f \in \mathcal{F}_n$, the set of Boolean functions on n variables, depends on its i^{th} variable, x_i, if there exists an input $\alpha \in \mathcal{F}_n$ such that $f(\alpha) \neq f(\alpha \oplus e_i)$, where e_i denotes the Boolean vector that is 0 everywhere except for a 1 at index i and $\oplus$ is bitwise* XOR*. If f depends on all of its variables, then we say f is a* non-degenerate *function. Conversely, we say f is a* degenerate *function if it does not depend on at least one variable.*

We now define simple extension as

▶ **Definition 2.** *Let $f \in \mathcal{F}_n$ be non-degenerate. A* simple extension *of f is either f itself or a function $g \in \mathcal{F}_{n+m}$ satisfying:*

1. *g is a non-degenerate function,*
2. *$CC(g) = CC(f) + m$, and*
3. *there exists a setting $k \in \{0,1\}^m$, called a* key*, such that for all $x \in \{0,1\}^n$, $g(x,k) = f(x)$.*

We denote the first n inputs of f and g by $x_1, \ldots x_n$ and will refer to the extra m inputs of g as *extension variables* and refer to them as $y_1, \ldots, y_m$. From the definition of simple extension above, we define the following decision problem.

▶ **Problem 1** (f-SEP)**.** *Let f be a sequence of Boolean functions $\{f_n\}_{n\in\mathbb{N}}$ such that each f_n is a non-degenerate function in $\mathcal{F}_n$. The f-Simple Extension Problem is defined as follows: Given $n \in \mathbb{N}$ and $tt(g)$ – the truth tables of a binary function $g \in \mathcal{F}_{n+m}$ – decide whether g is a simple extension of f_n.*

We extend this to partial functions,

▶ **Problem 2** (f-SEP*). *Given $n \in \mathbb{N}$ and a* partial *$tt(g)$, decide whether any completion of the truth table is a simple extension of f_n.*

2.2 An Explicit Reduction BPIS from to f-SEP*

Recall the formulation of BPIS from [19],

▶ **Problem 3** (BPIS). *The $2n \times 2n$ Bipartite Permutation Independent Set is defined as follows: Given $G = (V, E)$ a directed graph with vertex set $V = [n] \times [n]$. Decide whether there exists a permutation $\pi : [2n] \to [2n]$ such that:*

1. $\pi([n]) = [n]$,
2. $\pi(\{n + i : i \in [n]\}) = \{n + i : i \in [n]\}$,
3. *if* $((j, k), (j', k')) \in E$, *then either* $\pi(j) \neq k$ *or* $\pi(j' + n) \neq k' + n$

If the Exponential Time Hypothesis holds, then BPIS cannot be solved much faster than brute-forcing over all $2^{o(n \log n)}$ permutations [30]. BPIS is a natural problem to show hardness of circuit size problems since there are $O(2^{s \log s})$ circuits of size s. Reducing from BPIS implies, assuming ETH, that our problem cannot be solved much faster than by brute-forcing over all possible circuits.

The reduction from BPIS to f-SEP*, as it appears as part of the original proof [19], is somewhat *implicit*; the target problem, as written, could be described more simply as "determine whether a partial truth table is ever consistent with a monotone read-once formula." Since this is *easy* for total functions [2, 11], this view of the reduction cannot help us to extend the reduction technique to total MCSP. We will need the more general f-SEP and by reframing Ilango's proof we gain insight into how it might extend to total functions.

The connection to f-SEP* is *explicitly* stated, however, in the introduction of [19]. Here, the reduction maps to OR$_{4n}$-SEP* where each z variable is an extension variable. This is one way to frame the reduction, though non-degenerate functions computed by read-once formulas are simple extensions of *any* of their non-degenerate restrictions. When framing the reduction to f-SEP* explicitly, we find it more compelling to choose a different function $\hat{f}$, whose truth table is given by $\bigvee_{i \in [2n]}(y_i \wedge z_i)$ – because using $\hat{f}$ makes the intuitive description of Ilango's technique "reverse gate elimination" an obvious property of the reduction. Under this framing, the extension variables will instead be the x variables in Ilango's original proof. Despite this, targeting OR$_{4n}$-SEP* is *also* natural; afterwards, we discuss how it compares to $\hat{f}$. Informally, the two choices of base function provide distinct "channels" for ETH to imply hardness of f-SEP*.

Structural Lemmas. To encode BPIS in $\hat{f}$-SEP*, we first prove two lemmas which were essentially proven in tandem in [19] as Lemma 16. We separate them out here and stay faithful to the original arguments. Like Lemma 16, these lemmas establish structural properties of circuits computing our base function $\hat{f}$ and our eventual output $\hat{g}$.

▶ **Lemma 3.** *Let $\hat{f} : \{0,1\}^{2n} \times \{0,1\}^{2n} \to \{0,1\}$ be the Boolean function computed by $\bigvee_{i \in [2n]}(y_i \wedge z_i)$. If ψ be an optimal normalized[7] formula computing $\hat{f}$, then ψ, as a formula, is equal to $\bigvee_{i \in [2n]}(y_i \wedge z_i)$.*

[7] A formula is normalized if all negations are pushed down to the input level. Normalization does not affect the size of the formula, and thus f-SEP* still reduces to MCSP* even if we restrict ourselves to normalized formulas.

In particular, the only difference between two distinct optimal circuits for $\hat{f}$ is which binary tree of fanin 2 $\vee$ gates is used.

Proof. Observe that ψ must read all of its input variables and furthermore must do so positively. This is because (1) f depends on all of its variables and is monotone in them, and (2) ψ is normalized and thus no $\neg$ gates can appear internally in the circuit. We note that ψ must use a total of $4n-1$ gates since f is non-degenerate on $4n$ variables computable. We see that ψ must contain at least $2n-1$ $\vee$ gates: substituting $z = 1^{2n}$ and simplifying yields a monotone read once formula computing $\mathsf{OR}_{2n}(y_1, \ldots, y_{2n})$ which must consist of $2n-1$ $\vee$ gates that appear in ψ.

We now argue that each z_i feeds into an $\wedge$ gate. Assume otherwise, then observe that setting $z_i = 1$ and simplifying removes at least two gates (since $z_1 \vee p \equiv 1 \vee p \equiv 1$). The resulting read once formula for $\hat{f}|_{z_i \leftarrow 1}$ uses at most $4n-3$ gates. This is a contradiction since $\hat{f}|_{z_i \leftarrow 1}$ still depends on all of its remaining $4n-1$ variables: it cannot be computed by a circuit with fewer than $4n-2$ gates.

We now argue the other input to the $\wedge$ gate fed by z_i is y_i. Assume otherwise. Notice that since ψ is read-once, setting $z_i \leftarrow 0$ simplifying disconnects the other input to $\wedge$ gate and thus removes dependence on any variables it depends on. However, $\hat{f}|_{z_i \leftarrow 0}$ still depends on all of its variables besides y_i. Thus the $\wedge$ gate can only read y_i.

Since each z_i and y_i feed a distinct $\wedge$ gate, there are at least $2n$ $\wedge$ gates. Since there are $4n-1$ total gates, and at least $2n-1$ $\vee$ gates, we know that these $\wedge$ gates are the only $\wedge$ gates that appear. Thus the remainder of the circuit is a binary tree of $2n-1$ $\vee$ gates whose $2n$ leaves are the $2n$ $\wedge$ gates. ◀

Having established the structure of optimal circuits computing $\hat{f}$ we can further restrict its simple extensions. Extra restrictions to the truth table enforce that extension variables must be spliced into the circuit using $2n$ additional $\vee$ gates that each read a different y_i. This pairing of each y_i with a different x_j will define a permutation in which we can encode BPIS solutions.

▶ **Lemma 4.** *Let $g : \{0,1\}^{2n} \times \{0,1\}^{2n} \times \{0,1\}^{2n} \to \{0,1\}$ be any simple extension of $\hat{f}$ satisfying the following conditions:*

$$g(x,y,z) = \begin{cases} \hat{f}(y,z) & \text{if } x = 0^{2n} \\ \mathsf{OR}_{2n}(z_1, \ldots z_{2n}) & \text{if } x = 1^{2n} \\ \mathsf{OR}_{4n}(x_1, \ldots x_{2n}, y_1, \ldots y_{2n}) & \text{if } z = 1^{2n} \\ 0 & \text{if } z = 0^{2n} \end{cases}$$

If ϕ is an optimal normalized formula computing g then there exists a permutation $\pi : [2n] \to [2n]$ such that ϕ equals, as a formula, $\bigvee_{i \in [2n]}((x_{\pi(i)} \vee y_i) \wedge z_i)$.

Proof. Since $g(x,y,z) = \hat{f}(y,z)$ when $x = 0^{2n}$, we know that ϕ must read all y and z variables positively. Similarly, all x variables must be read positively, since $g(x,y,1^{2n}) = \mathsf{OR}_{4n}(x_1, \ldots x_{2n}, y_1, \ldots y_{2n})$. Note that these restrictions also imply that ϕ contains exactly $4n-1$ $\vee$ gates and $2n$ $\wedge$ gates since substituting and simplifying yields optimal formulas for those restrictions. From the lemma above, we know that when setting $x = 0^{2n}$ and simplifying, we obtain a circuit structurally equivalent to $\bigvee_{i \in [2n]}(y_i \wedge z_i)$. Therefore $2n$ $\vee$ gates must be removed during simplification. These $\vee$ gates cannot feed any remaining $\vee$ above the $\wedge$ gates, since otherwise setting $x = 1^{2n}$ would fix the circuit to be 1, rather than $\mathsf{OR}_{2n}(z_1, \ldots z_{2n})$. Similarly, z_i cannot feed any of these $\vee$ gates, as setting $x = 1^{2n}$ would remove dependence on that z_i since the circuit is a read-once formula. Thus each $\vee$ gate can

Table 2 BPIS requirements and the corresponding restrictions on $\hat{g}$.

BPIS Requirement on σ	Corresponding $\hat{g}$ Restriction	Impact If π Violates
$\sigma(\{1,\ldots,n\}) = \{1,\ldots,n\}$	$\mathsf{OR}_n(x_1,\ldots,x_n)$ when $z = 1^n0^n$ and $y = 0^{2n}$	If $\pi(i) = j \geq n$, then $z_j \leftarrow 0$ removes x_i in C_π
$\sigma(\{n+1,\ldots,2n\})$ $= \{n+1,\ldots,2n\}$	$\mathsf{OR}_n(x_{n+1},\ldots x_{2n})$ when $z = 0^n1^n$ and $y = 0^{2n}$	As above, $C_\pi\restriction_{z_j \leftarrow 0}$ will not depend on x_i
If $((j,k),(j',k')) \in E$ then $\sigma(j) \neq k$ or $\sigma(n+j') \neq \sigma(n+k')$	1 if $(x,y,z) = (\overline{e_k e_{k'}}, 0^{2n}, e_j e_{j'})$ where $\exists((j,k),(j',k')) \in E$	C_π wrongly outputs 0

only depend on the x and y variables. Observe that each y_i must feed into one of these $\vee$ gates instead of the $\wedge$ gate fed by z_i, as otherwise when we set $x = 1^{2n}$, the function would still depend on y_i. Since there are exactly $2n$ additional $\vee$ gates, and exactly $2n$ y and $2n$ x variables, its easy to see that each additional $\vee$ gate must read one x_i and one y_j. Therefore, as a formula, ϕ must be $\bigvee_{i\in[2n]}((x_{\pi(i)\vee y_i}) \wedge z_i)$ for some permutation π. ◀

An Explicit Reduction. We now provide an explicit reduction from BPIS to $\hat{f}$-SEP*. Given an instance G of BPIS we output $4n$ and the partial truth-table for a function $\hat{g}$ that is consistent with the requirements of Lemma 4. We add three additional restrictions (listed in Table 2) to ensure that any permutation π, whose corresponding circuit $C_\pi \equiv \bigvee_{i\in[2n]}\left((x_{\pi(i)} \vee y_i) \wedge z_i\right)$ is consistent with $\hat{g}$, *is* also a solution for G (and vice versa). All other rows of the truth table are left undefined (e.g. as $\star$). We summarize the requirements for any valid BPIS solution σ, the corresponding restriction, and how it enforces the requirement on π in Table 2.

This completes the reduction as any σ satisfying BPIS for G can be used to construct a read-once circuit consistent with $\hat{g}$ and vice versa. The arguments verifying this are the same as in [19], and we refer the reader there for the full details.

▶ **Lemma 5.** BPIS *reduces to* $\hat{f}$-SEP* *in* $2^{O(n)}$ *time.*

The Original Framing. In its introduction, [19] identifies $\hat{g}$ as a simple extension of OR_{4n}. Under this lens, the hardness comes from determining which OR_{4n} base circuit can have the z extension variables added. The additional truth-table restrictions on $\hat{g}$ force each z_i to be spliced in a particular way adjacent to y_i. Assuming ETH, there are $\Omega(n!)$ optimal base OR_{4n} circuits that must be checked via brute-force.

Implicit Circuit Lower Bounds & Enumeration of Optimal Circuits. From both presentations, we see that leveraging f-SEP involves explicit circuit size lower bounds. Indeed, both Lemma 3 and Lemma 4 prove formula lower bounds for specific non-degenerate functions. However, in a sense, circuit lower bounds are intrinsic to the reduction itself. This connection can be made rigorous: the reduction can used to produce explicit Boolean functions which enjoy non-vacuous lower bounds. On `no` instance of BPIS, the reduction outputs a partial truth table where every completion is non-degenerate but *not* a simple extension. Hence, the circuit complexity of these completions is not the vacuous $6n - 1$ lower bound obtained by knowing that functions produced are non-degenerate.

Furthermore, the reduction did not solely rely on the circuit complexity of $\hat{f}$ and $\hat{g}$. Lemmas 3 and 4 *tightly control* how base circuits and their extensions can be arranged; and this is pivotal for encoding BPIS permutations. This structural requirement can be formalized by observing the reduction is also an efficient *Levin reduction*.

Recall, from [33], that a Levin reduction is a many-one reduction that also efficiently maps witnesses, not just problem instances. More precisely, let R be a set of ordered pairs (x, w) where x is a yes-instance of a problem and w is an accompanying certificate. We define L_R, the language defined by R, to be the set of elements x such that $(x, w) \in R$ for some w. Then a Levin reduction between two languages A and B is an efficient many-one reduction r between problem instances paired with two efficient mappings m, ℓ between instance-witness pairs that satisfy (1) if $(x, w) \in R_A$ then $(r(x), m(x, w)) \in R_B$ and (2) $(t(x), w) \in R_B$ implies $(x, \ell(x, w)) \in R_A$.

For BPIS, the witnesses for an instance are simply the valid permutations σ and witnesses for $\hat{f}$-SEP are optimal circuits computing the extension. Let $\mathcal{R}_{\mathsf{BPIS}}$ and $\mathcal{R}_{\hat{f}\text{-}\mathsf{SEP}^*}$ be the sets of ordered pairs consisting of problem instances and all of their witnesses as described. The reduction admits linear time mappings between witnesses: given σ, simply construct $\bigvee_{i\in[2n]}((x_{\sigma(i)} \vee y_i) \wedge z_i)$ and given a circuit for $\hat{g}$, simply read off the permutation from the x variables.

3 The Structure of Optimal XOR Circuits

Similar to the characterization of optimal XOR circuits over the Red'kin basis $\mathcal{R}$ [26], we show:

▶ **Lemma 6.** Every *optimal* $(\neg)\mathsf{XOR}_n$ *circuit over the DeMorgan basis* $\mathcal{D}$ *partitions into a binary tree of* $(n-1)$ *sub-circuits computing* $(\neg)\mathsf{XOR}_2$ *– even when NOT gates are free.*

The structure of optimal circuits computing the XOR-function is a crucial ingredient for ruling it out as a candidate function. We carry out an elementary but intricate case analysis of restricting and eliminating gates from optimal XOR circuits. Essentially we extract more information from the proof of Schnorr's lower bound by using it to identify "templates" that must be found in *any* optimal XOR circuit. We push this process to the limit, fully characterizing the "shape" of all such circuits. Specifically,

- Schnorr's proof is essentially a technical lemma which says that any one-bit restriction will eliminate at least 3 costly gates [42]. This means that at the bottom level of every optimal XOR circuit, any variable must be fed into two distinct costly gates, and furthermore, one of these two must be fed into another costly gate. Any deviation from these properties will violate essential properties of the XOR-function, such as "XOR depends on all the input bits." Via a basic inductive argument and the fact that XOR is downward self-reducible, Schnorr's lower bound follows: $CC(\mathsf{XOR}_n) \geq 3(n-1)$.
- Schnorr's proof leaves the local structure of the optimal circuit computing XOR "open." Namely, it does not provide any information about the other inputs of the costly gates or where their outputs connect to the rest of the circuit, since we consider fan-in 2 and unbounded fan-out. However, we know that XOR circuit has a matching upper-bound of $3(n-1)$. In particular, this means *each one-bit restriction cannot remove more than* 3 *gates.* We also know that *each variable in optimal* XOR*-circuits must be read twice.*
- We leverage these two properties to show that in every optimal XOR circuit, any two distinct input variables x_i and x_j must be fed into a block $\mathcal{B}$ as shown in the left sub-figure of Figure 1. Specifically, we argue that any deviations from the block will violate at least one of the properties via exhaustive case analyses of gate elimination steps. Finally, we argue that this block $\mathcal{B}$ must compute either XOR_2 or $\neg\mathsf{XOR}_2$ and apply a basic inductive argument to obtain the desired structural characterization of any optimal circuit computing XOR_n as depicted in the right sub-figure of Figure 1.

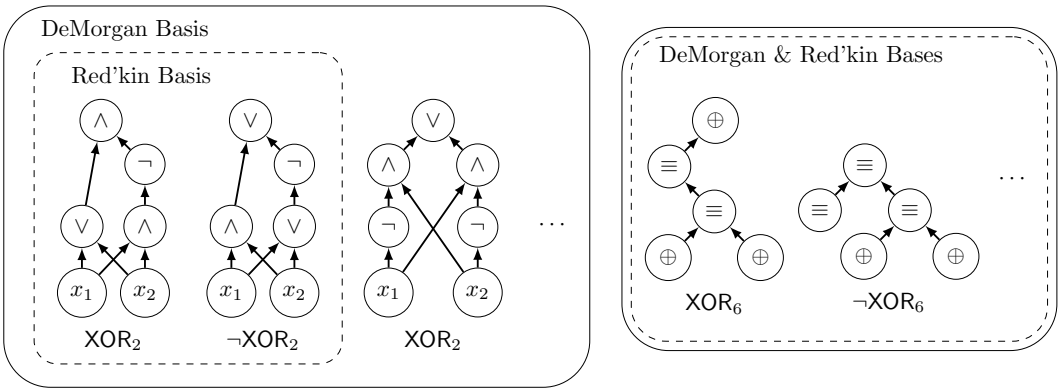


Figure 1 An example of the binary tree structure of optimal circuits computing XOR_6. The left sub-figure depicts possible $(\neg)\mathsf{XOR}_2$ blocks in the Red'kin and DeMorgan Bases. Notice each optimal Red'kin circuit is an optimal DeMorgan circuit, but not vice-versa. The right sub-figure depicts that the arrangement of XOR_2 blocks that make up XOR_6 circuits are shared by both bases.

Besides the linear size for optimal circuits computing XOR, our structural theorem yields two more properties that rule out XOR as a candidate function for MCSP-hardness via Simple Extension. That is, for optimal circuits computing XOR_n, (1) *the maximum fan-out is a constant*, and (2) *the number of such optimal circuits up to permutation of variables, is* $2^{O(n)}$.

4 A Fixed-Parameter Tractable Simple Extension Solver

It is easy to see that the approach of solving the Simple Extension Problem for XOR_n via brute-forcing over all possible circuits of size $CC(f) + m$ is super-polynomial in terms of the length of the input truth-tables. Using the following ingredients, we design Algorithm 1, a Fixed-Parameter Tractable (FPT) algorithm for the Simple Extension problem that depends on the following three parameters: (1) the number of optimal circuits for f (up to isomorphism & permutation of variables), (2) the maximum fanout of any node in any optimal circuit for f, and (3) $CC(f)$.

Algorithm 1 Informal Simple Extension Solver, taking input $n \in \mathbb{N},\ g \in \mathcal{F}_{n+m}$.

if there is no key to f in g or g is degenerate **then**
    return False
▷ If the above tests pass, then g is a non-degenerate extension of f. It remains to check simplicity.
for each isomorphism class $\mathcal{C}$ of optimal circuits for f **do**
    $F \leftarrow$ an arbitrary element of $\mathcal{C}$ with all gates labeled in topological order
    label the open nodes of F by an arbitrary permutation of $x_1, \ldots, x_n$
    for each reverse elimination E that adds exactly m costly gates to F **do**
        $\tilde{G} \leftarrow \texttt{Decode}(F, E)$
        if $\texttt{tt}(\tilde{G}) \simeq \texttt{tt}(g)$ **then** ▷ Test using the procedure of Theorem 9.
            return True
return False

4.1 Structured Simple Extension Circuits

By analyzing the behavior of optimal simple extension circuits under gate elimination, we can characterize the structure of *every* optimal circuit computing a simple extension. By definition, if g is a simple extension of f then there are restrictions of g's added variables (called *extension* variables and denoted y_i) that yield f. We call such a restriction a *key* to f in g. We first show that *circuits obtained by partially restricting with a key are themselves optimal simple extension circuits for intermediate extensions.* Building on this, we then develop convenient *all-stops restrictions* that order substitutions and simplification steps with the following properties: (1) *single-bit substitutions from this key in the given order eliminate exactly one costly gate at each step*, (2) *there exists such an all-stops restriction for any optimal circuit computing a simple extension.*

Combining these tools, we inductively show a robust structure arises in optimal simple extension circuits: *each extension variable occurs in an isolated read-once subformula that depends only on other extension variables* (referred to as the *Y-trees*). Formally,

▶ **Definition 7** (Y-Tree Decomposition)**.** *Let G be a circuit with two distinguished sets of inputs:* base variables X *and* extension variables Y*. A* Y-Tree Decomposition *of G is a set of triples $\langle \gamma, b, T \rangle$ where γ – referred to as a* combiner *– is a costly gate of G, bit $b \in \{0, 1\}$ designates an input of γ, and T is a sub-circuit of G rooted at the b child of γ such that*

1. *Each T is a read-once formula in only extension variables Y.*
2. *Each $y_i \in Y$ appears in at most one T.*
3. *Each T is* isolated *in G – gate γ is the unique gate reading from T, and it only reads the root of T.*
4. *The sub-circuit of G rooted at the $\neg b$ child of γ contains at least one X variable.*

The size *of a decomposition is the number of tuples – Y-trees and their associated combiner gates – present in the decomposition. The* weight *of a Y-tree decomposition is the number of extension variables that are read in some T. We say a Y-tree decomposition is* total *if its weight is $|Y|$, i.e. every extension variable appears. An example Y-tree decomposition of size three is depicted in Figure 2, where the shaded circles represent the circuity around each combiner γ connecting each $T_{\mathcal{Y}}$ to the rest of the circuit.*

When gate elimination is performed with a total key, these added Y-trees and their combiners are pruned to reveal an embedded optimal circuit for the base f function. We get the following structural insight:

▶ **Theorem 8.** *If G is a minimal circuit for a simple extension, then G has a total Y-tree decomposition.*

Y-tree decomposition forms the basis of our strategy: we brute force over every optimal base circuit and try to "splice in" every possible Y-tree – literally "reversing" the process of gate elimination with a key to f in g.

4.2 Encoding & Decoding the "Grafts" in a Y-Tree Decomposition

To efficiently search over candidate simple extension circuits, we devise an encoding scheme and corresponding decoding algorithm which efficiently captures the difference in local neighborhoods after each new Y-tree is spliced on top of an existing gate or input. Our final encoding must be $O(n + m)$ bits long to ensure brute-force runs in $2^{O(n+m)}$. We present our encoding as a communication problem to clarify the overhead and constraints involved.

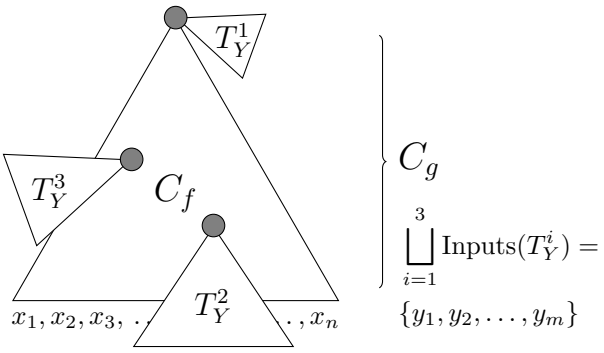
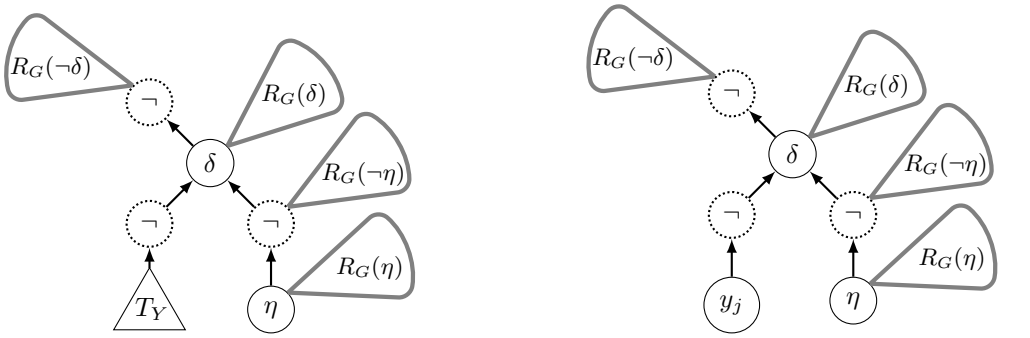


Figure 2 An example of a Y-Tree Decomposition of size three.



(a) Before any extension variable restrictions.

(b) All extension variables restricted except y_j.

Figure 3 The local neighborhood of a combiner δ and Y-tree $T_{\mathcal{Y}}$ grafted on an origin η.

Suppose g is a simple extension of f and Alice knows G, an optimal circuit for g. Alice can obtain an optimal circuit F computing f by simply restricting the y-variables of G with a key and performing gate elimination. Now consider the following communication problem: Bob (i.e., line 5 of Algorithm 1) knows F, and Alice would like to send him G using as few bits as possible. Because g is a simple extension of f, Alice can compute the Y-tree decomposition of optimal circuit G. The idea is to send Bob a sequence of instructions that tell him exactly how to graft each Y-Tree of G onto the gates of F, where all information is encoded relative to isomorphism-invariant properties of F.

We begin by illustrating the "grafting" idea for the algorithm. First consider the case where there is only a single Y-tree $T_{\mathcal{Y}}$ in the decomposition of G. Then there must be some costly gate η – an *origin* already present in F – that is combined with $T_{\mathcal{Y}}$ in G. Our decomposition theorem shows that every possible arrangement of this *graft* is depicted in Figure 3, which shows the local neighborhood[8] of the single Y-tree in G.

In Figure 3, both δ and η represent costly gates that *must* be present, the dotted negations represent NOT gates that *may* be present, and the cones represent connections to the rest of G. The notation $R_C(\beta)$ means "the set of costly gates that read gate β in circuit C." These sets are depicted by shaded cones, because at least one of them must be non-empty (otherwise, G would not be an optimal circuit) but we do not know which. Note how the Y-tree and potential negation atop it are *isolated* from the rest of the circuit, except for

[8] Think of this as "zooming in" to inspect one of the shaded circles in Figure 2.

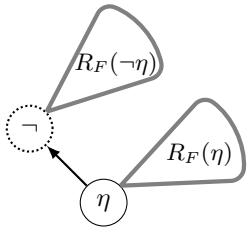


Figure 4 The origin η after the Y-tree has been pruned.

connections via the *combiner* gate δ. We will describe in some detail how this single graft can be transmitted and sketch the issues involved in efficiently coding *all* grafts and Y-trees for Bob.

First, Alice applies gate elimination to G to eliminate all but one of the y-variables in $T_{\mathcal{Y}}$. Because $T_{\mathcal{Y}}$ is an isolated formula in G, a single variable y_j is left in G at the end of this process: the local neighborhood transforms from Figure 3a to Figure 3b. Crucially, all gates outside $T_{\mathcal{Y}}$ are unaffected. Now, Alice runs a final step of gate elimination, substituting y_j according to a key to f in g. The result will be as depicted in Figure 4: the local neighborhood of η in F. The key observation is that *Bob has perfect knowledge of this structure* – it is simply a sub-circuit of F. If Alice can send (1) an identifier for η (2) a "diff" between the neighborhood of η in F compared to G (Fig. 3b vs. 4) and (3) a description of $T_{\mathcal{Y}}$ this would suffice for Bob to efficiently transform F into G.

The most straightforward protocol would send $O(\log(\#\mathsf{gates}(F)))$ bits to identify η for Bob. This is too expensive if there is more than one Y-tree in G. Recall that we can only tolerate runtime of $2^{O(n+m)}$ and intend to brute-force all possible codes. There could be as many as m Y-trees with a single variable each and thus m origins, so brute-forcing this simple code would require time $2^{O(m\log(n)+n)}$ for $\#\mathsf{gates}(F) = O(n)$ – super-polynomial in $2^{(n+m)}$ and therefore unacceptable.

Instead, Alice can send a $\#\mathsf{gates}(F)$-bit origin indicator vector χ, where χ_i is 1 if gate i of F is the origin of some Y-tree. It is feasible to brute-force these vectors in $2^{O(n)}$-time for any possible number of origins given a linear upper bound on the circuit size of f. Returning to the case where G has a single Y-tree, Bob identifies η by reading the single 1-bit of χ. For the local neighborhood of η, the following information must be transmitted to summarize the "diff" between F and G:

1. The costly functions computed by gates η and δ,
2. presence or absence of each possible NOT gate depicted in Figure 3b,
3. whether η is connected to the left or right input of δ,
4. the description of $T_{\mathcal{Y}}$, and
5. the sets of gates that read from each individual element of the graft, $R_G(\cdot)$.

Items 1 - 3 can be described by a constant-length bitstring, depending only on the enumeration of all possible "graft" sub-circuits implied by our characterization of gate elimination. We code the exact Y-Tree $T_{\mathcal{Y}}$ (item 4) explicitly for now and eliminate it in Section 4.3. Here, we are concerned with how to efficiently code the sets $R_G(\cdot)$ without sending explicit "pointers" to nodes of F, which remain too expensive per the discussion of transmitting η above.

To code these wire movements efficiently, Alice will exploit the relationship between the gates reading from η in F and the gates reading from η in G as determined by gate elimination. Bob knows the contents of $R_F(\eta)$ and $R_F(\neg\eta)$ – the sets of costly gates reading from $(\neg)\eta$ in F. Collect the gates that read from the graft, in both G and F:

$$R_G = R_G(\eta) \cup R_G(\neg\eta) \cup R_G(\delta) \cup \mathsf{R}_G(\neg\delta)$$
$$R_F = R_F(\eta) \cup R_F(\neg\eta)$$

Observe that $R_G \subseteq R_F$ because during the single run of gate elimination that transforms Figure 3b into Figure 4, all the wires reading the eliminated gate δ are "inherited" by η. Furthermore, *Bob knows η from χ and thus can reconstruct R_F*, so Alice can identify any wire relevant to the graft by coding "the j-th element of R_F." Here we must apply the assumption that optimal circuits for f have at most constant fanout, independent of n, to bound the length of these *relative* wire-identifiers by a constant. Thus, Alice can identify *exactly which* wires should be moved from η to read $(\neg)\delta$ in G. Inspecting Figure 3b again, she can also code *where* they move using constantly many bits, because there are only two options: reading from $\neg\delta$ or δ directly.

This discussion has outlined the base case of our encoding/decoding argument, where only a single Y-tree is transmitted to Bob. Notice that, if there are multiple *combiner-disjoint* Y-trees in G, an essentially identical strategy could encode all these Y-trees. However, reverse gate elimination can create somewhat more complex structures: specifically, we can have combiners δ_i grafted onto *a distinct combiner* δ_j, instead of a gate η that was present in the original circuit F. Even so, Alice can use the fact that every combiner δ_i has a *unique* origin η to identify the "first" combiner grafted onto η and instruct Bob to add subsequent η-derived combiners in depth-respecting order. See the full version for the precise statement and proofs about the grafting procedure [5].

4.3 Speeding Up Via Truth-table Isomorphism

Brute-forcing over the graft encodings described above is still catastrophically inefficient: we must eliminate the explicit description of Y-trees. Since each Y-tree is a read-once formula in the added m variables there are least $C_{m-1} \cdot m!$ such explicit Y-trees, where C_a is the a^{th} Catalan number [45]. The dominating term – $m!$ – comes from permuting the labels of the variables. The same issue arises if our base function f is symmetric: the number of optimal f circuits is $\Omega(n!)$.

We sidestep this issue and drastically improve the speed of brute-force search. If we "incorrectly" assign variables in the base circuit or in the Y-trees, the result is a circuit for a Boolean function that is *truth-table isomorphic* to g, i.e. their truth-tables are the same up to a permutation of the inputs. If h and g are truth-table isomorphic then they have the same circuit complexity. Thus it suffices to brute-force over unlabeled ("open") base circuits and Y-trees, assign variables to inputs arbitrarily, generate each circuit's truth table, and check if it is truth-table isomorphic to g. This final step is feasible, because truth-table isomorphism testing is efficient.

▶ **Theorem 9** (Corollary 1.3 of [31]). *Given the truth-tables for two Boolean functions, testing whether they are equivalent under permutation of variables can be done in time $c^{O(n)}$ where c is a constant.*

This results in our final algorithm, which runs in time $O(|L| \cdot 2^{O(\ell(s+m))})$ where $|L|$ is the number of optimal base f circuits up to permutation of variables, ℓ is the maximum fanout in any of those base circuits, $s = CC(f)$, and the asymptotic notation hides encoding overhead from grafting. As discussed above, for XOR these parameters are all sufficiently small and hence XOR-Simple Extension is in P.

References

1 Kazuyuki Amano and Jun Tarui. A well-mixed function with circuit complexity 5n: Tightness of the lachish-raz-type bounds. *Theor. Comput. Sci.*, 412(18):1646–1651, 2011. doi:10.1016/J.TCS.2010.12.040.

2 Dana Angluin, Lisa Hellerstein, and Marek Karpinski. Learning read-once formulas with queries. *J. ACM*, 40(1):185–210, 1993. doi:10.1145/138027.138061.

3 Eric Blais, Clément L. Canonne, Igor C. Oliveira, Rocco A. Servedio, and Li-Yang Tan. Learning circuits with few negations. In Naveen Garg, Klaus Jansen, Anup Rao, and José D. P. Rolim, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2015, Princeton, NJ, USA, August 24-26, 2015*, volume 40 of *LIPIcs*, pages 512–527. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2015. doi:10.4230/LIPIcs.APPROX-RANDOM.2015.512.

4 Norbert Blum and Martin Seysen. Characterization of all optimal networks for a simultaneous computation of AND and NOR. *Acta Informatica*, 21:171–181, 1984. doi:10.1007/BF00289238.

5 Marco Carmosino, Ngu Dang, and Tim Jackman. Simple circuit extensions for XOR in PTIME, 2025. doi:10.48550/arXiv.2511.16903.

6 Mahdi Cheraghchi, Valentine Kabanets, Zhenjian Lu, and Dimitrios Myrisiotis. Circuit lower bounds for mcsp from local pseudorandom generators. *ACM Transactions on Computation Theory (TOCT)*, 12(3):1–27, 2020. doi:10.1145/3404860.

7 Ludmila Glinskih and Artur Riazanov. MCSP is hard for read-once nondeterministic branching programs. In Armando Castañeda and Francisco Rodríguez-Henríquez, editors, *LATIN 2022: Theoretical Informatics - 15th Latin American Symposium, Guanajuato, Mexico, November 7-11, 2022, Proceedings*, volume 13568 of *Lecture Notes in Computer Science*, pages 626–640. Springer, 2022. doi:10.1007/978-3-031-20624-5_38.

8 Ludmila Glinskih and Artur Riazanov. Partial Minimum Branching Program Size Problem is ETH-hard. In Raghu Meka, editor, *16th Innovations in Theoretical Computer Science Conference (ITCS 2025)*, volume 325 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 54:1–54:22, Dagstuhl, Germany, 2025. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ITCS.2025.54.

9 Alexander Golovnev, Edward A. Hirsch, Alexander Knop, and Alexander S. Kulikov. On the limits of gate elimination. *J. Comput. Syst. Sci.*, 96:107–119, 2018. doi:10.1016/j.jcss.2018.04.005.

10 Alexander Golovnev, Rahul Ilango, Russell Impagliazzo, Valentine Kabanets, Antonina Kolokolova, and Avishay Tal. $\mathrm{Ac}^0[\mathrm{p}]$ lower bounds against MCSP via the coin problem. In Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, editors, *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, Patras, Greece, July 9-12, 2019*, volume 132 of *LIPIcs*, pages 66:1–66:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ICALP.2019.66.

11 Martin Charles Golumbic, Aviad Mintz, and Udi Rotics. Factoring and recognition of read-once functions using cographs and normality and the readability of functions associated with partial k-trees. *Discrete Applied Mathematics*, 154(10):1465–1477, 2006. doi:10.1016/j.dam.2005.09.016.

12 Shuichi Hirahara. Np-hardness of learning programs and partial MCSP. In *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022, Denver, CO, USA, October 31 - November 3, 2022*, pages 968–979. IEEE, 2022. doi:10.1109/FOCS54457.2022.00095.

13 Shuichi Hirahara, Igor C. Oliveira, and Rahul Santhanam. Np-hardness of minimum circuit size problem for OR-AND-MOD circuits. In Rocco A. Servedio, editor, *33rd Computational Complexity Conference, CCC 2018, June 22-24, 2018, San Diego, CA, USA*, volume 102 of *LIPIcs*, pages 5:1–5:31. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.CCC.2018.5.

14 Shuichi Hirahara and Rahul Santhanam. On the average-case complexity of MCSP and its variants. In Ryan O'Donnell, editor, *32nd Computational Complexity Conference, CCC 2017, Riga, Latvia, July 6-9, 2017*, volume 79 of *LIPIcs*, pages 7:1–7:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.CCC.2017.7.

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

16 Yizhi Huang, Rahul Ilango, and Hanlin Ren. Np-hardness of approximating meta-complexity: A cryptographic approach. *SIAM J. Comput.*, 54(4):819–886, 2025. doi:10.1137/23M1608483.

17 Rahul Ilango. Approaching MCSP from above and below: Hardness for a conditional variant and AC^0[p]. In *11th Innovations in Theoretical Computer Science Conference (ITCS 2020)*, pages 34:1–34:26. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ITCS.2020.34.

18 Rahul Ilango. Connecting perebor conjectures: Towards a search to decision reduction for minimizing formulas. In *35th Computational Complexity Conference (CCC 2020)*, pages 31:1–31:35. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020.

19 Rahul Ilango. Constant depth formula and partial function versions of mcsp are hard. *SIAM Journal on Computing*, 0(0):FOCS20–317–FOCS20–367, 2020. doi:10.1137/20M1383562.

20 Rahul Ilango. The minimum formula size problem is (ETH) hard. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*, pages 427–432. IEEE, 2021. doi:10.1109/FOCS52979.2021.00050.

21 Rahul Ilango, Bruno Loff, and Igor Carboni Oliveira. Np-hardness of circuit minimization for multi-output functions. In *Electronic Colloquium on Computational Complexity (ECCC)*, volume 27, page 21, 2020.

22 Kazuo Iwama and Hiroki Morizumi. An explicit lower bound of 5n - o(n) for boolean circuits. In Krzysztof Diks and Wojciech Rytter, editors, *Mathematical Foundations of Computer Science 2002, 27th International Symposium, MFCS 2002, Warsaw, Poland, August 26-30, 2002, Proceedings*, volume 2420 of *Lecture Notes in Computer Science*, pages 353–364. Springer, 2002. doi:10.1007/3-540-45687-2_29.

23 Stasys Jukna. *Boolean Function Complexity - Advances and Frontiers*, volume 27 of *Algorithms and combinatorics*. Springer, 2012. doi:10.1007/978-3-642-24508-4.

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

25 Klein and Paterson. Asymptotically optimal circuit for a storage access function. *IEEE Transactions on Computers*, C-29(8):737–738, 1980. doi:10.1109/TC.1980.1675657.

26 Yu A Kombarov. The minimal circuits for linear boolean functions. *Moscow University Mathematics Bulletin*, 66(6):260–263, 2011.

27 Yu A Kombarov. Complexity and structure of circuits for parity functions. *Journal of Mathematical Sciences*, 233:95–99, 2018.

28 Oded Lachish and Ran Raz. Explicit lower bound of 4.5n - o(n) for boolena circuits. In *Proceedings of the Thirty-Third Annual ACM Symposium on Theory of Computing*, STOC '01, pages 399–408, New York, NY, USA, 2001. Association for Computing Machinery. doi:10.1145/380752.380832.

29 Yanyi Liu and Rafael Pass. On one-way functions and kolmogorov complexity. In Sandy Irani, editor, *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 1243–1254. IEEE, 2020. doi:10.1109/FOCS46700.2020.00118.

30 Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Slightly superexponential parameterized problems. *SIAM Journal on Computing*, 47(3):675–702, 2018. doi:10.1137/16M1104834.

31 Eugene M Luks. Hypergraph isomorphism and structural equivalence of boolean functions. In *Proceedings of the thirty-first annual ACM symposium on Theory of computing*, pages 652–658, 1999. doi:10.1145/301250.301427.

32 William J Masek. Some np-complete set covering problems. *Unpublished manuscript*, 1979.

33 Noam Mazor and Rafael Pass. Gap MCSP is not (levin) np-complete in obfustopia. In Rahul Santhanam, editor, *39th Computational Complexity Conference, CCC 2024, July 22-25, 2024, Ann Arbor, MI, USA*, volume 300 of *LIPIcs*, pages 36:1–36:21. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPIcs.CCC.2024.36.

34 Cody D. Murray and Richard Ryan Williams. On the (non) np-hardness of computing circuit complexity. In David Zuckerman, editor, *30th Conference on Computational Complexity, CCC 2015, Portland, Oregon, USA, June 17-19, 2015*, volume 33 of *LIPIcs*, pages 365–380. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2015. doi:10.4230/LIPIcs.CCC.2015.365.

35 Wolfgang J. Paul. A 2.5 n-lower bound on the combinational complexity of boolean functions. In *Proceedings of the Seventh Annual ACM Symposium on Theory of Computing*, STOC '75, pages 27–36, New York, NY, USA, 1975. Association for Computing Machinery. doi:10.1145/800116.803750.

36 Wolfgang J. Paul. A 2.5 n-lower bound on the combinational complexity of boolean functions. *SIAM J. Comput.*, 6(3):427–443, 1977. doi:10.1137/0206030.

37 NP Red'kin. Proof of minimality of circuits consisting of functional elements. *Systems Theory Research: Problemy Kibernetiki*, pages 85–103, 1973.

38 Hanlin Ren and Rahul Santhanam. Hardness of KT characterizes parallel cryptography. In Valentine Kabanets, editor, *36th Computational Complexity Conference, CCC 2021, Toronto, Ontario, Canada (Virtual Conference), July 20-23, 2021*, volume 200 of *LIPIcs*, pages 35:1–35:58. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.CCC.2021.35.

39 Hanlin Ren and Rahul Santhanam. A relativization perspective on meta-complexity. In Petra Berenbrink and Benjamin Monmege, editors, *39th International Symposium on Theoretical Aspects of Computer Science, STACS 2022, Marseille, France (Virtual Conference), March 15-18, 2022*, volume 219 of *LIPIcs*, pages 54:1–54:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.STACS.2022.54.

40 Rahul Santhanam. Pseudorandomness and the minimum circuit size problem. In Thomas Vidick, editor, *11th Innovations in Theoretical Computer Science Conference, ITCS 2020, Seattle, Washington, USA, January 12-14, 2020*, volume 151 of *LIPIcs*, pages 68:1–68:26. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ITCS.2020.68.

41 Jürgen Sattler. Netzwerke zur simultanen berechnung boolescher funktionen. In Peter Deussen, editor, *Theoretical Computer Science, 5th GI-Conference, Karlsruhe, Germany, March 23-25, 1981, Proceedings*, volume 104 of *Lecture Notes in Computer Science*, pages 32–40. Springer, 1981. doi:10.1007/BFB0017293.

42 Claus-Peter Schnorr. Zwei lineare untere schranken für die komplexität boolescher funktionen. *Computing*, 13(2):155–171, 1974. doi:10.1007/BF02246615.

43 Claude. E. Shannon. The synthesis of two-terminal switching circuits. *The Bell System Technical Journal*, 28(1):59–98, 1949. doi:10.1002/j.1538-7305.1949.tb03624.x.

44 B. A. Trakhtenbrot. A survey of russian approaches to perebor (brute-force searches) algorithms. *Annals of the History of Computing*, 6(4):384–400, 1984. doi:10.1109/MAHC.1984.10036.

45 J. H. van Lint and R. M. Wilson. *Recursions and generating functions*, pages 109–1311. Cambridge University Press, 1992.

46 Uri Zwick. A 4n lower bound on the combinational complexity of certain symmetric boolean functions over the basis of unate dyadic boolean functions. *SIAM Journal on Computing*, 20(3):499–505, 1991. doi:10.1137/0220032.