Artificial Intelligence 295 (2021) 103460

Contents lists available at ScienceDirect

Artificial Intelligence

www.elsevier.com/locate/artint

New width parameters for SAT and #SAT ☆

Robert Ganian *, Stefan Szeider *

Algorithms and Complexity Group, TU Wien, Favoritenstrasse 9-11, 1040 Wien, Austria

ARTICLE INFO

Article history:
Received 15 April 2020
Received in revised form 2 December 2020
Accepted 26 January 2021
Available online 29 January 2021

Keywords:
SAT
Model counting
Parameterized complexity
Treewidth
Community structure

ABSTRACT

We study the parameterized complexity of the propositional satisfiability (SAT) and the more general model counting (#SAT) problems and obtain novel fixed-parameter algorithms that exploit the structural properties of input formulas. In the first part of the paper, we parameterize by the treewidth of the following two graphs associated with CNF formulas: the consensus graph and the conflict graph. Both graphs have as vertices the clauses of the formula; in the consensus graph two clauses are adjacent if they do not contain a complementary pair of literals, while in the conflict graph two clauses are adjacent if they do contain a complementary pair of literals. We show that #SAT is fixed-parameter tractable when parameterized by the treewidth of the former graph, but SAT is W[1]-hard when parameterized by the treewidth of the latter graph.

In the second part of the paper, we turn our attention to a novel structural parameter we call h-modularity which is loosely inspired by the well-established notion of community structure. The new parameter is defined in terms of a partition of clauses of the given CNF formula into strongly interconnected communities which are sparsely interconnected with each other. Each community forms a hitting formula, whereas the interconnections between communities form a graph of small treewidth. Our algorithms first identify the community structure and then use them for an efficient solution of SAT and #SAT, respectively.

1. Introduction

Propositional model counting (#SAT) is the problem of determining the number of models (satisfying truth assignments) of a given propositional formula in conjunctive normal form (CNF). This problem arises in several areas of artificial intelligence, in particular in the context of probabilistic reasoning [2,23,46,51]. The problem is well-known to be #P-complete [55], and remains #P-hard even for monotone 2CNF formulas and Horn 2CNF formulas. Thus, solving #SAT by exploiting the syntax of instances requires stronger restrictions than when aiming for SAT.

An alternative to restricting the syntax is to impose *structural restrictions* on the input formulas. Structural restrictions can be applied in terms of structural parameters, often defined on natural graphical models of CNF formulas. Among the most frequently used graphical models are *primal graphs* (sometimes called *variable interaction graphs* or VIGs), *dual graphs*, and *incidence graphs* (see Fig. 1 for definitions and examples).

☆ Preliminary and shortened versions of the results presented in this submission appeared in the proceedings of the SAT conference [19,20]. This article expands the exposition of those papers by providing full proofs, more detailed explanations, and a broader explanation of the context of the results.

* Corresponding authors.
E-mail addresses: rganian@ac.tuwien.ac.at (R. Ganian), sz@ac.tuwien.ac.at (S. Szeider).

https://doi.org/10.1016/j.artint.2021.103460

The most widely studied and prominent graph parameter is *treewidth*, which was introduced by Robertson and Seymour in their Graph Minors Project. Small treewidth indicates that a graph resembles a tree in a certain sense (e.g., trees have treewidth 1, cycles have treewidth 2, cliques on $k+1$ vertices have treewidth k). Many otherwise NP-hard graph problems are solvable in polynomial time for graphs of bounded treewidth. It is generally believed that many practically relevant problems actually do have low treewidth [7].

Depending on whether we consider the treewidth of the primal, dual, or incidence graph of a given CNF formula, we speak of the primal, dual, or incidence treewidth of the formula, respectively. It is known that the number of models of a CNF formula of size L with primal, dual, or incidence treewidth k can be computed in time $f(k)L^c$ for a computable function f and a constant c which is independent of k; in other words, #SAT is *fixed-parameter tractable* parameterized by primal, dual, or incidence treewidth (see, e.g., [49]). It is worth noting here that the study of treewidth-based algorithms is not restricted merely to model counting and propositional satisfiability: indeed, treewidth has been successfully applied to a large variety of problems which arise in artificial intelligence, such as constraint satisfaction [11,50], answer set programming [4,43], integer linear programming [18,22], abstract argumentation [14,27], and closed-world reasoning [24].

A completely different approach to understanding the structure of tractable #SAT instances is based on the notion of *community structure*. It is known that large networks often exhibit a certain structure, where nodes form strongly interconnected communities which are sparsely connected with each other; to what extent a network exhibits such a structure can be measured by its *modularity* [36,37,35,58]. Recently the community structure and modularity of practical SAT instances has been empirically studied, revealing an interesting correlation between the modularity and the solving time of state-of-the art SAT solvers. Interestingly, learnt clauses tend to lie within communities and learnt clauses of low Literal Block Distance (LBD) are shared by few communities [1,38]. These findings contribute towards a better understanding of the spectacular performance of today's SAT solvers on practical instances, which is generally not well understood and remains a challenge for the research community [56].

In this paper we consider structural parameters of CNF formulas which utilize treewidth to push beyond the state of the art of propositional model counting—both in terms of what was previously known about useful graphical models for #SAT instances, and in terms of what was known about the algorithmic applications of formulas exhibiting certain forms of community structure.

1.1. Algorithmically useful graphical models

In the first part of the paper, we consider the treewidth of two further graphical models: the *consensus graph* and the *conflict graph*, giving rise to the parameters *consensus treewidth* and *conflict treewidth*, respectively. Both graphs have as their vertices the clauses of the formula. In the consensus graph two clauses are adjacent if they do not contain a complementary pair of literals; in the conflict graph, two clauses are adjacent if they do contain a complementary pair of literals (see Fig. 1 later for examples). Here, we study the parameterized complexity of #SAT with respect to the new parameters and provide a comparison to known parameters under which #SAT is fixed-parameter tractable.

We remark that related notions have been studied in previous works. For instance, Kullmann [34] considered the symmetric conflict matrix of a CNF formula om m clauses, which is an $m \times m$ matrix where the entry at row i and column j gives the number of conflicts between the i-th and the j-th clause. This matrix is the adjacency matrix of the conflict multigraph [17], which arises from the conflict graph by adding as many parallel edges between two clauses as the clauses have conflicts. Scheder and Zumstein [52] considered conflict graphs in our sense, without parallel edges. The cited works [17,34,52] mainly focused on structural results on special classes of formulas and less on algorithms.

Our main result regarding consensus treewidth is a novel fixed-parameter algorithm for model counting (Theorem 1). The algorithm is based on dynamic programming along a tree decomposition of the consensus graph. This result is particularly interesting as none of the known parameters under which #SAT is fixed-parameter tractable dominates consensus treewidth, in the sense that there are instances of small consensus treewidth where all other known parameters (that yield the fixed-parameter tractability of #SAT) can be arbitrarily large (Proposition 3). Hence consensus treewidth pushes the state of the art for fixed-parameter tractability of #SAT further, and moreover does so via a parameter that forms a natural counterpart to the already established primal, dual and incidence treewidth parameters. We also note that the presented fixed-parameter algorithm generalizes the polynomial-time algorithm on hitting formulas [28–30,40] (see Subsection 2.2 in the preliminaries).

This positive result is complemented by our results on conflict treewidth: we show that conflict treewidth in its general form does not provide a parameter under which #SAT is fixed-parameter tractable, even for formulas without pure literals (subject to the well-established complexity theoretic assumption $\mathsf{W}[1] \neq \mathsf{FPT}$ [13]). In fact, we show that already the decision problem SAT for formulas without pure literals is W[1]-hard when parameterized by conflict treewidth, or even by a weaker parameter, the size of a smallest vertex cover of the conflict graph (Proposition 1).

1.2. An algorithmically applicable notion of modularity

The presence of a community structure with low modularity as exhibited in the mentioned studies [1,38] is not a *guarantee* for an instance to be easy; instead, the correlation between modularity and solving time is of statistical nature.

In fact, it is not difficult to show that SAT remains NP-hard for highly modular instances. More specifically, given any SAT formula F, one can use a padding process (i.e., the addition of multiple variable-disjoint dense satisfiable subformulas) to create an equisatisfiable formula F' whose size is linear in F and whose modularity can be better than any arbitrarily fixed threshold.

Here, we introduce the notion of *h-modularity* for CNF formulas and show that, when used as a parameter, it gives rise to a single-exponential fixed-parameter algorithm for SAT and a double-exponential fixed-parameter algorithm for #SAT. h-modularity is designed to naturally exhibit the core properties of modularity (i.e., having variables clustered into communities with strong connections inside each community but less connections between different communities) while also ensuring that formulas of small h-modularity have structural properties that can be used to obtain good worst-case runtime guarantees (the padding argument mentioned above shows that density of clauses or connections is not sufficient).

In particular, h-modularity is defined based on the partition of the set of clauses into subsets, which we call *h-communities*. Each h-community forms a strongly interconnected set of clauses. This is ensured by the requirement that any two clauses of an h-community "clash" in at least one variable (i.e., h-communities are hitting formulas, cf. Subsection 2.2). Furthermore, the h-communities are sparsely interconnected with each other, which is ensured by the requirement that a certain graph which represents the interaction between h-communities has small treewidth as well as h-communities are of small degree (graphs of small treewidth are sparse [45,31]). A formal definition of h-modularity is given in Section 5. Somewhat related to h-modularity is the notion of modular incidence treewidth of a CNF formula [42], which is the treewidth of the incidence graph after contraction of modules (i.e., sets of vertices with the same neighborhood). SAT is not fixed-parameter tractable parameterized by the modular treewidth of the input formula (unless $\mathsf{W}[1] \neq \mathsf{FPT}$). However, if the modular treewidth is bounded by a constant, then the number of satisfying assignments can be computed in polynomial time [42].

We show that h-modularity is incomparable with the parameters signed clique-width and clustering-width, hence h-modularity is not dominated by any well-known parameter that admits fixed-parameter tractability of SAT or #SAT. As a consequence, our algorithms which use h-modularity also push the frontiers of tractability for SAT and #SAT. We remark that our algorithms do not assume that the relevant "decompositions" (i.e., a suitable partitioning into h-communities) are provided on the input.

The connections between the newly introduced parameters with previously studied structural parameters that yield tractability for SAT and #SAT are summarized in Section 6.

2. Preliminaries

The set of natural numbers (that is, positive integers) will be denoted by $\mathbb{N}$. For $i \in \mathbb{N}$ we write $[i]$ to denote the set $\{1, \ldots, i\}$.

2.1. SAT and #SAT

We consider propositional formulas in conjunctive normal form (CNF), represented as sets of clauses. That is, a *literal* is a (propositional) variable x or a negated variable $\overline{x}$; a *clause* is a finite set of literals not containing a complementary pair x and $\overline{x}$; a *formula* is a finite set of clauses.

For a literal $l = \overline{x}$ we write $\overline{l} = x$; for a clause C we set $\overline{C} = \{\overline{l} : l \in C\}$. For a clause C, $\mathrm{var}(C)$ denotes the set of variables x with $x \in C$ or $\overline{x} \in C$, and the *width* of C is $|\mathrm{var}(C)|$. Similarly, for a formula F we write $\mathrm{var}(F) = \bigcup_{C \in F} \mathrm{var}(C)$. The *length* of a formula F is the total number of literals it contains, i.e., $\sum_{C \in F} |\mathrm{var}(C)|$. We say that two clauses C, D *overlap* if $C \cap D \neq \emptyset$; we say that C and D *clash* if C and $\overline{D}$ overlap. Note that two clauses can clash and overlap at the same time. Two clauses C, D are *adjacent* if $\mathrm{var}(C) \cap \mathrm{var}(D) \neq \emptyset$. A variable is *pure* if it only occurs as either a positive literal or as a negative literal; the literals of a pure variable are then called pure literals.

The *dual graph* of a formula F is the graph whose vertices are clauses of F and whose edges are defined by the adjacency relation of clauses. We will also make references to the *primal graph* and the *incidence graph* of a formula F. The former is the graph whose vertices are the variables of F and where two variables a, b are adjacent if and only if there exists a clause C such that $a, b \in \mathrm{var}(C)$, while the latter is the graph whose vertices are the variables and clauses of F and where two vertices a, b are adjacent if and only if a is a clause and $b \in \mathrm{var}(a)$ (see Fig. 1 for an illustration).

A *truth assignment* (or *assignment*, for short) is a mapping $\tau : X \to \{0, 1\}$ defined on some set X of variables. We extend τ to literals by setting $\tau(\overline{x}) = 1 - \tau(x)$ for $x \in X$. $F[\tau]$ denotes the formula obtained from F by removing all clauses that contain a literal x with $\tau(x) = 1$ and by removing from the remaining clauses all literals y with $\tau(y) = 0$; $F[\tau]$ is the *restriction* of F to τ. Note that $\mathrm{var}(F[\tau]) \cap X = \emptyset$ holds for every assignment $\tau : X \to \{0, 1\}$ and every formula F. An assignment $\tau : X \to \{0, 1\}$ *satisfies* a formula F if $F[\tau] = \emptyset$. A truth assignment $\tau : \mathrm{var}(F) \to \{0, 1\}$ that satisfies F is a *model* of F. We denote by $\#(F)$ the number of models of F. A formula F is *satisfiable* if $\#(F) > 0$. In the SAT problem, we are given a formula F and the task is to determine whether F is satisfiable. In the #SAT problem, we are also given a formula F and the task is to compute $\#(F)$.

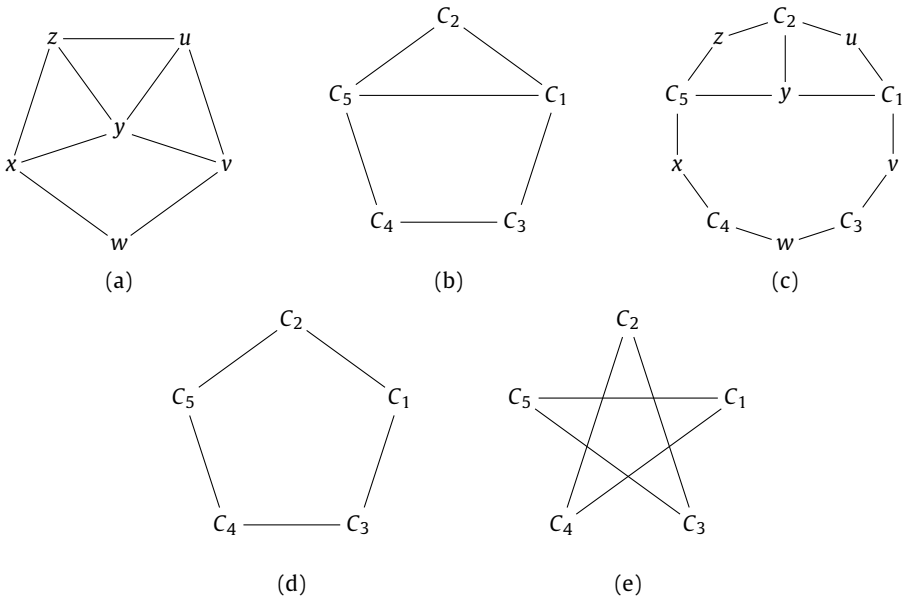

Fig. 1. The primal graph (a), dual graph (b), incidence graph (c), conflict graph (d) and consensus graph (e) of the formula $\{C_1, \ldots, C_5\}$ with $C_1 = \{u, \overline{v}, y\}$, $C_2 = \{\overline{u}, z, \overline{y}\}$, $C_3 = \{v, \overline{w}\}$, $C_4 = \{w, \overline{x}\}$, $C_5 = \{x, y, \overline{z}\}$. (a) The *primal graph* has as vertices the variables of the given formula, two variables are joined by an edge if they occur together in a clause. (b) The *dual graph* has as vertices the clauses, two clauses are joined by an edge if they share a variable. (c) The *incidence graph* is a bipartite graph where one vertex class consists of the clauses and the other consists of the variables; a clause and a variable are joined by an edge if the variable occurs in the clause. (d) The *conflict graph* has as vertices the clauses of the formula, two clauses are joined by an edge if they do contain a complementary pair of literals. (e) The *consensus graph* has as vertices the clauses of the formula, two clauses are joined by an edge if they do not contain a complementary pair of literals.

2.2. Hitting formulas

A *hitting formula* is a CNF formula with the property that any two of its clauses clash (see [28,29,40]). The same notion for DNF formulas is termed orthogonality [10]. The following result makes hitting formulas particularly attractive in the context of SAT and #SAT.

Fact 1 *([26]).* A hitting formula F with n variables has exactly $2^n - \sum_{C \in F} 2^{n-|C|}$ models.

The following observation will be implicitly used in several of our proofs.

Fact 2. Let F be a hitting formula, and let F' be obtained from F by an arbitrary sequence of clause deletions and restrictions under truth assignments. Then F' is also a hitting formula.

2.3. Parameterized complexity

Next we give a brief and rather informal review of the most important concepts of parameterized complexity. For an in-depth treatment of the subject we refer the reader to other sources [13,39].

The instances of a parameterized problem can be considered as pairs (I, k) where I is the *main part* of the instance and k is the *parameter* of the instance; the latter is usually a non-negative integer. A parameterized problem is *fixed-parameter tractable* (FPT) if instances (I, k) of size n (with respect to some reasonable encoding) can be solved in time $f(k)n^c$ where f is a computable function and c is a constant independent of k. Such algorithms are called *fixed-parameter algorithms*, and the function f is called the *parameter dependence*.

To obtain our lower bounds, we will need the notion of a parameterized reduction. Let L_1, L_2 be parameterized problems. A *parameterized reduction* (or fpt-reduction) from L_1 to L_2 is a mapping P from instances of L_1 to instances of L_2 such that

1. $(x, k) \in L_1$ if and only if $P(x, k) \in L_2$,
2. the mapping can be computed by a fixed-parameter algorithm w.r.t. parameter k, and
3. there is a computable function g such that $k' \leq g(k)$, where $(x', k') = P(x, k)$.

The class W[1] captures parameterized intractability and contains all parameterized decision problems that are fpt-reducible to Multicolored Clique (defined below) [13]. Showing W[1]-hardness for a problem rules out the existence of a fixed-parameter algorithm under the well-established assumption that W[1] $\neq$ FPT.

MULTICOLORED CLIQUE
Instance: A k-partite graph $G = (V, E)$ with a partition $V_1, \ldots, V_k$ of V.
Parameter: The integer k.
Question: Are there vertices $v_1, \ldots, v_k$ such that $v_i \in V_i$ and $\{v_i, v_j\} \in E$ for all i and j with $1 \leq i < j \leq k$ (i.e. the subgraph of G induced by $\{v_1, \ldots, v_k\}$ is a clique of size k)?

2.4. Treewidth

Let G be a simple, undirected, finite graph with vertex set $V = V(G)$ and edge set $E = E(G)$. A *tree decomposition* of G is a pair $(T, \{B_i : i \in I\})$ where $B_i \subseteq V$, T is a tree, and $I = V(T)$ such that:

1. for each edge $uv \in E$, there is an $i \in I$ such that $\{u, v\} \subseteq B_i$, and
2. for each vertex $v \in V$, $T[\{i \in I : v \in B_i\}]$ is a (connected) tree with at least one node.

The *width* of a tree decomposition is $\max_{i \in I} |B_i| - 1$. The *treewidth* [31,44] of G is the minimum width taken over all tree decompositions of G and it is denoted by $\mathbf{tw}(G)$. We call the elements of I *nodes* and B_i *bags*. As an example, consider the graphs depicted in Fig. 1: graphs (b), (d), (e) have treewidth 2, while graphs (a) and (c) have treewidth 3.

While it is possible to compute the treewidth exactly using a fixed-parameter algorithm [8], the asymptotically best running time is achieved by using the recent state-of-the-art 5-approximation algorithm of Bodlaender et al. [6]. We summarize below.

Fact 3 *([6]).* There exists an algorithm which, given an n-vertex graph G and an integer k, in time $2^{\mathcal{O}(k)} \cdot n$ either outputs a tree decomposition of G of width at most $5k + 4$ and $\mathcal{O}(n)$ nodes, or correctly determines that $\mathbf{tw}(G) > k$. Moreover, in time $k^{\mathcal{O}(k^3)} \cdot n$ it is possible to either output a tree decomposition of G of width at most k and $\mathcal{O}(n)$ nodes, or correctly determine that $\mathbf{tw}(G) > k$.

For other standard graph-theoretic notions not defined here, we refer to a standard textbook [12]. It is well known that, for every clique over $Z \subseteq V(G)$ in G, it holds that every tree decomposition of G contains an element B_i such that $Z \subseteq B_i$ [31]. Furthermore, if i separates a node j from another node l in T, then B_i separates $B_j \setminus B_i$ from $B_l \setminus B_i$ in G [31]; this *inseparability property* will be useful in some of our later proofs.

A tree decomposition $(T, \mathcal{B})$ of a graph G is *nice* if the following conditions hold:

1. T is rooted at a specifically marked node r, and $B_r = \emptyset$.
2. Every node of T has at most two children.
3. If a node t of T has two children t_1 and t_2, then $B_t = B_{t_1} = B_{t_2}$; in that case we call t a *join node*.
4. If a node t of T has exactly one child t', then exactly one of the following holds:
   (a) $|B_t| = |B_{t'}| + 1$ and $B_{t'} \subset B_t$; in that case we call t an *introduce node*.
   (b) $|B_t| = |B_{t'}| - 1$ and $B_t \subset B_{t'}$; in that case we call t a *forget node*.
5. If a node t of T is a leaf, then $|B_t| = 1$; we call these *leaf nodes*.

The main advantage of nice tree decompositions is that they allow the design of much more transparent dynamic programming algorithms, since one only needs to deal with four very specific types of nodes. It is well known (and easy to see) that for every fixed k, given a tree decomposition of a graph $G = (V, E)$ of width at most k and with $\mathcal{O}(|V|)$ nodes, one can construct in linear time a nice tree decomposition of G with $\mathcal{O}(|V|)$ nodes and width at most k [5]. We say that a vertex v was *forgotten* below a node $t \in V(T)$ if the subtree rooted at t contains a (forget) node s with a child s' such that $B_{s'} \setminus B_s = \{v\}$.

Finally, we summarize known algorithms for SAT and #SAT when parameterized by the treewidth of the three natural graph representations discussed in previous Subsection 2.1; we note that the original results assumed that a tree decomposition is supplied as part of the input, and we can obtain one using Fact 3 (even while retaining the running time bounds).

Fact 4 *([49,25]).* #SAT is FPT when parameterized by the treewidth of any of the following graphical models of the formula: the incidence graph, the primal graph, or the dual graph.

2.5. Other structural parameters of CNF formulas

The *signed incidence graph* of a CNF formula F is the incidence graph of F where additionally each edge between a variable x and a clause c is associated with the label "+" if x occurs as a positive literal in c, and analogously each edge between a variable x and a clause c is associated with the label "-" if x occurs as a negative literal in c.

Let k be a positive integer. A *signed k-graph* is a graph whose vertices are labeled by $[k]$; formally, the graph is equipped with a labeling function $\gamma : V(G) \to [k]$. We consider an arbitrary signed graph as a signed k-graph with all vertices labeled by 1. We call the signed k-graph consisting of exactly one vertex v (say, labeled by i) an initial signed k-graph and denote it by $i(v)$. The *signed clique-width* of a signed graph G is the smallest integer k such that G can be constructed from initial signed k-graphs by means of repeated application of the following three operations:

1. Disjoint union (denoted by $\oplus$);
2. Relabeling: changing all labels i to j (denoted by $p_{i\to j}$);
3. "+"-edge insertion: adding an edge labeled "+" between each vertex labeled by i and each vertex labeled by j, where $i \neq j$;
4. "-"-edge insertion: adding an edge labeled "-" between each vertex labeled by i and each vertex labeled by j, where $i \neq j$.

The signed clique-width of a CNF formula F is the signed clique-width of its signed incidence graph [54,9]. We also note that a decomposition for signed clique-width can be approximated by using signed rank-width [21].

The *clustering-width* [40] of a CNF formula F is the smallest number of variables whose deletion results in a variable-disjoint union of hitting formulas.

3. Consensus treewidth

Recall that the consensus graph of a CNF formula F is the graph G whose vertices are the clauses of F and which contains an edge ab if and only if clauses a and b do not clash. Observe that the consensus graph of a hitting formula is edgeless. The consensus treewidth of F, denoted $\mathbf{contw}(F)$, is then the treewidth of its consensus graph.

We now proceed to the main algorithmic result of this section. Our algorithm will in certain cases invoke the previously known algorithm [49] for #SAT parameterized by dual treewidth as a subroutine, and so we provide the full statement of its runtime below. We note that the runtime of that algorithm depends on the time required to multiply two n-bit integers, denoted δ.

Fact 5 *([49]).* Given a nice tree decomposition $(T, \mathcal{B})$ of the dual graph of a formula F, #SAT can be solved in time $\mathcal{O}(2^k(k\ell + \delta)N)$, where N is the number of nodes of T, k is its width, and ℓ is the maximum width of a clause in F.

In the literature there exist several algorithms for multiplying two n-bit integers; we refer the interested reader to Knuth's in-depth overview [32]. One of the most prominent of these algorithms is due to Schönhage and Strassen [32] and runs in time $\mathcal{O}(n \log n \log\log n)$. An even faster algorithm was later obtained by Fürer [16].

Theorem 1. *#SAT can be solved in time $2^{\mathcal{O}(k)} \cdot L(L + \delta)$, where L is the length of the formula, δ is the time required to multiply two L-bit integers, and k is the consensus treewidth.*[1]

Proof. Let F be an input formula over n variables, and let G be its consensus graph. Let $(T, \mathcal{B})$ be a nice tree decomposition of G of width at most $5k + 4$; recall that such $(T, \mathcal{B})$ can be computed in time $2^{\mathcal{O}(k)}$ by Fact 3. For brevity, we will use the following terminology: for a node t with bag B_t and a clause set $X \subseteq B_t$, we say that an assignment is X^t-validating if it satisfies all clauses in X but does not satisfy any clause in $B_t \setminus X$. For instance, if $X = \emptyset$ then a X^t-validating assignment cannot satisfy any clause in B_t, while if $X = B_t$ then a X^t-validating assignment must satisfy every clause in B_t.

Consider the following leaf-to-root dynamic programming algorithm $\mathcal{A}$ on T. At each bag B_t associated with a node t of T, $\mathcal{A}$ will compute two mappings φ_t^+, $\varphi_t^\sim$, each of which maps each $X \subseteq B_t$ to an integer between 0 and 2^n. These mappings will be used to store the number of X^t-validating assignments of $\mathrm{var}(F)$ under an additional restriction:

- in φ_t^+, we count only assignments which satisfy all clauses that were already forgotten below t, and
- in $\varphi_t^\sim$, we count only assignments which invalidate at least one clause that was already forgotten below t.

Since we assume that the root r of a nice tree decomposition is an empty bag, the total number of satisfying assignments of F is equal to $\varphi_r^+(\emptyset)$. The purpose of also keeping records for $\varphi_t^\sim$ will become clear during the algorithm; in particular, they will be needed to correctly determine the records for φ_t^+ at certain stages.

At each node t, let σ_t be the set of clauses which were forgotten below t; for example, $\sigma_r = F$ and $\sigma_\ell = \emptyset$ for each leaf ℓ of T. We now proceed to explain how $\mathcal{A}$ computes the mappings φ_t^+, $\varphi_t^\sim$ at each node t of T, starting from the leaves, along with arguing correctness of the performed operations.

[1] If arithmetic operations are assumed to have constant runtime, that is, $\delta = \mathcal{O}(1)$, then we obtain an upper bound on the runtime of $2^{\mathcal{O}(k)} \cdot L^2$.

1. *Leaf nodes.* Since σ_t is empty, $\varphi_t^{\sim}$ will map each subset of B_t to 0. As for φ_t^{+}, we observe that there are precisely $2^{n-|c|}$ many assignments which invalidate a clause $c \in B_t$. Hence we correctly set $\varphi_t^{+}(c) = 2^{n-|c|}$ and $\varphi_t^{+}(\emptyset) = 2^n - 2^{n-|c|}$.

2. *Forget nodes.* Let t be a forget node with child p and let $B_p \setminus B_t = \{c\}$. We begin by observing that the number of X^t-validating assignments which satisfy all clauses in σ_t is precisely equal to the number of $(X \cup \{c\})^p$-validating assignments which satisfy all clauses in σ_p. In other words, for each $X \subseteq B_t$ we correctly set $\varphi_t^{+}(X) = \varphi_p^{+}(X \cup \{c\})$.
   On the other hand, X^t-validating assignments which do not satisfy at least one clause in σ_t are partitioned into the following mutually exclusive cases:

   (a) $(X \cup \{c\})^p$-validating assignments which do not satisfy at least one clause in σ_p;
   (b) X^p-validating assignments which do not satisfy at least one clause in σ_p;
   (c) X^p-validating assignments which satisfy all clauses in σ_p.

   Hence, we correctly set $\varphi_t^{\sim}(X) = \varphi_p^{\sim}(X \cup \{c\}) + \varphi_p^{\sim}(X) + \varphi_p^{+}(X)$.

3. *Join nodes.* Let t be a join node with children p, q. Recall that $\sigma_p \cap \sigma_q = \emptyset$ and $\sigma_t = \sigma_p \cup \sigma_q$ due to the properties of tree decompositions. Furthermore, an assignment satisfies all clauses in σ_t if and only if it satisfies all clauses in both σ_p and σ_q. In other words, X^t-validating assignments which do not satisfy at least one clause in σ_t are partitioned into the following mutually exclusive cases (recall that $B_p = B_q$ by the definition of join nodes):

   (a) X^p-validating assignments which do not satisfy at least one clause in σ_p but satisfy all clauses in σ_q;
   (b) X^p-validating assignments which do not satisfy at least one clause in σ_q but satisfy all clauses in σ_p;
   (c) X^p-validating assignments which invalidate at least one clause in σ_p and also at least one clause in σ_q.

   Recall that B_t is a separator between σ_p and σ_q, which means that every clause in σ_p clashes with every clause in σ_q. That in turn implies that the number of assignments covered by point 3c must be equal to 0: every assignment that does not satisfy at least one clause in one of σ_p, σ_q must satisfy all clauses in the other set. Since we now know that every assignment which does not satisfy a clause in σ_p must satisfy all clauses in σ_q and vice-versa, we can correctly set $\varphi_t^{\sim}(X) = \varphi_p^{\sim}(X) + \varphi_q^{\sim}(X)$. Finally, to compute $\varphi_t^{+}(X)$ we can subtract $\varphi_t^{\sim}(X)$ from the total number of X^t-validating assignments (which is equal to the sum of $\varphi_p^{+}(X)$ and $\varphi_p^{\sim}(X)$ and hence is known to us), i.e., we set $\varphi_t^{+}(X) = \varphi_p^{+}(X) + \varphi_p^{\sim}(X) - \varphi_t^{\sim}(X)$.

4. *Introduce nodes.* Let t be an introduce node with child p and let $B_t = B_p \cup \{c\}$. For each $X \subseteq B_p$, we consider two cases and proceed accordingly. On one hand, if $\varphi_p^{\sim}(X) = 0$ (i.e., there exists no X^p-validating assignment invalidating at least one clause in σ_p), then clearly $\varphi_p^{\sim}(X) = \varphi_t^{\sim}(X) + \varphi_t^{\sim}(X \cup \{c\}) = 0$ and in particular $\varphi_t^{\sim}(X) = \varphi_t^{\sim}(X \cup \{c\}) = 0$. On the other hand, assume $\varphi_p^{\sim}(X) > 0$ and consider a X^p-validating assignment α which invalidates at least one clause in σ_p. Since c clashes with all clauses in σ_p, it follows that α must satisfy c. Consequently, we correctly set $\varphi_t^{\sim}(X) = \varphi_p^{\sim}(X \cup c)$ and $\varphi_t^{\sim}(X) = 0$. Since each subset of B_t is a subset of $B_p \cup \{c\}$, it follows that using the above rules $\mathcal{A}$ has computed the mapping $\varphi_t^{\sim}$ for all $X' \subseteq B_t$.
   The last remaining step is to compute $\varphi_t^{+}(X')$ for each $X' \subseteq B_t$. In order to do so, we will first use Fact 5 to compute the number $s_{X'}$ of all X'^t-validating assignments of F. Since we are now interested in assignments which must invalidate all clauses in $B_t \setminus X'$, we can construct the subformula F' from F by

   (a) removing all clauses except for those in X', i.e., $F' := X'$, and
   (b) assigning all variables which occur in $B_t \setminus X'$ in order to invalidate clauses outside of X'. Formally, for each clause $c \in B_t \setminus X'$, we apply the partial assignment $x \mapsto 0$ whenever $x \in c$ and the partial assignment $x \mapsto 1$ whenever $\overline{x} \in c$. If a contradiction arises for some variable, then we know that there exists no X'-validating assignment and hence set $s_{X'} = 0$.

   Clearly, F' can be constructed in time $\mathcal{O}(L)$ and satisfies $\#F' = s_{X'}$. Furthermore, since F' contains at most k clauses, we can construct a trivial nice tree decomposition of F' of width at most k containing at most $2k+1$ nodes in linear time by first consecutively introducing all of its nodes and then consecutively forgetting them. With this decomposition in hand, we invoke Fact 5 to compute $\#F'$ in time at most $2^k(kL + \delta)(2k + 1)$, i.e., $2^{\mathcal{O}(k)} \cdot (L + \delta)$. Once we compute $s_{X'}$, we use the fact that $s_{X'} = \varphi_t^{\sim}(X) + \varphi_t^{+}(X)$ and correctly set $\varphi_t^{+}(X) = s_{X'} - \varphi_t^{\sim}(X)$.

Observe that the time requirements for performing the above-specified operations at individual nodes of T are dominated by the time requirements for processing introduce nodes, upper-bounded by $2^k \cdot (L + 2^{\mathcal{O}(k)} \cdot (L + \delta)) = 2^{\mathcal{O}(k)} \cdot (L + \delta)$. Furthermore, a nice tree decomposition with at most $\mathcal{O}(L)$ nodes and width at most $5k + 4$ can be obtained in time $2^{\mathcal{O}(k)} \cdot L$ by Fact 3. Hence we conclude that it is possible to compute $\varphi_r^{+}(\emptyset) = \#(F)$ in time at most $2^{\mathcal{O}(k)} \cdot L(L + \delta)$. The correctness of the whole algorithm follows from the correctness of computing the mappings $\varphi_t^{\sim}$ and φ_t^{+} at each node t in T. □

4. Conflict treewidth

The algorithmic application of the consensus graph, as detailed above, gives rise to a natural follow-up question: what can we say about its natural counterpart, the *conflict graph*? Recall that the conflict graph of a CNF formula F is the graph G whose vertices are the clauses of F and which contains an edge ab if and only if clauses a and b clash. Observe that the conflict graph of a hitting formula is a complete graph, and that the conflict graph is the complement graph of the consensus graph. The conflict treewidth of F is then the treewidth of its conflict graph.

Since the conflict graph is a subgraph of the dual graph, conflict treewidth can be much (and in fact arbitrarily) smaller than the dual treewidth. However, unlike the case of dual treewidth, we will show that SAT does not admit a fixed-parameter algorithm parameterized by conflict treewidth (unless $\mathsf{W}[1] \neq \mathsf{FPT}$).

Proposition 1. *SAT is* W[1]*-hard when parameterized by conflict treewidth. Furthermore, SAT remains* W[1]*-hard when parameterized by the size of a minimum vertex cover of the conflict graph, even for instances without pure literals.*

Proof. We provide a parameterized reduction from Multicolored Clique. Given an instance G of Multicolored Clique over vertex set $V = V_1 \cup \cdots \cup V_k$, we construct a formula F over the variable set V (i.e., each vertex in G is a variable in F). We add the following clauses to F (observe that F contains no pure literals):

1. for each $i \in [k]$, we add one clause containing one positive literal of each variable $x \in V_i$;
2. for each $i \in [k]$ and each distinct $x, y \in V_i$, we add one clause $\{\overline{x}, \overline{y}\}$;
3. for each non-edge between distinct vertices x, y in G, we add one clause $\{\overline{x}, \overline{y}\}$.

F can clearly be constructed from G in polynomial time. The intuition behind the construction is the following: variables set to true correspond to the vertices of a multicolored clique, clauses in groups 1 and 2 enforce the selection of a single vertex from each color class, and the remaining clauses ensure that the result is a clique.

To formally prove that the reduction is correct, consider a solution X to G, and consider the assignment α which sets variables in X to true and all other variables to false. Since X contains precisely one vertex from each color class V_i, α clearly satisfies all clauses in groups 1 and 2. Now consider any clause in group 3, and observe that it can only be invalidated if both of its variables are set to true. However, since X is a clique it must hold that for each pair of distinct variables $x, y \in C$ we'll never have a clause in group 3 between x and y, and hence in particular each such clause will always contain at least one variable that is set to false and that therefore satisfies it.

On the other hand, consider a satisfying assignment α for F. Then clauses in group 1 ensure that at least one variable is set to true in each color class, and clauses in group 2 ensure that at most one variable is set to true in each color class. Finally, clauses in group 3 prevent α from setting two variables to true if they are the endpoints of a non-edge in G. Consequently, the variables set to true by α must form a solution to the multicolored clique instance G.

Finally, we argue that the parameter values are bounded by k, as claimed by the hardness result. Observe that all literals in clause groups 2 and 3 are negative, which means that whenever two clauses clash, at least one of them must be in group 1. Furthermore, recall that there are precisely k clauses in group 1. Hence the clauses in group 1 form a vertex cover of size k in the conflict graph of F. It is well known (and easy to verify) that the vertex cover is an upper bound on the treewidth of a graph. □

Observe that Proposition 1 implies that there exist instances where the conflict treewidth is arbitrarily smaller than the incidence treewidth (since SAT is known to be FPT when parameterized by the latter). In fact, it is not difficult to also directly construct a class of formulas with this behavior: consider an arbitrary SAT instance containing only positive literals of variables $x_1, \ldots, x_n$, plus a single clause $(\overline{x_1}, \overline{x_2}, \ldots, \overline{x_n})$—graphs in this class may have arbitrarily large incidence treewidth, but their conflict graphs will be stars. On the other hand, we can show that in the case of formulas of bounded clause width and without pure literals, conflict treewidth (denoted **conflict-tw**) is dominated by incidence treewidth.

Proposition 2. *For any formula F with clauses of width at most d and without pure literals, it holds that* $\mathbf{itw}(F) \leq (d+1) \cdot (\mathbf{conflict\text{-}tw}(F) + 1)$.

Proof. Let G be the conflict graph of F and $(T, \mathcal{B})$ be a tree decomposition of G of width k. Consider the structure $(T, \mathcal{B}')$ obtained as follows: for each $B_i \in \mathcal{B}$, we create a set B'_i in $\mathcal{B}'$ where $B'_i = B_i \cup \{x : \exists c \in B_i : x \in \mathrm{var}(c)\}$. Informally, the set $\mathcal{B}'$ is obtained by extending the bags in $(T, \mathcal{B})$ by the variables that occur in the clauses of that bag. We claim that $(T, \mathcal{B}')$ is a tree decomposition of the incidence graph G' of F.

Towards proving this claim, first observe that T is still a tree and each $B'_i \in \mathcal{B}'$ is a subset of $V(G')$. Furthermore, for any edge ab of G' between a clause a and variable b, it must hold that $a \in B_i$ for some $B_i \in \mathcal{B}$. By construction, B'_i must then contain both a and b and so condition 1 of the definition of tree decompositions is satisfied. As for condition 2, assume first for a contradiction that some vertex $v \in G'$ is not contained in any bag of $(T, \mathcal{B}')$. This clearly cannot happen if v is a clause, and so v must be a variable; furthermore, since F contains no pure literals, v must occur in at least two clauses.

It remains to show that all bags containing v induce a connected subtree of T. So, let us assume once more for a contradiction that this is not the case. By construction of $(T, \mathcal{B}')$ this implies that $(T, \mathcal{B})$ must contain a node t such that B_t separates some set of clauses containing v, say X_1, from all remaining clauses containing v, say X_2. Next, observe that $X_1 \cup X_2$ forms a complete bipartite graph in G: indeed, one side consists of all clauses containing v as a literal, while the other side consists of all clauses containing $\overline{v}$. But these two facts together contradict the inseparability property of tree decompositions: $X_1 \cup X_2$ induce a connected subgraph of G', and yet they are supposedly separated by B_t which does not intersect $X_1 \cup X_2$. Hence we conclude that no such node B_t exists and that the bags containing v indeed induce a connected subtree of T.

We conclude the proof by observing that the size of each bag $B'_i \in \mathcal{B}'$ is equal to $d+1$ times $|B_i|$, since we added at most d extra vertices for each vertex in B_i. □

As a consequence of Proposition 2, restricted to formulas of bounded clause width, #SAT is FPT when parameterized by conflict treewidth, since in this case the parameter is dominated by incidence treewidth [49]. We note that the domination is strict: for each $i \in \mathbb{N}$ there exists a formula F_i of clause width 2 and without pure literals such that $\mathbf{itw}(F_i) = 1$ and $\mathbf{contw}(F_i) \geq i$. Indeed, one such example is the formula $F_i = \{\{y, x_1\}, \{\overline{x_1}\}, \{y, x_2\}, \{\overline{x_2}\}, \ldots, \{y, x_i\}, \{\overline{x_i}\}\} \cup \{\{\overline{y}, z_1\}, \{\overline{z_1}\}, \{\overline{y}, z_2\}, \{\overline{z_2}\}, \ldots, \{\overline{y}, z_i\}, \{\overline{z_i}\}\}$.

5. h-Communities and h-modularity

Let F be a formula. We call a hitting formula $H \subseteq F$ a *hitting community* (or *h-community* in brief) in F. The *degree* $\mathbf{deg}(H)$ of an h-community H is the number of edges in the dual graph of F between a clause in H and a clause outside of H. A *hitting community structure* (or *h-structure* in brief) $\mathcal{P}$ is a partitioning of F into h-communities, and the degree $\mathbf{deg}(\mathcal{P})$ of $\mathcal{P}$ is $\max\{\mathbf{deg}(H) : H \in \mathcal{P}\}$. Intuitively, these notions are based on connections in the dual graph, whereas for a clause C in F we let $\mathbf{deg}(C)$ be the degree of C in the dual graph of F. Moreover, let $\mathbf{deg}(F)$ be the maximum degree in the dual graph of F.

To measure the treewidth of an h-structure $\mathcal{P}$, we construct a *community graph* G as follows. The vertices of G are h-communities in $\mathcal{P}$, and two vertices A, B in G are adjacent if and only if there exist clauses $C \in A$ and $D \in B$ which are adjacent. Then we let $\mathbf{tw}(\mathcal{P}) = \mathbf{tw}(G)$.

We define the term *h-modularity* to measure how "well-structured" a particular h-structure is. The h-modularity $\mathbf{h\text{-}mod}(\mathcal{P})$ of an h-structure $\mathcal{P}$ is $\max\{\mathbf{deg}(\mathcal{P}), \mathbf{tw}(\mathcal{P})\}$. The h-modularity $\mathbf{h\text{-}mod}(F)$ of a formula F is then defined as the minimum $\mathbf{h\text{-}mod}(\mathcal{P})$ over all h-structures $\mathcal{P}$ of F.

An h-structure $\mathcal{P}$ of F is called a *witness* of $\mathbf{h\text{-}mod}(F) \leq k$ if $\mathbf{h\text{-}mod}(\mathcal{P}) \leq k$. Given an h-structure $\mathcal{P}$ of F and a subformula $F' \subseteq F$, we denote by $\mathcal{P}[F']$ the h-structure induced by $\mathcal{P}$ on F'; observe that $\mathbf{h\text{-}mod}(\mathcal{P}[F']) \leq \mathbf{h\text{-}mod}(\mathcal{P})$.

We introduce some additional notation which will be useful later, always w.r.t. a fixed h-structure. A clause $C \in H$ is a *bridge clause* if there exists a clause outside of H adjacent to C. A variable x is a *bridge variable* if it occurs in a clause in one h-community and at least one other clause in another h-community. Notice that every clause containing a bridge variable is a bridge clause, and that h-structures of low h-modularity can still contain a large number of bridge variables, even in a single h-community.

We now formally state the main algorithmic results of this section.

Theorem 2. *SAT can be solved in time $\mathcal{O}(L^3) + k^{\mathcal{O}(k^3)} \cdot L^2$ and #SAT can be solved in time $2^{2^{\mathcal{O}(k)}} \cdot L^{\mathcal{O}(1)}$, where L and k are the length and the h-modularity of the input formula, respectively.*

Our approach for proving Theorem 2 can be separated into two main tasks: first, we compute an h-structure $\mathcal{P}$ of small h-modularity, and then we use $\mathcal{P}$ to solve the problem. Our techniques to achieve this are discussed in detail in the following two subsections.

5.1. Finding h-structures

Our approach for finding h-structures of small h-modularity consists of two steps. Generally speaking, we introduce a preprocessing procedure which we exhaustively apply until all clauses have a sufficiently small degree (Lemma 1), and once the degree of all clauses is sufficiently small we compute a tree decomposition of the dual graph and use it to find a suitable h-structure (Lemma 2). One of the technical obstacles we have to overcome is that the preprocessing procedure given by Lemma 1 only guarantees the preservation of h-modularity up to a certain bound; this effectively means that the h-structure obtained by Lemma 2 cannot overapproximate the optimal h-modularity by too much, as otherwise the preprocessing step would be invalidated.

Lemma 1. *There exists an algorithm which, given $q \in \mathbb{N}$ and a formula F of length L containing a clause C such that $\mathbf{deg}(C) > 3q+2$, runs in time $\mathcal{O}(L^2)$ and either correctly determines that $\mathbf{h\text{-}mod}(F) > q$, or outputs a strictly smaller subformula F' with the following property: if $\mathbf{h\text{-}mod}(F') \leq q$, then $\mathbf{h\text{-}mod}(F) = \mathbf{h\text{-}mod}(F')$. Furthermore, a witness $\mathcal{P}$ of $\mathbf{h\text{-}mod}(F) \leq q$ can be computed from F, F' and a witness $\mathcal{P}'$ of $\mathbf{h\text{-}mod}(F') \leq q$ in linear time.*

Proof. Let Z_0 be the set containing C and all clauses which are neighbors of C, let Z_1 be the subset of Z_0 containing clauses which have a neighbor outside of Z_0, and let $Z = Z_0 \setminus Z_1$. Let W be the subset of Z containing clauses which have at least $q+2$ neighbors in Z. We now make a series of tests:

1. if $|Z_1| > q$, then $\textbf{h-mod}(F) > q$;
2. if $|W| < q+3$, then $\textbf{h-mod}(F) > q$;
3. if W is not a hitting formula, then $\textbf{h-mod}(F) > q$;
4. if Z contains a clause which clashes with exactly $|W|-1$ clauses in W, then $\textbf{h-mod}(F) > q$;
5. let $B \in W$ be a clause with no neighbors outside W; if no such B exists, then $\textbf{h-mod}(F) > q$.

Otherwise we set $F' = F \setminus B$.

We prove correctness. Observe that if $|Z_1| > q$ then there exists no $\mathcal{P}$ of h-modularity at most q. Indeed, for each neighbor D of Z_1 outside of Z_0, it holds that D and C cannot be in the same h-community, since they are not adjacent. Hence each element of Z_1 increases the degree of the h-community containing C by at least 1; either due to the edge between C and that element, or the edge between D and that element. Hence we can assume that $|Z| \geq 2q+3$.

For the second test, observe that if $|W| < q+3$ then there exists no $\mathcal{P}$ of h-modularity at most q. Indeed, since the number of neighbors of C in Z is at least $2q+2$, at least $q+2$ of these neighbors must be in the same h-community as C if $\textbf{h-mod}(\mathcal{P}) \leq q$. This implies that at least $q+2$ of these neighbors would have to be pairwise-adjacent, and in particular would each have at least $q+2$ neighbors in Z. Then W necessarily must contain C and at least $q+2$ neighbors of C.

For the third test, if W is not a hitting formula, then any h-structure $\mathcal{P}$ of h-modularity at most q would need to partition W into (subsets of) at least two h-structures; let H_C be the hypothetical h-community containing C, and let $D \in W \setminus H_C$. Since D has $q+2$ neighbors in Z, there are at least $q+2$ edge-disjoint paths between D and C, and each of these paths contributes at least 1 to the degree of H_C. But then it follows that $\textbf{deg}(H_C) \geq q+2$, which would contradict $\textbf{h-mod}(\mathcal{P}) \leq q$, and hence W must be a hitting formula. Observe that this argument also implies that every clause in W is in fact adjacent to every other clause in W, and that every $\mathcal{P}$ of h-modularity at most q must contain an h-community H_C which contains W.

For the fourth test, assume there exists a clause D which clashes with exactly $|W|-1$ clauses in W. Consider any witness $\mathcal{P}$ of $\textbf{h-mod}(F) \leq q$, and let H_C be the h-community containing C. Since $D \notin H_C$ and there are at least $q+1$ edge-disjoint paths between D and C, the existence of D would imply that $\textbf{deg}(H_C) \geq q+1$.

For the fifth test, recall that for any clause $Q \in Z \setminus W$ it holds that $W \cup \{Q\}$ cannot be a hitting formula because Q cannot be adjacent to every clause in W. Hence every clause in W with a neighbor outside of W contributes at least 1 to the degree of any h-community containing W. Together with $|W| > q+2$ this implies that if no clause B exists, then $\textbf{h-mod}(F) > q$.

Finally, assume there exists a clause $B \in W$ with no neighbors outside of W and let $F' = F \setminus B$. If $\textbf{h-mod}(F') > q$ then the lemma already holds, so assume there exists a witness $\mathcal{P}'$ of $\textbf{h-mod}(F') \leq q$. Let $W' = W \setminus B$. Observe that W' must be contained in a single h-community $H' \in \mathcal{P}'$, since otherwise the fact that each clause of W' is adjacent to every other clause of W' would contradict the degree bound given by $\textbf{h-mod}(\mathcal{P}') \leq q$. Then let $\mathcal{P}$ be obtained from $\mathcal{P}'$ by adding B to H'. Observe that there cannot exist a clause $D \in H'$ such that D and B do not clash; since D clashes with every other clause in W, it follows that D would clash with $|W|-1$ clauses in W. Hence $B \cup H'$ is still an h-community. Furthermore, by our choice of B it holds that B contains no neighbors outside of W', and hence $\textbf{deg}(H') = \textbf{deg}(H' \cup \{B\})$ and in turn $\textbf{deg}(\mathcal{P}') = \textbf{deg}(\mathcal{P})$.

Finally, observe that, if we are given a witness $\mathcal{P}'$ of $\textbf{h-mod}(F') \leq q$, we can construct a witness of $\textbf{h-mod}(F)$ by adding B back into the unique h-community in $\mathcal{P}$ containing the neighbors of B (i.e., W'). □

Lemma 2. *There exists an algorithm which, given $k \in \mathbb{N}$ and a formula F of length L such that $\textbf{deg}(F) \leq 12k^2+2$, runs in time $k^{\mathcal{O}(k^3)} \cdot L$, and either outputs an h-structure $\mathcal{P}$ of F such that $\textbf{h-mod}(\mathcal{P}) \leq k^2+k$, or correctly determines that $\textbf{h-mod}(F) > k$.*

Proof. We first test whether the treewidth of the dual graph G of F is at most $k \cdot (12k^2+3)$; if not, then $\textbf{h-mod}(F) > k$, and if yes, we compute a tree decomposition of F. This can be achieved in time at most $k^{\mathcal{O}(k^3)} \cdot L$ by Fact 3. Next, we enumerate every inclusion-maximal clique in G of cardinality at least $k+2$ in time $\mathcal{O}(k^3) \cdot L$ by a simple traversal of the tree decomposition. Let U be the set of all such cliques. For each clique $K \in U$ we test whether K is a hitting formula and whether $\textbf{deg}(K) \leq k$; if not, then $\textbf{h-mod}(F) > k$. For each pair of cliques $K_1, K_2 \in U$ we test that they are pairwise disjoint; if not, then $\textbf{h-mod}(F) > k$. Let G' be the graph obtained from G by contracting each clique in U into a single vertex; that is, each $K \in U$ is replaced by a vertex adjacent to all neighbors of K. We test that $\textbf{deg}(G') \leq 2k$ and $\textbf{tw}(G') \leq k^2+k$; if not, then $\textbf{h-mod}(F) > k$. Finally, let $\mathcal{P}'$ be the vertex set of G'. Then $\mathcal{P}'$ is an h-structure witnessing $\textbf{h-mod}(F) \leq k^2+k$.

We prove correctness. First, assume for a contradiction that $\textbf{tw}(G) > k \cdot (12k^2+3)$ and that there exists a witness $\mathcal{P}'$ of $\textbf{h-mod}(F) \leq k$. Since $\textbf{deg}(F) \leq 12k^2+2$, every h-community in $\mathcal{P}'$ must have size at most $12k^2+3$. Let (β, T) be a width-k tree decomposition of the community graph of $\mathcal{P}'$, and let β' be obtained by replacing each h-community $H \in \mathcal{P}'$ with $\bigcup_{C \in H} C$. Then (β', T) is a tree decomposition of G of width at most $k \cdot (12k^2+3)$, contradicting our assumptions.

Next, assume that there exists a clique $K \in U$ which is not a hitting formula. Then any hypothetical h-structure $\mathcal{P}$ of F must partition K into several h-communities. Let $C, D \in K$ and $H \in \mathcal{P}$ be such that $C \in H$ and $D \notin H$. Since there exist $k+1$ edge-disjoint paths between C and D, this implies that $\mathbf{deg}(H) \geq k+1$ and hence $\mathbf{h\text{-}mod}(\mathcal{P}) > k$.

Similarly, assume that there exist inclusion-maximal cliques $K_1, K_2 \in U$ which intersect in some clause C. Then any hypothetical h-structure $\mathcal{P}$ must contain an h-community H containing C, and there must exist a clause $D \in K_1 \cup K_2$ such that $D \notin H$. As in the previous case, this gives rise to at least $k+1$ edge-disjoint paths between C and D and hence $\mathbf{h\text{-}mod}(\mathcal{P}) > k$. In particular, we conclude that each element of U must form an h-community in any hypothetical witness of $\mathbf{h\text{-}mod}(F) \leq k$. This in turn implies that if there exists an h-community $K \in U$ of degree at least $k+1$, then $\mathbf{h\text{-}mod}(F) > k$.

We proceed by considering the graph G'. Assume it contains a vertex v of degree at least $2k+1$. If v is a clause in F, then at most k neighbors of v can form an h-community with v (since we have contracted all cliques of cardinality at least $k+2$). This means that at least $k+1$ neighbors of v would contribute to the degree of the h-community containing v, which guarantees $\mathbf{h\text{-}mod}(F) > k$. On the other hand, if v is an element of U, then we already know that v itself must be an h-community in any witness of $\mathbf{h\text{-}mod}(F) \leq k$, and hence v having more than k neighbors also implies $\mathbf{h\text{-}mod}(F) > k$.

Next, consider the case $\mathbf{tw}(G') > k^2 + k$. Observe that each hitting subformula of F not contained in U contains at most $k+1$ clauses. Consider a width-k tree decomposition (β, T) of the community graph Q of a hypothetical witness of $\mathbf{h\text{-}mod}(F) \leq k$. By replacing, in β, each h-community $H \in V(Q) \setminus U$ with the set of clauses contained in H, we would obtain a tree decomposition of G' of width at most $k \cdot (k+1)$, contradicting our assumption. Hence we conclude that $\mathbf{h\text{-}mod}(F) > k$.

Finally, we summarize why $\mathcal{P}'$ is indeed an h-structure of G such that $\mathbf{h\text{-}mod}(\mathcal{P}') \leq k^2 + k$. The fact that $\mathcal{P}'$ is an h-structure follows by construction; indeed, each element in $\mathcal{P}'$ is either a single clause, or an element of U which is guaranteed to be a hitting formula. Regarding the h-modularity of $\mathcal{P}'$, recall that G' is the community graph of $\mathcal{P}'$ and that $\mathbf{tw}(G') \leq k^2 + k$. As for the degree bound, each vertex v in G' is either a clause C in F, which means that $\mathbf{deg}(v) \leq 2k$, or an element of K, in which case we have already tested that $\mathbf{deg}(v) \leq k$. □

Theorem 3. *There exists an algorithm which, given $k \in \mathbb{N}$ and a formula F of length L, runs in time $\mathcal{O}(L^3) + k^{\mathcal{O}(k^3)} \cdot L$, and either outputs an h-structure $\mathcal{P}$ of F such that $\mathbf{h\text{-}mod}(\mathcal{P}) \leq k^2 + k$, or correctly determines that $\mathbf{h\text{-}mod}(F) > k$.*

Proof. We begin by exhaustively applying Lemma 1 on F for $q = 4k^2$; let us denote the resulting formula F'. Then we apply Lemma 2 on F' to find an h-structure $\mathcal{P}'$ of F' such that $\mathbf{h\text{-}mod}(\mathcal{P}') \leq k^2 + k \leq q$. Finally, we use Lemma 1 to convert $\mathcal{P}'$ into an h-structure $\mathcal{P}$ of F. Correctness follows from the correctness of Lemmas 1 and 2. □

5.2. Using h-structures

With Theorem 3 in hand, we proceed to show how the identified h-structure of small h-modularity can be used to obtain fixed-parameter tractability of SAT and #SAT. The general strategy is to replace each h-community by a suitable object that represents all the satisfying assignments of this h-community. This way, variables only appearing in a single h-community are eliminated. In case of SAT, we represent an h-community by a set of clauses over the bridge variables of the h-community, and in the case of #SAT, we represent an h-community by a so-called valued constraint. This way, we reduce the problems SAT and #SAT parameterized by h-modularity to certain problems (SAT and SumProd, respectively) parameterized by primal treewidth. For solving the latter problems we can use known algorithms.

For making this general strategy work, we have to overcome the difficulty that the number of bridge variables of a single h-community can be arbitrarily large even when the input formula has small h-modularity. In the case of SAT we can handle this by replacing the input formula with a satisfiability-equivalent "matching-lean" subformula (we postpone the definition of this property to the proof of Theorem 4). This approach does not work for #SAT since this replacement does not preserve the number of models. However, by replacing equivalence classes of variables that appear in the same way in all clauses by 3-valued variables (which represent the three possibilities that all variables in the module are set to true, all are set to false, or some are set to true and some to false, respectively), we can reduce the number of variables for a single valued constraint so that we can make our overall strategy work.

We begin with the conceptually simpler case of SAT, which can not only be viewed as a way to gently introduce some of the considerations that will later be used for the more general #SAT, but is also of independent interest due to it having a significantly better running time than the #SAT algorithm (which will be provided in Theorem 6). Our solution relies on the following well-known result.

Fact 6 *([25]).* There exists an algorithm which takes as input a formula F of length L and a tree decomposition of the primal graph of F of width k, runs in time $2^{\mathcal{O}(k)} \cdot L^2$, and determines whether F is satisfiable.

Theorem 4. *Given a formula F' of length L and an h-structure $\mathcal{P}'$ of F', we can decide in time $2^{\mathcal{O}(\mathbf{h\text{-}mod}(\mathcal{P}')^2)} \cdot L^2$ whether F' is satisfiable.*

Proof. Our algorithm has three steps. First, we compute an equisatisfiable subformula F of F' where F has the following property: for every nonempty set X of variables of F there are at least $|X|+1$ clauses C of F such that some variable in X

occurs in C. Formulas with this property are called 1-*expanding* or *matching-lean*, and it is known that for any formula F' of length L, an equisatisfiable 1-expanding subformula F can be computed in time $\mathcal{O}(L^{3/2})$ [15,33,53]. We set $\mathcal{P} = \mathcal{P}'[F]$ and $k = \textbf{h-mod}(\mathcal{P}')$; note that $\textbf{h-mod}(\mathcal{P}) = \textbf{h-mod}(\mathcal{P}'[F]) \leq \textbf{h-mod}(\mathcal{P}') = k$. Observe that since each $H \in \mathcal{P}$ satisfies $\textbf{deg}(\mathrm{H}) \leq \mathrm{k}$, it follows that the number of bridge variables which occur in any clause in H is upper-bounded by k.

For the second step, we construct a formula I as follows. The variable set of I consists of all the bridge variables of $\mathcal{P}$. For each h-community $H \in \mathcal{P}$ containing bridge variables $X_H = \{x_1, \ldots, x_p\}$ and for each assignment α of variables in X_H, we test whether α satisfies H; if it does not, we add the clause C_α over X_α into I, where C_α is the unique clause which is not satisfied by α.

For the final third step, we compute a tree decomposition of the primal graph of I with width at most $k^2 + k$ by Fact 3, and then decide whether I is satisfiable by Fact 6. If it is, we output "YES", and otherwise we output "NO". The rest of the proof is dedicated to verifying the bound on the treewidth of I and arguing correctness.

We argue that the treewidth of the primal graph of I at most $k^2 + k$. Let (β, T) be a tree decomposition of the community graph G of $\mathcal{P}$ of width at most k. Consider the tree decomposition (γ, T) obtained from (β, T) by replacing each h-community H in β by X_H. Since F is 1-expanding and the variables of X_H only appear in at most $k+1$ clauses of F due to the degree bound, the cardinality of each X_H is upper-bounded by $k+1$. Consequently, the cardinality of each element in γ is at most $k^2 + k$.

Next, we show that (γ, T) is indeed a tree decomposition of the primal graph of I. For every edge ab in this graph, there exists at least one clause $C \in H$ which contains both variable a and variable b in its scope, and hence a, b are both bridge variables for H, which in turn means that a, b will both be present in every element of γ which used to contain H; this proves that the first property of tree decompositions is satisfied. For every bridge variable a, let $\mathcal{D}_a$ denote the set of h-communities which contain a. Since each pair of h-communities containing a are adjacent in the community graph of $\mathcal{P}$, $\mathcal{D}_a$ forms a clique in the community graph of $\mathcal{P}$ and hence there must exist an element θ_a of β which contains every h-community in $\mathcal{D}_a$. Since a occurs in an element of γ if and only if this originated from an element of β containing an h-community in $\mathcal{D}_a$, and since all h-communities in $\mathcal{D}_a$ occur in θ_a, we conclude that the nodes of T containing a are connected in (γ, T); this proves that the second property of tree decompositions is satisfied.

Finally, we argue that I is satisfiable if and only if F is satisfiable. Let τ_I be a satisfying assignment for I, and consider the assignment τ which assigns each bridge variable in F based on τ_I. The resulting instance, say F', consists of variable-disjoint h-communities. Furthermore, by the construction of each constraint in I, it holds that each h-community in F' is satisfiable, and hence both F' and F are satisfiable. On the other hand, let τ_F be a satisfying assignment for F, and consider the restriction τ of τ_F to the set of bridge variables. Then applying τ on F once again results in a satisfiable formula F' consisting of variable-disjoint h-communities. Furthermore, since each such h-community is satisfiable, it follows that τ also satisfies every clause in I. □

Our next goal is to show how h-structures of low h-modularity can be used to solve #SAT. To this end, we will make use of a reduction to the SumProd problem [3] (sometimes also called Valued #CSP [57]), which can be viewed as a generalization of the Constraint Satisfaction problem. An instance I of SumProd is a triple $(V, D, \mathcal{C})$, where V is a finite set of *variables*, D is a finite set of *domain values*, and $\mathcal{C}$ is a finite set of *valued constraints*. Each valued constraint C in $\mathcal{C}$ is a tuple (S_C, f_C), where S_C, the *constraint scope*, is a non-empty sequence $s_1, s_2, \ldots, s_r$ of distinct variables of V, and f_C, the *cost function*, is a function from D^r to $\mathbb{N}_0$.

An *assignment* is a mapping $\psi : V \to D$. Each assignment ψ results in a cost, $f_C(\psi)$, being assigned to each constraint C, where $f_C(\psi) = f_C((\psi(s_1), \psi(s_2), \ldots, \psi(s_r)))$. The task in the SumProd problem is to compute the value $\textbf{cost}(I)$, defined as the sum over all assignments of the products of cost functions for that assignment. In other words, $\textbf{cost}(I) = \sum_{\psi: V \to D} \prod_{C \in \mathcal{C}} f_C(\psi)$.

The *primal graph* G of a SumProd instance I is defined as follows. The vertices of G are the variables of I, and two vertices a, b of G are adjacent if and only if there exists a constraint whose scope contains both a and b. The *primal treewidth* of I, denoted $\textbf{ptw}(I)$, is the treewidth of the primal graph of I. The crucial property which we exploit is that primal treewidth allows a straightforward dynamic programming FPT algorithm for SumProd.

Fact 7 *([3]).* Let D be a fixed set. There exists an algorithm which takes as input an n-variable instance $I = (V, D, \mathcal{C})$ of SumProd and a tree decomposition of the primal graph of I of width k, runs in time $2^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$, and correctly outputs $\textbf{cost}(I)$.

Theorem 5. *There exists an algorithm which, given a formula F of length L and an h-structure $\mathcal{P}$ of F, runs in time $\mathcal{O}(3^{\textbf{h-mod}(\mathcal{P})} \cdot L^{\mathcal{O}(1)})$, and computes an instance $I = (V, D, \mathcal{C})$ of SumProd such that $\textbf{ptw}(I) \leq 2^{\mathcal{O}(\textbf{h-mod}(\mathcal{P}))}$, $D = \{0, 1, mix\}$, $|V| \leq L$ and $\textbf{cost}(I)$ is the number of models of F.*

Proof. Overview. We begin by informally outlining the idea of the algorithm, then provide a formal construction, and finally prove the runtime bounds and correctness.

Our goal is to capture the contribution of an h-community H to the total number of models of F by using only a small number of variables in I; specifically, the number of these variables should depend only on $\textbf{h-mod}(\mathcal{P})$. Unlike in Theorem 4,

here we cannot directly use 1-expanding subformulas, since these do not preserve the number of models. So instead we group bridge variables into modules, where two bridge variables are in the same module if and only if they occur in the same way in the same clauses; crucially, the number of modules which intersect with each H is bounded by a function of $\mathbf{h\text{-}mod}(\mathcal{P})$. Furthermore, every "mixed" assignment (mapping at least one variable to 0 and at least one to 1) of a module satisfies the same clauses as any other mixed assignment of that module, allowing us to aggregate all such assignments without loss of information. Then we construct our instance I so that each of its variables represents one module, and each constraint represents one h-community H. An assignment ψ of I then corresponds to determining whether all bridge variables of F in each module are assigned to 0, to 1, or mix.

Generally speaking, the cost function is then constructed so as to capture the contribution of each h-community to the total number of models. However, since many assignments in F can be aggregated into a single assignment in I due to the mix value, the cost function also needs to reflect this. To this end, each module is assigned (arbitrarily) to some valued constraint C and whenever that module is mapped to mix, f_C is increased by a factor corresponding to the number of assignments in F aggregated into this mixed assignment.

Construction. We now give the formal construction. Let B be the set of all bridge variables of $\mathcal{P}$. For $a, b \in B$, we let $a \equiv b$ if and only if for each clause Q, it holds that either a, b both occur positively in Q, or a, b both occur negatively in Q, or neither of a, b occurs in Q. We call an equivalence class of $\equiv$ a *module* and let V be the set of all modules. We say that a module a *occurs* in an h-community H if and only if every variable in a occurs in some clause in H (either positively or negatively). Let us fix an arbitrary mapping α which maps each module a to some h-community H such that a occurs in H.

For each h-community H we add a valued constraint $C_H = (S_{C_H}, f_{C_H})$ (we often drop the subscript H for brevity). $S_C = (s_1, \ldots, s_j)$ contains all modules which occur in H. For each $\psi : S_C \to D$, we compute the value of $f_C(\psi)$ as follows. Let H_ψ be a subset of H obtained by assigning for each $i \in [j]$:

1. all variables in s_i to 1 if and only if $\psi(s_i) = 1$,
2. all variables in s_i to 0 if and only if $\psi(s_i) = 0$, and
3. one arbitrary variable in s_i to 1 and all other variables in s_i to 0 if and only if $\psi(s_i) = \text{mix}$.

Next, we compute the number of models of H_ψ by Lemma 1 and denote it as sat_C^ψ. For each $a = s_i \in S_C$, we furthermore set $\text{fac}_{C,a}^\psi = 2^{|a|} - 2$ if $r_i = \text{mix}$ and $\text{fac}_{C,a}^\psi = 1$ otherwise. Finally, we set

$$f_{C_H}(\psi) = \text{sat}_{C_H}^\psi \cdot \prod_{a \in S_{C_H} : \alpha(a) = H} \text{fac}_{C_H,a}^\psi.$$

We note that if there is no a such that $\alpha(a) = H$, then $f_{C_H}(\psi) = \text{sat}_{C_H}^\psi \cdot 1$.

Properties of the construction. The running time follows from two observations. First, the equivalence classes of $\equiv$ can be computed in at most $\mathcal{O}(L^3)$ time, assuming that arithmetic operations take constant time. Second, the number of modules which occur in any h-community $H \in \mathcal{P}$ is upper-bounded by $3^{\mathbf{h\text{-}mod}(\mathcal{P})+1}$. Indeed, since each clause containing a bridge variable is a bridge clause and each such bridge clause after the first corresponds to at least one edge between H and $F \setminus H$ in the dual graph of F, the number of clauses where the modules in S_C occur is upper-bounded by $\mathbf{h\text{-}mod}(\mathcal{P}) + 1$. Moreover, if two bridge variables occur positively in the same clauses, occur negatively in the same clauses and are not contained in the same clauses, then they are equivalent and hence belong to the same module.

Since the number of modules which occur in any h-community $H \in \mathcal{P}$ is at most $3^{\mathbf{h\text{-}mod}(\mathcal{P})+1}$, it follows that the scope of each constraint C in I has cardinality at most $3^{\mathbf{h\text{-}mod}(\mathcal{P})+1}$. Let G be the primal graph of I, and let (β, T) be a tree decomposition of the community graph G' of F. Consider the tree decomposition (γ, T) obtained from (β, T) by replacing each h-community H in β by $\bigcup_{s \in S_{C_H}} s$. It follows that the cardinality of each element in γ is upper-bounded by $k \cdot 3^{\mathbf{h\text{-}mod}(\mathcal{P})+1}$.

Next, we show that (γ, T) is indeed a tree decomposition of the primal graph of I. For every edge ab in this graph, there exists at least one valued constraint C_H which contains both variable a and variable b in its scope, and hence a, b are both bridge variables for H, which in turn means that a, b will both be present in every element of γ which used to contain H; this proves that the first property of tree decompositions is satisfied. For every bridge variable a, let $\mathcal{D}_a$ denote the set of h-communities which contain a. Since each pair of h-communities containing a are adjacent in the community graph G' of $\mathcal{P}$, $\mathcal{D}_a$ forms a clique in G' and hence there must exist an element θ_a of β which contains every h-community in $\mathcal{D}_a$. Since a occurs in an element of γ if and only if this originated from an element of β containing an h-community in $\mathcal{D}_a$, and since all h-communities in $\mathcal{D}_a$ occur in θ_a, we conclude that the nodes of T containing a are connected in (γ, T); this proves that the second property of tree decompositions is satisfied.

Correctness of the construction. Finally, we argue that $\mathbf{cost}(I)$ is the number of models of F. From now on, we let τ denote an assignment of bridge variables in F, we use $c(\tau)$ to denote the number of models of $F[\tau]$, and we let $\psi : V \to \{0, 1, \text{mix}\}$ denote an assignment of all variables in I; to distinguish these two types of assignments, we will refer to assignments in I

as *pseudoassignments*. We say that an assignment τ is *associated* with a unique pseudoassignment ψ if, for each $s \in V$, it holds that $\psi(s) =$

- 0 if for each $x \in s$ it holds that $\tau(x) = 0$,
- 1 if for each $x \in s$ it holds that $\tau(x) = 1$,
- mix if there exist $x, y \in s$ such that $\tau(x) = 0$ and $\tau(y) = 1$.

Observe that if two assignments τ_1, τ_2 of bridge variables are both associated with the same ψ, then τ_1 satisfies exactly the same clauses as τ_2 and hence $F[\tau_1] = F[\tau_2]$; indeed, any bridge clause containing a variable which is assigned positively in τ_1 and negatively in τ_2 must also contain a variable which is assigned negatively in τ_1 and positively in τ_2, and hence must be satisfied by both assignments. We will denote by $F[\psi]$ the formula $F[\tau]$ where τ is any assignment associated with ψ, and we remark that $F[\psi]$ is uniquely defined for each ψ by the above observation (however, several distinct ψ may still have the same $F[\psi]$). In particular, this implies that if τ_1 and τ_2 are associated with the same ψ, then $c(\tau_1) = c(\tau_2)$; we denote this value $c(\psi)$.

We now propose the following way of counting the models of F: instead of computing the number of all models of F, we aggregate the models into "groups", compute the number of models in each group, and output the sum. In our case, these "groups" are defined based on the assignments τ of the bridge variables; the restriction of each model of F to the bridge variables results in a unique τ. At this point, we can say that the number of models of F is equal to $\sum_\tau c(\tau)$. Now we apply this aggregation strategy once more, this time by grouping assignments τ based on their association with ψ. Hence $\sum_\tau c(\tau) = \sum_\psi c(\psi) \cdot q_\psi$, where q_ψ is the total number of assignments τ associated with ψ. Next, we decompose $c(\psi)$ into a product of the number of assignments of individual variable-disjoint hitting formulas in $F[\psi]$, i.e., $\sum_\tau c(\tau) = \sum_\psi q_\psi \cdot \prod_{H \in F[\psi]} y_\psi^H$, where y_ψ^H is the number of models of a hitting formula H in $F[\psi]$, which is equal to $\mathrm{sat}_{C_H}^\psi$ in our construction.

Finally, we decompose q_ψ into a product of factors capturing the number of possible ways one can obtain τ from ψ by turning each "mixed" value in ψ into a specific "mixed" assignment of a module within τ; here, each module a contributes by a multiplicative factor of $2^{|a|} - 2$ (all possible assignments of $|a|$ variables, except for the all-0 and all-1 assignment). For instance, modules containing two variables result in a multiplicative factor of 2, since there are 2 mixed assignments of these two variables, while modules containing a single variable result in a multiplicative factor of 0, since a single variable can never lead to a mixed assignment. The factors of this product are then arbitrarily mapped (by α) to individual h-communities in $F[\psi]$ so that we may capture them by the cost functions on individual valued constraints, and we observe that each factor of this product is captured by a unique value of $\mathrm{fac}_{C_a}^\psi$ in our construction. Hence we conclude that $\sum_\tau c(\tau) = \sum_\psi \prod_{C_H \in \mathcal{C}} \left(\mathrm{sat}_{C_H}^\psi \cdot \prod_{a \in S_{C_H} : \alpha(a) = H} \mathrm{fac}_{C_H, a}^\psi\right) = \mathbf{cost}(I)$. □

Theorem 6. *Given a formula F of length L and an h-structure $\mathcal{P}$ of F, we can count in time $2^{2^{\mathcal{O}(\mathbf{h\text{-}mod}(\mathcal{P}))}} \cdot L^{\mathcal{O}(1)}$ the number of models of F.*

Proof. Let $k = \mathbf{h\text{-}mod}(\mathcal{P})$. We apply Theorem 5 to obtain an instance $I = (V, D, \mathcal{C})$ of SumProd such that $\mathbf{ptw}(I) \leq 2^{\mathcal{O}(k)}$ and $\mathbf{cost}(I)$ is the number of models of F. Next, we compute a tree decomposition of the primal graph of I of width $2^{\mathcal{O}(k)}$: either by observing that the algorithm of Theorem 5 implicitly also computes such a tree decomposition of I, or in time $2^{2^{\mathcal{O}(k)}} \cdot L$ by Fact 3. Finally, we use Fact 7 to solve I in time $2^{2^{\mathcal{O}(k)}} \cdot L^{\mathcal{O}(1)}$. □

We now have all the ingredients necessary to prove the main theorem stated at the beginning of this section.

Proof of Theorem 2. Let F be the given CNF formula and k the parameter. First we apply Theorem 3 to either find an h-structure $\mathcal{P}$ of F of h-modularity at most $k^2 + k$, or correctly determine that $\mathbf{h\text{-}mod}(F) > k$. To decide whether F is satisfiable, we now use Theorem 4. This results in a fixed-parameter algorithm for SAT parameterized by **h-mod** where the parameter dependence is dominated by the time needed to apply Theorem 3 for computing $\mathcal{P}$. To compute the number of models of F, we follow up on Theorem 3 by using Theorem 6. This results in a fixed-parameter algorithm for #SAT parameterized by **h-mod** with a double-exponential parameter dependence, where the dependence is dominated by the time needed to apply Theorem 6. □

6. Comparison to other parameters

Here, we compare the structural parameters investigated in this paper which give rise to new fixed-parameter algorithms to previously studied and established structural measures of CNF formulas.

We say that parameter X *dominates* parameter Y if there exists a computable function f such that for each formula F we have $X(F) \leq f(Y(F))$ [48]. In particular, if X dominates Y and SAT is FPT parameterized by X, then SAT is FPT parameterized by Y [48]. We say that two parameters are *incomparable* if neither dominates the other. We note that in our comparison,

we only consider parameters which are known to give rise to fixed-parameter algorithms for SAT (i.e., not *incidence clique-width* [41]) and can be used without requiring additional information from an oracle (i.e., not *PS-width* [47]).[2]

We begin by showing that consensus treewidth is incomparable with the signed clique-width, the clustering-width and h-modularity. The former claim implies that consensus treewidth is not dominated by the primal, dual, or incidence treewidth; indeed, all of these parameters are dominated by signed clique-width [54]. Furthermore, consensus treewidth is also not dominated by signed rank-width [21], which both dominates and is dominated by signed clique-width.

Proposition 3. *The following claims hold.*

1. *Signed clique-width and consensus treewidth are incomparable.*
2. *Clustering-width and consensus treewidth are incomparable.*
3. *h-modularity and consensus treewidth are incomparable.*

Proof. We prove these claims by showing that there exist classes of formulas such that each formula in the class has one parameter bounded while the other parameter can grow arbitrarily. For a formula F, let $\mathbf{scw}(F)$ and $\mathbf{clw}(F)$ denote its signed clique-width and clustering width, respectively.

Let us choose an arbitrary positive integer $i \in \mathbb{N}$. For the first claim, it is known that already the class of all hitting formulas has unbounded **scw** [40]. In particular, this means that there exists a hitting formula F_1 such that $\mathbf{scw}(F_1) \geq i$. Observe that the consensus graph of F_1 is edgeless, and hence $\mathbf{contw}(F_1) = 0$.

Conversely, consider the following formula $F_2 = \{c_1, \ldots, c_i\}$. The formula contains variables $x_1, \ldots x_i$, and each variable x_ℓ occurs only in clause c_ℓ. Since the incidence graph of F_2 is just a matching, its signed clique-width is bounded by a constant (in particular, it will be 2). However, the consensus graph of F_2 is a complete graph on i vertices, and it is known that such graphs have treewidth precisely $i-1$, hence $\mathbf{contw}(F_2) = i-1$.

We proceed similarly for the second and third claims; in fact, we can use a single construction to deal with both h-modularity and clustering width. Let us once again fix some $i \in \mathbb{N}$, let F'_1 be the union of two variable-disjoint hitting formulas each containing i clauses. Both h-modularity and clustering width have a value of 0 for variable-disjoint hitting formulas. However, the consensus graph of F'_1 is a complete bipartite graph with each side containing precisely i vertices, and it is well-known that such graphs have treewidth i; hence, $\mathbf{contw}(F'_1) = i$.

Conversely, consider the formula F'_2 over variable sets $Y = \{y_1, \ldots, y_i\}$ and $X = \{x_1, \ldots, x_i\}$. For each subset α of X, we will add two clauses to F'_2:

- c_α contains α as positive literals and $X \setminus \alpha$ as negative literals;
- c^y_α contains α as positive literals, $X \setminus \alpha$ as negative literals, and all variables in Y as positive literals.

We observe that for each α, clause c^y_α clashes with all other clauses except for c_α (and vice-versa for c_α). This implies that the consensus graph of F'_2 is a matching, and hence $\mathbf{contw}(F'_2) = 1$. On the other hand, note that for each distinct pair of subsets $\alpha, \beta \subseteq X$, the clauses c_α, c_β, c^y_α, c^y_β form a formula which is not a variable-disjoint union of hitting formulas. However, deleting a subset of X from F'_2 will only resolve this obstruction for choices of α and β which differed in X; for instance, even if we deleted all of X except for a single variable (w.l.o.g. say x_1), the resulting formula would still not be a disjoint union of hitting formulas (it would contain clauses $\{x_1\} \cup Y, \{x_1\}, \{\overline{x_1}\} \cup Y, \{\overline{x_1}\}$). Similarly, deleting any proper subset $Y' \subset Y$ will also clearly not result in a disjoint union of hitting formulas (it would, in fact, not change the consensus graph at all), and the same goes for any combination of deleting Y' along with a proper subset of X. Hence we conclude that $\mathbf{clw}(F'_2) \geq i$.

Finally, we argue that F'_2 has h-modularity at least i, and we will do so by following the arguments presented in the proof of Lemma 1. In particular, closely following that proof, let us fix $q = i$ and a clause $c \in F'_2$. Then:

1. the set Z_0 defined in the proof will be equal to F'_2;
2. the set Z_1 defined in the proof will be empty;
3. the set Z defined in the proof will be equal to F'_2;
4. the set W defined in the proof will be equal to F'_2;
5. since W is not a hitting formula, by point 3 of the proof it holds that F'_2 has h-modularity greater than $q = i$.

The above general constructions show that for any choice of i, one can produce formulas with a gap of at least i between consensus treewidth and any of the three other measures under consideration. □

We now complete our comparison by contrasting h-modularity to signed clique-width and clustering-width.

[2] SAT is not fixed-parameter tractable when parameterized by incidence clique-width, and SAT is only known to be fixed-parameter tractable when parameterized by PS-width when a suitable decomposition is provided as part of the input.

Proposition 4. *The following claims hold.*

1. *Signed clique-width and h-modularity are incomparable.*
2. *Clustering-width and h-modularity are incomparable.*

Proof. We prove both claims by showing that there exist classes of formulas such that each formula in the class has one parameter bounded while the other parameter can grow arbitrarily. For a formula F, let $\mathbf{scw}(F)$ and $\mathbf{clw}(F)$ denote its signed clique-width and clustering width, respectively. Our proof does not require a formal definition of these parameters, as we refer to known properties of these notions.

Let us choose an arbitrary $i \in \mathbb{N}$. For the first claim, it is known that already the class of all hitting formulas has unbounded **scw** [40]. In particular, this means that there exists a hitting formula F_1 such that $\mathbf{scw}(F_1) \geq i$. Recall that, since F_1 is a hitting formula, clearly $\mathbf{h\text{-}mod}(F_1) = 0$.

Conversely, consider the following formula $F_2 = \{C, C_1, \ldots, C_{i+2}\}$. The formula contains variables $x_1, \ldots x_{i+2}$, and each variable x_j occurs (either positively or negatively) in clause C and C_j. Then the incidence graph of F_2 is a tree and hence has treewidth 1. Since signed clique-width dominates the treewidth of the incidence graph, it follows that there exists a constant c independent of i such that $\mathbf{scw}(F_2) \leq c$ (in particular, one can check from the definition of **scw** that $c \leq 2$). On the other hand, the degree of any h-community H containing C is at least $i + 1$, and hence $\mathbf{h\text{-}mod}(F_2) \geq i + 1$.

We proceed similarly for the second claim; let $i \in \mathbb{N}$. Let F_1'' be a hitting formula, let F_1' be constructed by adding a variable z into an arbitrary clause in F_1'' and adding a clause Z containing only z (both occurrences can either be positive or negative). Observe that $\mathbf{clw}(F_1') = \mathbf{h\text{-}mod}(F_1') = 1$. Let F_1 then contain $i + 2$ disjoint copies of F_1'; clearly, $\mathbf{clw}(F_1) = i + 2$. However, since the h-modularity of a formula is equal to the maximum h-modularity over all of its connected components, it holds that $\mathbf{h\text{-}mod}(F_1) = 1$.

Conversely, let F_2' and F_2'' be variable-disjoint hitting formulas containing at least $i + 2$ clauses each, and let F_2 be obtained from a disjoint union of F_2' and F_2'' by adding a variable z which occurs (either positively or negatively) in $\lfloor i/2 \rfloor$ clauses in F_2' and in $\lfloor i/2 \rfloor$ clauses in F_2''. While F_2 is not a hitting formula, deleting z results in two variable-disjoint hitting formulas and hence $\mathbf{clw}(F_2) = 1$. On the other hand, the three inclusion-maximal h-communities in F_2 are F_2', F_2'' and possibly the set of clauses where z occurs; each of these have a degree which is greater than i. Consequently, it holds that $\mathbf{h\text{-}mod}(F_2) \geq i + 1$. □

7. Concluding remarks

We have introduced two novel structural parameters which give rise to fixed-parameter algorithms for counting the number of models of CNF formulas and provide worst-case performance guarantees for instances that are not accessible by known methods.

The introduced notion of consensus treewidth generalizes and, in some sense, builds upon the classical #SAT algorithm on hitting formulas [26], and our positive results for this parameter show that it is worthwhile to consider further graphical models in addition to the already established ones such as primal, dual, and incidence graphs.

Our investigation of h-modularity naturally leads to the question of how the notion of h-community structure can be further generalized, for example by using a suitably defined property for the communities that generalizes hitting formulas. This way, we hope that ultimately one can build bridges between empirically observed problem hardness and theoretical worst-case upper bounds.

Finally, we note that while we established the W[1]-hardness of SAT parameterized by conflict treewidth, to the best of our knowledge it remains open whether SAT and #SAT can be solved in polynomial time when the conflict treewidth is a fixed constant.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

The authors acknowledge support by the Austrian Science Fund (FWF, projects Y1329, P31336, and P32441), as well the Vienna Science and Technology Fund (WWTF, project ICT19-065).

References

[1] C. Ansótegui, M.L. Bonet, J. Giráldez-Cru, J. Levy, The fractal dimension of SAT formulas, in: S. Demri, D. Kapur, C. Weidenbach (Eds.), Automated Reasoning - Proceedings of the 7th International Joint Conference, IJCAR 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 19–22, 2014, in: Lecture Notes in Computer Science, vol. 8562, Springer Verlag, 2014, pp. 107–121.

[2] F. Bacchus, S. Dalmao, T. Pitassi, Algorithms and complexity results for #SAT and Bayesian inference, in: 44th Annual IEEE Symposium on Foundations of Computer Science (FOCS'03), 2003, pp. 340–351.

[3] F. Bacchus, S. Dalmao, T. Pitassi, Solving #SAT and Bayesian inference with backtracking search, J. Artif. Intell. Res. 34 (2009) 391–442.
[4] B. Bliem, M. Moldovan, M. Morak, S. Woltran, The impact of treewidth on ASP grounding and solving, in: C. Sierra (Ed.), Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19–25, 2017, 2017, pp. 852–858.
[5] H.L. Bodlaender, T. Kloks, Efficient and constructive algorithms for the pathwidth and treewidth of graphs, J. Algorithms 21 (2) (1996) 358–402.
[6] H.L. Bodlaender, P.G. Drange, M.S. Dregi, F.V. Fomin, D. Lokshtanov, M. Pilipczuk, A $c^k n$ 5-approximation algorithm for treewidth, SIAM J. Comput. 45 (2) (2016) 317–378.
[7] H.L. Bodlaender, A tourist guide through treewidth, Acta Cybern. 11 (1993) 1–21.
[8] H.L. Bodlaender, A linear-time algorithm for finding tree-decompositions of small treewidth, SIAM J. Comput. 25 (6) (1996) 1305–1317.
[9] B. Courcelle, J.A. Makowsky, U. Rotics, On the fixed parameter complexity of graph enumeration problems definable in monadic second-order logic, Discrete Appl. Math. 108 (1–2) (2001) 23–52.
[10] Y. Crama, P.L. Hammer, Boolean Functions: Theory, Algorithms, and Applications, Encyclopedia of Mathematics and Its Applications, vol. 142, Cambridge University Press, Cambridge, 2011.
[11] R. Dechter, Bucket elimination: a unifying framework for reasoning, Artif. Intell. 113 (1–2) (1999) 41–85.
[12] R. Diestel, Graph Theory, 4th edition, Graduate Texts in Mathematics, vol. 173, Springer Verlag, New York, 2010.
[13] R.G. Downey, M.R. Fellows, Fundamentals of Parameterized Complexity, Texts in Computer Science, Springer Verlag, 2013.
[14] P.E. Dunne, Computational properties of argument systems satisfying graph-theoretic constraints, Artif. Intell. 171 (10–15) (2007) 701–729.
[15] H. Fleischner, O. Kullmann, S. Szeider, Polynomial-time recognition of minimal unsatisfiable formulas with fixed clause-variable difference, Theor. Comput. Sci. 289 (1) (2002) 503–516.
[16] M. Fürer, Faster integer multiplication, SIAM J. Comput. 39 (3) (2009) 979–1005.
[17] N. Galesi, O. Kullmann, Polynomial time SAT decision, hypergraph transversals and the Hermitian rank, in: H.H. Hoos, D.G. Mitchell (Eds.), Theory and Applications of Satisfiability Testing, 7th International Conference, SAT 2004, Revised Selected Papers, Vancouver, BC, Canada, May 10–13, 2004, in: Lecture Notes in Computer Science, vol. 3542, Springer Verlag, 2005, pp. 89–104.
[18] R. Ganian, S. Ordyniak, The complexity landscape of decompositional parameters for ILP, Artif. Intell. 257 (2018) 61–71.
[19] R. Ganian, S. Szeider, Community structure inspired algorithms for SAT and #SAT, in: M. Heule, S. Weaver (Eds.), Theory and Applications of Satisfiability Testing - SAT 2015 - Proceedings of the 18th International Conference, Austin, TX, USA, September 24–27, 2015, in: Lecture Notes in Computer Science, vol. 9340, Springer, 2015, pp. 223–237.
[20] R. Ganian, S. Szeider, New width parameters for model counting, in: Theory and Applications of Satisfiability Testing - SAT 2017 - Proceedings of the 20th International Conference, Melbourne, VIC, Australia, August 28 - September 1, 2017, 2017, pp. 38–52.
[21] R. Ganian, P. Hlinený, J. Obdrzálek, Better algorithms for satisfiability problems for formulas of bounded rank-width, Fundam. Inform. 123 (1) (2013) 59–76.
[22] R. Ganian, S. Ordyniak, M.S. Ramanujan, Going beyond primal treewidth for (M)ILP, in: Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, San Francisco, California, USA, February 4–9, 2017, 2017, pp. 815–821.
[23] C.P. Gomes, A. Sabharwal, B. Selman, Model counting, in: A. Biere, M. Heule, H. van Maaren, T. Walsh (Eds.), Handbook of Satisfiability, vol. 185, IOS Press, 2009, pp. 633–654.
[24] G. Gottlob, R. Pichler, F. Wei, Bounded treewidth as a key to tractability of knowledge representation and reasoning, Artif. Intell. 174 (1) (2010) 105–132.
[25] G. Gottlob, F. Scarcello, M. Sideri, Fixed-parameter complexity in AI and nonmonotonic reasoning, Artif. Intell. 138 (1–2) (2002) 55–86.
[26] K. Iwama, CNF-satisfiability test by counting and polynomial average time, SIAM J. Comput. 18 (2) (1989) 385–391.
[27] E.J. Kim, S. Ordyniak, S. Szeider, Algorithms and complexity results for persuasive argumentation, Artif. Intell. 175 (2011) 1722–1736.
[28] H. Kleine Büning, O. Kullmann, Minimal unsatisfiability and autarkies, in: A. Biere, M.J.H. Heule, H. van Maaren, T. Walsh (Eds.), Handbook of Satisfiability, in: Frontiers in Artificial Intelligence and Applications, vol. 185, IOS Press, 2009, pp. 339–401 (Chapter 11).
[29] H. Kleine Büning, X. Zhao, Satisfiable formulas closed under replacement, in: H. Kautz, B. Selman (Eds.), Proceedings for the Workshop on Theory and Applications of Satisfiability, in: Electronic Notes in Discrete Mathematics, vol. 9, Elsevier Science Publishers, North-Holland, 2001.
[30] H. Kleine Büning, X. Zhao, On the structure of some classes of minimal unsatisfiable formulas, Discrete Appl. Math. 130 (2) (2003) 185–207.
[31] T. Kloks, Treewidth: Computations and Approximations, Springer Verlag, Berlin, 1994.
[32] D.E. Knuth, How fast can we multiply?, in: The Art of Computer Programming, vol. II: Seminumerical Algorithms, 3rd edition, Addison-Wesley, 1998, pp. 294–318 (Chapter 4.3.3).
[33] O. Kullmann, Lean clause-sets: generalizations of minimally unsatisfiable clause-sets, Discrete Appl. Math. 130 (2) (2003) 209–249.
[34] O. Kullmann, The combinatorics of conflicts between clauses, in: E. Giunchiglia, A. Tacchella (Eds.), Sixth International Conference on Theory and Applications of Satisfiability Testing, S. Margherita Ligure - Portofino, Italy, May 5–8, 2003 (SAT 2003), in: Lecture Notes in Computer Science, vol. 2919, Springer Verlag, 2004.
[35] M.E.J. Newman, M. Girvan, Finding and evaluating community structure in networks, Phys. Rev. E 69 (2) (2004) 026113.
[36] M.E.J. Newman, The structure and function of complex networks, SIAM Rev. 45 (2) (2003) 167–256.
[37] M.E.J. Newman, Modularity and community structure in networks, Proc. Natl. Acad. Sci. 103 (23) (2006) 8577–8582.
[38] Z. Newsham, V. Ganesh, S. Fischmeister, G. Audemard, L. Simon, Impact of community structure on SAT solver performance, in: C. Sinz, U. Egly (Eds.), Theory and Applications of Satisfiability Testing - SAT 2014 - Proceedings of the 17th International Conference, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 14–17, 2014, in: Lecture Notes in Computer Science, vol. 8561, Springer Verlag, 2014, pp. 252–268.
[39] R. Niedermeier, Invitation to Fixed-Parameter Algorithms, Oxford Lecture Series in Mathematics and Its Applications, Oxford University Press, Oxford, 2006.
[40] N. Nishimura, P. Ragde, S. Szeider, Solving #SAT using vertex covers, Acta Inform. 44 (7–8) (2007) 509–523.
[41] S. Ordyniak, D. Paulusma, S. Szeider, Satisfiability of acyclic and almost acyclic CNF formulas, Theor. Comput. Sci. 481 (2013) 85–99.
[42] D. Paulusma, F. Slivovsky, S. Szeider, Model counting for CNF formulas of bounded modular treewidth, Algorithmica 76 (1) (2016) 168–194.
[43] R. Pichler, S. Rümmele, S. Szeider, S. Woltran, Tractable answer-set programming with weight constraints: bounded treewidth is not enough, Theory Pract. Log. Program. 14 (2) (2014) 141–164.
[44] N. Robertson, P.D. Seymour, Graph minors. II. Algorithmic aspects of tree-width, J. Algorithms 7 (3) (1986) 309–322.
[45] D.J. Rose, On simple characterizations of k-trees, Discrete Math. 7 (1974) 317–322.
[46] D. Roth, On the hardness of approximate reasoning, Artif. Intell. 82 (1–2) (1996) 273–302.
[47] S.H. Sæther, J.A. Telle, M. Vatshelle, Solving #SAT and MAXSAT by dynamic programming, J. Artif. Intell. Res. 54 (2015) 59–82.
[48] M. Samer, S. Szeider, Fixed-parameter tractability, in: A. Biere, M. Heule, H. van Maaren, T. Walsh (Eds.), Handbook of Satisfiability, IOS Press, 2009, pp. 425–454 (Chapter 13).
[49] M. Samer, S. Szeider, Algorithms for propositional model counting, J. Discret. Algorithms 8 (1) (2010) 50–64.
[50] M. Samer, S. Szeider, Constraint satisfaction with bounded treewidth revisited, J. Comput. Syst. Sci. 76 (2) (2010) 103–114.
[51] T. Sang, P. Beame, H.A. Kautz, Performing Bayesian inference by weighted model counting, in: Proceedings, The Twentieth National Conference on Artificial Intelligence and the Seventeenth Innovative Applications of Artificial Intelligence Conference, Pittsburgh, Pennsylvania, USA, July 9–13, 2005, AAAI Press/The MIT Press, 2005, pp. 475–482.

[52] D. Scheder, P. Zumstein, How many conflicts does it need to be unsatisfiable?, in: Theory and Applications of Satisfiability Testing - SAT 2008, Proceedings of the 11th International Conference, SAT 2008, Guangzhou, China, May 12–15, 2008, in: Lecture Notes in Computer Science, vol. 4996, Springer Verlag, 2008, pp. 246–256.
[53] S. Szeider, Minimal unsatisfiable formulas with bounded clause-variable difference are fixed-parameter tractable, J. Comput. Syst. Sci. 69 (4) (2004) 656–674.
[54] S. Szeider, On fixed-parameter tractable parameterizations of SAT, in: E. Giunchiglia, A. Tacchella (Eds.), Theory and Applications of Satisfiability, 6th International Conference, SAT 2003, Selected and Revised Papers, in: Lecture Notes in Computer Science, vol. 2919, Springer Verlag, 2004, pp. 188–202.
[55] L.G. Valiant, The complexity of computing the permanent, Theor. Comput. Sci. 8 (2) (1979) 189–201.
[56] M.Y. Vardi, Boolean satisfiability: theory and engineering, Commun. ACM 57 (3) (2014) 5.
[57] S. Živný, The Complexity of Valued Constraint Satisfaction Problems, Cognitive Technologies, Springer, 2012.
[58] W. Zhang, G. Pan, Z. Wu, S. Li, Online community detection for large complex networks, in: F. Rossi (Ed.), IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3–9, 2013, IJCAI/AAAI, 2013.