Model Counting for CNF Formulas of Bounded Modular Treewidth*

Daniel Paulusma[1], Friedrich Slivovsky[2], and Stefan Szeider[3]

1 **School of Engineering and Computing Sciences, Durham University**
Durham, DH1 3LE, UK daniel.paulusma@durham.ac.uk
2 **Institute of Information Systems, Vienna University of Technology**
A-1040 Vienna, Austria fslivovsky@gmail.com
3 **Institute of Information Systems, Vienna University of Technology**
A-1040 Vienna, Austria stefan@szeider.net

Abstract

The modular treewidth of a graph is its treewidth after the contraction of modules. Modular treewidth properly generalizes treewidth and is itself properly generalized by clique-width. We show that the number of satisfying assignments of a CNF formula whose incidence graph has bounded modular treewidth can be computed in polynomial time. This provides new tractable classes of formulas for which #SAT is polynomial. In particular, our result generalizes known results for the treewidth of incidence graphs and is incomparable with known results for clique-width (or rank-width) of signed incidence graphs. The contraction of modules is an effective data reduction procedure. Our algorithm is the first one to harness this technique for #SAT. The order of the polynomial time bound of our algorithm depends on the modular treewidth. We show that this dependency cannot be avoided subject to an assumption from Parameterized Complexity.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Satisfiability, Model Counting, Parameterized Complexity

Digital Object Identifier 10.4230/LIPIcs.STACS.2013.55

1 Introduction

Given a set $\{x_1, \ldots, x_n\}$ of variables, consider a propositional formula $F_{m,n}$ defined as follows. $F_{m,n}$ consists of m distinct clauses each containing n literals over $\{x_1, \ldots, x_n\}$, so that every variable occurs in every clause. It is easy to see that $F_{m,n}$ has exactly $2^n - m$ satisfying assignments. The vertices of the incidence graph of $F_{m,n}$ (i.e., the bipartite graph whose vertex classes consist of variables and clauses, and a variable is adjacent to the clauses it occurs in) can be partitioned into two large modules (a module in a graph is a set S of vertices such that for any vertex $v \notin S$, every vertex in S is a neighbor of v or every vertex in S is a non-neighbor of v). By contracting these modules, the incidence graph reduces to a single edge.

Contraction of modules is an important preprocessing step for a wide range of combinatorial optimization problems [14]. The aim of this paper is to harness its power for propositional model counting (#SAT), a well-studied problem with various applications in artifical intelligence, such as probabilistic inference [1]. We consider CNF formulas whose incidence graph

* Slivovsky and Szeider's research was supported by the ERC, grant reference 239962. Paulusma's research was supported by EPSRC, grant reference EP/G043434/1.

30th Symposium on Theoretical Aspects of Computer Science (STACS'13).
Editors: Natacha Portier and Thomas Wilke; pp. 55–66

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany


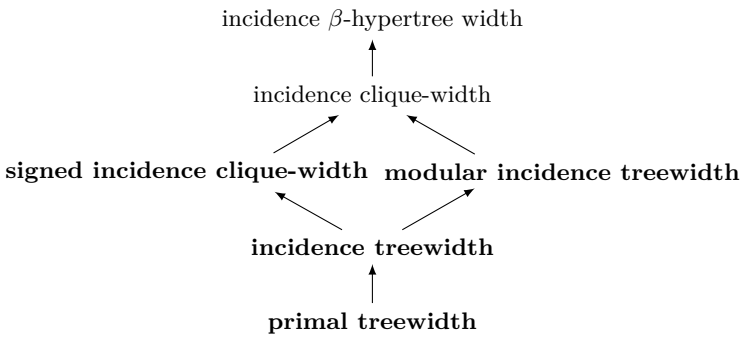

Figure 1 Hierarchy of structural parameters. An arc from parameter p to parameter q reads as "q is bounded whenever p is bounded." Bold type indicates parameters that are known to render #SAT polynomial-time tractable when bounded.

has bounded treewidth after contraction of modules (we call this the *modular incidence treewidth*). We will prove that #SAT is polynomial-time tractable for such formulas.

▶ **Theorem 1.** *The number of satisfying assignments of a CNF formula F with modular incidence treewidth at most k can be computed in time $\ell^{\mathcal{O}(k)}$, where ℓ is the length of F.*

This result characterizes new tractable classes for a notoriously hard problem. #SAT is #P-complete in general [24] and remains #P-hard even for monotone 2CNF formulas and Horn 2CNF formulas. It is NP-hard to approximate the number of models of a formula with n variables is within $2^{n^{1-\varepsilon}}$ for $\varepsilon > 0$. Again, this hardness result still holds for monotone 2CNF formulas and Horn 2CNF formulas [22]. These syntactic restrictions do no lead to tractability of #SAT.

By contrast, modular incidence treewidth is a so-called *structural* parameter. Structural restrictions are applied in terms of parameters (invariants) of graphs or hypergraphs associated with formulas. Figure 1 illustrates the relation of modular incidence treewidth to other structural parameters. For a detailed discussion, see Section 1.1 below. Our algorithmic result presented in Section 3 is achieved by dynamic programming on a tree decomposition of the *modular incidence graph* (the graph obtained from the incidence graph of a formula by contracting modules). Vertices in the modular incidence graph represent entire *modules* (i.e., sets of variables or sets of clauses), whose size cannot be bounded in terms of the modular incidence treewidth alone. Algorithms for #SAT on formulas of bounded treewidth typically rely on data structures indexed by the subsets of variables and clauses associated with a bag of the tree decomposition [23]. The number of subsets of variables and clauses occurring in even a single module can be exponential in the length of the input formula, so these algorithms do not yield tractability in our case. It is a significant challenge to encode the information required to perform dynamic programming in space polynomially bounded by the input size. Our main technical contribution is the use of *projections* in solving this task. We define an equivalence relation on assignments based on their projections onto a particular formula. This formula is determined by the boundary of a subgraph induced by the decomposition. The resulting equivalence relation is sufficiently precise while its rank can still be polynomially bounded. This allows our algorithm to run in polynomial time. Note that the order of the polynomial time bound in Theorem 1 is a function in the modular incidence treewidth. The hardness result presented in Section 4 shows that one cannot replace this function by a constant (subject to an assumption from Parameterized Complexity).

1.1 Related structural parameters

For sake of comparing two structural parameters p and q of CNF formulas, we say p *dominates* q if there is a function f such that $p(F) \leq f(q(F))$ for all formulas F. Parameters p and q are *equivalent* if p dominates q and q dominates p. We say p is *more general* than q if p dominates q but not the other way around. Two parameters p and q are *incomparable* if neither p dominates q nor q dominates p. The *primal treewidth* of a formula is the treewidth of its primal graph. The primal graph has as vertices the variables of the given formula, and two variables are joined by an edge if they occur together in a clause. It is well known that #SAT on formulas of bounded primal treewidth is linear-time tractable [23]. A similar result has been shown in terms of the *branchwidth* of formulas [1], a parameter that is equivalent to primal treewidth. The *incidence treewidth* of a formula is the treewidth of its incidence graph. This parameter is known to be more general than primal treewidth [13]. Again, #SAT on formulas of bounded incidence treewidth is linear-time tractable [10, 23]. *Clique-width* is a graph invariant based on graph grammars [4]. *Signed clique-width* is a variant of clique-width for directed graphs [7]. The *signed incidence clique-width* of a formula corresponds to the signed clique-width of its *signed incidence graph*, which is obtained from the incidence graph by orientating edges so as to indicate positive or negative occurrences of variables. A polynomial-time algorithm for #SAT on formulas of bounded signed incidence clique-width is due to Fischer, Makowsky, and Ravve [10]. Clique-width is typically approximated by means of another parameter known as *rank-width* [20]. A class of graphs has bounded rank-width if and only if it has bounded clique-width. But while it is open whether, for fixed $k \geq 4$, graphs of clique-width at most k can be recognized in polynomial time [9], this is known to be the case for graphs of rank-width at most k [15]. Ganian, Hlinený, and Obdrzálek proposed a polynomial-time algorithm for #SAT for formulas of bounded *signed incidence rank-width*, a parameter that is equivalent to signed incidence clique-width [12]. Signed incidence clique-width and signed incidence rank-width are currently the most general structural parameters based on width measures for which #SAT is known to be polynomial time-tractable. The following two examples show that modular incidence treewidth is incomparable with these parameters and more general than incidence treewidth. Due to space constraints, we will only sketch the proofs and refer to known results wherever possible.

▶ **Example 2** (Fischer, Makowsky, and Ravve [10])**.** Let $x_1, \ldots, x_m$ be distinct variables. The formula φ_m is defined as the set of clauses $C_{i,j}$ for $1 \leq i, j \leq m$ and $i \neq j$, where $C_{i,j} = (\{x_1, \ldots, x_m\} \setminus \{x_i, x_j\}) \cup \{\neg x_i, \neg x_j\}$. The signed incidence clique-width of φ_m tends to infinity with m. The (unsigned) incidence graph corresponds to the complete bipartite graph $K_{n,m}$ for $n = \binom{m}{2}$. As in our initial example, module contraction reduces $K_{n,m}$ to a single edge, so the modular incidence treewidth of φ_m is 1 for arbitrary m.

▶ **Example 3.** Let $x_1, \ldots, x_m, y_1, \ldots, y_m$ be distinct variables. We let ψ_m consist of the clauses C_i for $1 \leq i \leq m$ where $C_i = \{y_i, x_1, \ldots, x_m\}$, along with m singleton clauses $\{x_1\}, \ldots, \{x_m\}$. The incidence graph $I(\psi_m)$ of ψ_m has no nontrivial modules (it is *prime*), so the modular incidence treewidth and incidence treewidth of ψ_m coincide. Since $I(\psi_m)$ contains $K_{m,m}$ as a subgraph, its treewidth is at least m. By contrast, it can be shown that the signed incidence clique width of ψ_m is at most 5 for arbitrary m.

▶ **Proposition 4.** *Modular incidence treewidth and signed incidence clique-width are incomparable.*

It is readily verified that modular incidence treewidth dominates incidence treewidth: by contracting modules, we obtain an induced subgraph of the incidence graph, and the treewidth

of a graph is bounded from below by the treewidth of any of its subgraphs. Also note that the sequence of formulas from Example 2 has unbounded treewidth. By combining these facts, we can conclude that modular incidence treewidth is more general than incidence treewidth. Incidence clique-width is known to be more general than signed incidence clique-width [10]. It is also more general than modular incidence treewidth.

▶ **Proposition 5.** *Incidence clique-width is more general than modular incidence treewidth.*

Proof. It is well known that there is a function that provides an upper bound on the clique-width of any graph in terms of its treewidth, and that clique-width is invariant under contraction of modules [7]. It follows that incidence clique-width dominates modular incidence treewidth. Because incidence clique-width dominates signed incidence clique-width, the sequence of formulas described in Example 3 has bounded incidence clique-width. We conclude that incidence clique-width is more general than modular incidence treewidth. ◀

The incidence β-hypertree width is parameter that is yet more general than incidence clique-width [13]. At this time, it remains open whether #SAT is polynomial-time tractable on formulas for which one of these parameters is bounded.

We note that tractability of #SAT for formulas of bounded primal treewidth, bounded incidence treewidth, or bounded signed incidence clique-width can also be established using algorithmic meta-theorems by Courcelle, Makowsky, and Rotics [5, 6]. The hardness result presented in Section 4 implies that our Theorem 1 cannot be proved in this way.

2 Preliminaries

Let X and Y be sets and let $f : X \to Y$ be a function. We write $f^{-1}(y) = \{x \in X \mid f(x) = y\}$. For a subset $X' \subseteq X$, we let $f|_{X'}$ denote the restriction of f to X'. If $Y = 2^Z$ for some set Z and $f(x) = \{z\}$ for some $z \in Z$, then we may write $f(x) = z$ instead. If $g : X^* \to Y^*$ is a function with $g(x) = f(x)$ for all $x \in X \cap X^*$, then the function $f \cup g : X \cup X^* \to Y \cup Y^*$ is defined as $(f \cup g)(x) = f(x)$ if $x \in X$ and $(f \cup g)(x) = g(x)$ if $x \in X^* \setminus X$.

We assume an infinite supply of propositional *variables*. A *literal* is a variable x or a negated variable $\overline{x}$; we put $\mathrm{var}(x) = \mathrm{var}(\overline{x}) = x$; if $y = \overline{x}$ is a literal, then we write $\overline{y} = x$. For a set S of literals we put $\overline{S} = \{\,\overline{x} \mid x \in S\,\}$; S is *tautological* if $S \cap \overline{S} \neq \emptyset$. A *clause* is a finite non-tautological set of literals. A finite set of clauses is a *CNF formula* (or *formula*, for short). The *length* of a formula F is given by $\sum_{C \in F} |C|$. The *union* of two clauses C and D denoted CD is the union of the literals of C and D. A variable x *occurs* in a clause C if $x \in C \cup \overline{C}$. We let $\mathrm{var}(C)$ denote the set of variables that occur in C. A variable x *occurs* in a formula F if it occurs in one of its clauses, and we let $\mathrm{var}(F) = \bigcup_{C \in F} \mathrm{var}(C)$.

Let F be a formula. The *incidence graph* of F is the bipartite graph $I(F)$ with vertex set $\mathrm{var}(F) \cup F$ and edge set $\{\,Cx \mid C \in F \text{ and } x \in \mathrm{var}(C)\,\}$. Two vertices are *twins* if they have the same neighbors in $I(F)$. The equivalence classes of the twin relation are called *modules*. By the definition of $I(F)$, twins either consist of two variables or of two clauses. If the vertices of a module correspond to clauses, then we call the module a *clause module*; otherwise we call it a *variable module*. By definition, all clauses of any clause module $\mathcal{C}$ of F contain all variables of any variable module X of F if and only if one clause of $\mathcal{C}$ contains at least one variable from X. This implies that the set of variable modules of $\mathcal{C}$ is a subset of the set of variable modules of F. For a set of clause or variable modules $\mathcal{S}$, we let $\langle \mathcal{S} \rangle = \bigcup_{S \in \mathcal{S}} S$ denote the union of the elements of $\mathcal{S}$. The *modular incidence graph* $I^*(F)$ is the bipartite graph obtained from $I(F)$ after removing all but one vertices of each module. A *truth assignment* is a mapping $\tau : X \to \{\,0, 1\,\}$ defined on some set $X \subseteq \mathrm{var}(F)$ of variables.

For $x \in X$, we define $\tau(\overline{x}) = 1 - \tau(x)$. A truth assignment τ *satisfies* a clause C if C contains some literal x with $\tau(x) = 1$. If τ satisfies all clauses of F, then τ *satisfies* F; in that case we call F satisfiable. The *satisfiability* (SAT) problem is that of testing whether a given formula is satisfiable. *Propositional model counting* (#SAT) is a generalization of SAT that asks for the number of satisfying truth assignments.

Let X be a set of variables. For a clause C, we let $C^X = \{\, \ell \in C \mid \mathrm{var}(\ell) \in X \,\}$. For a formula F, we let $F^X = \{\, C^X \mid C \in F \,\} \setminus \{\emptyset\}$. For a truth assignment τ, we let $\mathbf{clause}(\tau) = \tau^{-1}(0) \cup \overline{\tau^{-1}(1)}$. This leads to the following lemma.

▶ **Lemma 6.** *Let $\tau : X \to \{0,1\}$ be a truth assignment, and let C be a clause. Then C is not satisfied by τ if and only if $C^X = \mathbf{clause}(\tau|_{X \cap \mathrm{var}(C)})$.*

We now define the notion of a projection, which plays an important role in our paper. As an aside, Kaski, Koivisto, and Nederlof [16] recently showed that SAT can be solved in polynomial time for formulas with a bounded number of projections. Let F be a formula and X be a set of variables. We refer to the set of clauses of F *not* satisfied by a truth assignment $\sigma : X \to \{0,1\}$ as the *(negative) projection* of σ on F denoted $F(\sigma)$. We denote the set of all these projections by $\mathscr{P}_{(F,X)} = \{\, F(\sigma) \mid \sigma : X \to \{0,1\} \,\}$. If $X \supseteq \mathrm{var}(F)$, then we may write $\mathscr{P}_F$ instead, as $\mathscr{P}_{(F,X)} = \mathscr{P}_{(F,\mathrm{var}(F))}$ holds in that case. Note that F is satisfiable if and only if the *empty* projection $\emptyset$ belongs to $\mathscr{P}_F$, and that the number of satisfying truth assignments of F is equal to $|\{\sigma : \mathrm{var}(F) \to \{0,1\} \mid F(\sigma) = \emptyset\}|$. The following lemma states a useful property of projections.

▶ **Lemma 7.** *Let F be a formula and let X, Y be two sets of variables. Let $\sigma : X \to \{0,1\}$ and $\tau : Y \to \{0,1\}$ be two truth assignments that agree on $X \cap Y$. Then $F(\sigma \cup \tau) = F(\sigma) \cap F(\tau)$.*

For a clause C and a formula F we let $\mathbf{select}(F, C) = \{\, C' \in F \mid C \subseteq C' \,\}$. We will now prove two useful lemmas. The first lemma is for clause modules $\mathcal{C}$. It implies that every truth assignment on $\mathrm{var}(\langle \mathcal{C} \rangle)$ either satisfies $\mathcal{C}$ or does not satisfy a unique clause of $\mathcal{C}$. The second lemma is similar but with respect to variable modules X.

▶ **Lemma 8.** *Let $\mathcal{C}$ be a clause module of a formula F, and let τ be a truth assignment defined on a set X of variables. Then $\mathcal{C}(\tau) = \mathbf{select}(\mathcal{C}, \mathbf{clause}(\tau|_{X \cap \mathrm{var}(\mathcal{C})}))$.*

Proof. Let $C \in \mathcal{C}$. Because $\mathcal{C}$ is a clause module, $\mathrm{var}(C) = \mathrm{var}(\mathcal{C})$. Then, by using Lemma 6 and the definitions of **clause** and **select**, we find that $C \in \mathcal{C}(\tau)$ if and only if C is not satisfied by τ if and only if $C^X = \mathbf{clause}(\tau|_{X \cap \mathrm{var}(C)}) = \mathbf{clause}(\tau|_{X \cap \mathrm{var}(\mathcal{C})})$ if and only if $C \in \mathbf{select}(\mathcal{C}, \mathbf{clause}(\tau|_{X \cap \mathrm{var}(\mathcal{C})}))$. ◀

▶ **Lemma 9.** *Let X be a variable module of a formula F, and let τ be a truth assignment defined on a superset of X. If $F^X(\tau) \neq \emptyset$, then $F^X(\tau) = \mathbf{clause}(\tau|_X)$.*

Proof. Let $C \in F^X$. Because X is a variable module, $\mathrm{var}(C) = X$. Lemma 6 tells us that C is not satisfied by τ if and only if $C = \mathbf{clause}(\tau|_{X \cap \mathrm{var}(C)}) = \mathbf{clause}(\tau|_X)$. ◀

We also need the following lemma.

▶ **Lemma 10.** *Let X be a variable module of a formula F. Let $E = \{\sigma : X \to \{0,1\} \mid F^X(\sigma) = \Pi\}$ for some $\Pi \in \mathscr{P}_{F^X}$. Then $|E| = 1$ if $\Pi \neq \emptyset$, and $|E| = 2^{|X|} - |F^X|$ if $\Pi = \emptyset$.*

Proof. First suppose that $\Pi \neq \emptyset$. By Lemma 9, the only truth assignment $\tau : X \to \{0,1\}$ with $F^X(\tau) = \Pi$ is the truth assignment τ_X with $F^X(\tau_X) = \mathbf{clause}(\tau_X)$. Hence, $|E| = 1$ in this case. Now suppose that $\Pi = \emptyset$. The number of truth assignments defined on X is equal

to $2^{|X|}$. By Lemma 9, each such truth assignment τ_X does not satisfy one unique clause with set of variables X, namely the clause $\mathbf{clause}(\tau_X)$. Then there are exactly $2^{|X|} - |F^X|$ truth assignments τ_X that do satisfy F^X, i.e., that have $F^X(\tau_X) = \emptyset = \Pi$. Hence, in this case, $|E| = 2^{|X|} - |F^X|$. ◀

We finish this section with some terminology on tree decompositions. Let $G = (V_G, E_G)$ be a finite, undirected graph with neither self-loops nor multiple edges. A *tree decomposition* of G is a triple (T, χ, r), where $T = (V_T, E_T)$ is a tree rooted at r and $\chi : V_T \to 2^{V_G}$ is a labeling of the vertices of T (called *nodes*) by subsets of V_G (called *bags*) such that the following conditions hold:

1. $\bigcup_{t \in V_T} \chi(t) = V_G$,
2. for each edge $uv \in E_G$, there is a node $t \in V_T$ with $\{u, v\} \subseteq \chi(t)$,
3. for each vertex $x \in V_G$, the set of nodes t with $x \in \chi(t)$ forms a connected subtree of T.

The *width* of a tree decomposition (T, χ) is the size of a largest bag $\chi(t)$ minus 1. The *treewidth* of G is the minimum width over all possible tree decompositions of G. A tree decomposition (T, χ, r) is *nice* if T is a binary tree such that the nodes of T belong to one of the following four types:

A. a *leaf node* t is a leaf of T,
B. an *introduce node* t has one child t' and $\chi(t) \setminus \{v\} = \chi(t')$ for some vertex $v \in V_G$,
C. a *forget node* t has one child t' and $\chi(t') \setminus \{v\} = \chi(t)$ for some vertex $v \in V_G$,
D. a *join node* t has two children t_1, t_2' and $\chi(t) = \chi(t_1) = \chi(t_2)$.

Kloks [17] showed that every tree decomposition of a graph G can be converted in linear time to a nice tree decomposition, such that the size of the largest bag does not increase, and the corresponding tree has at most $4|V_G|$ nodes.

Let F be a formula. We call the treewidth of $I^*(F)$ the *modular incidence treewidth* of F. Let (T, χ, r) be a nice tree decomposition of $I^*(F)$. For $t \in V_T$, we write $\chi_c(t)$ and $\chi_v(t)$ to denote the sets of clause modules and variable modules in $\chi(t)$, respectively. Note that $\chi(t) = \chi_c(t) \cup \chi_v(t)$. Moreover, we let $\mathcal{X}_t$ and $\mathcal{F}_t$ denote the set of variable modules and the set of clause modules occurring in the subtree rooted at t, respectively. We write $X_t = \langle \mathcal{X}_t \rangle$ and $F_t = \langle \mathcal{F}_t \rangle$. Note that $X_r = \mathrm{var}(F)$ and $F_r = F$.

3 Solving #SAT for Formulas of Bounded Modular Treewidth

In this section, we present an algorithm for computing the number of satisfying truth assignments of a formula F. This algorithm runs in polynomial time provided that the modular incidence treewidth of F is bounded. We begin by explaining the main ideas.

Let F be a formula and X be a set of variables. We can partition truth assignments defined on X into equivalence classes with respect to a relation $\sim_{(F,X)}$, which is defined as follows. Let $\sigma, \tau : X \to \{0, 1\}$ be two distinct truth assignments. Then $\sigma \sim_{(F,X)} \tau$ if and only if σ and τ satisfy exactly the same set of clauses of F, or equivalently, if and only if $F(\sigma) = F(\tau)$. Due to the latter equivalence, we can speak about *the* projection of an equivalence class of $\sim_{(F,X)}$ on F. Recall that the number of satisfying truth assignments of F is equal to $|\{\sigma : \mathrm{var}(F) \to \{0, 1\} \mid F(\sigma) = \emptyset\}|$, which is the size of the equivalence class of $\sim_{(F,\mathrm{var}(F))}$ corresponding to the empty projection.

Now let (T, χ, r) be a nice tree decomposition of $I^*(F)$. We will apply dynamic programming over (T, χ, r). As is usual, we start in the leaves of the tree and, using the parent-child

relation, move to nodes closer to the root, and we stop after having processed the root. For each node $t \in V_T$, we define the formula

$$F_t^* \;=\; \big\{F^X \mid X \in \chi_v(t)\big\} \cup F_t \;=\; \big\{F^X \mid X \in \chi_v(t)\big\} \cup \langle\chi_c(t)\rangle \cup F_t \setminus \langle\chi_c(t)\rangle,$$

and we compute the sizes of those equivalence classes $[\tau]$ of $\sim_{(F_t^*, X_t)}$ that consist of truth assignments τ with $(F_t \setminus \langle\chi_c(t)\rangle)(\tau) = \emptyset$; we call such equivalence classes *transferable*. Below we explain the reasons why we do this.

First, the union of the transferable equivalence classes of $\sim_{(F_r^*, X_r)}$ that consist of truth assignments τ with $\langle\chi_c(r)\rangle(\tau) = \emptyset$ in addition to $(F_r \setminus \langle\chi_c(t)\rangle)(\tau) = (F \setminus \langle\chi_c(t)\rangle)(\tau) = \emptyset$ contains all satisfying truth assignments of F. Note that such truth assignments may not satisfy some formula F^X for some $X \in \chi_v(r)$, but in that case only formulas in $F^X \setminus F_r = F^X \setminus F$ are not satisfied, and these are irrelevant for our output.

Second, we do not have to compute the sizes of any non-transferable equivalence classes of $\sim_{(F_t^*, X_t)}$. The reason for this is that these equivalence classes only contain truth assignments τ that cannot be extended to satisfying truth assignments of F. This can be seen as follows. Let τ be a truth assignment from a non-transferable equivalence class of $\sim_{(F_t^*, X_t)}$. By definition, $F_t \setminus \langle\chi_c(t)\rangle$ contains a clause C not satisfied by τ. Then C must contain at least one variable $x \in X_r \setminus X_t$ in order to be satisfied by an extension of τ. Let $\mathcal{C}$ be the clause module that contains C. Let X be the variable module that contains x. Then $X\mathcal{C} \in I^*(F)$. Hence, by condition 2 of the definition of a tree decomposition, there exists a node $t' \in V_T$ with $\{X, \mathcal{C}\} \subseteq \chi(t')$. Because $x \in X_r \setminus X_t$, we find that $X \in \mathcal{X}_r \setminus \mathcal{X}_t$. Because $X \in \chi(t')$, this means that t' is not a node of the subtree of T rooted at t. Because $\mathcal{C} \in \chi(t')$, we then find that $\mathcal{C} \in \mathcal{F}_r \setminus \mathcal{F}_t$. However, as $C \in F_t \setminus \langle\chi_c(t)\rangle$, we also have $\mathcal{C} \in \mathcal{F}_t \setminus \chi_c(t)$. This violates condition 3 of the definition of a tree decomposition. Hence, non-transferable equivalence classes may be discarded during our dynamic programming.

Third, we must keep track of how truth assignments that not yet satisfy all clauses in F can be extended to truth assignments that do satisfy F in a later stage of the dynamic programming. In particular, such truth assignments may not yet satisfy clauses C that belong to clause modules in $\chi_c(t)$ or that contain variables from variable modules in $\chi_v(t)$; in the latter case their restriction C^X belongs to F^X for some $X \in \chi_v(t)$. In order to do this bookkeeping we must partition truth assignments that satisfy $F_t \setminus \langle\chi_c(t)\rangle$ into equivalence classes of truth assignments that satisfy exactly the same clauses of any F^X with $X \in \chi_v(t)$ and exactly the same clauses of any $\mathcal{C} \in \chi_c(t)$. The reason why the partitioning does not cause an exponential blow-up if the modular incidence treewidth of F is bounded is due to two of our lemmas from Section 2. For a clause module $\mathcal{C} \in \chi_c(t)$, the number of equivalence classes of $\sim_{(\mathcal{C}, X_t)}$ on is bounded by $|\mathcal{C}|+1$ due to Lemma 8. For a variable module $X \in \chi_v(t)$, the number of equivalence classes of $\sim_{(F^X, X_t)}$ is bounded by $|F|+1$ due to Lemma 9. Hence, the total number of different transferable equivalence classes of $\sim_{(F_t^*, X_t)}$ is at most

$$\prod_{\mathcal{C} \in \chi_c(t)} (|\mathcal{C}|+1) \cdot \prod_{X \in \chi_v(t)} (|F|+1) \leq (|F|+1)^{|\chi_c(t)|+|\chi_v(t)|} = (|F|+1)^{|\chi(t)|} \leq (|F|+1)^k, \qquad (1)$$

where k denotes the treewidth of $I^*(F)$, i.e., the modular incidence treewidth of F. We observe that this bound is polynomial if k is fixed.

In order to describe the transferable equivalence classes, we use some terminology introduced by Ganian, Hlinený and Obdrzálek [12], which we adjusted for our purposes. Let $t \in T$. A *shape* for t is a pair of mappings (α, θ) where α has domain $\chi_v(t)$ with $\alpha(X) \in \mathscr{P}_{F^X}$ for all $X \in \chi_v(t)$ and θ has domain $\chi_c(t)$ with $\theta(\mathcal{C}) \in \mathscr{P}_{(\mathcal{C}, X_t)}$ for all $\mathcal{C} \in \chi_c(t)$. An assignment $\tau : X_t \to \{0,1\}$ is said to be *of shape* (α, θ) if it satisfies the following three conditions:

(a) $F^X(\tau) = \alpha(X)$ for all $X \in \chi_v(t)$
(b) $\mathcal{C}(\tau) = \theta(\mathcal{C})$ for all $\mathcal{C} \in \chi_c(t)$
(c) $(F_t \setminus \langle \chi_c(t) \rangle)(\tau) = \emptyset$.

In other words, the set of assignments τ that are of shape (α, θ) describes exactly one transferable equivalence class of $\sim_{(F_t^*, X_t)}$. From now we denote this class by $N_t(\alpha, \theta)$, and we write $n_t(\alpha, \theta) = |N_t(\alpha, \theta)|$. We denote the set of all shapes for t that correspond to a transferable equivalence class by $\mathscr{S}_t$. By (1), we have $|\mathscr{S}_t| \leq (|F| + 1)^k$ for all nodes $t \in V_T$. Also note that any truth assignment $\tau : X_t \to \{0, 1\}$ has a (unique) shape if and only if $(F_t \setminus \langle \chi_c(t) \rangle)(\tau) = \emptyset$. We sometimes denote the shape of such a truth assignment τ by $(\alpha_\tau^t, \theta_\tau^t)$, where $\alpha_\tau^t(X) = F^X(\tau)$ for all $X \in \chi_v(t)$ and $\theta_\tau^t(\mathcal{C}) = \mathcal{C}(\tau)$ for all $\mathcal{C} \in \chi_c(t)$. Because equivalence classes are nonempty by definition, not all pairs (α, θ) with $\alpha(X) \in \mathscr{P}_{F^X}$ for all $X \in \chi_v(t)$ and $\theta(\mathcal{C}) \in \mathscr{P}_{(\mathcal{C}, X_t)}$ for all $\mathcal{C} \in \chi_c(t)$ form a shape for a node $t \in V_T$. We make this more explicit in our next lemma (see condition (ii) in particular).

▶ **Lemma 11.** *Let $(\alpha, \theta) \in \mathscr{S}_t$ with $t \in V_T$, and let $\chi_v^*(t) \subseteq \chi_v(t)$. Moreover, let $\tau : X_t \to \{0, 1\}$ satisfy $F^X(\tau) = \alpha(X)$ for all $X \in \chi_v^*(t)$. For all $\mathcal{C} \in \chi_c(t)$, the following three conditions hold:*

(i) If $\mathcal{C}$ has no variable modules in $\chi_v^*(t)$, then $\mathcal{C}(\tau|_{\langle \chi_v^*(t) \rangle}) = \mathcal{C}$.
(ii) If $\mathcal{C}$ has some variable module $X \in \chi_v^*(t)$ with $\alpha(X) = \emptyset$, then $\mathcal{C}(\tau|_{\langle \chi_v^*(t) \rangle}) = \mathcal{C}(\tau) = \emptyset$, and moreover, $\theta(\mathcal{C}) = \emptyset$ should $\tau \in N_t(\alpha, \theta)$.
(iii) If $\mathcal{C}$ has exactly $p \geq 1$ variable modules $X_1, \ldots, X_p$ in $\chi_v^*(t)$ and $\alpha(X_i) \neq \emptyset$ for $i = 1, \ldots, p$, then $\mathcal{C}(\tau|_{\langle \chi_v^*(t) \rangle}) = \mathbf{select}(\mathcal{C}, \alpha(X_1) \cdots \alpha(X_p))$.

We will now give the exact details of our dynamic programming, i.e., how we compute all sizes $n_t(\alpha, \theta)$ over all $t \in V_T$ in order to be able to compute the desired output

$$n = \sum_{(\alpha, \theta) \in \mathscr{S}_r^*} n_r(\alpha, \theta),$$

where $\mathscr{S}_r^*$ consists of those $(\alpha, \theta) \in \mathscr{S}_r$ with $\theta(\mathcal{C}) = \emptyset$ for all $\mathcal{C} \in \chi_c(r)$. Note that $\mathscr{S}_r^* = \emptyset$ is possible; in that case F is not satisfiable and $n = 0$.

Recall that the nodes of a tree of a nice tree decomposition can be partitioned into four types of nodes. Our next four lemmas show how to compute the sizes $n_t(\alpha, \theta)$ for these four types, i.e., for leaf nodes, introduce nodes, forget nodes and join nodes t, respectively.

▶ **Lemma 12.** *Let t be a leaf node and $(\alpha, \theta) \in \mathscr{S}_t$. Then $n_t(\alpha, \theta) = \prod_{X \in \alpha^{-1}(\emptyset)} (2^{|X|} - |F^X|)$.*

Let $(\alpha, \theta) \in \mathscr{S}_t$ for some $t \in V_T$. We define a mapping g with domain $\chi_c(t) \times \chi_v(t)$ as follows. When $X \in \chi_v(t)$ is not a variable module of a clause $\mathcal{C} \in \chi_c(t)$, we let $g(\mathcal{C}, X) = \mathcal{C}$. Otherwise, we let $g(\mathcal{C}, X) = \emptyset$ if $\alpha(X) = \emptyset$, and $g(\mathcal{C}, X) = \mathbf{select}(\mathcal{C}, \alpha(X))$ if $\alpha(X) \neq \emptyset$.

▶ **Lemma 13.** *Let $t \in T$ be an introduce node with child t', such that $\chi(t) \setminus \{S\} = \chi(t')$ for a module $S \in \chi(t)$. Let $(\alpha, \theta) \in \mathscr{S}_t$. Moreover, let $\alpha' = \alpha|_{\chi_v(t')}$ and $\theta' = \theta|_{\chi_c(t')}$.*

(i) If $S \in \chi_v(t)$, then $n_t(\alpha, \theta) = \begin{cases} \sum_{\theta^* \in \mathscr{T}} n_{t'}(\alpha', \theta^*) & \text{if } \alpha(S) \neq \emptyset \\ (2^{|S|} - |F^S|) \sum_{\theta^* \in \mathscr{T}} n_{t'}(\alpha', \theta^*) & \text{if } \alpha(S) = \emptyset, \end{cases}$

where $\mathscr{T} = \{\, \theta^* \mid (\alpha', \theta^*) \in \mathscr{S}_{t'}$ and $\theta^*(\mathcal{C}) \cap g(\mathcal{C}, S) = \theta(\mathcal{C})$ for all $\mathcal{C} \in \chi_c(t) \,\}$.

(ii) If $S \in \chi_c(t)$, then $n_t(\alpha, \theta) = n_{t'}(\alpha, \theta')$.

▶ **Lemma 14.** *Let $t \in T$ be a forget node with child t', such that $\chi(t) = \chi(t') \setminus \{S\}$ for a module $S \in \chi(t')$. Let $(\alpha, \theta) \in \mathscr{S}_t$.*

(i) If $S \in \chi_v(t')$, then $n_t(\alpha,\theta) = \sum_{\Pi \in \mathscr{P}_{F^S}} n_{t'}(\alpha \cup \{(S,\Pi)\}, \theta)$.

(ii) If $S \in \chi_c(t')$, then $n_t(\alpha,\theta) = n_{t'}(\alpha, \theta \cup \{(S,\emptyset)\})$.

▶ **Lemma 15.** *Let $t \in T$ be a join node with children t_1 and t_2. Let $(\alpha,\theta) \in \mathscr{S}_t$. Moreover, let $\mathscr{T}_{1,2} = \{\, (\theta_1,\theta_2) \mid (\alpha,\theta_1) \in \mathscr{S}_{t_1}, (\alpha,\theta_2) \in \mathscr{S}_{t_2}$, and $\theta_1(\mathcal{C}) \cap \theta_2(\mathcal{C}) = \theta(\mathcal{C})$ for all $\mathcal{C} \in \chi_c(t) \,\}$. Then the following equality holds:*

$$n_t(\alpha,\theta) = \frac{1}{\prod_{X \in \alpha^{-1}(\emptyset)} (2^{|X|} - |F^X|)} \sum_{(\theta_1,\theta_2) \in \mathscr{T}_{1,2}} n_{t_1}(\alpha,\theta_1) \cdot n_{t_2}(\alpha,\theta_2).$$

We are now ready to present the proof of our main result, which we restate below.

Theorem 1. *The number of satisfying assignments of a CNF formula F with modular incidence treewidth at most k can be computed in time $\ell^{\mathcal{O}(k)}$, where ℓ is the length of F.*

Proof. Let F be a formula with modular incidence treewidth at most k. We first construct $I(F)$ and perform module contraction to obtain $I^*(F)$. Clearly, this can be done in time $\mathcal{O}(\ell^c)$ for some constant c independent of F. By using Bodlaender's algorithm [2] we obtain in linear time a tree decomposition of $I^*(F)$ of width at most k. Recall that Kloks [17] showed that such a tree decomposition can be converted in linear time to a nice tree decomposition (T,χ,r) of width at most k, and at most $4|V_{I^*(F)}| \leq 4\ell$ nodes. Also recall that the desired output is

$$n = \sum_{(\alpha,\theta) \in \mathscr{S}_r^*} n_r(\alpha,\theta),$$

where $\mathscr{S}_r^*$ consists of those $(\alpha,\theta) \in \mathscr{S}_r$ with $\theta(\mathcal{C}) = \emptyset$ for all $\mathcal{C} \in \chi_c(r)$. In order to compute n, we compute the sizes $n_t(\alpha,\theta)$ for all $t \in V_T$. Here we follow a bottom-up approach starting at the leaves. For each node $t \in V_T$, we use one of the Lemmas 12–15 depending on the type of t, i.e., whether t is a leaf, introduce, forget or join node, respectively. The correctness of our algorithm follows from these lemmas and some extra arguments: when any new clause module $\mathcal{C}$ is introduced in a node t (that is either a leaf or an introduce node) we find that $\theta(\mathcal{C}) = \mathcal{C}(\tau) = \mathcal{C}(\tau|_{\langle \chi_v(t) \rangle})$ should τ be a truth assignment in $N_t(\alpha,\theta)$ for some pair (α,θ), and then Lemma 11 tells us that $\theta(\mathcal{C})$ must be fixed to some set of clauses. Hence, if $\theta(\mathcal{C})$ is not equal to this set of clauses, then we must discard the pair (α,θ).

If t is a leaf node, then we first determine which pairs (α,θ) may belong to $\mathscr{S}_t$. We do this by using Lemma 11, where we choose $\chi_v^*(t) = \chi_v(t) = \mathcal{X}_t$; the latter equality follows from the fact that t is a leaf. We then find that there can only exist a truth assignment $\tau \in N_t(\alpha,\theta)$ if the following three conditions hold for all $\mathcal{C} \in \chi_c(t)$.

(i) $\theta(\mathcal{C}) = \mathcal{C}(\tau) = \mathcal{C}(\tau|_{\langle \chi_v(t) \rangle}) = \mathcal{C}$ if $\mathcal{C}$ has no variable modules in $\chi_v(t)$.
(ii) $\theta(\mathcal{C}) = \emptyset$ if $\mathcal{C}$ has a variable module in $\chi_c(t)$ with $\alpha(X) = \emptyset$.
(iii) $\theta(\mathcal{C}) = \mathcal{C}(\tau) = \mathcal{C}(\tau|_{\langle \chi_v(t) \rangle}) = \mathbf{select}(\mathcal{C}, \alpha(X_1) \cdots \alpha(X_p))$ If $\mathcal{C}$ has exactly $p \geq 1$ variable modules $X_1, \ldots, X_p$ in $\chi_v(t)$ and $\alpha(X_i) \neq \emptyset$ for $i = 1, \ldots, p$.

Note that checking these three conditions for all $\mathcal{C} \in \chi_c(t)$ takes linear time for a given pair (α,θ). If these conditions are violated for some $\mathcal{C} \in \chi_c(t)$, then we discard the pair (α,θ). Otherwise, i.e., if these conditions are satisfied for all $\mathcal{C} \in \chi_c(t)$, then we apply Lemma 12. If we find that $n_t(\alpha,\theta) = 0$, then $(\alpha,\theta) \notin \mathscr{S}_t$ (as equivalence classes are nonempty by definition)

and we discard this pair. If t is a introduce node that introduces a variable module, then we apply Lemma 13 (i). Here, we find that a pair $(\alpha, \theta) \in \mathscr{S}_t$ only if $n_t(\alpha, \theta) > 0$. If t is a introduce node that introduces a clause module $\mathcal{C}$, then we we first determine which pairs (α, θ) may belong to $\mathscr{S}_t$. We do this by using Lemma 11, where we choose $\chi_v^*(t) = \chi_v(t)$. Because $\mathcal{C}$ is introduced by t, we find that $\mathcal{C} \notin \mathcal{F}_t \setminus \chi_c(t)$. Hence, by definition of a tree decomposition, $\mathcal{C}$ contains no variable modules from $\mathcal{X}_t \setminus \chi_v(t)$. This means that $\mathcal{C}(\tau) = \mathcal{C}(\tau|_{\langle \chi_v(t) \rangle})$ should τ be a truth assignment in $N_t(\alpha, \theta)$. Lemma 11 tells us that this can only happen if the above conditions (i)–(iii) hold. Note that checking these three conditions for $\mathcal{C}$ takes linear time for a given pair (α, θ). If these conditions are satisfied for $\mathcal{C}$, then we apply Lemma 13 (ii). If we find that $n_t(\alpha, \theta) = 0$, then $(\alpha, \theta) \notin \mathscr{S}_t$ and we discard this pair. If t is a forget node that forgets a variable module or a clause module, then we apply Lemmas 14 (i) and (ii), respectively. If t is a join node, then we apply Lemma 15. In all these three cases, we find that a pair $(\alpha, \theta) \in \mathscr{S}_t$ only if $n_t(\alpha, \theta) > 0$.

As soon as we are of distance two from a node t, we can forget the sizes $n_t(\alpha, \theta)$. Recall that $|\mathscr{S}_t| \leq (|F| + 1)^k$ for all nodes $t \in V_T$ due to (1). This bound on the number of transferable equivalence classes for a node t has the following two consequences. First, as can be seen from the equations in Lemmas 12–15, it means that it takes at most $p(\ell)(|F|+1)^k(|F|+1)^k = (|F|+1)^{2k}$ time to compute the size $n_t(\alpha, \theta)$ of an equivalence class $N_t(\alpha, \theta)$, where $p(\ell)$ is a polynomial that only depends on ℓ, which also includes the additional time necessary to verify whether a pair (α, θ) may belong to $\mathscr{S}_t$ in case of Lemma 12 and Lemma 13 (ii). Second, it means that for each node we must compute and verify at most $(|F| + 1)^k$ sizes $n_t(\alpha, \theta)$. As the total number of nodes is at most 4ℓ, we find that the total running time is at most $4\ell \cdot (|F| + 1)^k \cdot p(\ell) \cdot (|F| + 1)^{2k} = \ell^{\mathcal{O}(k)}$. ◀

4 A (Parameterized) Hardness Result

The polynomial-time algorithm developed in the proof of Theorem 1 runs in time $\ell^{\mathcal{O}(k)}$ for formulas of length ℓ and modular incidence treewidth at most k. That is, the order of the polynomial depends on k. The question arises whether this dependency is necessary: *Is there a better algorithm with a running time of, say, $\mathcal{O}(\ell^c)$ where c is a constant independent of k?* We give a negative answer subject to the complexity theoretic assumption W[1] $\neq$ FPT from the area of Parameterized Complexity.

We briefly review basic concepts of Parameterized Complexity; for more information we refer to other sources [8, 11, 18]. An instance of a parameterized problem is a pair (x, k), where x is the *main part* and k (usually a non-negative integer) is the *parameter*. A parameterized problem is *fixed-parameter tractable* if it can be solved in time $\mathcal{O}(f(k)|x|^c)$ where f is a computable function and c is a constant independent of k. FPT denotes the class of all fixed-parameter tractable decision problems. Parameterized Complexity offers a completeness theory similar to the theory of NP-completeness for non-parameterized problems. A parameterized problem P *fpt-reduces* to a parameterized problem Q if we can transform an instance (x, k) of P into an instance (x', k') of Q with $k' \leq g(k)$ in time $\mathcal{O}(f(k)|x|^c)$ (f, g are arbitrary computable functions, c is a constant) such that (x, k) is a yes-instance of P if and only if (x', k') is a yes-instance of Q. A parameterized complexity class is the class of parameterized decision problems fpt-reducible to a certain parameterized decision problem. Of particular interest is the class W[1] which is considered as the parameterized analog to NP. For example, the Clique problem (given a graph G and an integer k, decide whether G contains a *k-clique* a complete subgraph on k vertices), parameterized by k, is well-known to be W[1]-complete. It is believed that FPT $\neq$ W[1], and there is strong theoretical evidence

that supports this belief; for example, FPT = W[1] implies that the Exponential Time Hypothesis fails [11].

Ordyniak, Paulusma, and Szeider [19] showed that satisfiability is W[1]-hard when parameterized by the incidence β-hypertree width (see Section 1.1), using an fpt-reduction from the following problem, which is W[1]-complete [21] (a k-partite graph is *balanced* if its k partition classes are of the same size): The input is a balanced k-partite graph $G = (V_1, \ldots, V_k, E)$, the parameter is k. The question is whether G contains a k-clique.

Let $G = (V_1, \ldots, V_k)$ be a balanced k-partite graph for $k \geq 2$. The reduction [19] maps the instance (G, k) to an instance (F, k) such that k is an upper bound on the β-hypertree width of $I(F)$. By taking a closer look at the incidence graph $I(F)$, we will show that the modular incidence treewidth of F can also be bounded by a function of k. Let $V_i = \{v_1^i, \ldots, v_n^i\}$. The incidence graph $I(F)$ of F is structured as follows. One vertex class (corresponding to variables of F) contains the vertices of G plus new vertices z_j^i for $1 \leq i \leq k$ and $1 \leq j \leq n-1$. The other vertex class (corresponding to clauses of F) consists of vertices $C_{u,v}$ for $u \in V_i, v \in V_j$ $(i \neq j)$ and $uv \notin E(G)$ such that $N_{I(F)}(C_{u,v}) = V_i \cup V_j$. Moreover, for $1 \leq i \leq k$, it contains the vertices $D_1^i, \ldots, D_n^i$ with $N_{I(F)}(D_1^i) = \{z_1^i, v_1^i, v_2^i\}$, $N_{I(F)}(D_j^i) = \{z_j^i, z_{j-1}^i, v_{j+1}^i\}$ for $2 \leq j \leq n-1$ and $N_{I(F)}(D_n^i) = \{z_{n-1}^i\}$.

The set of vertices $C_{u,v}$ can be partitioned into modules $\mathcal{C}_1, \ldots, \mathcal{C}_m$, where $m \leq \binom{k}{2}$. By deleting these modules, we obtain a graph $I'(F)$ that consists of k connected components corresponding to the subgraphs of $I(F)$ induced by $\{v_1^i, \ldots, v_n^i, z_1^i, \ldots, z_{n-1}^i, D_1^i, \ldots, D_n^i\}$ for $1 \leq i \leq k$. Note that these component are trees, so the treewidth of $I'(F)$ is 1. Thus the graph obtained from $I'(F)$ by contracting modules has treewidth 1. We can turn the corresponding tree decomposition into a tree decomposition of $I^*(F)$ by simply adding the set of clause modules $\{\mathcal{C}_1, \ldots, \mathcal{C}_m\}$ to each bag. So the modular incidence treewidth of F is at most $m + 1 \leq \binom{k}{2} + 1$. This proves the following result.

▶ **Theorem 16.** *The* Satisfiability *problem is* W[1]*-hard, when parameterized by an upper bound on the modular incidence treewidth of the input formula.*

That is, already deciding whether $\#(F) > 0$ for a formula F of length ℓ and bounded modular incidence treewidth cannot be done in time $\mathcal{O}(\ell^c)$ for constant c unless FPT = W[1] (where $\#(F)$ denotes the number of satisfying truth assignments of F). In particular, this implies a negative answer to the question raised at the beginning of this section.

5 Conclusion

In this paper, we proved that #SAT becomes polynomial-time tractable on formulas of bounded *modular incidence treewidth*. Modular incidence treewidth combines treewidth and *module contraction*, a powerful preprocessing technique widely used in combinatorial optimization. The resulting parameter is incomparable with the most general structural parameters for which #SAT is known to be tractable.

With this result, we approach the frontier of tractability from a new direction. On the other side, one can find incidence β-hypertree width and incidence clique-width. It remains open whether #SAT becomes tractable when these parameters are bounded. We think that this work is a significant step towards proving tractability of #SAT on formulas of bounded clique-width. Graphs of bounded clique-width that do not contain large bipartite subgraphs are known to have bounded treewidth [3]. This gives us reason to believe that our techniques carry over to the case of bounded incidence clique-width.

References

1 F. Bacchus, S. Dalmao, and T. Pitassi. Algorithms and complexity results for #SAT yand Bayesian inference. In *FOCS'03*, pages 340–351, 2003.

2 H. L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25(6):1305–1317, 1996.

3 B. Courcelle. The monadic second-order logic of graphs XIV: Uniformly sparse graphs and edge set quantifications. *Theoretical Computer Science*, 299(1–3):1 – 36, 2003.

4 B. Courcelle, J. Engelfriet, and G. Rozenberg. Handle-rewriting hypergraph grammars. *J. of Computer and System Sciences*, 46(2):218–270, 1993.

5 B. Courcelle, J. A. Makowsky, and U. Rotics. Linear time solvable optimization problems on graphs of bounded clique-width. *Theory Comput. Syst.*, 33(2):125–150, 2000.

6 B. Courcelle, J. A. Makowsky, and U. Rotics. On the fixed parameter complexity of graph enumeration problems definable in monadic second-order logic. *Discr. Appl. Math.*, 108(1-2):23–52, 2001.

7 B. Courcelle and S. Olariu. Upper bounds to the clique-width of graphs. *Discr. Appl. Math.*, 101(1-3):77–114, 2000.

8 R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer Verlag, 1999.

9 M. R. Fellows, F. A. Rosamond, U. Rotics, and S. Szeider. Clique-width is NP-complete. *SIAM J. Discrete Math.*, 23(2):909–939, 2009.

10 E. Fischer, J. A. Makowsky, and E. R. Ravve. Counting truth assignments of formulas of bounded tree-width or clique-width. *Discr. Appl. Math.*, 156(4):511–529, 2008.

11 J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer Verlag, 2006.

12 R. Ganian, P. Hlinený, and J. Obdrzálek. Better algorithms for satisfiability problems for formulas of bounded rank-width. In K. Lodaya and M. Mahajan, editors, *FSTTCS 2010*, volume 8 of *LIPIcs*, pages 73–83. 2010.

13 G. Gottlob and R. Pichler. Hypergraphs in model checking: Acyclicity and hypertree-width versus clique-width. In *ICALP 2001*, volume 2076 of *Lecture Notes in Computer Science*, pages 708–719, 2001.

14 M. Habib and C. Paul. A survey of the algorithmic aspects of modular decomposition. *Computer Science Review*, 4(1):41–59, 2010.

15 P. Hlinený and S. il Oum. Finding branch-decompositions and rank-decompositions. *SIAM J. Comput.*, 38(3):1012–1032, 2008.

16 P. Kaski, M. Koivisto, and J. Nederlof. Homomorphic hashing for sparse coefficient extraction. In *IPEC 2012*, 2012.

17 T. Kloks. *Treewidth: Computations and Approximations*. Springer Verlag, 1994.

18 R. Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Oxford University Press, 2006.

19 S. Ordyniak, D. Paulusma, and S. Szeider. Satisfiability of acyclic and almost acyclic CNF formulas. In *FSTTCS 2010*, volume 8 of *LIPIcs*, pages 84–95. 2010.

20 S. Oum and P. Seymour. Approximating clique-width and branch-width. *J. Combin. Theory Ser. B*, 96(4):514–528, 2006.

21 K. Pietrzak. On the parameterized complexity of the fixed alphabet shortest common supersequence and longest common subsequence problems. *J. of Computer and System Sciences*, 67(4):757–771, 2003.

22 D. Roth. On the hardness of approximate reasoning. *Artificial Intelligence*, 82(1-2):273–302, 1996.

23 M. Samer and S. Szeider. Algorithms for propositional model counting. *J. of Discrete Algorithms*, vol. 8, no. 1, pp. 50–64, 2010.

24 L. G. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8(2):189–201, 1979.