Solving MaxSAT and #SAT on structured CNF formulas

Sigve Hortemo Sæther, Jan Arne Telle, and Martin Vatshelle

Department of Informatics, University of Bergen, Norway

Abstract. In this paper we propose a structural parameter of CNF formulas and use it to identify instances of weighted MaxSAT and #SAT that can be solved in polynomial time. Given a CNF formula we say that a set of clauses is projection satisfiable if there is some complete assignment satisfying these clauses only. Let the ps-value of the formula be the number of projection satisfiable sets of clauses. Applying the notion of branch decompositions to CNF formulas and using ps-value as cut function, we define the ps-width of a formula. For a formula given with a decomposition of polynomial ps-width we show dynamic programming algorithms solving weighted MaxSAT and #SAT in polynomial time. Combining with results of 'Belmonte and Vatshelle, Graph classes with structured neighborhoods and algorithmic applications, Theor. Comput. Sci. 511: 54-65 (2013)' we get polynomial-time algorithms solving weighted MaxSAT and #SAT for some classes of structured CNF formulas. For example, we get $\mathcal{O}(m^2(m+n)s)$ algorithms for formulas F of m clauses and n variables and total size s, if F has a linear ordering of the variables and clauses such that for any variable x occurring in clause C, if x appears before C then any variable between them also occurs in C, and if C appears before x then x occurs also in any clause between them. Note that the class of incidence graphs of such formulas do not have bounded clique-width.

1 Introduction

Given a CNF formula, propositional model counting (#SAT) is the problem of computing the number of satisfying assignments, and maximum satisfiability (MaxSAT) is the problem of determining the maximum number of clauses that can be satisfied by some assignment. Both problems are significantly harder than simply deciding if a satisfying assignment exists. #SAT is #P-hard [11] even when restricted to Horn 2-CNF formulas, and to monotone 2-CNF formulas [22]. MaxSAT is NP-hard even when restricted to Horn 2-CNF formulas [15], and to 2-CNF formulas where each variable appears at most 3 times [20]. Both problems become tractable under certain structural restrictions obtained by bounding width parameters of graphs associated with formulas, see for example [9,10,23,25]. For earlier work on width decompositions in this setting see e.g. [8,1]. The work we present here is inspired by the recent results of Paulusma et al [18] and Slivovsky and Szeider [24] showing that #SAT is solvable in polynomial time

when the incidence graph $I(F)$ of the input formula F has bounded modular treewidth, and more strongly, bounded symmetric clique-width.

We extend these results in several ways. We give algorithms for both #SAT and MaxSAT, and also weighted MaxSAT, finding the maximum weight of satisfiable clauses, given a set of weighted clauses. We introduce the parameter ps-width, and express the runtime of our algorithms as a function of ps-width.

Theorem 3 *Given a formula F over n variables and m clauses and of total size s, and a decomposition of F of* ps*-width k, we solve* #SAT*, and weighted* MaxSAT *in time $\mathcal{O}(k^3 s(m+n))$.*

Thus, given a decomposition having a ps-width k that is *polynomially-bounded* in the number of variables n and clauses m of the formula, we get polynomial-time algorithms. These are dynamic programming algorithms similar to the one given for #SAT in [24], but we believe that the ps-width parameter is a better measure of the inherent runtime bottleneck of #SAT and MaxSAT when using this type of dynamic programming. The essential combinatorial result enabling this improvement is Lemma 5 of this paper. The algorithm of [24] solves #SAT in time $(n+m)^{\mathcal{O}(w)}$ for w being the symmetric clique-width of the decomposition, and is thus a polynomial-time algorithm if given a decomposition with *constantly bounded* w. The result of Theorem 3 encompasses this, since we show via the concept of MIM-width [26], that any formula with constantly bounded symmetric clique-width also has polynomially bounded ps-width.

We show that a relatively rich class of formulas, including classes of unbounded clique-width, have polynomially bounded ps-width. This is shown using the concept of MIM-width of graphs, introduced in the thesis of Vatshelle [26]. See Figure 1. In particular, this holds for classes of formulas having incidence graphs that can be represented as intersection graphs of certain objects, like interval graphs [2]. We prove this also for bigraph bipartizations of these graphs, which are obtained by imposing a bipartition on the vertex set and keeping only edges between the partition classes. Some such bigraph bipartizations have been studied previously, in particular the interval bigraphs. The interval bigraphs contain all bipartite permutation graphs, and these latter graphs have been shown to have unbounded clique-width [4].

By combining an alternative definition of interval bigraphs [13] with a fast recognition algorithm [17,19] we arrive at the following. Say that a CNF formula F has an interval ordering if there exists a linear ordering of variables and clauses such that for any variable x occurring in clause C, if x appears before C then any variable between them also occurs in C, and if C appears before x then x occurs also in any clause between them.

Theorem 10 *Given a CNF formula F over n variables and m clauses and of total size s, we can in time $\mathcal{O}((m+n)s)$ decide if F has an interval ordering (yes iff $I(F)$ is an interval bigraph), and if yes we solve* #SAT *and weighted* MaxSAT *with a runtime of $\mathcal{O}(m^2(m+n)s)$.*

The algorithms of Theorem 10 may be of interest for practical applications, as there are no big hidden constants in the runtimes.

Our paper is organized as follows. In Section 2 we give formal definitions. We will be using a type of decomposition that originates in the theory of graphs and matroids where it is known as branch decomposition, see [12,21]. The standard approach is to apply this type of decomposition to the incidence graph of a formula, and evaluate its width using as cut function a graph parameter, as done in [24]. The cut function we will use is not a graph parameter, but rather the ps-value of a formula, being the number of distinct subsets of clauses that are satisfied by some complete assignment. We thus prefer to apply the decomposition directly to the formula and not to its incidence graph, although the translation between the two will be straightforward. We define cuts of formulas and ps-width of a formula. Note that a formula can have ps-value exponential and ps-width polynomial. In Section 3 we present dynamic programming algorithms that given a formula and a decomposition solves #SAT and weighted MaxSAT, proving Theorem 3. In Section 4 we investigate classes of formulas having decompositions of low ps-width, basically proving the correctness of the hierarchy presented in Figure 1. In Section 5 we consider formulas having an interval ordering and prove Theorem 10. We end in Section 5 with some open problems.

2 Framework

A *literal* is a propositional *variable* or a negated variable, x or $\neg x$, a *clause* is a set of literals, and a *formula* is a multiset of clauses. For a formula F, $\texttt{cla}(F)$ denotes the clauses in F. For a clause C, $\texttt{lit}(C)$ denotes the set of literals in C and $\texttt{var}(C)$ denotes the variables of the literals in $\texttt{lit}(C)$. For a (multi-) set S of variables and clauses, $\texttt{var}(S)$ denotes the variables of S and $\texttt{cla}(S)$ denotes the clauses. All sets of clauses mentioned in this paper are multisets. For a formula F, $\texttt{var}(F)$ denotes the union $\bigcup_{C \in \texttt{cla}(F)} \texttt{var}(C)$. For a set X of variables, an *assignment* of X is a function $\tau : X \to \{0, 1\}$. For a literal ℓ, we define $\tau(\ell)$ to be $1 - \tau(\texttt{var}(\ell))$ if ℓ is a negated variable ($\ell = \neg x$ for some variable x) and to be $\tau(\texttt{var})$ otherwise ($\ell = x$ for some variable x). A clause C is said to be *satisfied* by an assignment τ if there exists at least one literal $\ell \in \texttt{lit}(C)$ so that $\tau(\ell) = 1$. All clauses an assignment τ do not satisfy are said to be *unsatisfied* by τ. We notice that this means an empty clause will be unsatisfied by all assignments. A formula is satisfied by an assignment τ if τ satisfies all clauses in $\texttt{cla}(F)$.

The problem #SAT, given a formula F, asks how many distinct assignments of $\texttt{var}(F)$ satisfy F. The optimization problem weighted MaxSAT, given a formula F and weight function $w : \texttt{cla}(F) \to \mathbb{N}$, asks what assignment τ of $\texttt{var}(F)$ maximizes $\sum_C w(C)$ for all $C \in \texttt{cla}(F)$ satisfied by τ. The problem MaxSAT is weighted MaxSAT where all clauses have weight one. When given a CNF formula F, we use s to denote the total size of F. More precisely, the total size of F is $s = |\texttt{cla}(F)| + \sum_{C \in \texttt{cla}(F)} |\texttt{lit}(C)|$. For weighted MaxSAT, we

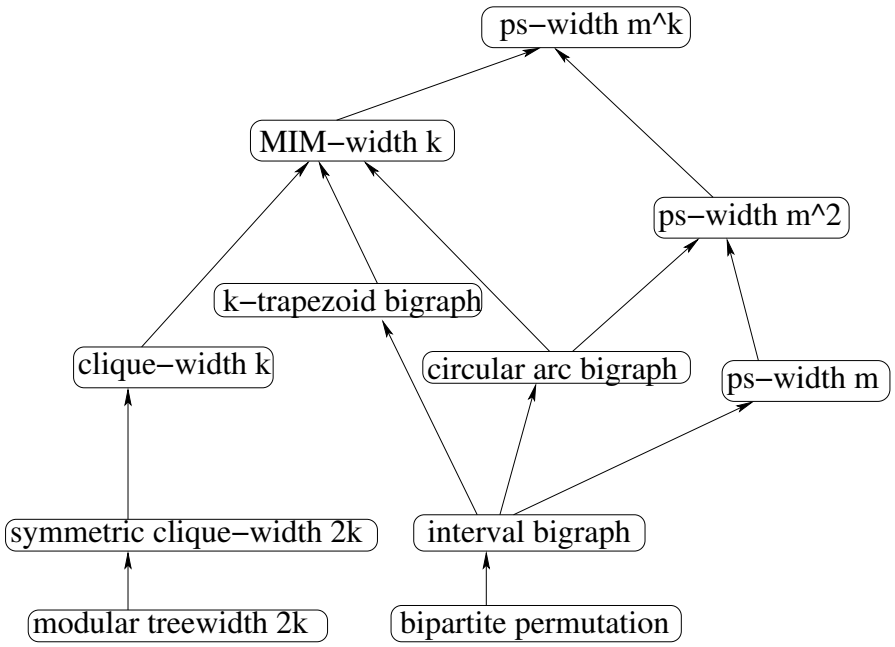

Fig. 1. A hierarchy of structural parameters and classes of bipartite graphs, where k is a constant and F a CNF formula having m clauses. An arc from P to Q means 'any formula (or incidence graph of a formula) that has a decomposition of type P, also has a decomposition of type Q'. The lack of an arc means that no such relation holds, i.e. this is a Hasse diagram.

assume the sum of all the weights are at most $2^{O(\texttt{cla}(F))}$, and thus we can do summation on the weights in time linear in $\texttt{cla}(F)$.

For a set A, with elements from a universe U we denote by $\overline{A}$ the elements in $U \setminus A$, as the universe is usually given by the context.

2.1 Cut of a formula

In this paper, we will solve MaxSAT and #SAT by the use of dynamic programming. We will be using a divide and conquer technique where we solve the problem on smaller subformulas of the original formula F and then combine the solutions to each of these smaller formulas to form a solution to the entire formula F. Note however, that the solutions found for a subformula will depend on the interaction between the subformula and the remainder of the formula. We use the following notation for subformulas.

For a clause C and set X of variables, by $C|_X$ we denote the clause $\{\ell \in C : \texttt{var}(\ell) \in X\}$. We say $C|_X$ is the clause C *induced* by X. Unless otherwise specified, all clauses mentioned in this paper is from the set $\texttt{cla}(F)$ (e.g., if we write $C|_x \in \texttt{cla}(F')$, we still assume $C \in \texttt{cla}(F)$). For a formula F and subsets

$\mathcal{C} \subseteq \texttt{cla}(F)$ and $X \subseteq \texttt{var}(F)$, we say the subformula $F_{\mathcal{C},X}$ of F *induced* by $\mathcal{C}$ and X is the formula consisting of the clauses $\{C_i|_X : C_i \in \mathcal{C}\}$. That is, $F_{\mathcal{C},X}$ is the formula we get by removing all clauses not in $\mathcal{C}$ followed by removing each literal that consists of a variable not in X. For a set $\mathcal{C}$ of clauses, we denote by $\mathcal{C}|_X$ the set $\{C|_X : C \in \mathcal{C}\}$. As with a clause, for an assignment τ over a set X of variables, we say the assignment τ *induced* by $X' \subseteq X$ is the assignment $\tau|_{X'}$ where the domain is restricted to X'.

For a formula F and sets $\mathcal{C} \subseteq \texttt{cla}(F)$, $X \subseteq \texttt{var}(F)$, and $S = \mathcal{C} \cup X$, we call S a *cut* of F and note that it breaks F into four subformulas $F_{\mathcal{C},X}$, $F_{\overline{\mathcal{C}},X}$, $F_{\mathcal{C},\overline{X}}$, and $F_{\overline{\mathcal{C}},\overline{X}}$. See Figure 2. One important fact we may observe from this definition is that a clause C in F is satisfied by an assignment τ of $\texttt{var}(F)$, if and only if C (induced by X or $\overline{X}$) is satisfied by τ in at least one of the formulas of any cut of F.

2.2 Projection satisfiable sets and ps-value of a formula

For a formula F and assignment τ of all the variables in $\texttt{var}(F)$, we denote by $\texttt{sat}(F, \tau)$ the set $\mathcal{C} \subseteq \texttt{cla}(F)$ so that each clause in $\mathcal{C}$ is satisfied by τ, and each clause not in $\mathcal{C}$ is unsatisfied by τ. If for a set $\mathcal{C} \subseteq \texttt{cla}(F)$ we have $\texttt{sat}(F, \tau) = \mathcal{C}$ for some τ over $\texttt{var}(F)$, then $\mathcal{C}$ is known as a *projection* (see e.g. [16,24]) and we say $\mathcal{C}$ is *projection satisfiable* in F. We denote by $\texttt{PS}(F)$ the family of all projection satisfiable sets in F. That is,

$$\texttt{PS}(F) = \{\texttt{sat}(F, \tau) : \tau \text{ is an assignment of } \texttt{var}(F)\}.$$

The cardinality of this set, $\texttt{PS}(F)$, is referred to as the ps-value of F.

2.3 The ps-width of a formula

We define a *branch decomposition* of a formula F to be a pair (T, δ) where T is a rooted binary tree and δ is a bijective function from the leaves of T to the clauses and variables of F. If all the non-leaf nodes (also referred to as *internal* nodes) of T induce a path, we say that (T, δ) is a *linear* branch decomposition. For a non-leaf node v of T, we denote by $\delta(v)$ the set $\{\delta(l) : l \text{ is a leaf in the subtree rooted in } v\}$. Based on this, we say that the decomposition (T, δ) of formula F induces certain cuts of F, namely the cuts defined by $\delta(v)$ for each node v in T.

For a formula F and branch decomposition (T, δ), for each node v in T, by F_v we denote the formula induced by the clauses in $\texttt{cla}(F) \setminus \delta(v)$ and the variables in $\delta(v)$, and by $F_{\overline{v}}$ we denote the formula on the complement sets; i.e. the clauses in $\delta(v)$ and the variables in $\texttt{var}(F) \setminus \delta(v)$. In other words, if $\delta(v) = \mathcal{C} \cup X$ with $\mathcal{C} \subseteq \texttt{cla}(F)$ and $X \subseteq \texttt{var}(F)$ then $F_v = F_{\overline{\mathcal{C}},X}$ and $F_{\overline{v}} = F_{\mathcal{C},\overline{X}}$. To simply the notation, we will for a node v in a branch decomposition and a set $\mathcal{C}$ of clauses denote by $\mathcal{C}|_v$ the set $\mathcal{C}|_{\texttt{var}(F_v)}$. We define the ps-*value* of the cut $\delta(v)$ to be

$$\texttt{ps}(\delta(v)) = \max\{|PS(F_v)|, |PS(F_{\overline{v}})|\}$$

We define the ps-*width* of a branch decomposition to be

$$\mathtt{psw}(T,\delta) = \max\{\mathtt{ps}(\delta(v)) : v \text{ is a node of } T\}$$

We define the ps-*width* of a formula F to be

$$\mathtt{psw}(F) = \min\{\mathtt{psw}(T,\delta) : (T,\delta) \text{ is a branch decompositions of } F\}$$

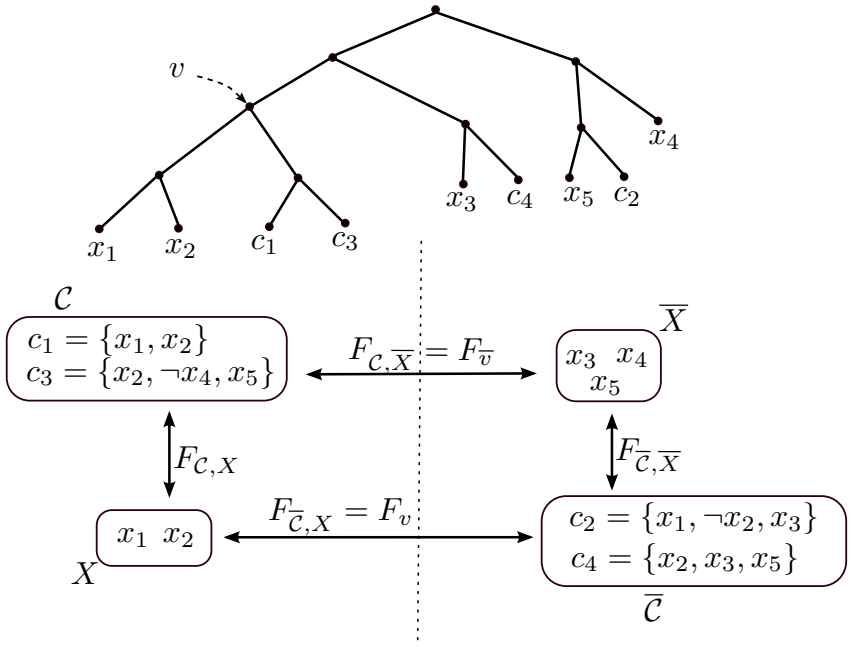

Fig. 2. On top is a branch decomposition of a formula F with $\mathtt{var}(F) = \{x_1, x_2, x_3, x_4, x_5\}$ and the 4 clauses $\mathtt{cla}(F) = \{c_1, c_2, c_3, c_4\}$ as given in the boxes. The node v of the tree defines the cut $\delta(v) = \mathcal{C} \uplus X$ where $\mathcal{C} = \{c_1, c_3\}$ and $X = \{x_1, x_2\}$. On the bottom is an illustration of the 4 subformulas defined by this cut. For example, $F_{\overline{\mathcal{C}},X} = \{\{x_1, \neg x_2\}, \{x_2\}\}$ and $F_{\mathcal{C},\overline{X}} = \{\emptyset, \{\neg x_4, x_5\}\}$. We have $F_v = F_{\overline{\mathcal{C}},X}$ and $F_{\overline{v}} = F_{\mathcal{C},\overline{X}}$ with projection satisfiable sets of clauses $\mathtt{PS}(F_v) = \{\{c_2|_v\}, \{c_4|_v\}, \{c_2|_v, c_4|_v\}\}$ and $\mathtt{PS}(F_{\overline{v}}) = \{\emptyset, \{c_3|_{\overline{v}}\}\}$ and the ps-value of this cut is $\mathtt{ps}(\delta(v)) = \max\{|PS(F_v)|, |PS(F_{\overline{v}})|\} = 3$.

Note that the ps-value of a cut is a symmetric function. That is, the ps-value of cut S equals the ps-value of the cut $\overline{S}$. See Figure 2 for an example.

3 Dynamic programming for MaxSAT and #SAT

Given a branch decomposition (T,δ) of a CNF formula F over n variables and m clauses and of total size s, we will give algorithms that solve MaxSAT and

#SAT on F in time $\mathcal{O}(\mathtt{psw}(T,\delta)^3 s(m+n))$. Our algorithms are strongly inspired by the algorithm of [24], but in order to achieve a runtime polynomial in ps-width, and also to solve MAXSAT, we make some changes.

In a pre-processing step we will need the following which, for each node v in T computes the sets $\mathtt{PS}(F_v)$ and $\mathtt{PS}(F_{\overline{v}})$.

Theorem 1. *Given a CNF formula F of n variables and m clauses with a branch decomposition (T,δ) of $\mathtt{ps}$-width k, we can in time $\mathcal{O}(k^2\log(k)m(m+n))$ compute the sets $\mathtt{PS}(F_v)$ and $\mathtt{PS}(F_{\overline{v}})$ for each v in T.*

Proof. We notice that for a node v in T with children c_1 and c_2, we can express $\mathtt{PS}(F_v)$ as

$$\mathtt{PS}(F_v) = \left\{ (C_1 \cup C_2)|_v \cap \mathtt{cla}(F_v) : \begin{array}{l} C_1|_{c_1} \in \mathtt{PS}(F_{c_1}), \text{ and} \\ C_2|_{c_2} \in \mathtt{PS}(F_{c_2}) \end{array} \right\} .$$

Similarly, for sibling s and parent p of v in T, the set $\mathtt{PS}(F_{\overline{v}})$ can be expressed as

$$\mathtt{PS}(F_{\overline{v}}) = \left\{ (C_p \cup C_s)|_{\overline{v}} \cap \mathtt{cla}(F_{\overline{v}}) : \begin{array}{l} C_p|_{\overline{p}} \in \mathtt{PS}(F_{\overline{p}}), \text{ and} \\ C_s|_s \in \mathtt{PS}(F_s) \end{array} \right\} .$$

By transforming these recursive expressions into a dynamic programming algorithm, as done in Procedure 1 and Procedure 2 below, we are able to calculate all the desired sets as long as we can compute the sets for the base cases $\mathtt{PS}(F_l)$ when l is a leaf of T, and $\mathtt{PS}(F_{\overline{r}})$ for the root r of T. However, these formulas contain at most one variable, and thus we can easily construct their set of projection satisfiable clauses in linear amount of time for each of the formulas. For the rest of the formulas, we construct the formulas using Procedure 1 and Procedure 2. As there are at most twice as many nodes in T as there are clauses and variables in F, the procedures will run at most $\mathcal{O}(|\mathtt{cla}(F)|+|\mathtt{var}(F)|)$ times. In each run of the algorithms, we iterate through at most k^2 pairs of projection satisfiable sets, and do a constant number of set operations that might take $\mathcal{O}(|\mathtt{cla}(F)|)$ time each. Then we sort the list of at most k^2 sets of clauses. When we sort, we can expect the runtime of comparing two elements to spend time linear in $|\mathtt{cla}(F)|$, so the total runtime for sorting L and deleting duplicates takes at most $\mathcal{O}(k^2\log(k)|\mathtt{cla}(F)|)$ time. This results in a total runtime of $\mathcal{O}(k^2\log(k)|\mathtt{cla}(F)|(|\mathtt{cla}(F)|+|\mathtt{var}(F)|))$ for all the nodes of T combined. □

We first give the algorithm for MaxSAT and then briefly describe the changes necessary for solving weighted MaxSAT and #SAT.

Our algorithm relies on the following binary relation, $\leq$, on assignments τ and τ' related to a cut $S = \mathcal{C} \cup X$ with $\mathcal{C} \subseteq \mathtt{cla}(F)$, $X \subseteq \mathtt{var}(F)$. For $\mathcal{C}'|_{\overline{X}} \in \mathtt{PS}(F_{\mathcal{C},\overline{X}})$, we define $\tau' \leq_S^{\mathcal{C}'} \tau$ if it holds that $|\mathtt{sat}(F,\tau') \setminus \mathcal{C}'| \leq |\mathtt{sat}(F,\tau) \setminus \mathcal{C}'|$. Note that for each cut $S = \mathcal{C} \cup X$ and each $\mathcal{C}'|_{\overline{X}} \in \mathtt{PS}(F_{\mathcal{C},\overline{X}})$ this gives a total preorder (transitive, reflexive and total) on assignments. The largest elements of this total preorder will be important for our algorithm, as they satisfy the maximum number of clauses under the given restrictions.

Procedure 1: Generating $\texttt{PS}(F_v)$
input: $\texttt{PS}(F_{c_1})$ and $\texttt{PS}(F_{c_2})$ for children c_1 and c_2 of v in branch decomposition **output:** $\texttt{PS}(F_v)$
$L \leftarrow$ empty list of projection satisfiable clause-sets **for** each $(C_1\|_{c_1}, C_2\|_{c_2}) \in \texttt{PS}(F_{c_1}) \times \texttt{PS}(F_{c_2})$ **do** add $(C_1 \cup C_2)\|_v \cap \texttt{cla}(F_v)$ to L sort L lexicographically by what clauses each element contains remove duplicates in L by looking only at consecutive elements **return** L

Procedure 2: Generating $\texttt{PS}(F_{\overline{v}})$
input: $\texttt{PS}(F_s)$ and $\texttt{PS}(F_{\overline{p}})$ for sibling s and parent p of v in branch decomposition **output:** $\texttt{PS}(F_{\overline{v}})$
$L \leftarrow$ empty list of projection satisfiable clause-sets **for** each $(C_s\|_s, C_p\|_{\overline{p}}) \in \texttt{PS}(F_s) \times \texttt{PS}(F_{\overline{p}})$ **do** add $(C_s \cup C_p)\|_{\overline{v}} \cap \texttt{cla}(F_{\overline{v}})$ to L sort L lexicographically by what clauses each element contains remove duplicates in L by looking only at consecutive elements **return** L

Given (T, δ) of a formula F our dynamic programming algorithm for MaxSAT will generate, for each node v in T, a table $\texttt{Tab}_\texttt{v}$ indexed by pairs of $\{(C_1, C_2) : C_1|_v \in \texttt{PS}(F_v), C_2|_{\overline{v}} \in \texttt{PS}(F_{\overline{v}})\}$. For projection satisfiable sets $C_v|_v \in \texttt{PS}(F_v)$ and $C_{\overline{v}}|_{\overline{v}} \in \texttt{PS}(F_{\overline{v}})$ the contents of the table at this index $\texttt{Tab}_\texttt{v}(C_v, C_{\overline{v}})$ should be an assignment $\tau : \texttt{var}(\delta(v)) \to \{0, 1\}$ satisfying the following constraint:

$$\begin{aligned}\texttt{Tab}_\texttt{v}(C_v, C_{\overline{v}}) = \tau \text{ such that } \texttt{sat}(F_v, \tau) = C_v|_v \text{ and } \tau' \leq^{C_{\overline{v}}}_{\delta(v)} \tau \text{ for any} \\ \tau' : \texttt{var}(\delta(v)) \to \{0, 1\} \text{ having } \texttt{sat}(F_v, \tau') = C_v|_v\end{aligned} \tag{1}$$

Let us give some intuition for this constraint. Our algorithm uses the technique of 'expectation from the outside' introduced in [5,6]. The partial assignment τ to variables in $\texttt{var}(\delta(v))$ stored at $\texttt{Tab}_\texttt{v}(C_v, C_{\overline{v}})$ will be combined with partial assignments to variables in $\texttt{var}(F) \setminus \texttt{var}(\delta(v))$ satisfying $C_{\overline{v}}$. These latter partial assignments constitute 'the expectation from the outside'. Constraint (1) implies that τ, being a largest element of the total preorder, will be a best combination with this expectation from the outside since it satisfies the maximum number of remaining clauses.

By bottom-up dynamic programming along the tree T we compute the tables of each node of T. For a leaf l in T, generating $\texttt{Tab}_l$ can be done easily in linear time since the formula F_v contains at most one variable. For an internal node v of T, with children c_1, c_2, we compute $\texttt{Tab}_\texttt{v}$ by the algorithm described in Procedure 3. There are 3 tables involved in this update, one at each child and one at the parent. A pair of entries, one from each child table, may lead to an

update of an entry in the parent table. Each table entry is indexed by a pair, thus there are 6 indices involved in a single potential update. A trick first introduced in [6] allows us to loop over triples of indices and for each triple compute the remaining 3 indices forming the 6-tuple involved in the update, thereby reducing the runtime.

Procedure 3: Computing $\mathtt{Tab}_v$ for inner node v with children c_1, c_2

input: $\mathtt{Tab}_{c_1}$, $\mathtt{Tab}_{c_2}$
output: $\mathtt{Tab}_v$

1. initialize $\mathtt{Tab}_v : \mathtt{PS}(F_v) \times \mathtt{PS}(F_{\overline{v}}) \to \{\mathtt{unassigned}\}$ // *dummy entries*
2. **for** each $(C_{c_1}|_{c_1}, C_{c_2}|_{c_2}, C_{\overline{v}}|_{\overline{v}})$ in $\mathtt{PS}(F_{c_1}) \times \mathtt{PS}(F_{c_2}) \times \mathtt{PS}(F_{\overline{v}})$ **do**
3. $\quad C_{\overline{c_1}} \leftarrow (C_{c_2} \cup C_{\overline{v}}) \cap \delta(c_1)$
4. $\quad C_{\overline{c_2}} \leftarrow (C_{c_1} \cup C_{\overline{v}}) \cap \delta(c_2)$
5. $\quad C_v \leftarrow (C_{c_1} \cup C_{c_2}) \setminus \delta(v)$
6. $\quad \tau \leftarrow \mathtt{Tab}_{c_1}(C_{c_1}, C_{\overline{c_1}}) \uplus \mathtt{Tab}_{c_2}(C_{c_2}, C_{\overline{c_2}})$
7. $\quad \tau' \leftarrow \mathtt{Tab}_v(C_v, C_{\overline{v}})$
8. $\quad$ **if** $\tau' = \mathtt{unassigned}$ or $\tau' \leq^{C_{\overline{v}}}_{\delta(v)} \tau$ **then** $\mathtt{Tab}_v(C_v, C_{\overline{v}}) \leftarrow \tau$
9. **return** $\mathtt{Tab}_v$

Lemma 2. *For a CNF formula F of total size s and an inner node v, of a branch decomposition (T, δ) of* ps*-width k, Procedure 3 computes $\mathtt{Tab}_v$ satisfying Constraint (1) in time $\mathcal{O}(k^3 s)$.*

Proof. We assume $\mathtt{Tab}_{c_1}$ and $\mathtt{Tab}_{c_2}$ satisfy Constraint (1). Procedure 3 loops over all triples in $\mathtt{PS}(F_{c_1}) \times \mathtt{PS}(F_{c_2}) \times \mathtt{PS}(F_{\overline{v}})$. From the definition of ps-width of (T, δ) there are at most k^3 such triples. Each operation inside an iteration of the loop take $\mathcal{O}(s)$ time and there is a constant number of such operations. Thus the runtime is $\mathcal{O}(k^3 s)$.

To show that the output $\mathtt{Tab}_v$ of Procedure 3 satisfies Constraint (1), we will prove that for any $C|_v \in \mathtt{PS}(F_v)$ and $C'|_{\overline{v}} \in \mathtt{PS}(F_{\overline{v}})$ the value of $\mathtt{Tab}_v(C, C')$ satisfies Constraint (1). That is, we will assure that the content of $\mathtt{Tab}_v(C, C')$ is an assignment τ so that $\mathtt{sat}(F_v, \tau) = C|_v$ and for all other assignments τ' over $\mathtt{var}(\delta(v))$ so that $\mathtt{sat}(F_v, \tau') = C|_v$, we have $\tau' \leq^{C'}_{\delta(v)} \tau$.

Let us assume for contradiction, that $\mathtt{Tab}_v(C, C')$ contains an assignment τ but there exists an assignment τ' over $\mathtt{var}(\delta(v))$ so that $\mathtt{sat}(F_v, \tau') = C|_v$, and we do not have $\tau' \leq^{C'}_{\delta(v)} \tau$. As τ is put into $\mathtt{Tab}_v(C, C')$ only if it is an assignment over $\mathtt{var}(\delta(v))$ and $\mathtt{sat}(F_v, \tau) = C|_v$. So, what we need to show to prove that $\mathtt{Tab}_v$ is correct is that in fact $\tau' \leq^{C'}_{\delta(v)} \tau$:

First, we notice that τ' consist of assignments $\tau'_1 = \tau'|_{\mathtt{var}(\delta(c_1))}$ and $\tau'_2 = \tau'|_{\mathtt{var}(\delta(c_2))}$ where τ'_1 is over the variables in $\mathtt{var}(\delta(c_1))$ and τ'_2 is over $\mathtt{var}(\delta(c_2))$. Let $C_1|_{c_1} = \mathtt{sat}(F_{c_1}, \tau'_1)$ and $C_2|_{c_2} = \mathtt{sat}(F_{c_2}, \tau'_2)$ and let $C'_1 = (C_2 \cup C') \cap \delta(c_1)$ and $C'_2 = (C_1 \cup C') \cap \delta(c_2)$. By how $\mathtt{Tab}_{c_1}$ and $\mathtt{Tab}_{c_2}$ is defined, we know for the assignment τ_1 in $\mathtt{Tab}_{c_1}(C_1, C'_1)$ and τ_2 in $\mathtt{Tab}_{c_2}(C_2, C'_2)$, we have $\tau'_1 \leq^{C'_1}_{\delta(c_1)} \tau_1$ and $\tau'_2 \leq^{C'_2}_{\delta(c_2)} \tau_2$. From our definition of the total preorder $\leq$ for assignments, we can

deduce that $\tau_1' \uplus \tau_2' \leq^{C'}_{\delta(v)} \tau_1 \uplus \tau_2$;

$$\begin{aligned}
&|\mathtt{sat}(F, \tau_1' \uplus \tau_2') \setminus C'| \\
=&|\mathtt{sat}(F, \tau_1') \setminus C_1'| - |C_1 \cap C'| + |\mathtt{sat}(F, \tau_2') \setminus C_2'| - |C_2 \cap C'| - |C_1 \cap C_2| \\
\leq&|\mathtt{sat}(F, \tau_1) \setminus C_1'| - |C_1 \cap C'| + |\mathtt{sat}(F, \tau_2) \setminus C_2'| - |C_2 \cap C'| - |C_1 \cap C_2| \\
=&|\mathtt{sat}(F, \tau_1 \uplus \tau_2) \setminus C'|\,.
\end{aligned}$$

However, since $\tau_1 \uplus \tau_2$ at the iteration of the triple $(C_1|_{c_1}, C_2|_{c_2}, C'|_{\overline{v}})$ in fact is considered by the algorithm to be set as $\mathtt{Tab}_v(C, C')$, it must be the case that $\tau_1 \uplus \tau_2 \leq^{C'}_{\delta(v)} \tau$. As $\leq^{C'}_{\delta(v)}$ clearly is a transitive relation, we conclude that $\tau' \leq^{C'}_{\delta(v)} \tau$. □

Theorem 3. *Given a formula F over n variables and m clauses and of total size s, and a branch decomposition (T, δ) of F of* ps*-width k, we solve* MaxSAT*,* #SAT*, and weighted* MaxSAT *in time $\mathcal{O}(k^3 s(m+n))$.*

Proof. To solve MaxSAT, we first compute $\mathtt{Tab}_r$ for the root node r of T. This requires that we first compute $\mathtt{PS}(F_v)$ and $\mathtt{PS}(F_{\overline{v}})$ for all nodes v of T, and then, in a bottom up manner, compute $\mathtt{Tab}_v$ for each of the $\mathcal{O}(m+n)$ nodes in T. The former part we can do in $\mathcal{O}(k^3 s(m+n))$ time by Theorem 1, and the latter part we do in the same amount of time by Lemma 2.

At the root r of T we have $\delta(r) = \mathtt{var}(F) \cup \mathtt{cla}(F)$. Thus $F_r = \emptyset$ and $F_{\overline{r}}$ contains only empty clauses, so that $PS(F_r) \times PS(F_{\overline{r}})$ contains only $(\emptyset, \emptyset)$. By Constraint (1) and the definition of the $\leq$ total preorder on assignments, the assignment τ stored in $\mathtt{Tab}_r(\emptyset, \emptyset)$ is an assignment of $\mathtt{var}(F)$ maximizing $|\mathtt{sat}(F, \tau)|$, the number of clauses satisfied, and hence is a solution to MaxSAT.

For a weight function $w : \mathtt{cla}(F) \to \mathbb{N}$, by redefining $\tau_1 \leq^B_A \tau_2$ to mean $w(\mathtt{sat}(F, \tau_1) \setminus B) \leq w(\mathtt{sat}(F, \tau_2) \setminus B)$ both for the definition of $\mathtt{Tab}$ and for Procedure 3, we are able to solve the more general problem weighted MaxSAT in the same way.

For the problem #SAT, we care only about assignments satisfying all the clauses of F, and we want to decide the number of distinct assignments doing so. This requires a few alterations. Firstly, alter the definition of the contents of $\mathtt{Tab}_v(C, C')$ in Constraint (1) to be the number of assignments τ over $\mathtt{var}(\delta(v))$ where $\mathtt{sat}(F_v, \tau) = C|_v$ and $\mathtt{cla}(\delta(v)) \setminus C' \subseteq \mathtt{sat}(F, \tau)$. Secondly, when computing $\mathtt{Tab}_l$ for the leaves l of T, we set each of the entries of $\mathtt{Tab}_l$ to either zero, one, or two, according to the definition. Thirdly, we alter the algorithm to compute $\mathtt{Tab}_v$ (Procedure 3) for inner nodes. We initialize $\mathtt{Tab}_{\mathtt{v}}(C, C')$ to be zero at the start of the algorithm, and substitute lines 6, 7 and 8 of Procedure 3 by the following line which increases the table value by the product of the table values at the children

$$\mathtt{Tab}_{\mathtt{v}}(C_v, C_{\overline{v}}) \leftarrow \mathtt{Tab}_{\mathtt{v}}(C_v, C_{\overline{v}}) + \mathtt{Tab}_{\mathtt{c_1}}(C_{c_1}, C_{\overline{c_1}}) \cdot \mathtt{Tab}_{\mathtt{c_2}}(C_{c_2}, C_{\overline{c_2}})$$

This will satisfy our new constraint of $\mathtt{Tab}_v$ for internal nodes v of T. The value of $\mathtt{Tab}_r(\emptyset, \emptyset)$ at the root r of T will be exactly the number of distinct assignments satisfying all clauses of F. □

The bottleneck giving the cubic factor k^3 in the runtime of Theorem 3 is the number triples in $\mathtt{PS}(F_{\overline{v}}) \times \mathtt{PS}(F_{c_1}) \times \mathtt{PS}(F_{c_2})$ for any node v with children c_1 and c_2. When (T, δ) is a linear branch decomposition, it is always the case that either c_1 or c_2 is a leaf of T. In this case either $|\mathtt{PS}(F_{c_1})|$ or $|\mathtt{PS}(F_{c_2})|$ is a constant. Therefore, for linear branch decompositions $\mathtt{PS}(F_{\overline{v}}) \times \mathtt{PS}(F_{c_1}) \times \mathtt{PS}(F_{c_2})$ will contain no more than $\mathcal{O}(k^2)$ triples. Thus we can reduce the runtime of the algorithm by a factor of k.

Theorem 4. *Given a formula F over n variables and m clauses and of total size s, and a linear branch decomposition (T, δ) of F of* ps*-width k, we solve* #SAT*,* MaxSAT*, and weighted* MaxSAT *in time $\mathcal{O}(k^2 s(m+n))$.*

4 CNF formulas of polynomial ps-width

In this section we investigate classes of CNF formulas having decompositions with ps-width polynomially bounded in the total size s of the formula. In particular, we show that this holds whenever the incidence graph of the formula has constant MIM-width (maximum induced matching-width). We also show that a large class of bipartite graphs, using what we call bigraph bipartizations, have constant MIM-width.

Let us start by defining bigraph bipartizations. For a graph G and subset of vertices $A \subseteq V(G)$ the bipartite graph $G[A, \overline{A}]$ is the subgraph of G containing all edges of G with exactly one endpoint in A. We call $G[A, \overline{A}]$ a bigraph bipartization of G, note that G has a bigraph bipartization for each subset of vertices. For a graph class X define the class of X bigraphs as the bipartite graphs H for which there exists $G \in X$ such that H is isomorphic to a bigraph bipartization of G. For example, H is an interval bigraph if there is some interval graph G and some $A \subseteq V(G)$ with H isomorphic to $G[A, \overline{A}]$.

To establish the connection to MIM-width we need to look at induced matchings in the incidence graph of a formula. The incidence graph of a formula F is the bipartite graph $I(F)$ having a vertex for each clause and variable, with variable x adjacent to any clause C in which it occurs. An induced matching in a graph is a subset M of edges with the property that any edge of the graph is incident to at most one edge in M. In other words, for any 3 vertices a, b, c, if ab is an edge in M and bc is an edge then there does not exist an edge cd in M. The number of edges in M is called the size of the induced matching. The following result provides an upper bound on the ps-value of a formula in terms of the maximum size of an induced matching of its incidence graph.

Lemma 5. *Let F be a CNF formula and let k be the maximum size of an induced matching in $I(F)$. We then have $|\mathtt{PS}(F)| \leq |\mathtt{cla}(F)|^k$.*

Proof. Let $\mathcal{C} \in \mathtt{PS}(F)$ and $\mathcal{C}_f = \mathtt{cla}(F) \setminus \mathcal{C}$. Thus, there exists a complete assignment τ such that the clauses not satisfied by τ are $\mathcal{C}_f = \mathtt{cla}(F) \setminus \mathtt{sat}(F, \tau)$. Since every variable in $\mathtt{var}(F)$ appears in some clause of F this means that $\tau|_{\mathtt{var}(\mathcal{C}_f)}$ is the unique assignment of the variables in $\mathtt{var}(\mathcal{C}_f)$ which do not

satisfy any clause of $\mathcal{C}_f$. Let $\mathcal{C}_f^{'} \subseteq \mathcal{C}_f$ be an inclusion minimal set such that $\mathtt{var}(\mathcal{C}_f) = \mathtt{var}(\mathcal{C}_f^{'})$, hence $\tau|_{\mathtt{var}(\mathcal{C}_f)}$ is also the unique assignment of the variables in $\mathtt{var}(\mathcal{C}_f)$ which do not satisfy any clause of $\mathcal{C}_f^{'}$. An upper bound on the number of different such minimal $\mathcal{C}_f^{'}$, over all $\mathcal{C} \in \mathtt{PS}(F)$, will give an upper bound on $|\mathtt{PS}(F)|$. For every $C \in \mathcal{C}_f^{'}$ there is a variable v_C appearing in C and no other clause of $\mathcal{C}_f^{'}$, otherwise $\mathcal{C}_f^{'}$ would not be minimal. Note that we have an induced matching M of $I(F)$ containing all such edges v_C, C. By assumption, the induced matching M can have at most k edges and hence $|\mathcal{C}_f^{'}| \leq k$. There are at most $|\mathtt{cla}(F)|^k$ sets of at most k clauses and the lemma follows. □

In order to lift this result on the ps-value of F, i.e $|\mathtt{PS}(F)|$, to the ps-width of F, we use MIM-width of the incidence graph $I(F)$, which is defined using branch decompositions of graphs. A branch decomposition of the formula F, as defined in Section 2, can also be seen as a branch decomposition of the incidence graph $I(F)$. Nevertheless, for completeness, we formally define branch decompositions of graphs and MIM-width.

A branch decomposition of a graph G is a pair (T, δ) where T is a rooted binary tree and δ a bijection between the leaf set of T and the vertex set of G. For a node w of T let the subset of $V(G)$ in bijection δ with the leaves of the subtree of T rooted at w be denoted by V_w. We say the decomposition defines the cut $(V_w, \overline{V_w})$. The MIM-value of a cut $(V_w, \overline{V_w})$ is the size of a maximum induced matching of $G[V_w, \overline{V_w}]$. The MIM-width of (T, δ) is the maximum MIM-value over all cuts $(V_w, \overline{V_w})$ defined by a node w of T. The MIM-width of graph G, denoted $mimw(G)$, is the minimum MIM-width over all branch decompositions (T, δ) of G. As before a *linear branch decomposition* is a branch decomposition where inner nodes of the underlying tree induces a path.

We now give an upper bound on the ps-value of a formula in terms of the MIM-width of any graph G such that the incidence graph of the formula is a bigraph bipartization of G.

Theorem 6. *Let F be a CNF formula of m clauses, G a graph, and (T, δ_G) a (linear) branch decomposition of G of MIM-width k. If for a subset $A \subseteq V(G)$ the graph $G[A, \overline{A}]$ is isomorphic to $I(F)$, then we can in linear time produce a (linear) branch decomposition (T, δ_F) of F having ps-width at most m^k.*

Proof. Since each variable and clause in F has a corresponding node in $I(F)$, and each node in $I(F)$ has a corresponding node in G, by defining δ_F to be the function mapping each leaf l of T to the variable or clause in F corresponding to the node $\delta_G(l)$, (T, δ_T) is going to be a branch decomposition of F. For any cut $(A, \overline{A})$ induced by a node of (T, δ_F), let $C \subseteq \mathtt{cla}(F)$ be the clauses corresponding to vertices in A and $X \subseteq \mathtt{var}(F)$ the variables corresponding to vertices in A. The cut $S = C \cup X$ of F defines the two formulas $F_{C,\overline{X}}$ and $F_{\overline{C},X}$, and it holds that $I(F_{C,\overline{X}})$ and $I(F_{\overline{C},X})$ are induced subgraphs of $G[A, \overline{A}]$ and hence by Lemma 5, we have $|\mathtt{PS}(F_{C,\overline{X}})| \leq |\mathtt{cla}(F)|^{\mathtt{mim}(A)}$, and likewise we have $|\mathtt{PS}(F_{\overline{C},X})| \leq |\mathtt{cla}(F)|^{\mathtt{mim}(A)}$. Since the ps-width of the decomposition is the maximum ps-value of each cut, the theorem follows. □

Note that by taking $G = I(F)$ and $A = \texttt{cla}(F)$ and letting (T, δ_G) be a branch decomposition of G of minimum MIM-width, we get the following weaker result.

Corollary 7. *For any CNF formula F over m clauses, the* ps*-width of F is no larger than $m^{\texttt{mimw}(I(F))}$.*

In his thesis, Vatshelle [26] shows that MIM-width of any graph G is at most the clique-width of G. Furthermore, the clique-width has been shown by Courcelle [7] to be at most twice the symmetric clique-width. Thus, we can conclude that MIM-width is bounded on any graph class with a bound on the symmetric clique-width, in accordance with Figure 1.

Many classes of graphs have intersection models, meaning that they can be represented as intersection graphs of certain objects, i.e. each vertex is associated with an object and two vertices are adjacent iff their objects intersect. The objects used to define intersection graphs usually consist of geometrical objects such as lines, circles or polygons. Many well known classes of intersection graphs have constant MIM-width, as in the following which lists only a subset of the classes proven to have such bounds in [2,26].

Theorem 8 ([2,26]). *Let G be a graph. If G is a:*
interval graph then $\texttt{mimw}(G) \leq 1$.
circular arc graph then $\texttt{mimw}(G) \leq 2$.
k-trapezoid graph then $\texttt{mimw}(G) \leq k$.
Moreover there exist linear decompositions satisfying the bound.

Let us briefly mention the definition of these graph classes. A graph is an interval graph if it has an intersection model consisting of intervals of the real line. A graph is a circular arc graph if it has an intersection model consisting of arcs of a circle. To build a k-trapezoid we start with k parallel line segments $(s_1, e_1), (s_2, e_2), ..., (s_k, e_k)$ and add two non-intersecting paths s and e by joining s_i to s_{i+1} and e_i to e_{i+1} respectively by straight lines for each $i \in \{1, ..., k-1\}$. The polygon defined by s and e and the two line segments $(s_1, e_1), (s_k, e_k)$ forms a k-trapezoid. A graph is a k-trapezoid graph if it has an intersection model consisting of k-trapezoids. See [3] for information about graph classes and their containment relations. Combining Theorems 6 and 8 we get the following.

Corollary 9. *Let F be a CNF formula containing m clauses. If $I(F)$ is a:*
interval bigraph then $\texttt{psw}(F) \leq m$.
circular arc bigraph then $\texttt{psw}(F) \leq m^2$.
k-trapezoid bigraph then $\texttt{psw}(F) \leq m^k$.
Moreover there exist linear decompositions satisfying the bound.

5 Interval bigraphs and formulas having interval orders

We will in this section show one class of formulas where we can find linear branch decompositions having ps-width $\mathcal{O}(|\texttt{cla}(F)|)$. Let us recall the definition

of interval ordering. A CNF formula F has an interval ordering if there exists a linear ordering of variables and clauses such that for any variable x occurring in clause C, if x appears before C then any variable between them also occurs in C, and if C appears before x then x occurs also in any clause between them. By a result of Hell and Huang [13] it follows that a formula F has an interval ordering if and only if $I(F)$ is a interval bigraph.

Theorem 10. *Given a CNF formula F over n variables and m clauses and of total size s, we can in time $\mathcal{O}((m+n)s)$ decide if F has an interval ordering (yes iff $I(F)$ is an interval bigraph), and if yes we solve* #SAT *and weighted* MaxSAT *with a runtime of $\mathcal{O}(m^2(m+n)s)$.*

Proof. Using the characterization of [13] and the algorithm of [19] we can in time $\mathcal{O}((m+n)s)$ decide if F has an interval ordering and if yes, then we find it. From this interval ordering we build an interval graph G such that $I(F)$ is a bigraph bipartization of G, and construct a linear branch decomposition of G having MIM-width 1 [2]. From such a linear branch decomposition we get from Theorem 6 that we can construct another linear branch decomposition of F having ps-width $\mathcal{O}(m)$. We then run the algorithm of Theorem 4. □

6 Conclusion

In this paper we have proposed a structural parameter of CNF formulas, called ps-width or projection-satisfiable-width. We showed that weighted MaxSAT and #SAT can be solved in polynomial time on formulas given with a decomposition of polynomially bounded ps-width. Using the concept of interval bigraphs we also showed a polynomial time algorithm that actually finds such a decomposition, for formulas having an interval ordering.

Could one devise such an algorithm also for the larger class of circular arc bigraphs, or maybe even for the even larger class of k-trapezoid bigraphs? In other words, is the problem of recognizing if a bipartite input graph is a circular arc bigraph, or a k-trapezoid bigraph, polynomial-time solvable?

It could be interesting to give an algorithm solving MaxSAT and/or #SAT directly on the interval ordering of a formula, rather than using the more general notion of ps-width as in this paper. Maybe such an algorithm could be of practical use?

Also of practical interest would be to design a heuristic algorithm which given a formula finds a decomposition of relatively low ps-width, as has been done for boolean-width in [14].

Finally, we hope the essential combinatorial result enabling the improvements in this paper, Lemma 5, may have other uses as well.

References

1. Bacchus, F., Dalmao, S., Pitassi, T.: Algorithms and complexity results for# sat and bayesian inference. In: Foundations of Computer Science, 2003. Proceedings. 44th Annual IEEE Symposium on. pp. 340–351. IEEE (2003)

2. Belmonte, R., Vatshelle, M.: Graph classes with structured neighborhoods and algorithmic applications. Theor. Comput. Sci. 511, 54–65 (2013)
3. Brandstädt, A., Le, V.B., Spinrad, J.P.: Graph Classes: A Survey, Monographs on Discrete Mathematics and Applications, vol. 3. SIAM Society for Industrial and Applied Mathematics, Philadelphia (1999)
4. Brandstädt, A., Lozin, V.V.: On the linear structure and clique-width of bipartite permutation graphs. Ars Comb. 67 (2003)
5. Bui-Xuan, B.M., Telle, J.A., Vatshelle, M.: H-join decomposable graphs and algorithms with runtime single exponential in rankwidth. Discrete Applied Mathematics 158(7), 809–819 (2010)
6. Bui-Xuan, B.M., Telle, J.A., Vatshelle, M.: Boolean-width of graphs. Theoretical Computer Science 412(39), 5187–5204 (2011)
7. Courcelle, B.: Clique-width of countable graphs: a compactness property. Discrete Mathematics 276(1-3), 127–148 (2004)
8. Darwiche, A.: Recursive conditioning. Artificial Intelligence 126(1), 5–41 (2001)
9. Fischer, E., Makowsky, J.A., Ravve, E.V.: Counting truth assignments of formulas of bounded tree-width or clique-width. Discrete Applied Mathematics 156(4), 511–529 (2008)
10. Ganian, R., Hlinený, P., Obdrzálek, J.: Better algorithms for satisfiability problems for formulas of bounded rank-width. Fundam. Inform. 123(1), 59–76 (2013)
11. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman (1979)
12. Geelen, J.F., Gerards, B., Whittle, G.: Branch-width and well-quasi-ordering in matroids and graphs. J. COMBIN. THEORY SER. B 84(2), 270–290 (2002)
13. Hell, P., Huang, J.: Interval bigraphs and circular arc graphs. Journal of Graph Theory 46(4), 313–327 (2004)
14. Hvidevold, E.M., Sharmin, S., Telle, J.A., Vatshelle, M.: Finding good decompositions for dynamic programming on dense graphs. In: Marx, D., Rossmanith, P. (eds.) IPEC. Lecture Notes in Computer Science, vol. 7112, pp. 219–231. Springer (2011)
15. Jaumard, B., Simeone, B.: On the complexity of the maximum satisfiability problem for horn formulas. Inf. Process. Lett. 26(1), 1–4 (1987)
16. Kaski, P., Koivisto, M., Nederlof, J.: Homomorphic hashing for sparse coefficient extraction. In: Proceedings of the 7th international conference on Parameterized and Exact Computation. pp. 147–158. Springer-Verlag (2012)
17. Müller, H.: Recognizing interval digraphs and interval bigraphs in polynomial time. Discrete Applied Mathematics 78(1-3), 189–205 (1997)
18. Paulusma, D., Slivovsky, F., Szeider, S.: Model counting for CNF formulas of bounded modular treewidth. In: Portier, N., Wilke, T. (eds.) STACS. LIPIcs, vol. 20, pp. 55–66. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2013)
19. Rafiey, A.: Recognizing interval bigraphs by forbidden patterns. CoRR abs/1211.2662 (2012)
20. Raman, V., Ravikumar, B., Rao, S.S.: A simplified NP-complete MAXSAT problem. Inf. Process. Lett. 65(1), 1–6 (1998)
21. Robertson, N., Seymour, P.D.: Graph minors X. obstructions to tree-decomposition. J. COMBIN. THEORY SER. B 52(2), 153–190 (1991)
22. Roth, D.: A connectionist framework for reasoning: Reasoning with examples. In: Clancey, W.J., Weld, D.S. (eds.) AAAI/IAAI, Vol. 2. pp. 1256–1261. AAAI Press / The MIT Press (1996)
23. Samer, M., Szeider, S.: Algorithms for propositional model counting. J. Discrete Algorithms 8(1), 50–64 (2010)

24. Slivovsky, F., Szeider, S.: Model counting for formulas of bounded clique-width. In: Cai, L., Cheng, S.W., Lam, T.W. (eds.) ISAAC. Lecture Notes in Computer Science, vol. 8283, pp. 677–687. Springer (2013)
25. Szeider, S.: On fixed-parameter tractable parameterizations of SAT. In: Giunchiglia, E., Tacchella, A. (eds.) SAT. Lecture Notes in Computer Science, vol. 2919, pp. 188–202. Springer (2003)
26. Vatshelle, M.: New width parameters of graphs. Ph.D. thesis, The University of Bergen (2012)