Beating Brute Force for (Quantified) Satisfiability of Circuits of Bounded Treewidth

Daniel Lokshtanov[*] Ivan Mikhailin [†] Ramamohan Paturi [‡] Pavel Pudlak[§]

October 31, 2017

Abstract

We investigate the algorithmic properties of circuits of bounded treewidth. Here the treewidth of a circuit C is defined as the treewidth of the underlying undirected graph of C, after the vertices corresponding to input gates have been removed. Thus, boolean formulae correspond to circuits of treewidth 1.

- Our first main result is an algorithm for counting the number of satisfying assignments of circuits with n input gates, treewidth ω, and at most $s \cdot n$ gates. The running time of our algorithm is $2^{n(1-\frac{1}{O(s\cdot\omega\cdot4^{\omega})})}$, which for formulae instantiates to $2^{n(1-1/O(s))}$. This is the first algorithm to achieve exponential speed-up over brute force for the satisfiability of linear size circuits with treewidth bounded by a constant greater than 1. For treewidth 1, i.e., boolean formulae, our algorithm significantly outperforms the previously fastest $2^{n(1-1/O(s^2))}$ time satisfiability algorithm by Santhanam [32].
- Our second main result is an algorithm for True Quantified Boolean Circuit Satisfiability for circuits of treewidth ω, in which every input gate has fan-out at most s. The running time of our algorithm is $2^{n(1-\frac{1}{O(s\cdot\omega\cdot4^{\omega})})}$. Our algorithm is the first to achieve exponential speed-up over brute force for such circuits. Indeed, even for quantified boolean formulae where every variable appears at most s times, the previously best known algorithm by Santhanam [32] has running time $2^{n(1-1/O(f(s)\cdot\log n))}$.
- Utilizing the structural properties of low treewidth circuits which helped us obtain improved exponential-time algorithms for satisfiability, we also show that the number of wires of any constant treewidth circuit that computes the majority function must be super-linear.

1 Introduction

Satisfiability testing is both a canonical **NP**-complete problem [10, 24] and one of the most successful general approaches to solving real-world constraint satisfaction problems. Optimized CNF-SAT heuristics are used to address a variety of combinatorial search problems successfully in practice, such as circuit and protocol design verification. Algorithms for the satisfiability problem are also central to complexity theory, as demonstrated by Williams [39], who showed that any improvement (by even a superpolynomial factor compared to exhaustive search) for the satisfiability problem for general circuits implies circuit lower bounds. Furthermore he has

[*]Department of Informatics, University of Bergen, Norway, e-mail: `daniello@ii.uib.no`

[†]Department of Computer Science and Engineering, University of California, San Diego, La Jolla, CA 92093, USA, e-mail:`imikhail@eng.ucsd.edu`

[‡]Department of Computer Science and Engineering, University of California, San Diego, La Jolla, CA 92093, USA, e-mail:`paturi@cs.ucsd.edu`

[§]Institute of Mathematics of the Czech Academy of Sciences, 115 67 Praha 1, Czech Republic, e-mail: `pudlak@math.cas.cz`

successfully used the connection to prove superpolynomial size bounds for $\mathbf{ACC}^0$ circuits using a novel satisfiability algorithm for $\mathbf{ACC}^0$ circuits, solving a long standing open problem [40].

This raises the questions: For which circuit models do improved (over exhaustive search by at least a superpolynomial factor) satisfiability algorithms exist? How does the amount of improvement over exhaustive search relate to the expressive power of the model (and hence to lower bounds)? Can satisfiability heuristics for stronger models than CNF be useful for real-world instances?

A considerable amount of research has gone into devising improved algorithms for checking the satisfiability of various circuit models up to depth $o(\log n/\log\log n)$ (see, for example, [26, 35, 30, 29, 36, 5, 21, 27, 18, 40, 19, 6, 20, 9, 25]), although the question of improved algorithms for checking the satisfiability of linear size bounded depth threshold circuits is still unresolved. The improvements range from superpolynomial factors to exponential factors depending on the circuit model, size and depth. For *unrestricted depth* circuits, the progress is minimal. Improved algorithms have only been known for formulas over Boolean and the complete basis [32, 37, 8].

Our general goal is to move beyond formulas to obtain improved algorithms for unrestricted depth circuits. It is a tantalizing question whether linear size circuits of unrestricted depth (even of logarithmic depth) admit satisfiability algorithms with a superpolynomial speedup over exhaustive search, the possibility of which has consequences for superlinear circuit lower bounds. All the known improved algorithms for satisfiability are such that the 'improvement' decreases exponentially in depth so that we get no speedup over exhaustive search when the depth is at least $c\log n$ for all sufficiently large c. In this paper, we consider unrestricted depth circuits with *restricted treewidth* as natural generalizations of formulas (circuits with treewidth 1) and obtain improved satisfiability algorithms for such circuits.

Treewidth is also a natural parameter to measure the structural complexity of a circuit. It should be noted that Alekhnovich and Razborov [1] and Allender et al. [2] have previously investigated the time and space complexity of CNF-SAT from the perspective of the treewidth of the underlying graph. For a CNF ϕ with n input variables and m clauses, they define the treewidth ω' of ϕ as the treewidth of the clause-variable graph of ϕ. Alekhnovich and Razborov [1] showed that the satisfiability of such CNF can be solved in time $2^{O(\omega')}|\phi|^{O(1)}$ and space $2^{O(\omega')}|\phi|^{O(1)}$. Allender et al. improved the space bound to $|\phi|^{O(1)}$ with a corresponding time bound of $2^{O(\omega')\log|\phi|}|\phi|^{O(1)}$. These results also apply to circuits, as a circuit with m gates and n inputs of treewidth ω' can be converted to a CNF with m input gates and treewidth $O(\omega')$ with a 1-1 correspondence between the satisfying assignments. This shows that satisfiability for such circuits can be checked in time $O(\texttt{poly}(m)2^{O(\omega')})$ where the definition of the treewidth includes input gates. While these results shed light on the connection between the the structural properties of the CNF and the complexity of their satisfiability, treewidth ω' as defined can be as large as n for CNF and thus these results do not offer any improvements even for CNF-SAT.

In this paper, we define treewidth ω of a circuit as the treewidth of the circuit after the input gates are removed. According to our relaxed definition of treewidth, the treewidth of CNFs and formulas is at most 1. We make modest progress towards the general goal by showing improved algorithms for the satisfiability of such circuits whereas previously such improvements all only known for formulas. In particular, we prove the following results in this paper.

- Our first main result (Theorem 1) is an algorithm for counting the number of satisfying assignments of circuits with n input gates, treewidth ω, and at most $s\cdot n$ gates. The running time of our algorithm is $2^{n(1-\frac{1}{O(s\cdot\omega\cdot 4^{\omega})})}$, which for formulae instantiates to $2^{n(1-1/O(s))}$. This is the first algorithm to achieve exponential speed-up over brute force for the satisfiability of linear size circuits with treewidth bounded by a constant greater than 1. For treewidth 1, i.e., boolean formulae, our algorithm significantly outperforms the previously fastest $2^{n(1-1/O(s^2))}$ time satisfiability algorithm by Santhanam [32][1].

[1]The $1/O(s^2)$ bound on the savings in the algorithm of Santhanam was derived by Chen et al. [8]

- Our second main result (Theorem 2) is an algorithm for True Quantified Boolean Circuit Satisfiability for circuits of treewidth ω, in which every input gate has fan-out at most s. The running time of our algorithm is $2^{n(1-\frac{1}{O(s\cdot\omega\cdot 4^{\omega})})}$. Our algorithm is the first to achieve exponential speed-up over brute force for such circuits. Indeed, even for quantified boolean formulae where every variable appears at most s times, the previously best known algorithm by Santhanam [32] has running time $2^{n(1-1/O(f(s)\cdot\log n))}$.

- Utilizing the structural properties of low treewidth circuits which helped us obtain improved exponential-time algorithms for satisfiability, we also show that the number of wires of any constant treewidth circuit that computes the majority function must be super-linear (Theorem 3).

Remarks. The algorithm for satisfiability of circuits with $s\cdot n$ gates is given in Theorem 1. Note that the statement of Theorem 1 never explicitly mentions that the number of gates in the circuit should be at most $s\cdot n$. Instead the requirement is that the number of gates that have at least one wire directly from the input gates is at most $s\cdot n$. Clearly this is a weaker requirement.

For *formulas* we have that $\omega = 1$. Thus, for formulas, and more generally for circuits where the treewidth ω of $C - I(C)$ is constant, Theorem 1 yields a factor $2^{n/O(s)}$ improvement over exhaustive search. This is an *exponential* improvement over $O(2^n)$ for circuits of *linear* wire-size, and *sub-exponential* improvement over $O(2^n)$ for $s = O(n^{\delta})$ for $\delta < 1$, i.e circuits of *sub-quadratic* gate-size. The sub-exponential improvement over $O(2^n)$ for circuits of sub-quadratic wire-size holds all the way up to $\omega = o(\log n)$, i.e. *sub-logarithmic* treewidth.

Our algorithms all use exponential space, which limits their practical applicability. However, while the time usage of our algorithms is close to 2^n, the space usage of our algorithms is approximately $2^{n/2}$. Thus, the space used is about the square root of the time, which means that time is just as likely to become a bottleneck as space is.

As stated, the algorithms of Theorems 1 and 2, and the lower bound of Theorem 3 applies to un-bounded fan-in circuits over the De Morgan basis. However, any circuit C of fan-in d, treewidth ω, and gate-size $s\cdot n$ over the arbitrary Boolean basis can be converted into a circuit C' of treewidth $\omega\cdot d$, and gate-size $2^d\cdot s\cdot n$ computing the same function. Hence the conclusions of Theorems 1, 2 and 3 also hold for constant fan-in circuits over the arbitrary Boolean basis (at the cost of a constant factor reduction in the savings for Theorems 1 and 2).

It would be interesting to see how close our savings of $1/O(s)$ in the exponent for formula satisfiability are optimal. For formulas in CNF with gate-size $s\cdot n$, there are algorithms that achieve savings $1/O(\log s)$ in the exponent [12]. Hence perhaps one could hope for savings of this form even for formula satisfiability. Nevertheless, even an algorithm with savings $1/O(s^{\delta})$ for $\delta < 1$ would definitely be of interest.

Methods. In 1966, Nečiporuk [28] proved a $\Omega(n^2/\log n)$ size lower bound for formulas computing the element distinctness function over the complete binary basis. The main ingredient of this proof is an upper bound on the number of sub-functions of a function computable by a size-bounded formula. More concretely, let $f : \{0,1\}^n \to \{0,1\}$ be a function. For $X + Y = n$ we can split the input into two parts and think of f as a function from $\{0,1\}^X \times \{0,1\}^Y$ to $\{0,1\}$. Then each $y \in \{0,1\}^Y$ naturally defines a *subfunction* $f_{|_y} : \{0,1\}^X \to \{0,1\}$ where $f_{|_y}(x) = f(x,y)$. For general functions f each $y \in \{0,1\}^Y$ could result in a different subfunction $f_{|_y} : \{0,1\}^X \to \{0,1\}$. However, if the function f is structured one could hope that many different choices for $y \in \{0,1\}^Y$ will result in the function $f_{|_y}$ being the same. This is precisely what Nečiporuk proved, provided that f is computed by a formula, and that the input gates corresponding to y do not have a large fan-out.

Specifically, if f is computed by a formula C (that is, a circuit C that becomes a tree after removing the input gates $I(C)$), and the total fan-out of the input gates corresponding to X is at most p, then the number of different subfunctions $f_{|y}$ that can be obtained by selecting different $y \in \{0,1\}^Y$ is at most $2^{O(p)}$. In a recent paper, de Oliveira Oliveira [13] generalized Nečiporuk's bound to circuits of bounded treewidth. In particular, de Oliveira Oliveira proved that if $C - I(C)$ has treewidth ω instead of being a tree, then the number of different subfunctions $f_{|y}$ that can be obtained by selecting different $y \in \{0,1\}^Y$ is at most $2^{2^{O(\omega)}p}$.

The bounds of Nečiporuk and de Oliveira Oliveira suggest the following approach to counting the number of satisfying assignments of a circuit, provided that the input circuit C has treewidth ω after the input gates have been removed. First, find a partitioning of the input gates of C into X and Y such that $|X| = \epsilon \cdot n$ for a constant $\epsilon < 1/4$, but the total number p of wires leading out of X is small enough that $2^{2^{O(\omega)}p} \leq 2^{n/2}$. Such a partitioning can be found provided the circuit C is small enough. We count the number of pairs x and y such that $C(x,y) = 1$. We do this in two stages - go over every possible y, and for every y we count the number of choices for x such that $C_{|y}(x) = 1$. From the upper bound of de Oliveira Oliveira we know that the number of different subfunctions $C_{|y}$ that the algorithm will encounter is upper bounded by $2^{n/2}$. When we encounter a subfunction $C_{|y}$ we have not seen before, we try all possibilities for x and return the number of choices for x that resulted in $C(x,y) = 1$. When we encounter a subfunction $C_{|y}$ that we *have* seen before, we do not need to try all possibilities for x, we can just return the same number of satisfying assignments as we did the last time we saw this subfunction. There are $2^{n(1-\epsilon)}$ choices for y. For at most $2^{n/2}$ of these choices the subfunction $C_{|y}$ has not been previously encountered. For each of these choices we spend $2^{\epsilon n}$ time going over all choices of x. Thus the total running time is upper bounded by $2^{n(1-\epsilon)} + 2^{n/2} \cdot 2^{\epsilon \cdot n} = O(2^{n(1-\epsilon)})$.

The above argument has a serious gap - how do we check efficiently whether we have seen a subfunction $C_{|y}$ before, without evaluating it on all inputs x? Indeed, if C is unsatisfiable then *all* subfunctions would evaluate to 0 for every input, but determining this amounts to solving the satisfiability problem for each choice of y! We overcome this problem by noticing that the bounds of Nečiporuk and de Oliveira Oliveira can be made constructive in the following sense. We construct an efficiently computable "hash function" that given C and y outputs a "hash value" between 1 and $2^{2^{O(\omega)}p}$. If y and y' produce the same hash value, then $C_{|y'} = C_{|y}$. Armed with such a hash function the algorithm can easily be made to work by replacing the question "has this subfunction been encountered before?" with "has a subfunction with the same hash value been encountered before?". This is precisely the approach taken in our algorithm for counting then number of satisfying assignments of a linear size bounded treewidth circuit, encapsulated in Theorem 1.

The algorithm for *quantified* circuit satisfiability (Theorem 2) follows the same approach. The only important difference is that now we have to pick a partition of the input gates into X and Y such that all of Y is quantified before all of X. For this reason our algorithm only works for the more restricted setting where all input gates have bounded fan-out, as opposed to the average fan-out being bounded.

Our algorithms are based on memoization, which has previously been used extensively for the design of exact exponential time algorithms (see e.g. Fomin and Kratsch [15], Robson [31] or Williams [38]). Indeed, the algorithm of Santhanam [33] for QBF satisfiability and of Chen et al. [8] for counting satisfying assignments of a formula are both based on memoization. Therefore our main contribution is to show that, when based on the lower bounds of Nečiporuk and de Oliveira Oliveira, memoization can be made to work surprisingly well for (quantified) satisfiability of bounded treewidth circuits. Interestingly, Santhanam hints in his survey [34] that his algorithm for formula satisfiability [33] is inspired by the lower bound of Nečiporuk.

The connection between the lower bound and the algorithm in the work of Santhanam [33] is more subtle, whereas in our case it is very direct.

Both of our algorithms are based on decompositions of the input circuit into $\epsilon \cdot n$ parts, controlling the interaction between each of the parts and the input gates. Our circuit size lower bound for the $\mathsf{Majority}_n$ function (Theorem 3) is based on the observation that, for circuits of linear wire-size and constant treewidth (of $C - I(C)$), the decomposition can be "scaled up" to a decomposition where the number of parts is constant, the number of wires between different parts is (essentially) constant, and each part depends only on $n/2$ of the input gates.

If C computes a symmetric function f this decomposition leads to a constant-cost multi-party communication protocol for f in the Number on the Forehead (NOF) model. Here each part of the decomposition corresponds to a player in the communication protocol. For $\mathsf{Majority}_n$, such a protocol is known not to exist [7]. Together these two facts yield Theorem 3, that linear wire-size circuits C with $\mathsf{tw}(C - I(C)) = O(1)$ can not compute the $\mathsf{Majority}_n$ function.

2 Preliminaries

Graph notation. In this paper we deal with graphs that are simple, finite, and either directed or undirected. Given a graph G we will refer by $E(G)$ to the edge set E of G and by $V(G)$ to the vertex set V of G. For vertices u, v in $V(G)$ we will use notation uv for the ordered pair (u, v). Thus, if G is an undirected graph with an edge between u and v then this edge is denoted both by uv and by vu. If G is a directed graph then uv denotes an edge with its head in v and tail in u. For any non-empty subset $W \subseteq V$, the subgraph of G induced by W is denoted by $G[W]$ and for ease of notation $G - W$ is used for the induced subgraph $G[V \setminus W]$. For a directed graph G the *underlying undirected graph* is a graph with vertex set $V(G)$ and an (undirected) edge uv between vertices u and v whenever at least one of uv and vu is an edge in the original graph G.

For an undirected graph G, the *neighborhood* of a vertex v is $N_G(v) = \{u \in V : uv \in E(G)\}$ and the *closed neighborhood* of v is $N_G[v] = N(v) \cup \{v\}$. For a vertex set $W \subseteq V(G)$ we similarly define $N_G[W] = \bigcup_{v \in W} N_G[v]$ and $N_G(W) = N_G[W] \setminus W$. For a directed graph, the *in-neighborhood* of a vertex v in a graph G is $N_G^-(v) = \{u \in V : uv \in E(G)\}$ and the *out-neighborhood* of v in a graph G is $N_G^+(v) = \{u \in V : vu \in E(G)\}$. For vertex sets W we define $N_G^-(W) = \left(\bigcup_{v \in W} N_G^-(v)\right) \setminus W$ and $N_G^+(W) = \left(\bigcup_{v \in W} N_G^+(v)\right) \setminus W$. The *in-degree* and *out-degree* of a vertex v in a directed graph G is $|N_G^-(v)|$ and $|N_G^+(v)|$ respectively. The *degree* of a vertex v in an undirected graph G is $|N_G(v)|$ an the degree of v in a directed graph is its degree in the underlying undirected graph. In a directed graph a *sink* is a vertex of out-degree 0, and a *source* is a vertex of in-degree 0.

A *walk* in a graph G is a sequence $v_1, v_2, \ldots, v_t$ of vertices in G such that for every $i \leq t-1$, $v_i v_{i+1} \in E(G)$. Notice that this definition works both in directed and in undirected graphs, but in directed graphs the edges are required to point in the direction of the walk. A *path* in a graph G is a walk in which every vertex appears at most once, and a *cycle* in G is a walk where every vertex appears at most once, except for the first and last vertex, which are the same. A graph is *acyclic* if it does not contain any cycle. Directed acyclic graphs are commonly referred to as DAGs. An undirected graph is *connected* if there is a path between every pair of vertices. A *tree* is an undirected graph that is both connected and acyclic.

Every DAG G admits a *topological ordering* [14], which is an ordering of the vertices $\sigma : V(G) \to \{1, \ldots, |V(G)|\}$ such that for every edge $uv \in E(G)$, $\sigma(u) < \sigma(v)$. Hence, every DAG has at least one source and at least one sink (the vertices u and v with $\sigma(u) = 1$ and $\sigma(v) = |V(G)|$ respectively).

Treewidth. A *tree decomposition* of an undirected graph G is a pair (T, χ) consisting of a tree T and a function $\chi : V(T) \to 2^{V(G)}$ satisfying the following properties. For every $uv \in E(G)$,

$\{u, v\} \subseteq \chi(v)$ for some $v \in V(T)$; and for every vertex $v \in V(G)$ the set $\{u \in V(T) : v \in \chi(u)\}$ is non-empty and induces a connected subtree of T. The elements of the range $\{\chi(v) : v \in V(T)\}$ of χ are called *bags* of T. In a *rooted* tree-decomposition (T, χ) the tree T is rooted, and the root vertex of T is denoted by $r(T)$. The *width* of a tree decomposition (T, χ) of G is denoted by $\mathsf{tw}(T, \chi, G)$ and is defined to be the maximum over vertices $u \in V(T)$ of $|\chi(u)| - 1$. In other words, the width of a tree decomposition is the maximum size of a bag, minus one. The *treewidth* of G is denoted by $\mathsf{tw}(G)$ and is defined to be the minimum over all tree decompositions (T, χ) of G of $\mathsf{tw}(T, \chi, G)$. We extend the function χ to vertex sets in the following way: for a subset S of $V(T)$, $\chi(S) = \bigcup_{v \in S} \chi(v)$. For a directed graph G, a tree decomposition of G is a tree decomposition of the underlying undirected graph. Similarly, the treewidth of G is the treewidth of the underlying undirected graph.

Circuits. A *boolean circuit* (or just *circuit*) is a directed acyclic graph C, in which every vertex other than the sources are labeled with the symbols $\wedge$, $\vee$, or $\neg$. Every gate labeled $\neg$ has in-degree exactly equal to 1, and out-degree at most 1. The vertices of C are called *gates*, with the *sources* of C called *input gates*, and the sinks of C called *output gates*. Gates that are neither input nor output gates are called *normal* gates. The set of input gates, output gates and normal gates of a circuit C are referred to as $I(C)$, $O(C)$ and $N(C)$ respectively.[2]

A gate labeled $\wedge$, $\vee$, or $\neg$ is called an *and-gate*, *or-gate* or *not-gate*, respectively. We say that a gate u *feeds into* a gate v if $uv \in E(C)$. The *fan-in* of a gate is its in-degree, and the *fan-out* is its out-degree. The *fan-in* and *fan-out* of a circuit C is the maximum fan-in and fan-out of its gates. In the description of circuits we will often use the following short-hand: when we say that a gate u *feeds negatively* into a gate v, we mean that there is a not gate w such that u feeds into w and w feeds into v.

We will identify 0 with the meaning "false" and 1 with the meaning "true". An *evaluation* of a circuit C is an assignment to each gate of C a value in {0,1} such that: for every and-gate assigned 1 every gate feeding into it is assigned 1, for every and-gate assigned 0 at least one gate feeding into it is assigned 0, for every or-gate assigned 1 at least one feeding into it is assigned 1, for every or-gate assigned 0 every gate feeding into it is assigned 0, for every not-gate assigned 1 the unique gate feeding into it is assigned 0, and for every not-gate assigned 0 the unique gate feeding into it is assigned 1. For every assignment of values to the input gates of C there is exactly one evaluation of C that assigns precisely these values to the input gates.

Given a circuit C we can define a function $f : 2^{I(C)} \to 2^{O(C)}$ as follows. Given a subset X of $I(C)$ consider the unique evaluation of C that assigns 1 to the input gates in X and 0 to the other input gates, and let O be the subset of $O(C)$ gates assigned 1 by this evaluation. We will abuse notation and denote by $C : 2^{I(C)} \to 2^{O(C)}$ the function f computed by the circuit C. If the elements of $I(C)$ and $O(C)$ are ordered as $\{i_1, i_2, \ldots, i_n\}$ and $\{o_1, o_2, \ldots, o_m\}$ respectively then the function C can also be thought of as a function from $\{0,1\}^n$ to $\{0,1\}^m$. For a circuit C with n input gates and 1 output gate, a *satisfying assignment* is an assignment $x \in \{0,1\}^n$ such that $C(x) = 1$.

For a function $f : I(C) \to O(C)$, subset X of $I(C)$ and subset R of X we define the *restriction* of f by R to be a function $f_{|_R} : 2^{I(C) \setminus X} \to 2^{O(C)}$ defined as follows. For every $Z \subseteq I(C) \setminus X$ we have $f_{|_R}(Z) = f(Z \cup R)$. The set R is sometimes provided as an assignment $r : X \to \{0,1\}$ where $r(x) = 1$ if $x \in R$ and $f(x) = 0$ otherwise, and we will use $f_{|_r}$ as shorthand for $f_{|_R}$. Most commonly this notation will be used in the context of a function C computed by a circuit.

[2]Note that (perhaps unfortunately) the functional notation $N()$ refers to two different functions depending on whether the argument is a circuit or a vertex set. For vertex sets S, $N(S)$ returns the neighbors of S, whereas for circuits $N(C)$ returns the set of normal gates of C. Which one is being used should always be clear from context.

We may assume that in addition to the input gates, there are two input gates where one is set to 1 and the other is set to 0. Every evaluation of the circuit assigns 1 to the gate 1 and 0 to the gate 0. We will refer to these as *constant* gates. Throughout the algorithm, by *forcing* a gate g to 0 (or to 1) we mean removing all wires feeding into g and adding a new wire from 0 (or 1) into g. Note that if g is an input gate then after g is forced to 0 or 1, g is no longer an input gate. For a circuit C, gate g and value $v \in 0, 1$ we will refer by $C^{g \leftarrow v}$ to the circuit obtained from C by forcing g to v. This notation can be extended to forcing a set of gates according to an assignment to all of them. If X is a subset of the gates of C and $r : X \to \{0, 1\}$ then $C^{X \leftarrow r}$ refers to the circuit obtained from C by forcing each gate g in X to $r(g)$. Note that if X is a set of input gates then the circuit $C^{X \leftarrow r}$ computes the function $C_{|_r}$.

The *gate-size* of a circuit is the total number of gates, *excluding* the not-gates. The wire-size of a circuit is the total number of wires, that is $|E(C)|$. A circuit C has *linear* gate-size (or wire-size) if the gate-size (wire-size) is upper bounded by $O(n)$ where $n = |I(C)|$. Similarly, *quadratic* gate-size (or wire-size) means that gate-size (wire-size) is upper bounded by $O(n^2)$. In this paper, when we refer to the *treewidth* of a circuit C, we will always mean the treewidth of the underlying graph of $C - I(C)$, unless explicitly specified otherwise.

Our paper differs slightly from existing literature in how circuits are defined and how their size is measured. However the differences are purely notational. More concretely we demand that every not-gate has fan-out 1, while this is typically not required. On the other hand we do not count the not-gates towards the size of the circuit, while these gates are usually counted. Notice that a not-gate with fan-out x can easily be replaced by x not-gates of fan-out 1. This will at most double the total number of wires and gates (if you count not-gates) and not change the number of gates other than not-gates. Further, this operation can not increase the treewidth of the circuit, because on the underlying graph undirected it is a contraction operation, followed by a series of edge subdivisions (see [11, 14] for a definition of these graph operations).

3 Satisfiability Algorithms

In this section we will prove our main algorithmic results. We will assume that we have been given as input a circuit C with n input gates and *a single output gate* o, together with a tree decomposition of $C - I(C)$ of width at most ω. We will assume without loss of generality that the output gate o is an and-gate. If it is not, we may add a new and-gate and make o feed into the new gate, making the new gate the new output gate.

Splitting gates. The first ingredient both of our algorithms and of our lower bounds is a simple "splitting" operation on gates. Given the circuit C and a normal gate h of C, *splitting* h results in a circuit C' which is identical to C, with the following differences. The gate h is replaced by 4 gates h^{in}, h^{out}, h^L, and h^R. The gate h^{in} is an input gate, h^{out} is a gate of the same type as h, and h^L, and h^R are both or-gates. Further, h^{in} feeds into h^L and feeds negatively into h^R. On the other hand h^{out} feeds negatively into h^L and feeds into h^R. Both h^L and h^R feed directly into the output gate. Finally, every wire $uh \in E(C)$ is replaced by the wire uh^{out} in C'. Every wire $hu \in E(C)$ is replaced by the wire $h^{in}u$ in C'. This concludes the construction of C'.

Notice that h^{in} feeds into all the out-neighbors of h, while h^{out} is being fed into by all the in-neighbors of h. Thus the naming convention seems counter-intuitive, and it is tempting to swap the names of h^{in} and h^{out}. However this would result in h^{out} being an input gate, which also would not be ideal.

Let $e : V(C) \to \{0, 1\}$ be an evaluation of C, and define a function $e' : V(C') \to \{0, 1\}$ by setting $e'(g) = e(g)$ for every gate $g \in V(C') \setminus \{h^{in}, h^{out}, h^L, h^R\}$, $e'(h^{in}) = e'(h^{out}) = e(h)$ and $e'(h^L) = e'(h^R) = 1$. It is easy to verify that e' is an evaluation of C'. Similarly, let e' be an evaluation of C' that assigns 1 to the output gate. It follows that both h^L, h^R are assigned

1, which in turn means that h^{in} and h^{out} must both be assigned 0 or both be assigned 1. Let $e : V(C) \to \{0,1\}$ be the function such that for every gate $g \in V(C) \setminus \{h\}$ we have $e(g) = e'(g)$ and $e(h) = e'(h^{out})$. It is easy to verify that e is an evaluation of C.

The circuit C' has one additional input gate h^{in} in addition to the inputs $I(C)$. Hence C' computes a function from $\{0,1\}^{n+1}$ to $\{0,1\}$, where we associate the last coordinate of the input to the value assigned to h^{in}. This leads to the following observation.

Observation 1. *For every $x \in \{0,1\}^n$ such that $C(x) = 1$ there exists exactly one $y \in \{0,1\}$ such that $C'(x,y) = 1$. For every $x \in \{0,1\}^n$ and $y \in \{0,1\}$, if $C'(x,y) = 1$ then $C(x) = 1$.*

We will often need to split all the gates $h \in H$ for a set H of normal gates in C. From the definition of splitting it follows that the order in which we split the gates in H does not matter - the resulting circuit C' is the same. The circuit C' has $m = |H|$ additional input gates $h_1^{in}, \dots, h_m^{in}$ in addition to the inputs $I(C)$. Hence C' computes a function from $\{0,1\}^{n+m}$ to $\{0,1\}$, where we associate the m last coordinates of the input to the values assigned to $h_1^{in}, \dots, h_m^{in}$. Together with Observation 1 this leads to the following lemma.

Lemma 1. *Let C' be the circuit obtained from C by splitting a set H of m normal gates. For every $x \in \{0,1\}^n$ such that $C(x) = 1$ there exists exactly one $y \in \{0,1\}^m$ such that $C'(x,y) = 1$. For every $x \in \{0,1\}^n$ and $y \in \{0,1\}^m$, if $C'(x,y) = 1$ then $C(x) = 1$.*

Propeller Decompositions. The second ingredient of our algorithms and lower bounds is a decomposition lemma for graphs of bounded treewidth. A *propeller decomposition* of an undirected graph G is a partition of $V(G)$ into disjoint vertex sets H, B_1, B_2, $\dots, B_t$ such that for every $i \le t$, $N(B_i) \subseteq H$. The set H is called the *hub* of the decomposition, and the sets B_1, B_2, $\dots, B_t$ are called *blades*. The *width* of the propeller decomposition is defined as $\max_i |N(B_i)|$. A propeller decomposition of a *directed* graph is simply a propeller decomposition of the underlying undirected graph. The following lemma is a variation of Fomin et al. [17, Lemma 5], (see an extended version [16] for the proof) with essentially the same proof, which we include for completeness.

Lemma 2. *There exists a linear time algorithm that given a graph G, a tree decomposition (T, χ) of G of width ω, and a vertex set $Z \subseteq V(G)$ computes a propeller decomposition H, B_1, B_2, $\dots, B_t$ of width at most $2(\omega + 1)$, such that $t \le 2|Z|$, $Z \subseteq H$, and $|H| \le 2|Z|(\omega + 1)$.*

Towards the proof of Lemma 2 we need the notion of least common ancestor-closure in trees. For a rooted tree T and vertex set M in $V(T)$ the least common ancestor-closure (*LCA-closure*) **LCA-closure**(M) is obtained by the following process. Initially, set $M' = M$. Then, as long as there are vertices x and y in M' whose least common ancestor w is not in M', add w to M'. When the process terminates, output M' as the LCA-closure of M. The following (folklore) lemma summarizes two basic properties of LCA closures (see [16] for a proof).

Lemma 3 ([16])**.** *Let T be a tree, $M \subseteq V(T)$ and $M' =$ **LCA-closure**(M). Then $|M'| \le 2|M|$ and for every connected component C of $T \setminus M'$, $|N(C)| \le 2$.*

With Lemma 3 in hand we are ready to prove Lemma 2.

Proof of Lemma 2. For every $v \in Z$ add a node u in $V(T)$ such that $v \in \chi(u)$ to a set M'. We have that $M' \le |Z|$. Let M' be the set of marked nodes and set $M =$ **LCA-closure**(M'). By Lemma 3, $M \le 2|M'| \le 2|Z|$. Let $Q_1, Q_2, \dots, Q_\ell$ be the connected components of $T \setminus M$. By Lemma 3 we have that for every $i \le t$, $|N_T(Q_i)| \le 2$.

Without loss of generality all components $Q_1, Q_2, \dots, Q_t$ have exactly 2 neighbors in T while all components $Q_{t+1}, Q_{t+2}, \dots, Q_\ell$ have one neighbor. Contracting each component Q_i to a single vertex leaves a tree where the vertices corresponding to the components $Q_1, Q_2, \dots, Q_\ell$

have degree 1 or 2. Hence $t \leq |M| - 1$, and for each component Q_i with 1 neighbor in M there is a component Q_j with 2 neighbors in M such that $N_T(Q_i) \subseteq N_T(Q_j)$. Build $Q'_1, Q'_2, \ldots, Q'_t$ as follows. Initially, for every $i \leq t$, set $Q'_i = Q_i$. Then, for every j from $t+1$ to ℓ add Q_j to the lowest indexed Q'_i such that $N_T(Q_j) \subseteq N_T(Q_i)$. It follows that $M, Q'_1, \ldots, Q'_t$ is a partition of $V(T)$, and that each Q'_i has $|N_T(Q'_i)| \leq 2$.

Define $H = \bigcup_{u \in M} \chi(u)$ and for each $1 \leq i \leq t$ set $B_i = \bigcup_{u \in Q'_i} \chi(u) \setminus H$. Since every vertex of G appears in a bag of the tree-decomposition, $H, B_1, \ldots, B_t$ forms a partition of $V(G)$. By construction we have that for every i, $N(B_i) \subseteq H$. Furthermore, since $|N_T(Q'_i)| \leq 2$ we have $|N(B_i)| \leq 2(\omega + 1)$. Finally, the choice of M implies $Z \subseteq H$ and, since $|M| \leq 2|Z|$, we have that $|H| \leq 2|Z|(\omega+1)$. It is easy to implement a procedure that computes $H, B_1, \ldots, B_t$ in this way in linear time. □

3.1 Algorithmic Version of the Nečiporuk-Oliveira Subfunction Bound

We are now ready to prove our main technical result - a constructive bound on the number of subfunctions of a function computable by a circuit of bounded treewidth.

Lemma 4. *There exists an algorithm $\mathcal{A}$ that takes as input a C circuit with n input gates and gate-size m, a tree decomposition (T, χ) of $C - I(C)$ of width ω, a partition of $I(C)$ into $X \cup R$ and an assignment $r : R \to \{0,1\}$, runs in time $O(4^\omega \cdot m)$, and outputs a string $\mathcal{A}(C, (T, \chi), X, r) \in \{0,1\}^\ell$, where $\ell \leq 6(|N^+_C(X)| + 1) \cdot (\omega + 1) \cdot 2^{2(\omega+1)}$. Furthermore, for any two assignments r and r' to R such that $\mathcal{A}(C, (T, \chi), X, r) = \mathcal{A}(C, (T, \chi), X, r')$, the functions $C_{|r}$ and $C_{|r'}$ are equal.*

Proof. We first describe the algorithm $\mathcal{A}$. The algorithm first applies Lemma 2 to the underlying undirected graph of $C - I(C)$ and with the vertex set $Z = N^+_C(X) \cup \{o\}$, where o is the output gate of C. Lemma 2 yields a propeller decomposition $H, B_1, B_2, \ldots, B_t$ of $C - I(C)$ of width at most $2(\omega + 1)$ such that the hub H has size at most $2(|N^+_C(X)| + 1)(\omega + 1)$, $N^+_C(X) \cup \{o\} \subseteq H$, and the number t of blades is at most $2(|N^+_C(X)| + 1)$. Next the algorithm splits all the gates in $H \setminus \{o\}$ and obtains a new circuit C' with $m = |H| - 1$ input gates in addition to $I(C)$. By Lemma 1 we have that for every $x \in \{0,1\}^n$ such that $C(x) = 1$ there exists exactly one $y \in \{0,1\}^m$ such that $C'(x, y) = 1$. Furthermore, for every $x \in \{0,1\}^n$ and $y \in \{0,1\}^m$, if $C'(x, y) = 1$ then $C(x) = 1$.

We now summarize the structural properties of the circuit C'. Each gate $h \in H \setminus \{o\}$ in the circuit C corresponds to 4 gates h^{in}, h^{out}, h^L and h^Rin C'. We define $H^{in} = \{h^{in} : h \in H\}$, $H^{out} = \{h^{out} : h \in H\}$, $H^L = \{h^L : h \in H\}$ and $H^R = \{h^R : h \in H\}$. In the circuit C' we have that $N^+_{C'}(X) \subseteq H^{out} \cup \{o\}$, and that $|H^{out}| \leq 2(|N^+_C(X)| + 1) \cdot (\omega + 1)$. For each blade B_i we have that $N^+_{C'}(B_i) \subseteq H^{out} \cup \{o\}$ and that $|N^+_{C'}(B_i)| \leq 2(\omega + 1)$. Further, $N^-_{C'}(B_i) \subseteq H^{in} \cup R$ and $|N^-_{C'}(B_i) \cap H^{in}| \leq 2(\omega + 1)$.

For every $i \leq t$ define C^i to be the sub-circuit of C' induced by $B_i \cup N^+_{C'}(B_i) \cup N^-_{C'}(B_i)$. The input gates of C^i are precisely $N^-_{C'}(B_i)$ which consists of (a subset of) R plus a set $I^i = N^-_{C'}(B_i) \setminus R$ of at most $2(\omega + 1)$ gates from H^{in}. The circuit C^i has at most $2(\omega + 1)$ output gates, namely $O^i = N^+_{C'}(B_i)$. Hence the sub-circuit C^i computes a function $C^i : 2^{R \cup I^i} \to 2^{O_i}$. It follows that $C^i_{|r}$ is a function from 2^{I^i} to 2^{O^i}.

The algorithm $\mathcal{A}$ constructs the circuits C^i for every $i \leq t$. For every $i \leq t$ it then uses the circuit C^i to evaluate $C^i_{|r}(P)$ for every $P \subseteq I^i$ and records the output $C^i_{|r}(P)$ as a string 0's and 1's of length $2(\omega + 1)$. In the last phase of the algorithm, for each gate g in $H^{out} \cup \{o\}$ it records a 1 or a 0 as follows. If g is an and-gate the algorithm records a 0 for g if at least one gate in R feeding into g is assigned 0 by r, or if at least one gate in R feeding negatively into g is assigned 1 by r. Otherwise the algorithm records 1 for g. If g is an or-gate the algorithm records a 1 for g if at least one gate in R feeding into g is assigned 1 by r, or if at least one gate in R feeding negatively into g is assigned 0 by r. Otherwise the algorithm records 0 for g. This concludes the description of the algorithm $\mathcal{A}$

In total the algorithm records at most $2(\omega+1)\cdot 2^{2(\omega+1)}$ bits for each blade of the propeller decomposition, plus $|H^{out}|+1=|H|\leq 2(|N_C^+(X)|+1)(\omega+1)$ bits at the end of the procedure. The number of blades in the propeller decomposition is at most $2(|N_C^+(X)|+1)$. Hence the total number of bits recorded by the algorithm is at most $6(|N_C^+(X)|+1)\cdot(\omega+1)\cdot 2^{2(\omega+1)}$.

It remains to prove that for any two assignments r and r' to R such that $\mathcal{A}(C,(T,\chi),X,r)=\mathcal{A}(C,(T,\chi),X,r')$, the functions $C_{|r}$ and $C_{|r'}$ are equal. Towards this it suffices to show that for every $X'\subseteq X$ such that $C_{|r}(X')=1$ we have $C_{|r'}(X')=1$. Suppose now that $C_{|r}(X')=1$. By Lemma 1 there exists an evaluation f of C' that assigns 1 to every element in X', 0 to every element in $X\setminus X'$, agrees with r on R, and assigns 1 to o. Consider now the evaluation f' of C' that assigns 1 to every element in X', 0 to every element in $X\setminus X'$, agrees with r' on R, and agrees with f on H^{in}. We first prove that f and f' agree an all gates in H^{out}. Let g be an arbitrarily chosen or-gate in H^{out}. We will prove that if f assigns 1 to g, then f' also assigns 1 to g and if f' assigns 1 to g, then f also assigns 1 to g.

If f assigns 1 to g then there exists a gate g' such that either f assigns 1 to g' and g' feeds positively into g or f assigns 0 to g' and g' feeds negatively into g. The gate g' can either be in X, in R, in H^{in}, or in B_i for some $i\leq t$. If g' is in X or H^{in} then f and f' agree on g' (by choice of f') and hence f' also assigns 1 to g. If g' is in R then the algorithm $\mathcal{A}$ when ran on $(C,(T,\chi),X,r)$ recorded 1 for g in the last phase. Hence $\mathcal{A}$ when ran on $(C,(T,\chi),X,r')$ also recorded 1 for g in the last phase. Hence there exists a gate g'' such that r' assigns 1 to g'' and g'' feeds positively into g or r' assigns 0 to g'' and g'' feeds negatively into g. We conclude that also in this case f' assigns 1 to g. Finally, if g' is in B_i, then set P to be the subset of $N_{C'}^-(B_i)\setminus R$ that f assigns 1. We have that $g\in C^i_{|r}(P)$, in other words $C^i_{|r}(P)$ outputs 1 for g. Since the recordings of the algorithm for $C^i_{|r}(P)$ and $C^i_{|r'}(P)$ are the same, it follows that $C^i_{|r'}(P)$ also outputs 1 for g. But this means that there is some gate $g''\in B_i$ that such that f' assigns 1 to g'' and g'' feeds positively into g or f' assigns 0 to g'' and g'' feeds negatively into g. Hence f' assigns 1 to g. The proof that if f' assigns 1 to g then f also assigns 1 to g is identical.

Consider now an arbitrarily chosen and-gate g in H^{out}. An identical argument to the one above shows that if f assigns 0 to g, then f' also assigns 0 to g and if f' assigns 0 to g, then f also assigns 0 to g. Hence f and f' agree on all gates in H^{out}.

Since f assigns 1 to o it follows that f assigns 1 to all gates in $H^L\cup H^R$. Thus, for every $h\in H$ we have that f assigns the same value to h^{in} and h^{out}. Since f and f' agree both on h^{in} and h^{out} it follows that f' also assigns 1 to all gates in $H^L\cup H^R$. Finally we need to argue that f' assigns 1 to o. Since all gates in $H^L\cup H^R$ are 1 this follows from an argument identical to the one for and-gates in H^{out}. This concludes the proof. $\square$

3.2 Counting Satisfying Assignments

Lemma 5. *There is an algorithm that takes as input a circuit C with n input gates and gate-size m, such that at most $s\cdot n$ wires are incident to $I(C)$, and $\omega=\mathsf{tw}(C-I(C))=o(\log n)$, runs in time $2^{n(1-\epsilon)}m^{O(1)}$, where*

$$\epsilon=\left(48\cdot s\cdot(\omega+1)\cdot 4^{\omega}\right)^{-1},$$

and outputs the number of satisfying assignments to C.

Proof. First, using Bodlaender's algorithm [3], the algorithm computes a tree decomposition (T,χ) of $C-I(C)$ of width at most ω, in time $2^{O(\omega^3)}m=2^{o(n)}m$. Then, the algorithm picks a subset X of $I(C)$ of size $\epsilon \cdot n$ such that $|N_C^+(X)| \leq \epsilon \cdot s \cdot n$. Such a set exists by a simple averaging argument. For example, X can be chosen to be the set of the first ϵn gates of $I(C)$ when the gates are ordered by their fan-out in increasing order. Then, the algorithm sets $R = I(C) \setminus X$, and proceeds to iterate over every assignment $r : R \to \{0,1\}$. For each r the algorithm computes the number c_r of satisfying assignments to $C_{|r}$ as follows. The algorithm first applies Lemma 4 to r, and computes a string $q \in \{0,1\}^\ell$, where $\ell \leq 6(|N_C^+(X)| + 1) \cdot (\omega + 1) \cdot 2^{2(\omega+1)}$. It then looks up whether the string q has been output in a previous iteration for some other assignment r' to R. If this is the case, then by Lemma 4 we have that $c_r = c_{r'}$, so the algorithm records this and proceeds to the next iteration. If the string q has not been previously encountered, the algorithm computes c_r by trying all possible assignments to X and stores the pair (q, c_r). After iterating over all choices of r the algorithm outputs $\sum_r c_r$ as the total number of satisfying assignments to C. Since each satisfying assignment to C corresponds to exactly one satisfying assignment to exactly one circuit $C_{|r}$, the correctness of the algorithm follows.

We now turn to the running time analysis. For each of the $2^{n(1-\epsilon)}$ choices of r the algorithm applies Lemma 4 once, which takes $n^{O(1)}$ time since $\omega = o(\log n)$. Furthermore, in at most 2^ℓ of the iterations (once for each distinct q), the algorithm spends $2^{\epsilon n}$ time to compute c_r. In all other iterations, computation of c_r takes time $O(n)$ as it requires a single lookup in an exponentially large table (or binary search tree) that stored the pairs (q, c_r). Thus the total time is upper bounded by $2^{(1-\epsilon)n} n^{O(1)} + 2^\ell 2^{\epsilon n} n^{O(1)}$. Since $\ell \leq 6(s \cdot \epsilon \cdot n + 1) \cdot (\omega + 1) \cdot 4^{\omega+1}$ we have that $2^\ell 2^{\epsilon n} \leq 2^{(n/2)+o(n)}$. Hence the running time is upper bounded by $2^{(1-\epsilon)n} n^{O(1)}$, as claimed. $\square$

Observe that the requirement in Lemma 5 that at most $s \cdot n$ wires are incident to $I(C)$ is weaker than demanding that the wire-size of C is at most $s \cdot n$. Next we strengthen Lemma 5 to circuits of bounded gate-size, rather than wire-size. In a circuit C where the treewidth of $C - I(C)$ is bounded by ω, the number of wires in $C - I(C)$ is within a factor ω of the number gates. However the number of wires from $I(C)$ to $N_C^+(I(C))$ can be as large as quadratic in $|N_C^+(I(C))|$. We obtain the strengthening of Lemma 5 to bounded gate-size by a standard fan-out reducing procedure for the input gates.

Theorem 1. *There is an algorithm that takes as input a circuit C with n input gates and gate-size m, such that $|N_C^+(I(C))| \leq s \cdot n$, and $\omega = \mathsf{tw}(C - I(C)) = o(\log n)$, runs in time $2^{n(1-\frac{\epsilon}{10})} m^{O(1)}$, where*

$$\epsilon = (48 \cdot s \cdot (\omega + 1) \cdot 4^\omega)^{-1},$$

and outputs the number of satisfying assignments to C.

Proof. First, using Bodlaender's algorithm [3], the algorithm computes a tree decomposition (T, χ) of $C - I(C)$ of width at most ω, in time $2^{O(\omega^3)} m = 2^{o(n)} m$. From now on, we assume that the decomposition (T, χ) is given as input.

The algorithm is recursive, and proceeds as follows. If no input gate (not counting the constant gates) has fan-out more than $6 \cdot s$ the algorithm applies Lemma 5 to solve the instance. Otherwise the algorithm picks an input gate g with fan-out at least $6 \cdot s$. It then calls itself recursively to count the number of satisfying assignments where g is set to 1, and the number of satisfying assignments when g is set to 0.

Before making the recursive call the algorithm performs the following simplification procedure. When g is set to 1 then, for every or-gate g' that g feeds positively into we force g' to 1. For every and-gate g' that g feeds negatively into we force g' to 0. When g is set to 0 then, for every or-gate g' that g feeds negatively into we force g' to 1. For every and-gate g' that g feeds positively into we force g' to 0. After the simplification procedure the algorithm removes the gate g and all wires leading out of g.

It is easy to see that the simplification procedure produces circuits where the number of satisfying assignments to the simplified circuit is equal to the number of satisfying assignments to C where the input gate g is set to 1 or 0 respectively. The correctness of the algorithm follows from this fact, together with the correctness of the algorithm of Lemma 5. We now proceed with the running time analysis.

Consider a node in the recursion tree from which the algorithm make two recursive calls, one with the input gate g set to 1 and one with g set to 0. For any gate g' that g feeds into, g' will be forced in at exactly one of the two recursive calls. Indeed, if g' is an and-gate and g feeds positively into g' then g' is forced when g is set to 0. If g' is an and-gate and g feeds negatively into g' then g' is forced when g is set to 1. If g' is an or-gate and g feeds positively into g' then g' is forced when g is set to 1. Finally, if g' is an or-gate and g feeds negatively into g' then g' is forced when g is set to 0.

Hence, in at least one of the two recursive calls, at least $3 \cdot s$ gates in $N_C^+(I(C))$ are forced. For each internal node of the recursion tree, pick *exactly* one edge corresponding to a recursive call where at least $3 \cdot s$ gates in $N_C^+(I(C))$ are forced, and call this edge a *heavy* edge. Since every internal node in the recursion tree has exactly one heavy and one *light* (not heavy) edge coming out of it, each leaf of the recursion tree is uniquely identified by the length ℓ of the root to leaf path, and the positions in the path of the heavy edges. In a leaf node for which the root to leaf path has length ℓ the number of input gates is $n - \ell$ and the total number of wires leading out from the input gates is at most $(n - \ell) \cdot 4 \cdot s$. Further, if the root to leaf path has at least h heavy edges, then at most $s \cdot n - 3 \cdot s \cdot h$ normal gates in C are being fed into from the non-constant input gates. It follows that $h \leq n/3$. It follows that the running time of the algorithm is bounded by

$$\sum_{\ell \leq n} \sum_{h \leq \min(\ell, n/3)} \binom{\ell}{h} \cdot 2^{(n-\ell)(1-\epsilon)} \cdot m^{O(1)}.$$

For $\ell \geq 9n/10$ the terms of the sum are upper bounded by $\binom{9n/10}{n/3} \cdot 2^{n/10} n^{O(1)} \leq 1.95^n \cdot m^{O(1)}$. For $\ell \leq 9n/10$ the sum is upper bounded by $2^\ell \cdot 2^{(n-\ell)(1-\epsilon)} \cdot m^{O(1)} \leq 2^{n(1-\frac{\epsilon}{10})} m^{O(1)}$. Hence the total running time is upper bounded by $2^{n(1-\frac{\epsilon}{10})} m^{O(1)}$, as claimed. □

3.3 Quantified Boolean Circuit SAT

Informally, a quantified circuit is simply a quantified boolean formula in prenex normal form where the formula that is used to evaluate the predicate once the variables have been instantiated is replaced by a circuit. Formally, a *quantified circuit* is a circuit C with n input gates equipped with a bijection $\sigma : \{1, \ldots, n\} \to I(C)$ and a *quantifier sequence* $q_1, q_2, \ldots, q_n$ where each q_i is a symbol in $\{\forall, \exists\}$.

We now give an inductive definition for what it means for a quantified circuit to *evaluate to* 1 or *evaluate to* 0. A quantified circuit with a single input gate g and quantifier $q_1 = \exists$ evaluates to 1 if $C(\{g\}) = 1$ or $C(\emptyset) = 1$. A quantified circuit with a single input gate g and quantifier $q_1 = \forall$ evaluates to 1 if $C(\{g\}) = C(\emptyset) = 1$. Otherwise the quantified circuit evaluates to 0. A quantified circuit with $n \geq 2$ gates with $q_1 = \exists$ evaluates to 1 if at least one of the two quantified circuits $C^{\sigma(1) \leftarrow 1}$ or $C^{\sigma(1) \leftarrow 0}$ with quantifier sequence $q_2, \ldots, q_n$ evaluate to 1. A quantified circuit with $n \geq 2$ gates with $q_1 = \forall$ evaluates to 1 if both quantified circuits $C^{\sigma(1) \leftarrow 1}$ or $C^{\sigma(1) \leftarrow 0}$ with quantifier sequence $q_2, \ldots, q_n$ evaluate to 1. Otherwise the quantified circuit evaluates to 0. We will sometimes say that the quantified circuit is "true" or "false" instead of saying it evaluates to 1 or 0 respectively.

In the Quantified Boolean Circuit SAT problem the input is a quantified circuit $(C, \sigma, q_1, \ldots, q_n)$. The task is to determine whether the quantified circuit is true or false. The recursive definition of evaluating the circuit naturally gives rise to a $2^n \cdot n^{O(1)}$ time algorithm for the problem. Here we give an algorithm that is significantly faster provided that $C - I(C)$ has bounded treewidth, and that every gate in $I(C)$ has bounded fan-out. The proof of Theorem 2 closely follows the proof of Lemma 5, the key difference is that the subset X of input gates that are used to generate the propeller decomposition is no longer chosen greedily, but set to $\{\sigma(\lfloor n(1-\epsilon)\rfloor), \sigma(\lfloor n(1-\epsilon)\rfloor + 1, \ldots, \sigma(n))\}$ instead.

Theorem 2. *There is an algorithm that takes as input a Quantified Circuit $(C, \sigma, q_1, \ldots, q_n)$ with n input gates and gate-size m, such that each input gate has fan-out at most s, and $\omega = \mathsf{tw}(C - I(C)) = o(\log n)$, runs in time $2^{n(1-\epsilon)}m^{O(1)}$, where*

$$\epsilon = (48 \cdot s \cdot (\omega + 1) \cdot 4^{\omega})^{-1},$$

and outputs whether $(C, \sigma, q_1, \ldots, q_n)$ is true.

Proof. First, using Bodlaender's algorithm [3], the algorithm computes a tree decomposition (T, χ) of $C - I(C)$ of width at most ω, in time $2^{O(\omega^3)}m = 2^{o(n)}m$. Then, the algorithm sets $X \subseteq I(C)$ to be $\{\sigma(\lfloor n(1-\epsilon)\rfloor), \sigma(\lfloor n(1-\epsilon)\rfloor + 1, \ldots, \sigma(n)\}$. Observe that $|N_C^+(X)| \leq \epsilon \cdot s \cdot n$. Then, the algorithm sets $R = I(C) \setminus X$, and commences with the algorithm for evaluating quantified circuits that follows directly from the definition. However, every time the algorithm arrives at a recursive call when the set of remaining input gates is *exactly* X it proceeds as follows.

Let r be the assignment that is forced to R in this recursive call. The algorithm needs to return c_r, which is defined to be 0 or 1 according to whether the quantified circuit

$$(C^{R \leftarrow r}, \sigma, q_{\lfloor n(1-\epsilon)\rfloor}, q_{\lfloor n(1-\epsilon)\rfloor+1}, \ldots, q_n)$$

evaluates to 0 or to 1.

The algorithm first applies Lemma 4 to r, and computes a string $s \in \{0,1\}^{\ell}$, where $\ell \leq 6(|N_C^+(X)| + 1) \cdot (\omega + 1) \cdot 2^{2(\omega+1)}$. It then looks up whether the string s has been output in a previous recursive call for some other assignment r' to R. If it has, then by Lemma 4 we have that $c_r = c_{r'}$ since $C_{\big|r} = C_{\big|r'}$ and whether or not the considered quantified circuit evaluates to 0 or 1 depends only on the function computed by the circuit $C^{R \leftarrow r}$ and not on the (structure of the) circuit itself. In this case, the algorithm returns $c_r = c_{r'}$.

If the string s has not been previously encountered, the algorithm computes c_r using the algorithm implicit in the definition of evaluating quantified circuits. It then stores that s has been previously encountered, and that when s was encountered the algorithm returned c_r. Then the algorithm returns c_r. The correctness of the algorithm follows directly from the definition of evaluation of quantified circuits together with Lemma 4.

We now turn to the running time analysis. For each of the $2^{n(1-\epsilon)}$ choices of r the algorithm applies Lemma 4 once, which takes $m^{O(1)}$ time since $\omega = o(\log n)$. Furthermore, in at most 2^{ℓ} of the recursive calls (the ones where s has not been previously encountered) the algorithm spends $2^{\epsilon n}$ time to compute c_r. In all other recursive calls this is done in time $O(n)$ by a single lookup in an exponentially large table (or binary search tree) storing all previously encountered strings. Thus the total time is upper bounded by $2^{(1-\epsilon)n}m^{O(1)} + 2^{\ell}$. Since $\ell \leq 6(s \cdot \epsilon \cdot n + 1) \cdot (\omega + 1) \cdot 4^{\omega+1}$ we have that $2^{\ell} \leq 2^{(n/2)+o(n)}$. Hence the running time is upper bounded by $2^{(1-\epsilon)n}m^{O(1)}$, as claimed. ☐

Notice that the only place where we used that the fan-out of all input gates is at most s, is to bound $N_C^+(X)$. Hence Theorem 2 also applies to all circuits where the total fan-out of the $\epsilon \cdot n$ gates that are quantified last is at most $\epsilon \cdot s \cdot n$.

4 Lower Bounds

In this section we will prove that constant treewidth circuits with a linear number of wires can not compute the $\mathsf{Majority}_n$ function. The proof hinges on the classic result by Chandra, Furst and Lipton [7] that in the Number On The Forehead (NOF) model of communication complexity, $\mathsf{Majority}_n$ does not admit a constant cost communication protocol with a constant number of players. We will prove that every symmetric function computable by a constant treewidth circuit with a linear number of wires does admit such a protocol. Together the two results yield the desired lower bound for $\mathsf{Majority}_n$.

Theorem 3. *For every constant s and ω there is no circuit C that computes $\mathsf{Majority}_n$, such that at most $s \cdot n$ wires lead out of $I(C)$ and that the treewidth $C - I(C)$ is at most ω.*

The Number on the Forehead Model We now introduce the notions from communication complexity that we will use. For in-depth treatment, see [23]. In the Number On The Forehead (NOF) model there are k players that together have to compute a function $f : \{0,1\}^n \to \{0,1\}$. The players know n and the function f to be computed prior to deciding on the protocol. Once they have decide on a protocol an input $x \in \{0,1\}^n$ is given as input, and the players together have to output $f(x)$.

Each player sees all bits of x except a subset of n/k "anti-private" bits that are seen by everyone else except her. In particular, for every player i there is a set $Q_i \subseteq \{1,\dots,n\}$ of size $\lfloor n/k \rfloor$ or $\lceil n/k \rceil$ such that player i has access to the input bits $\{1,\dots,n\} \setminus Q_i$. The sets $Q_1,\dots,Q_k$ form a partition of $\{1,\dots,n\}$, and the players know this partition.

Communication between the players happens by broadcast, i.e when a player transmits a message all the other players receive the message. The cost of sending the message is the length of the message. Any one of the players may output the function value. There are two variants of the Number On The Forehead model - the *best-partition* and the *worst-partition* version. In the *best-partition* model it is sufficient that there exists a partition $Q_1,\dots,Q_k$, such that for every input $x \in \{0,1\}^n$ the protocol successfully computes $f(x)$. In the *worst-partition* model the protocol has to successfully compute $f(x)$ for every input x and partition $Q_1,\dots,Q_k$.

A function $f : \{0,1\}^n \to \{0,1\}$ is *symmetric* if $f(x)$ only depends on the number of input bits set to 1. The function $\mathsf{Majority}_n : \{0,1\}^n \to \{0,1\}$ function outputs 1 if at least $n/2$ bits are set to 1. Hence $\mathsf{Majority}_n$ is symmetric. For symmetric functions there is no difference between the best-partition and the worst-partition models. Our lower bounds for $\mathsf{Majority}_n$ rely on the following result.

Proposition 1 ([7])**.** *For every constant c and k, $\mathsf{Majority}_n$ does not admit a best-partition NOF protocol with cost c and k players.*

One-Way Communication Protocols In a *one-way* communication protocol with k players the i'th player has access to a subset $S_i \subseteq \{1,\dots n\}$ of the input bits. The *access size* of the protocol is $\max_i |S_i|$. Each bit is seen by at least one player, and the players know the family $S_1,\dots,S_k$.

Each player gets to sends a single message to a *center*. The center receives the messages from all the players, but has no access to the input. The center then has to determine and output the function value. None of the players other than the center can see the messages of any of the other players. The *cost* of the communication protocol is the total number of bits sent from the players to the center.

We will only consider *best-partition* one-way communication protocols where there has to exist a collection $S_1,\dots,S_k$ of sets (each S_i of cardinality at most the access size) such that for every x the protocol successfully computes $f(x)$ given that each player i has access to the bits in S_i. Note that $S_1,\dots,S_k$ does *not* have to be a partition of $\{1,\dots,n\}$, nevertheless we refer to this model as the best-partition model for a nomenclature consistent with the one for the NOF model. We start by relating One-Way communication protocols for $\mathsf{Majority}_n$ and communication protocols in the NOF model.

Lemma 6. *For every constant k and c, if $\mathsf{Majority}_n$ admits a best-partition one-way communication protocol with k players, access size $n/2$, and cost c, then $\mathsf{Majority}_n$ admits a best-partition NOF protocol with k players and cost c.*

Proof. Suppose that $\mathsf{Majority}_n$ admits a best-partition one-way communication protocol with k players, access size $n/2$, and cost c. We then give a best-partition NOF protocol with cost c and k players.

Let $S_1, \ldots, S_k \subseteq \{1, \ldots, 5n\}$ be the access sets the players for computing $\mathsf{Majority}_{5n}$ in the one-way protocol. Each S_i has size at most $5n/2$. Select $Q_1, \ldots Q_k \subseteq \{1, \ldots, 5n\}$ such that for every i, $|Q_i| \in \{\lfloor n/k \rfloor, \lceil n/k \rceil\}$, Q_i is disjoint from S_i, $\sum_i |Q_i| = n$ and the sets Q_i are disjoint. Such a collection $Q_1, \ldots Q_k$ exists because it can be picked greedily: first select $\lfloor n/k \rfloor$ elements disjoint from S_1 and put them into Q_1. Then select $\lceil n/k \rceil$ elements from $\{1, \ldots, 5n\} \setminus S_2 \cup Q_1$ and put them into Q_2. In general, in step i pick $\lfloor n/k \rfloor$ or $\lceil n/k \rceil$ elements disjoint from $S_i \cup Q_1 \cup \ldots \cup Q_{i-1}$ and insert them into Q_i. Since $|S_i \cup Q_1 \cup \ldots \cup Q_{i-1}| \leq 5n/2 + n \leq 7n/2$ this will always be possible. Because $\mathsf{Majority}_{5n}$ is a symmetric function, without loss of generality $Q_1, \ldots Q_k$ forms a partition of $\{1, \ldots, n\}$. We give a NOF protocol for $\mathsf{Majority}_n$ with partition $Q_1, \ldots Q_k$.

On input $x \in \{0,1\}^n$ For every $i \leq k$, player i simulates the behavior of player i in the one-way protocol for $\mathsf{Majority}_{5n}$ on input (x, y) where y is a string of $2n$ 1's followed by $2n$ 0's. Note that $\mathsf{Majority}_n(x) = \mathsf{Majority}_{5n}(x, y)$. Further, player i of the NOF protocol can simulate the behavior of player i in the one-way protocol because Q_i and S_i are disjoint. In the NOF model, communication happens by broadcast, so player 1 can act as the center of the one-way protocol for $\mathsf{Majority}_{5n}$, and output the value of $\mathsf{Majority}_{5n}(x, y) = \mathsf{Majority}_n(x)$. The cost of the NOF protocol for $\mathsf{Majority}_n$ is the same as the cost of the one-way protocol for $\mathsf{Majority}_{5n}$, which is a constant c independent of n. This concludes the proof. □

One-Way Communication Protocols from Bounded Treewidth Circuits. We now prove the main technical ingredient of our lower bound – that every function computable by a linear wire-size bounded treewidth circuit admits a one-way protocol with a constant number of players, constant cost, and access size $n/2$.

Lemma 7. *Let C be a circuit with n input gates, such that at most $s \cdot n$ wires are incident to $I(C)$ and $\mathsf{tw}(C - I(C)) \leq \omega$ for a positive integer ω. Then, for every d, the function $C : 2^{I(C)} \to 2^{\{o\}}$ admits a one-way multi-party communication protocol with at most $8 \cdot s \cdot d$ players, each seeing at most n/d bits of the input, and each sending at most $12 \cdot (\omega + 1) \cdot 4^{\omega}$ bits to the center.*

Towards the proof of Lemma 7 we will first give a propeller decomposition lemma similar to Lemma 2, but decomposing the graph into "equally sized" blades rather than trying to exclude some fixed set from the blades. Similar arguments commonly appear in graph algorithms, see e.g [4, 22]. In the following, when given a weight function $w : V(G) \to \mathbb{N}$ on the vertices of a graph G, we will write $w(S)$ for subset S of $V(G)$, meaning $w(S) = \sum_{v \in S} w(v)$.

Lemma 8. *There exists a linear time algorithm that given a graph G, a weight function $w : V(G) \to \mathbb{Z}^+$, a tree decomposition (T, χ) of G of width ω and an integer r, computes a propeller decomposition H, B_1, B_2, $\ldots B_t$ of width at most $2(\omega + 1)$, such that $t \leq 5r$, $|H| \leq 2r(\omega + 1)$, and for every $i \leq t$, $w(B_i) \leq w(V(G))/r$.*

Proof. We will consider the tree decomposition as rooted with root r. It is well known (see e.g. [11]) that given a tree decomposition (T, χ) of a graph G one can in polynomial time construct a rooted tree decomposition (T', χ') of G of the same width, such that every node u in T has at most two children (neighbors that are further away from the root). We will therefore assume that in T every node has at most two children. For every vertex $u \in V(T)$ we define T_u to be the subtree of T rooted at u. We perform the following marking procedure.

Initially $M' = \emptyset$ and $w' : V(G) \to \mathbb{N}$ is equal to w. Then, if such a vertex exists, pick a lowermost vertex $u \in V(T)$ such that $w'(\chi(V(T_u))) \geq w(V(G))/r$. Add u to M' and, for every $v \in \chi(V(T_u))$, set $w'(v)$ to 0. When no vertex u with $w'(\chi(V(T_u))) \geq w(V(G))/r$ exists terminate the marking procedure. Note that every time a vertex u is added to M', $w'(V(G))$ is decreased by at least $w(V(G))/r$. Thus the marking procedure terminates after at most r rounds, with $|M'| \leq r$. Set $M = \textbf{LCA-closure}(M')$. By Lemma 3, $|M'| \leq 2|M| \leq 2r$.

Let $Q_1, Q_2 \ldots Q_t$ be the connected components of $T \setminus M$. By Lemma 3 we have that for every $i \leq t$, $|N_T(Q_i)| \leq 2$. Further, since T has maximum degree 3 it follows that $t \leq 2|M| + 1 \leq 5r$.

Define $H = \chi(M)$ and for each $1 \leq i \leq t$ set $B_i = \chi(Q_i) \setminus H$. Since every vertex of G appears in a bag of the tree-decomposition, $H, B_1, \ldots B_t$ forms a partition of $V(G)$. By construction we have that for every i, $N(B_i) \subseteq H$. Furthermore, since $|N_T(Q_i)| \leq 2$ we have $|N(B_i)| \leq 2(\omega+1)$. Finally, the marking procedure implies that for every $i \leq t$, $w(B_i) \leq w(V(G))/r$. This concludes the proof. □

Proof of Lemma 7. We will assume without loss of generality that every gate of C has at most one wire feeding into it from $I(C)$. To justify this assumption, suppose C does not have this property. For every wire uv with u in $I(C)$ we will make a new or-gate g and replace the wire uv with the wires ug and gv. Call the resulting circuit C^*. Clearly C and C^* compute the same function. Furthermore, since adding leaves to a graph does not increase treewidth (except possibly from 0 to 1), the treewidth of $C^* - I(C^*)$ is also at most ω. Additionally, the number of wires feeding out of $I(C^*)$ is also at most $s \cdot n$. Hence we may prove Lemma 7 for C^*, and the same conclusion will hold for C. Thus, from now on we assume that every gate of C has at most one wire feeding into it from $I(C)$.

Next, define a weight function w on the gates of $C - I(C)$ where the weight of every gate g is $|N^-_C(g) \cap I(C)|$. In other words, the weight of a gate is the number of wires feeding in to the gate from the input gates. From the assumption on the number of wires feeding out of $I(C)$ it follows that the total weight of all gates is at most $s \cdot n$. We then apply Lemma 8 on $C - I(C)$ with this weight function, and $r = d \cdot s$. This results in a propeller decomposition H, $B_1, \ldots B_t$ of width at most $2(\omega + 1)$, such that $t \leq 5 \cdot d \cdot s$, $|H| \leq 2 \cdot d \cdot s(\omega + 1)$, and for every $i \leq t$, $w(B_i) \leq n/d$. If the output gate o is not in H, but rather in some B_i then remove o from B_i and add o to H. This will increase the size H to at most $2 \cdot d \cdot s(\omega + 1) + 1$ and the width of the decomposition to at most $2(\omega + 1) + 1$. The circuit C' is obtained from C by splitting all the gates in $H \setminus \{o\}$. We abuse notation and will refer by $I(C)$ both to the input gates of C and to the input gates of C' that are copies of gates of C.

We now summarize the structural properties of the circuit C'. Each gate $h \in H \setminus \{o\}$ in the circuit C corresponds to 4 gates h^{in}, h^{out}, h^L and h^Rin C'. We define $H^{in} = \{h^{in} : h \in H \setminus \{o\}\}$, $H^{out} = \{h^{out} : h \in H \setminus \{o\}\}$, $H^L = \{h^L : h \in H \setminus \{o\}\}$ and $H^R = \{h^R : h \in H \setminus \{o\}\}$. In the circuit C' we have that $|H^{out}| = |H^{in}| \leq 2(|N^+_C(X)| + 1) \cdot (\omega + 1)$. For each blade B_i we have that $N^+_{C'}(B_i) \subseteq H^{out} \cup \{o\}$ and that $|N^+_{C'}(B_i)| \leq 2(\omega + 1)$. Further, $N^-_{C'}(B_i) \subseteq H^{in} \cup I(C)$, $|N^-_{C'}(B_i) \cap I(C)| \leq n/d$ (by the properties of the propeller decomposition), and $|N^-_{C'}(B_i) \cap H^{in}| \leq 2(\omega + 1)$. From the properties of splitting (Lemma 1) it follows that for every $X \subseteq I(C)$, if $C(X) = 1$ there exists exactly one $Y \subseteq H^{in}$ such that $C'(X \cup Y) = 1$. Further, if $C'(X \cup Y) = 1$ then $C(X) = 1$.

We define a one-way communication protocol for the function computed by C with at most $t + 3 \cdot d \cdot s$ players. Players $1, \ldots, t$ correspond to the blades $B_1, \ldots B_t$ of the propeller decomposition, while the remaining players and the center correspond to the hub. For every $i \leq t$ player i has access to the bits in $I(C) \cap N^-_{C'}(B_i)$. By construction, each of these players has access to at most n/d bits. There are at most $|H| \leq 2 \cdot d \cdot s(\omega + 1) + 1 \leq 3 \cdot d \cdot s \cdot (\omega + 1)$ gates in $I(C)$ with wires to $H^{out} \cup \{o\}$. The remaining $3 \cdot d \cdot s$ players distribute these gates among themselves arbitrarily, each of them having access to $(\omega + 1)$ bits each.

On input $X \subseteq I(C)$ the players proceed as follows. For every $i \leq t$, player i will consider the sub-circuit of C' induced by $P_i = B_i \cup N^-_{C'}(B_i) \cup N^+_{C'}(B_i)$. The sub-circuit $C'[P_i]$ of C'

induced by P_i has at most $2(\omega + 1) + 1$ output gates. Player i has access to $X \cap N^{-}_{C'}(B_i)$. Further, $|N^{-}_{C'}(B_i) \cap H^{in}| \leq 2(\omega + 1)$. For each $Q \subseteq N^{-}_{C'}(B_i) \cap H^{in}$ she evaluates $C'[P_i]$ on input $(X \cap N^{-}_{C'}(B_i)) \cup Q$ and sends the result to the center. Since there are at most $2^{2(\omega+1)}$ choices for Q, and for each choice of Q the result can be encoded in at most $2(\omega + 1) + 1$ bits, the total number of bits player i transmits to the center is at most $12 \cdot (\omega + 1) \cdot 4^{\omega}$. The remaining players transmit to the center the value of the gates that they have access to. Each of these players transmits at most $\omega + 1$ bits.

The center receives the messages from all of the players, and then, for every $Y \subseteq H^{out}$ can evaluate the circuit $C'(X \cup Y)$ without *without the knowledge of* X because the messages of the players describe precisely the output of all of the sub-circuits P_i and the value of all gates in $I(C)$ feeding directly into $H^{out} \cup \{o\}$. If there exists a Y such that the center concludes that $C'(X \cup Y)$ evaluates to 1, it outputs that $C(X) = 1$. Otherwise it outputs that $C(X) = 0$. □

Lower bound for $\mathsf{Majority}_n$. We are now in position to put everything together and prove Theorem 3.

Proof of Theorem 3. Suppose $\mathsf{Majority}_n$ had a circuit C such that at most $s \cdot n$ wires lead out of $I(C)$ and that the treewidth $C - I(C)$ is at most ω. By Lemma 7 with $d = 2$, $\mathsf{Majority}_n$ has a one-way protocol with constant cost, a constant number of players, and access size $n/2$. From Lemma 6 it then follows that $\mathsf{Majority}_n$ has a NOF-protocol with a constant number of players and constant cost. This contradicts Proposition 1 that such a protocol does not exist. □

References

[1] M. Alekhnovich and A. Razborov, *Satisfiability, branch-width and tseitin tautologies*, computational complexity, 20 (2011), pp. 649–678.

[2] E. Allender, S. Chen, T. Lou, P. A. Papakonstantinou, and B. Tang, *Width-parametrized sat: Time–space tradeoffs*, Theory of Computing, 10 (2014), pp. 297–339.

[3] H. L. Bodlaender, *A linear-time algorithm for finding tree-decompositions of small treewidth*, SIAM J. Comput., 25 (1996), pp. 1305–1317.

[4] H. L. Bodlaender, F. V. Fomin, D. Lokshtanov, E. Penninkx, S. Saurabh, and D. M. Thilikos, *(meta) kernelization*, J. ACM, 63 (2016), pp. 44:1–44:69.

[5] C. Calabro, R. Impagliazzo, and R. Paturi, *A duality between clause width and clause density for* sat, in CCC '06: Proceedings of the 21st Annual IEEE Conference on Computational Complexity, Washington, DC, USA, 2006, IEEE Computer Society, pp. 252–260.

[6] ——, *On the exact complexity of evaluating quantified k-*cnf, in Parameterized and Exact Computation, V. Raman and S. Saurabh, eds., vol. 6478 of Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 2010, pp. 50–59. 10.1007/978-3-642-17493-3_7.

[7] A. K. Chandra, M. L. Furst, and R. J. Lipton, *Multi-party protocols*, in Proceedings of the 15th Annual ACM Symposium on Theory of Computing, 25-27 April, 1983, Boston, Massachusetts, USA, 1983, pp. 94–99.

[8] R. Chen, V. Kabanets, A. Kolokolova, R. Shaltiel, and D. Zuckerman, *Mining circuit lower bound proofs for meta-algorithms*, computational complexity, 24 (2015), pp. 333–392.

[9] S. Chen and P. A. Papakonstantinou, *Depth-reduction for composites*, 2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS), (2016), pp. 99–108.

[10] S. Cook, *The complexity of theorem-proving procedures*, in Proceedings of the Third Annual ACM Symposium on Theory of Computing, 1971, pp. 151–158.

[11] M. Cygan, F. V. Fomin, L. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh, *Parameterized Algorithms*, Springer, 2015.

[12] E. Dantsin and E. A. Hirsch, *Worst-case upper bounds*, in Handbook of Satisfiability, 2009, pp. 403–424.

[13] M. de Oliveira Oliveira, *Size-treewidth tradeoffs for circuits computing the element distinctness function*, in 33rd Symposium on Theoretical Aspects of Computer Science, STACS 2016, February 17-20, 2016, Orléans, France, 2016, pp. 56:1–56:14.

[14] R. Diestel, *Graph Theory*, Springer, Berlin, second ed., electronic ed., February 2000.

[15] F. V. Fomin and D. Kratsch, *Exact Exponential Algorithms*, Texts in Theoretical Computer Science. An EATCS Series, Springer, 2010.

[16] F. V. Fomin, D. Lokshtanov, N. Misra, and S. Saurabh, *Planar f-deletion: Approximation and optimal FPT algorithms*, CoRR, abs/1204.4230 (2012).

[17] ——, *Planar f-deletion: Approximation, kernelization and optimal FPT algorithms*, in 53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2012, New Brunswick, NJ, USA, October 20-23, 2012, 2012, pp. 470–479.

[18] T. Hertli, *3-sat faster and simpler - unique-sat bounds for ppsz hold in general*, in Proceedings of the 52nd Annual IEEE Symposium on Foundations of Computer Science, oct. 2011, pp. 277 –284.

[19] R. Impagliazzo, W. Matthews, and R. Paturi, *A Satisfiability Algorithm for* $\mathbf{AC}^0$, in Proceedings of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms, 2012.

[20] R. Impagliazzo, R. Paturi, and S. Schneider, *A satisfiability algorithm for sparse depth two threshold circuits*, in Proceedings of the 54rd Annual IEEE Symposium on Foundations of Computer Science, 2013.

[21] K. Iwama, K. Seto, T. Takai, and S. Tamaki, *Improved randomized algorithms for 3-sat*, in Algorithms and Computation, O. Cheong, K.-Y. Chwa, and K. Park, eds., vol. 6506 of Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 2010, pp. 73–84.

[22] E. J. Kim, A. Langer, C. Paul, F. Reidl, P. Rossmanith, I. Sau, and S. Sikdar, *Linear kernels and single-exponential algorithms via protrusion decompositions*, ACM Trans. Algorithms, 12 (2016), pp. 21:1–21:41.

[23] E. Kushilevitz and N. Nisan, *Communication complexity*, Cambridge University Press, 1997.

[24] L. Levin, *Universal sorting problems*, Problems of Information Transmission, 9 (1973), pp. 265–266.

[25] D. Lokshtanov, R. Paturi, S. Tamaki, R. Williams, and H. Yu, *Beating brute force for systems of polynomial equations over finite fields*, in Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA '17, Philadelphia, PA, USA, 2017, Society for Industrial and Applied Mathematics, pp. 2190–2202.

[26] B. Monien and E. Speckenmeyer, *Solving satisfiability in less than 2^n steps*, Discrete Appl. Math., 10 (1985), pp. 287–295.

[27] R. Moser and D. Scheder, *A Full Derandomization of Schöning's k-SAT Algorithm*, in Proceedings of the Forty-Fourth Annual ACM Symposium on Theory of Computing, 2011.

[28] E. I. Nečiporuk, *On a boolean function*, Doklady Akademii Nauk SSSR, 169 (1966), pp. 765–.

[29] R. Paturi, P. Pudlák, M. Saks, and F. Zane, *An improved exponential-time algorithm for k-sat*, Journal of the ACM, 52 (2005), pp. 337–364. Preliminary version in *39th Annual IEEE Symposium on Foundations of Computer Science*, pages 628–637, 1998.

[30] R. Paturi, P. Pudlák, and F. Zane, *Satisfiability coding lemma*, Chicago Journal of Theoretical Computer Science, (1999). Preliminary version in *38th Annual Symposium on Foundations of Computer Science*, 566-574, 1997.

[31] J. M. Robson, *Algorithms for maximum independent sets*, Journal of Algorithms, 7 (1986), pp. 425–440.

[32] R. Santhanam, *Fighting perebor: New and improved algorithms for formula and qbf satisfiability*, in Proceedings of the 2010 IEEE 51st Annual Symposium on Foundations of Computer Science, FOCS '10, Washington, DC, USA, 2010, IEEE Computer Society, pp. 183–192.

[33] ——, *Fighting perebor: New and improved algorithms for formula and qbf satisfiability*, in Proceedings of the 2010 IEEE 51st Annual Symposium on Foundations of Computer Science, FOCS '10, Washington, DC, USA, 2010, IEEE Computer Society, pp. 183–192.

[34] ——, *Ironic complicity: Satisfiability algorithms and circuit lower bounds*, Bulletin of the EATCS, 106 (2012), pp. 31–52.

[35] U. Schöning, *A probabilistic algorithm for k -sat based on limited local search and restart*, Algorithmica, 32 (2002), pp. 615–623.

[36] R. Schuler, *An algorithm for the satisfiability problem of formulas in conjunctive normal form*, Journal of Algorithms, 54 (2005), pp. 40–44.

[37] K. Seto and S. Tamaki, *A satisfiability algorithm and average-case hardness for formulas over the full binary basis*, computational complexity, 22 (2013), pp. 245–274.

[38] R. Williams, *Algorithms for quantified boolean formulas*, in SODA, 2002, pp. 299–307.

[39] ——, *Improving exhaustive search implies superpolynomial lower bounds*, in Proceedings of the 42nd ACM symposium on Theory of computing, STOC '10, New York, NY, USA, 2010, ACM, pp. 231–240.

[40] ——, *Non-Uniform ACC Circuit Lower Bounds*, in Proceedings of the Twenty-Sixth Annual IEEE Conference on Computational Complexity, 2011.