Algorithms for Propositional Model Counting

Marko Samer[*,a,1], Stefan Szeider[b]

[a] *Department of Computer Science, TU Darmstadt, Germany*
[b] *Department of Computer Science, Durham University, UK*

Abstract

We present algorithms for the propositional model counting problem #SAT. The algorithms utilize tree decompositions of certain graphs associated with the given CNF formula; in particular we consider primal, dual, and incidence graphs. We describe the algorithms coherently for a direct comparison and with sufficient detail for making an actual implementation reasonably easy. We discuss several aspects of the algorithms including worst-case time and space requirements.

Key words: Model counting, propositional satisfiability, parameterized algorithms

1. Introduction

Propositional model counting (#SAT) is the problem of determining the number of satisfying truth assignments (models) of a given propositional formula in conjunctive normal form (CNF). This problem arises in several areas of artificial intelligence, in particular in the context of probabilistic reasoning [3, 27]. However, since the problem is #P-complete [31], it is very unlikely that it can be solved in polynomial time. #SAT remains #P-hard even for monotone 2CNF formulas and Horn 2CNF formulas, and it is NP-hard to approximate the number of models of a formula with n variables within $2^{n^{1-\varepsilon}}$ for $\varepsilon > 0$. This approximation hardness holds also for monotone 2CNF formulas and Horn 2CNF formulas [27]. Thus, in contrast to the decision problem SAT, restricting the syntax of instances does not lead to tractability.

An alternative to restricting the syntax is to impose *structural restrictions* on the input formulas. Structural restrictions can be applied in terms of certain parameters (invariants) of graphs or hypergraphs associated with formulas. In this paper we will mainly consider the following graphs (more exact definitions are given in Section 2.3, examples are shown in Figure 1). The *primal graph* has as vertices the variables of the given formula, two variables are joined by an edge if they occur together in a clause. Symmetrically, the *dual graph* has as vertices the clauses of the formula, two clauses are joined by an edge if they share a variable. Finally, the *incidence graph* is a bipartite graph

[*] Corresponding author.
Email addresses: samer@cs.tu-darmstadt.de (Marko Samer), stefan.szeider@durham.ac.uk (Stefan Szeider)
[1] This research was carried out during the author's postdoc position at the University of Durham.

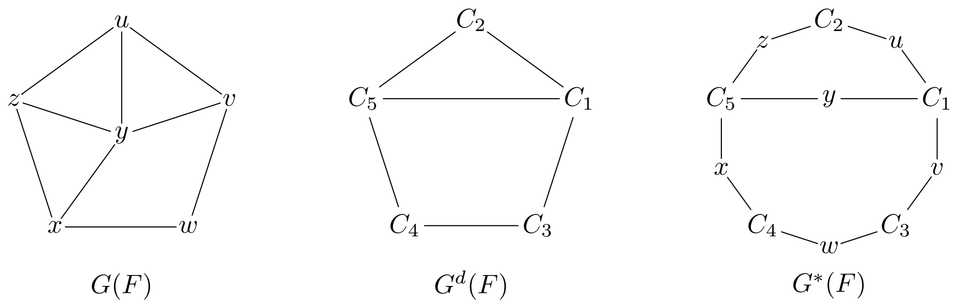


Figure 1: Graphs associated with the CNF formula $F = \{C_1, \ldots, C_5\}$ with $C_1 = \{u, \neg v, \neg y\}$, $C_2 = \{\neg u, z\}$, $C_3 = \{v, \neg w\}$, $C_4 = \{w, \neg x\}$, $C_5 = \{x, y, \neg z\}$; the primal graph $G(F)$, the dual graph $G^d(F)$, and the incidence graph $G^*(F)$.

where one vertex class consists of the clauses of the given formula, and the other consists of the variables; a clause and a variable are joined by an edge if the variable occurs in the clause. Primal and incidence graphs have been widely studied in the literature on satisfiability and constraint satisfaction, whereas dual graphs have received less attention.

We apply structural restrictions on CNF formulas by bounding the graph parameter treewidth of the associated graphs. Treewidth, introduced by Robertson and Seymour in their Graph Minors Project, indicates in a certain sense the "tree-likeness" of a graph. Many otherwise NP-hard graph problems such as Hamiltonicity and 3-colorability are solvable in polynomial time for graphs of bounded treewidth. It is generally believed that many practically relevant problems actually do have low treewidth [4]. Treewidth is based on certain decompositions of graphs, called tree decompositions, where sets of vertices ("bags") of a graph are arranged at the nodes of a tree such that certain conditions are satisfied (see Section 2.1). If a graph has treewidth k then it admits a tree decomposition of *width* k, i.e., a tree decomposition where all bags have size at most $k+1$. Depending on whether we consider the treewidth of the primal, dual, or incidence graph of the given CNF formula, we speak of the primal, dual, or incidence treewidth of the formula, respectively.

Owing to a general result on Monadic Second Order Logic of Courcelle, Makowsky, and Rotics [9], the model counting problem can be solved in polynomial time for formulas of bounded primal, dual, or incidence treewidth. However, the algorithms obtained via this general method are impractical. For getting practical results, one needs to design tailor-suited algorithms for the particular problem domain. As the algorithms under consideration are typically exponential in the treewidth, small improvements can have a strong impact on the practicability.

Contributions of this paper

We propose three efficient model counting algorithms that utilize small primal, dual, and incidence treewidth of instances. We present the three algorithms in a coherent fashion that allows a direct comparison of several aspects. We describe the algorithms at a level of detail that makes an implementation reasonably straightforward.

Our three algorithms follow the principle of dynamic programming: we start at leaf nodes of the tree decomposition and work our way up in the tree, computing at each node some information (stored in a table) on the subgraph thus far encountered. More details on the dynamic programming process and an analysis of space requirements is given in Section 3.4.

The following table summarizes worst-case runtimes of the algorithms. Here k_1, k_2, k_3 and N_1, N_2, N_3 denote the width and number of nodes of the given tree decomposition of the primal, dual, and incidence graph, respectively; d and l denote the maximum number of occurrences over all variables and the cardinality of a largest clause of the given CNF formula, respectively. For the bounds on the runtimes we assume arithmetic operations to have constant runtime; in Section 3 we provide a refined analysis based on bit complexity.

primal graph	dual graph	incidence graph
$\mathcal{O}(2^{k_1} k_1\, d\, N_1)$	$\mathcal{O}(2^{k_2} k_2\, l\, N_2)$	$\mathcal{O}(2^{k_3} k_3\, (l + 2^{k_3})\, N_3)$

Note that all three algorithms are *fixed-parameter algorithms* with respect to the corresponding treewidth parameter. A fixed-parameter algorithm solves instances of size n and parameter k in time $\mathcal{O}(f(k)n^c)$ where f denotes a computable function and c denotes a constant that is independent of the parameter k [12, 14, 24]. The main advantage of fixed-parameter algorithms is that the runtime increases moderately when n becomes large, in contrast to algorithms with runtime of, say, $\mathcal{O}(n^k)$.

The *incidence treewidth algorithm* is superior to the other two algorithms if the input formula has large clauses and contains variables that occur in many clauses. Such instances have large primal and large dual treewidth since a clause containing more than n literals causes the primal treewidth to be at least n, a variable occurring in more than n clauses causes the dual treewidth to be at least n (this follows from the fact that if a graph contains a complete subgraph on more than n vertices then the treewidth of the graph is at least n [20]).

However, our results indicate that the *primal treewidth algorithm* as well as the *dual treewidth algorithm* are exponentially faster then the incidence treewidth algorithm, imposing an exponential factor of 2^k instead of 4^k. Thus, although one can simulate the primal and dual treewidth algorithms by the incidence treewidth algorithm (a CNF formula of primal or dual treewidth k has incidence treewidth at most $k + 1$ [22]), such a simulation increases the runtime exponentially.

We also study space requirements of the three algorithms in terms of the maximum number of tables that need to be kept simultaneously in memory during the dynamic programming process. We analyze the table requirements and explain how optimal bottom-up traversals can be computed efficiently.

In summary, our analysis indicates that each of the three algorithms has its advantages and disadvantages. One needs to choose the right algorithm depending on context and area of the application under consideration.

Related work

Fischer, Makowsky, and Ravve [13] propose a fixed-parameter algorithm for #SAT with respect to the incidence treewidth. Their algorithm is based on a recursive splitting of the given formula according to a tree decomposition of the incidence graph, making use of the inclusion-exclusion principle. The time complexity stated in [13] is similar to the one we obtain for our incidence treewidth algorithm.

Branchwidth is a graph parameter that is related to treewidth by a constant factor [26]. Bacchus, Dalmao, and Pitassi [3] propose an algorithm that solves #SAT in time $n^{\mathcal{O}(1)}2^{\mathcal{O}(k)}$ for formulas with n variables whose formula hypergraphs have branchwidth k. The algorithm is based on the DPLL procedure and uses caching techniques for an efficient reuse of solutions for sub-problems; the branch decomposition provides an ordering of the variables as processed by the DPLL procedure. A fixed-parameter algorithm for the decision problem SAT with respect to primal treewidth has previously been proposed by Gottlob, Scarcello, and Sideri [17].

A different approach for solving #SAT is due to Nishimura, Ragde, and Szeider [25]. They present a fixed-parameter algorithm for computing strong backdoor sets with respect to cluster formulas, which yields a fixed-parameter algorithm for #SAT. In terms of generality, the corresponding parameter *clustering-width* is incomparable with incidence treewidth.

The *clique-width of directed incidence graphs* of CNF formulas provides a parameterization that is strictly more general than the treewidth parameters considered above. The directed (or signed) incidence graph is obtained from the incidence graph by indicating positive or negative occurrences of variables by the orientation of the corresponding edge. Fixed-parameter tractability of #SAT follows via the meta-theorem of Courcelle, Makowsky, and Rotics [8] on counting problems expressible in a certain fragment of Monadic Second Order Logic (MSO_1), yielding an algorithm that is double-exponential in the width of the clique-width decomposition. A single-exponential algorithm is due to Fisher, Makowsky, and Ravve [13]. However, both algorithms rely on clique-width approximation algorithms. The known polynomial-time algorithms for that purpose admit an exponential approximation error [19] and are of limited practical value.

The various treewidth parameters can be defined analogously for instances of the constraint satisfaction problem (CSP), considering constraints (i.e., relations) instead of clauses when forming the graphs. From the work of Gottlob et al. [17] it follows that the Boolean CSP is fixed-parameter tractable with respect to the parameter primal treewidth. In contrast to SAT and #SAT, this result cannot be generalized to the more general parameter incidence treewidth (subject to a complexity theoretic assumption): Samer and Szeider [29] show that the Boolean CSP (also known as *generalized satisfiability*) parameterized by the incidence treewidth is W[1]-hard. W[1] is a complexity class in parameterized complexity theory; there is strong theoretical evidence that W[1]-hard problems are not fixed-parameter tractable [12]. Of related interest is a dichotomy theorem for generalized satisfiability counting problems due to Creignou and Hermann [10].

In the context of constraint satisfaction several hypergraph parameters have been considered, such as hypertree-width [16], spread-cut width [7], and fractional hypertree-width [18]. For instances of unbounded arity (i.e., the associated hypergraphs have hyperedges of arbitrary size) these parameters are strictly more general than incidence treewidth. In the following we provide arguments that indicate that these hypergraph parameters have no apparent significance for the problems SAT and #SAT.

A hypergraph is *acyclic* if there is a tree decomposition (of its primal graph) whose number of nodes equals the number of hyperedges and for each hyperedge there is a tree-node that contains exactly the vertices of the hyperedge in its bag (cf. Gottlob et al. [16]). Note that if a hyperedge contains all the vertices of a hypergraph, then the hypergraph is acyclic and all the above mentioned hypergraph parameters equal 1.

Proposition 1. SAT *and* #SAT *remain* NP*-hard and* #P*-hard, respectively, for CNF formulas with acyclic primal hypergraphs.*

Proof. Let F be an arbitrary CNF formula and let x be a new variable not occurring in F. Consider the CNF formula F' obtained from F by adding the clause $C = var(F) \cup \{x\}$. The primal hypergraph of F', obtained by dropping negations and considering clauses as hyperedges, is acyclic. Since x is a pure literal, F and F' are equivalent with respect to satisfiability. Now let τ_0 be the assignment that sets all variables of F to 0. If τ_0 satisfies F, then F' has exactly twice as many models as F minus one (i.e., $\#(F') = 2\,\#(F) - 1$); otherwise, if τ_0 does not satisfy F, then F' has exactly twice as many models as F (i.e., $\#(F') = 2\,\#(F)$). $\square$

A similar construction can be applied with respect to the *dual hypergraph* whose vertices are the clauses and which contains for every variable y a hyperedge consisting of all the clauses that contain y or $\neg y$.

Proposition 2. SAT *and* #SAT *remain* NP*-hard and* #P*-hard, respectively, for CNF formulas with acyclic dual hypergraphs.*

Proof. Let F be an arbitrary CNF formula. Take a new variable x and obtain from F the formula F' by replacing every clause C with $C' = C \cup \{x\}$ and by adding the unit clause $\{\neg x\}$. The dual hypergraph of F' is acyclic. Clearly F and F' are equivalent with respect to satisfiability and have exactly the same number of models (i.e., $\#(F') = \#(F)$). $\square$

2. Preliminaries

2.1. Tree Decompositions

Let $G = (V(G), E(G))$ be a graph, $T = (V(T), E(T))$ be a tree, and χ be a labeling of the vertices of T by sets of vertices of G. We refer to the vertices of T as "nodes" to avoid confusion with the vertices of G. The tuple (T, χ) is a *tree decomposition* of G if the following three conditions hold:

1. For every $v \in V(G)$ there exists a node $t \in V(T)$ such that $v \in \chi(t)$.
2. For every $vw \in E(G)$ there exists a node $t \in V(T)$ such that $v, w \in \chi(t)$.
3. For any three nodes $t_1, t_2, t_3 \in V(T)$, if t_2 lies on the path from t_1 to t_3, then $\chi(t_1) \cap \chi(t_3) \subseteq \chi(t_2)$ ("Connectedness Condition").

The *width* of a tree decomposition (T, χ) is defined by $\max_{t \in V(T)} |\chi(t)| - 1$. The *treewidth* $tw(G)$ of a graph G is the minimum width over all its tree decompositions. For constant k, there exists a linear-time algorithm that checks whether a given graph has treewidth at most k and, if so, outputs a tree decomposition of minimum width [5]. However, the huge constant factor in the runtime of this algorithm makes it practically infeasible. For our purposes, however, it suffices to obtain tree decompositions of small but not necessarily minimal width. There exist several powerful tree decomposition heuristics that construct tree decompositions of small width for many cases that are relevant in practice [6, 23].

In this paper we also consider a special type of tree decompositions. The triple (T, χ, r) is a *nice tree decomposition* of G if (T, χ) is a tree decomposition, the tree T is rooted at node r, and the following three conditions hold [20]:

1. Every node of T has at most two children.
2. If a node t of T has two children t_1 and t_2, then $\chi(t) = \chi(t_1) = \chi(t_2)$; in that case we call t a *join node*.
3. If a node t of T has exactly one child t', then exactly one of the following prevails:
   (a) $|\chi(t)| = |\chi(t')| + 1$ and $\chi(t') \subset \chi(t)$; in that case we call t an *introduce node*.
   (b) $|\chi(t)| = |\chi(t')| - 1$ and $\chi(t) \subset \chi(t')$; in that case we call t a *forget node*.

It is known that one can transform efficiently any tree decomposition of width k of a graph with n vertices into a nice tree decomposition of width at most k and at most $4n$ nodes [20].

Let (T, χ, r) be a nice tree decomposition of a graph G. For each node t of T let T_t denote the subtree of T rooted at t; furthermore, let G_t denote the subgraph of G that is induced by the set $V_t = \bigcup_{t' \in V(T_t)} \chi(t')$ of vertices. Observe that $(T_t, \chi|_{V(T_t)}, t)$ is a nice tree decomposition of G_t.

2.2. Propositional Formulas

We consider propositional formulas F in conjunctive normal form (CNF) represented as set of clauses. Each clause in F is a finite set of *literals*, and a literal is a negated or unnegated propositional *variable*. For example,

$$F = \{\{\neg x, y, z\}, \{\neg y, \neg z\}, \{x, \neg y\}\}$$

represents the propositional formula $(\neg x \vee y \vee z) \wedge (\neg y \vee \neg z) \wedge (x \vee \neg y)$. For a clause C we denote by $\mathit{var}(C)$ the set of variables that occur (negated or unnegated) in C; for a formula F we put $\mathit{var}(F) = \bigcup_{C \in F} \mathit{var}(C)$. The *size* of a clause is its cardinality.

A *truth assignment* is a mapping $\tau : X \to \{0,1\}$ defined on some set X of variables. We extend τ to literals by setting $\tau(\neg x) = 1 - \tau(x)$ for $x \in X$. A truth assignment $\tau : X \to \{0,1\}$ *satisfies* a clause C if for some variable $x \in \mathit{var}(C) \cap X$ we have $x \in C$ and $\tau(x) = 1$, or $\neg x \in C$ and $\tau(x) = 0$. A truth assignment $\tau : X \to \{0,1\}$ *falsifies* a clause C if $\mathit{var}(C) \subseteq X$ and for every variable $x \in \mathit{var}(C)$ we have $x \in C$ and $\tau(x) = 0$, or $\neg x \in C$ and $\tau(x) = 1$. An assignment satisfies (resp. falsifies) a set A of clauses if it satisfies (resp. falsifies) every clause in A. A set A of clauses is *satisfiable* (resp. *falsifiable*) if there exists a truth assignment that satisfies (resp. falsifies) A; otherwise F is *unsatisfiable* (resp. *unfalsifiable*). Note that a set A of clauses is unfalsifiable if and only if the union of A contains a complementary pair of literals. For a formula F, we call a truth assignment $\tau : \mathit{var}(F) \to \{0,1\}$ a *model* of F if τ satisfies F. We denote by $\#(F)$ the number of models of F. Thus, F is satisfiable if and only if $\#(F) \geq 1$. The *propositional satisfiability problem* SAT is the problem of deciding whether a given propositional formula in CNF is satisfiable. The *propositional model counting problem* #SAT is the problem of computing $\#(F)$ of a given propositional formula F in CNF.

2.3. Primal, Dual, and Incidence Treewidth

The *primal graph* $G(F)$ of a CNF formula F is the graph with vertex set $\mathit{var}(F)$; two variables x, y are joined by an edge if and only if $x, y \in \mathit{var}(C)$ for some clause $C \in F$. The *primal treewidth* (or *treewidth*, for short) $\mathit{tw}(F)$ of a CNF formula F is the treewidth of its primal graph, that is $\mathit{tw}(F) = \mathit{tw}(G(F))$.

The *dual graph* $G^d(F)$ of a CNF formula F is the graph with vertex set F; two clauses C, C' are joined by an edge if and only if $var(C) \cap var(C') \neq \emptyset$. The *dual treewidth* $tw^d(F)$ of a CNF formula F is the treewidth of its dual graph, that is $tw^d(F) = tw(G^d(F))$.

The *incidence graph* $G^*(F)$ of a CNF formula F is the bipartite graph with vertex set $F \cup var(F)$; a variable x and a clause C are joined by an edge if and only if $x \in var(C)$. The *incidence treewidth* $tw^*(F)$ of a CNF formula F is the treewidth of its incidence graph, that is $tw^*(F) = tw(G^*(F))$.

3. The Fixed-Parameter Algorithms

Since the number of models of a CNF formula can be exponential in the number of its variables (and thus may become too large to be stored in a single data word), we consider in the following the bit complexity of our algorithms, i.e., instead of assuming that arithmetic operations have constant runtime we bound their runtime by the number of bit operations (cf. Aho, Hopcroft, and Ullman [1], pages 22–23). To this aim, we introduce δ to denote the runtime of multiplying two n-bit integers, the computationally most expensive arithmetic operation in our algorithms. In the literature there exist several algorithms for multiplying two n-bit integers; we refer the interested reader to Knuth's in-depth overview [21]. One of the most prominent of these algorithms is due to Schönhage and Strassen [21, 30] and runs in time $\mathcal{O}(n \log n \log\log n)$. Thus, we can assume that $\delta = \mathcal{O}(n \log n \log\log n)$, where n is the number of variables of the given CNF formula. Recently, Fürer [15] presented an even faster algorithm. If arithmetic operations are assumed to have constant runtime, that is, $\delta = \mathcal{O}(1)$, we easily obtain the runtimes listed in the introduction from the runtimes stated in Theorems 1, 2, and 3.

3.1. Primal Treewidth

For this section, let (T, χ, r) be a nice tree decomposition of the primal graph $G(F)$ of a CNF formula F. Let k denote the width of (T, χ, r) and let t be a node of T. For each truth assignment $\alpha : \chi(t) \to \{0, 1\}$ we define $N(t, \alpha)$ as the set of truth assignments $\tau : V_t \to \{0, 1\}$ for which the following two conditions hold:

1. $\tau(x) = \alpha(x)$ for all variables $x \in \chi(t)$.
2. There is no clause in F that is falsified by τ.

We represent the values of $n(t, \alpha) = |N(t, \alpha)|$ for all $\alpha : \chi(t) \to \{0, 1\}$ by a table M_t with $|\chi(t)| + 1$ columns and $2^{|\chi(t)|}$ rows. The first $|\chi(t)|$ columns of M_t contain Boolean values encoding $\alpha(x)$ for variables $x \in \chi(t)$. The last entry of each row contains the integer $n(t, \alpha)$.

Lemma 1. *Let t be a* join node *of T with children t_1, t_2. Then, for each truth assignment $\alpha : \chi(t) \to \{0, 1\}$, we have*

$$n(t, \alpha) \quad = \quad n(t_1, \alpha) \cdot n(t_2, \alpha).$$

PROOF. In the following, we will show that the mapping $f : \tau \mapsto (\tau|_{V_{t_1}}, \tau|_{V_{t_2}})$ is a bijection from the set $N(t, \alpha)$ into the set $M = \{(\tau_1, \tau_2) \mid \tau_1 \in N(t_1, \alpha)$ and $\tau_2 \in N(t_2, \alpha)\}$. The above equality follows then immediately.

It is easy to see that f is a mapping from $N(t,\alpha)$ into M. To show that f is injective, let $\tau, \sigma \in N(t,\alpha)$ such that $f(\tau) = f(\sigma)$. Then, since $\tau|_{V_{t_1}} = \sigma|_{V_{t_1}}$ and $\tau|_{V_{t_2}} = \sigma|_{V_{t_2}}$, we know that $\tau = \sigma$. To show that f is surjective, let $(\tau_1, \tau_2) \in M$. Now let us define the truth assignment $\tau : V_t \to \{0,1\}$ by $\tau|_{V_{t_1}} = \tau_1$ and $\tau|_{V_{t_2}} = \tau_2$. For the sake of contradiction, let us assume that there exists a clause $C \in F$ which is falsified by τ. Since C is not falsified by τ_1 and τ_2, we know that $var(C) \nsubseteq V_{t_1}$ and $var(C) \nsubseteq V_{t_2}$. Thus, there exist variables $x, y \in var(C)$ with $x \in V_{t_1} \setminus V_{t_2}$ and $y \in V_{t_2} \setminus V_{t_1}$. By the definition of join nodes, this implies that $x, y \notin \chi(t) = \chi(t_1) = \chi(t_2)$. Moreover, by Condition 2 of a tree decomposition, there must be a node $t' \in V(T)$ such that $x, y \in \chi(t')$. Clearly, $t' \notin V(T_t)$. This, however, contradicts the Connectedness Condition in the definition of a tree decomposition of the primal graph since, for example, there are nodes $t' \in V(T) \setminus V(T_t)$ and $t'_1 \in V(T_{t_1})$ such that $x \in \chi(t') \cap \chi(t'_1)$ but $x \notin \chi(t)$. Hence, we have $\tau \in N(t,\alpha)$. Consequently, f is indeed a bijection from $N(t,\alpha)$ into M. □

Lemma 2. *Let t be an* introduce node *with child t' and $\chi(t) = \chi(t') \cup \{x\}$ for a variable x. Then, for each truth assignment $\alpha : \chi(t) \to \{0,1\}$, we have*

$$n(t,\alpha) \quad = \quad \begin{cases} 0 & \text{if } \alpha \text{ falsifies some } C \in F; \\ n(t', \alpha|_{\chi(t')}) & \text{otherwise.} \end{cases}$$

Proof. Note that, by definition, $N(t,\alpha) = \emptyset$ if α falsifies some clause $C \in F$. Thus, let us assume that α falsifies no clause $C \in F$. In the following, we will show that the mapping $f : \tau \mapsto \tau|_{V_{t'}}$ is a bijection from the set $N(t,\alpha)$ into the set $N(t', \alpha|_{\chi(t')})$. The above equality follows then immediately.

It is easy to see that f is a mapping from $N(t,\alpha)$ into $N(t', \alpha|_{\chi(t')})$. To show that f is injective, let $\tau, \sigma \in N(t,\alpha)$ such that $f(\tau) = f(\sigma)$. Then, since $\tau|_{V_{t'}} = \sigma|_{V_{t'}}$ and $\tau(x) = \alpha(x) = \sigma(x)$ for the single variable $x \in V_t \setminus V_{t'}$, we know that $\tau = \sigma$. To show that f is surjective, let $\tau' \in N(t', \alpha|_{\chi(t')})$. Now let us define the truth assignment $\tau : V_t \to \{0,1\}$ by $\tau|_{V_{t'}} = \tau'$ and $\tau(x) = \alpha(x)$ for the single variable $x \in V_t \setminus V_{t'}$. For the sake of contradiction, let us assume that there exists a clause $C \in F$ which is falsified by τ. Since C is not falsified by τ' and α, we know that $var(C) \nsubseteq V_{t'}$ and $var(C) \nsubseteq \chi(t)$. Thus, there exist variables $x, y \in var(C)$ with $x \in \chi(t) \setminus V_{t'}$ and $y \in V_{t'} \setminus \chi(t)$. This, however, contradicts the definition of a tree decomposition of the primal graph. Hence, we have $\tau \in N(t,\alpha)$. Consequently, f is indeed a bijection from $N(t,\alpha)$ into $N(t', \alpha|_{\chi(t')})$. □

Lemma 3. *Let t be a* forget node *with child t' and $\chi(t) = \chi(t') \setminus \{x\}$ for a variable x. Then, for each truth assignment $\alpha : \chi(t) \to \{0,1\}$, we have*

$$n(t,\alpha) \quad = \quad n(t', \alpha \cup \{(x,0)\}) + n(t', \alpha \cup \{(x,1)\}).$$

Proof. It is easy to see that $N(t,\alpha) = N(t', \alpha \cup \{(x,0)\}) \cup N(t', \alpha \cup \{(x,1)\})$. The above equality follows then immediately. □

Lemma 4. *Let t be a* leaf node*. Then, for each truth assignment $\alpha : \chi(t) \to \{0,1\}$, we have*

$$n(t,\alpha) \quad = \quad \begin{cases} 0 & \text{if } \alpha \text{ falsifies some } C \in F; \\ 1 & \text{otherwise.} \end{cases}$$

Proof. Since $V_t = \chi(t)$ for every leaf node t, we know that for each truth assignment $\tau : V_t \to \{0,1\}$ there exists exactly one truth assignment $\alpha : \chi(t) \to \{0,1\}$ (and vice versa) such that $\tau(x) = \alpha(x)$ for all variables $x \in V_t$. Hence, the above equality follows immediately. $\square$

By using these equalities, we can now construct the tables M_t from the leaves to the root according to the following lemma.

Lemma 5. *Let t be a node of T. Given the tables of the children of t, we can compute the table M_t in time $\mathcal{O}(2^k(kd+\delta))$, where d is the maximum number of occurrences over all variables.*

Proof. To check the runtime of computing M_t, let $q = |\chi(t)|$; since we assume that the width of the tree decomposition under consideration is k, we have $q \leq k+1$. Now let us distinguish between the different kinds of nodes.

(i) Let t be a *join node* with children t_1, t_2. We compute the table M_t from tables M_{t_1} and M_{t_2} according to Lemma 1 as follows: For each of the 2^q choices of α in table M_t, we go simultaneously through the corresponding rows in the tables M_{t_1} and M_{t_2} and set the last entry of row $M_t(\alpha)$ to $n(t_1, \alpha) \cdot n(t_2, \alpha)$. Hence, we can compute M_t in time $\mathcal{O}(2^q\delta) \subseteq \mathcal{O}(2^k(kd+\delta))$.

(ii) Let t be an *introduce node* with child t'. We compute the table M_t from table $M_{t'}$ according to Lemma 2 as follows: For each of the 2^q choices of α in table M_t, we check whether α falsifies some clause in F, which can be accomplished in time $\mathcal{O}(qd)$. If so, we set the last entry of row $M(t, \alpha)$ to 0; otherwise, we search for row $M_{t'}(\alpha|_{\chi(t')})$ and set the last entry of row $M(t, \alpha)$ to $n(t', \alpha|_{\chi(t')})$, which can be accomplished in time $\mathcal{O}(q+\delta)$. Hence, we can compute M_t in time $\mathcal{O}(2^q(qd+\delta)) \subseteq \mathcal{O}(2^k(kd+\delta))$.

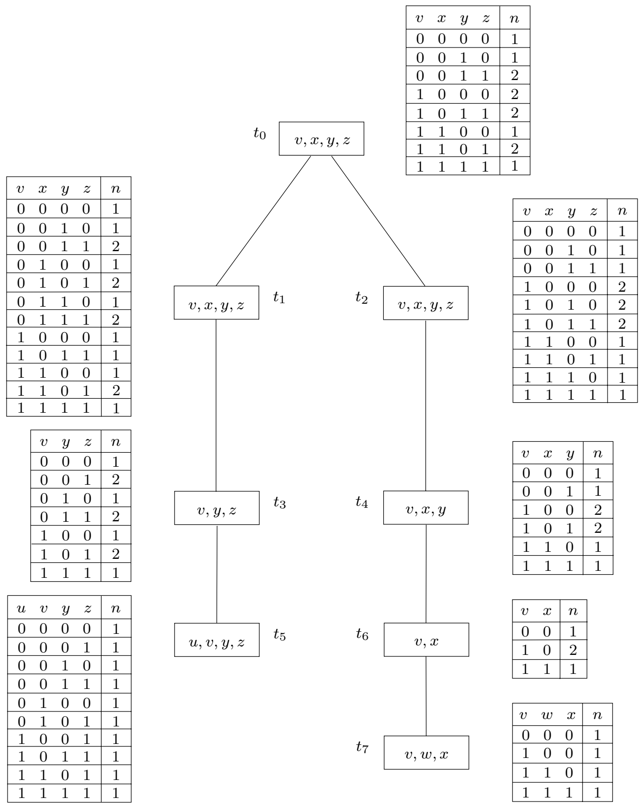
(iii) Let t be a *forget node* with child t'. We compute the table M_t from table $M_{t'}$ according to Lemma 3 as follows: For each of the 2^q choices of α in table M_t, we search for the rows $M_{t'}(\alpha \cup \{(x,0)\})$ and $M_{t'}(\alpha \cup \{(x,1)\})$, which can be accomplished in time $\mathcal{O}(q)$. We set the last entry of row $M_t(\alpha)$ to $n(t', \alpha \cup \{(x,0)\}) + n(t', \alpha \cup \{(x,1)\})$. Hence, we can compute M_t in time $\mathcal{O}(2^q(q+\delta)) \subseteq \mathcal{O}(2^k(kd+\delta))$.

(iv) Let t be a *leaf node*. We compute the table M_t according to Lemma 4 as follows: For each of the 2^q choices of α in table M_t, we check whether α falsifies some clause in F, which can be accomplished in time $\mathcal{O}(qd)$. If so, we set the last entry of row $M(t, \alpha)$ to 0; otherwise, we set it to 1. Hence, we can compute table M_t in time $\mathcal{O}(2^q qd) \subseteq \mathcal{O}(2^k(kd+\delta))$. $\square$

Theorem 1. *Given a nice tree decomposition of the primal graph of a CNF formula F, we can compute $\#(F)$ in time $\mathcal{O}(2^k(kd+\delta)\,N)$; d denotes the maximum number of occurrences over all variables in F, k denotes the width and N the number of nodes of the tree decomposition.*

Proof. Let (T, χ, r) be a nice tree decomposition of the primal graph of F; let k and N be the width and number of nodes of (T, χ, r) respectively. Starting from the leaf nodes of T, we compute the tables M_t for all nodes t of T in a bottom-up ordering. Each table can be computed by Lemma 5 in time $\mathcal{O}(2^k(kd+\delta))$. Since we have

$$\#(F) = \sum_{\alpha:\chi(r)\to\{0,1\}} n(r, \alpha),$$



v	x	y	z	n
0	0	0	0	1
0	0	1	0	1
0	0	1	1	2
1	0	0	0	2
1	0	1	1	2
1	1	0	0	1
1	1	0	1	2
1	1	1	1	1

v	x	y	z	n
0	0	0	0	1
0	0	1	0	1
0	0	1	1	2
0	1	0	0	1
0	1	0	1	2
0	1	1	0	1
0	1	1	1	2
1	0	0	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	2
1	1	1	1	1

v	x	y	z	n
0	0	0	0	1
0	0	1	0	1
0	0	1	1	1
1	0	0	0	2
1	0	1	0	2
1	0	1	1	2
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

v	y	z	n
0	0	0	1
0	0	1	2
0	1	0	1
0	1	1	2
1	0	0	1
1	0	1	2
1	1	1	1

v	x	y	n
0	0	0	1
0	0	1	1
1	0	0	2
1	0	1	2
1	1	0	1
1	1	1	1

u	v	y	z	n
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	1
0	1	0	1	1
1	0	0	1	1
1	0	1	1	1
1	1	0	1	1
1	1	1	1	1

v	x	n
0	0	1
1	0	2
1	1	1

v	w	x	n
0	0	0	1
1	0	0	1
1	1	0	1
1	1	1	1

Figure 2: Solving #SAT on a nice tree decomposition of the primal graph

we can read off $\#(F)$ from the table M_r at the root r. $\square$

An example of this algorithm on the tree decomposition of the primal graph in Figure 1 is shown in Figure 2. Note that, for simplicity, we have omitted those rows from the tables where $n(t, \alpha) = 0$. From table M_{t_0} we can read off that there are exactly $1 + 1 + 2 + 2 + 2 + 1 + 2 + 1 = 12$ models of the corresponding CNF formula. Let us remark that our above algorithm is related to Yannakakis's algorithm [33] for deciding whether an acyclic constraint satisfaction instance has a solution.

3.2. Dual Treewidth

For this section, let (T, χ, r) be a nice tree decomposition of the dual graph $G^d(F)$ of a CNF formula F. Let k denote the width of (T, χ, r) and let t be a node of T. For each subset $A \subseteq \chi(t)$ we define $N(t, A)$ as the set of truth assignments $\tau : var(V_t) \rightarrow \{0, 1\}$ for which the following two conditions hold:

1. Every clause in A is falsified by τ.
2. Every clause in $V_t \setminus \chi(t)$ is satisfied by τ.

We represent the values of $n(t, A) = |N(t, A)|$ for all $A \subseteq \chi(t)$ by a table M_t with $|\chi(t)|+1$ columns and $2^{|\chi(t)|}$ rows. The first $|\chi(t)|$ columns of M_t contain Boolean values encoding membership of C in A for clauses $C \in \chi(t)$. The last entry of each row contains the integer $n(t, A)$.

Lemma 6. *Let t be a* join node *of T with children t_1, t_2. Then, for each set $A \subseteq \chi(t)$, we have*

$$n(t, A) \quad = \quad \frac{n(t_1, A) \cdot n(t_2, A)}{2^{|var(\chi(t)) \setminus var(A)|}}.$$

Proof. In the following, we will show that the mapping $f : \tau \mapsto (\tau|_{var(V_{t_1})}, \tau|_{var(V_{t_2})})$ is a bijection from the set $N(t, A)$ into the set $M = \{(\tau_1, \tau_2) \mid \tau_1 \in N(t_1, A),\ \tau_2 \in N(t_2, A),\ \tau_1(x) = \tau_2(x) \text{ for all } x \in var(V_{t_1}) \cap var(V_{t_2})\}$ and that $|M| = |N(t_1, A)|\,|N(t_2, A)|/2^{|var(\chi(t)) \setminus var(A)|}$. The above equality follows then immediately.

First, let us show that f is a mapping from $N(t, A)$ into M. To this aim, let $\tau \in N(t, A)$ and $f(\tau) = (\tau_1, \tau_2)$. Note that $\tau_i(x) = \tau(x)$ for all $x \in var(V_{t_i})$, $i = 1, 2$. Thus, we know that every clause in $A \subseteq \chi(t) = V_{t_1} \cap V_{t_2}$ is falsified by τ_i and every clause in $(V_t \setminus \chi(t)) \cap V_{t_i} = V_{t_i} \setminus \chi(t_i)$ is satisfied by τ_i, that is, $\tau_1 \in M_1$ and $\tau_2 \in M_2$. To show that f is injective, let $\tau, \sigma \in N(t, A)$ such that $f(\tau) = f(\sigma)$. Then, since $\tau|_{var(V_{t_1})} = \sigma|_{var(V_{t_1})}$ and $\tau|_{var(V_{t_2})} = \sigma|_{var(V_{t_2})}$, we know that $\tau = \sigma$. To show that f is surjective, let $(\tau_1, \tau_2) \in M$. Now let us define the truth assignment $\tau : var(V_t) \to \{0, 1\}$ by $\tau|_{var(V_{t_1})} = \tau_1$ and $\tau|_{var(V_{t_2})} = \tau_2$. Since $A \subseteq \chi(t) = V_{t_1} \cap V_{t_2}$ and $V_t \setminus \chi(t) = (V_{t_1} \cup V_{t_2}) \setminus \chi(t) = (V_{t_1} \setminus \chi(t_1)) \cup (V_{t_2} \setminus \chi(t_2))$, it is easy to see that $\tau \in N(t, A)$. Consequently, f is indeed a bijection from $N(t, A)$ into M.

What remains to show is that $|M| = |N(t_1, A)||N(t_2, A)|/2^{|var(\chi(t)) \setminus var(A)|}$. To this aim, note first that, by the definition of a dual graph and a tree decomposition, $var(V_{t_1}) \cap var(V_{t_2}) = var(\chi(t)) = var(\chi(t_1)) = var(\chi(t_2))$. Moreover, for $\tau_1 \in N(t_1, A)$ and $\tau_2 \in N(t_2, A)$, it holds that $\tau_1(x) = \tau_2(x)$ for all $x \in var(A)$. Now, for $i = 1, 2$, let $X_i = var(\chi(t_i)) \setminus (var(A) \cup var(V_{t_i} \setminus \chi(t_i)))$ be the set of variables in $var(\chi(t_i))$ that do not occur in falsified clauses in A or in satisfied clauses in $V_{t_i} \setminus \chi(t_i)$. It holds that $X = X_1 \cap X_2 = var(\chi(t)) \setminus (var(A) \cup var(V_t \setminus \chi(t)))$, i.e., $X = X_1 \cap X_2$ is the set of variables in $var(\chi(t))$ that do not occur in falsified clauses in A or in satisfied clauses in $V_t \setminus \chi(t)$. It is thus easy to see that if $\tau \in N(t_i, A)$, then for every truth assignment $\tau' : var(V_{t_i}) \to \{0, 1\}$ with $\tau'(x) = \tau(x)$ for all $x \in var(V_{t_i}) \setminus X_i$ it holds that $\tau' \in N(t_i, A)$. Moreover, if $(\tau_1, \tau_2) \in M$, then for every truth assignment $\tau_1' : var(V_{t_1}) \to \{0, 1\}$ and $\tau_2' : var(V_{t_2}) \to \{0, 1\}$ with $\tau_1'(x) = \tau_1(x)$ for all $x \in var(V_{t_1}) \setminus X$, $\tau_2'(x) = \tau_2(x)$ for all $x \in var(V_{t_2}) \setminus X$, and $\tau_1'(x) = \tau_2'(x)$ for all $x \in X$ it holds that $(\tau_1', \tau_2') \in M$. Now let $M_1 = \{\tau|_{var(V_{t_1}) \setminus X} \mid \tau \in N(t_1, A)\}$, $M_2 = \{\tau|_{var(V_{t_2}) \setminus X} \mid \tau \in N(t_2, A)\}$, and $M' = \{(\tau_1, \tau_2) \mid \tau_1 \in M_1,\ \tau_2 \in M_2,\ \tau_1(x) = \tau_2(x) \text{ for all } x \in (var(V_{t_1}) \cap var(V_{t_2})) \setminus X\}$. By our observation above, we know that $|N(t_1, A)| = 2^{|X|}|M_1|$, $|N(t_2, A)| = 2^{|X|}|M_2|$, and $|M| = 2^{|X|}|M'|$. Our next step is to compute $|M'|$. To this aim, we partition M_1 and M_2 into equivalence classes such that $\tau, \tau' \in M_i$ are in the same equivalence class if and only if $\tau(x) = \tau'(x)$ for all $x \in var(V_{t_i}) \setminus X_i$. It is easy to see that there are exactly $2^{|X_i \setminus X|}$ truth assignments in each equivalence class of M_i and therefore exactly $|M_i|/2^{|X_i \setminus X|}$ equivalence classes of M_i, for $i = 1, 2$. Now let us consider w.l.o.g. any equivalence classes $E_1 \subseteq M_1$ and $E_2 \subseteq M_2$. Let $\tau_1 \in E_1$ and $\tau_2 \in E_2$. By our definition of the equivalence classes, we know that for all $\tau_i' \in E_i$ it holds that $\tau_i'(x) = \tau_i(x)$ for all $x \in var(\chi(t_i)) \setminus X_i$. Moreover, note that $((var(\chi(t_1)) \setminus var(A)) \setminus X_1) \cap ((var(\chi(t_2)) \setminus var(A)) \setminus X_2) = \emptyset$. Otherwise, there exist $C_1 \in V_{t_1} \setminus \chi(t_1)$ and $C_2 \in V_{t_2} \setminus \chi(t_2)$ such that $var(C_1) \cap var(C_2) \neq \emptyset$, which contradicts the definition of a dual graph or a tree decomposition. Thus, we know that there exists exactly one $\tau_1' \in E_1$ such that $\tau_1'(x) = \tau_2(x)$ for all $x \in (var(\chi(t_2)) \setminus var(A)) \setminus X_2 = X_1 \setminus X$. Symmetrically, there exists exactly one $\tau_2' \in E_2$ such that $\tau_2'(x) = \tau_1(x)$ for all $x \in (var(\chi(t_1)) \setminus var(A)) \setminus X_1 = X_2 \setminus X$. Hence, since $(var(\chi(t)) \setminus var(A)) \setminus X = ((var(\chi(t_1)) \setminus var(A)) \setminus X_1) \cup (X_1 \setminus X) = ((var(\chi(t_2)) \setminus var(A)) \setminus X_2) \cup (X_2 \setminus X)$, we know that $\tau_1'(x) = \tau_2'(x)$ for all $x \in (var(\chi(t)) \setminus var(A)) \setminus X$, that is, $(\tau_1', \tau_2') \in M'$. So we have $|M'| = (|M_1|/2^{|X_1 \setminus X|})(|M_2|/2^{|X_2 \setminus X|}) = |M_1||M_2|/2^{|(X_1 \cup X_2) \setminus X|} = |N(t_1, A)||N(t_2, A)|/2^{|(X_1 \cup X_2) \setminus X| + 2|X|} = |N(t_1, A)||N(t_2, A)|/2^{|X_1 \cup X_2| + |X|}$. Consequently, by putting our results together, we obtain $|M| = 2^{|X|}(|N(t_1, A)||N(t_2, A)|/2^{|X_1 \cup X_2| + |X|}) = |N(t_1, A)||N(t_2, A)|/2^{|X_1 \cup X_2|}$. □

Lemma 7. *Let t be an* introduce node *with child t' and $\chi(t) = \chi(t') \cup \{C\}$ for a clause C. Then, for each set $A \subseteq \chi(t)$, we have*

$$n(t, A) \quad = \quad \begin{cases} 0 & \text{if } A \text{ is unfalsifiable;} \\ n(t', A) \cdot 2^{|var(C) \setminus var(\chi(t'))|} & \text{otherwise, if } C \notin A; \\ \dfrac{n(t', A \setminus \{C\})}{2^{|var(C) \cap (var(\chi(t')) \setminus var(A \setminus \{C\}))|}} & \text{otherwise, if } C \in A. \end{cases}$$

Proof. Note that, by definition, $N(t, A) = \emptyset$ if there is no truth assignment $\tau : var(V_t) \to \{0, 1\}$ that falsifies A. Thus, let us assume that A is falsifiable. Now we define $M_1 = \{\tau : var(V_t) \to \{0, 1\} \mid \text{there exists } \tau' \in N(t', A) \text{ such that } \tau(x) = \tau'(x) \text{ for all } x \in var(V_{t'})\}$ and $M_2 = \{\tau \in N(t', A \setminus \{C\}) \mid \tau(x) = 0 \text{ if } x \in C \text{ and } \tau(x) = 1 \text{ if } \neg x \in C$ for all $x \in var(C) \cap var(V_{t'})\}$. In the following, we will show that (i) if $C \notin A$ the mapping $f : \tau \mapsto \tau$ is a bijection from the set $N(t, A)$ into the set M_1 and $|M_1| = |N(t', A)|2^{|var(C) \setminus var(\chi(t'))|}$ and that (ii) if $C \in A$ the mapping $g : \tau \mapsto \tau|_{V_{t'}}$ is a bijection from the set $N(t, A)$ into the set M_2 and $|M_2| = |N(t', A \setminus \{C\})|/2^{|var(C) \cap (var(\chi(t')) \setminus var(A \setminus \{C\}))|}$. The above equality follows then immediately.

(i) Since $C \notin A$ and $V_t \setminus \chi(t) = V_{t'} \setminus \chi(t')$, it is easy to see that f is a bijection from $N(t, A)$ into M_1. To compute $|M_1|$, note that $var(C) \cap var(V_{t'}) \subseteq var(\chi(t'))$. Otherwise, there exists $C' \in V_{t'} \setminus \chi(t')$ such that $var(C) \cap var(C') \neq \emptyset$, which contradicts the definition of a dual graph or a tree decomposition. Thus, we have $|M_1| = |N(t', A)|2^{|var(V_t) \setminus var(V_{t'})|} = |N(t', A)|2^{|var(C) \setminus var(\chi(t'))|}$.

(ii) First let us show that g is a mapping from $N(t, A)$ into M_2. To this aim, let $\tau \in N(t, A)$ and $g(\tau) = \tau'$. Since $C \in A$, we know that $\tau(x) = 0$ if $x \in C$ and $\tau(x) = 1$ if $\neg x \in C$ for all $x \in var(C)$. Thus, we have $\tau'(x) = \tau(x) = 0$ if $x \in C$ and $\tau'(x) = \tau(x) = 1$ if $\neg x \in C$ for all $x \in var(C) \cap var(V_{t'})$, that is, $\tau' \in M_2$. To show that g is injective, let $\tau, \sigma \in N(t, A)$ such that $g(\tau) = g(\sigma)$. Then, since $\tau|_{var(V_{t'})} = \sigma|_{var(V_{t'})}$, $\tau(x) = \sigma(x) = 0$ if $x \in C$, and $\tau(x) = \sigma(x) = 1$ if $\neg x \in C$ for all $x \in var(V_t) \setminus var(V_{t'}) \subseteq var(C)$, we know that $\tau = \sigma$. To show that g is surjective, let $\tau' \in M_2$. Now let us define the truth assignment $\tau : var(V_t) \to \{0, 1\}$ by $\tau|_{var(V_{t'})} = \tau'$, $\tau(x) = 0$ if $x \in C$, and $\tau(x) = 1$ if $\neg x \in C$ for all $x \in var(C) \setminus var(V_{t'})$. Since $V_t \setminus \chi(t) = V_{t'} \setminus \chi(t')$, it is easy to see that $\tau \in N(t, A)$. Consequently, g is indeed a bijection from $N(t, A)$ into M_2. To compute $|M_2|$, note that if $\tau \in N(t', A \setminus \{C\})$, then for every truth assignment $\tau' : var(V_{t'}) \to \{0, 1\}$

with $\tau'(x) = \tau(x)$ for all $x \in var(A \setminus \{C\}) \cup var(V_{t'} \setminus \chi(t'))$ it holds that $\tau' \in N(t', A \setminus \{C\})$. However, we know that for all $\tau \in N(t', A \setminus \{C\})$ it holds that $\tau(x) = 0$ if $x \in C$ and $\tau(x) = 1$ if $\neg x \in C$ for all $x \in var(C) \cap var(A \setminus \{C\})$. Otherwise, A is not falsifiable, which contradicts our assumption. Moreover, we have $var(C) \cap var(V_{t'} \setminus \chi(t')) = \emptyset$. Otherwise, there exists $C' \in V_{t'} \setminus \chi(t')$ such that $var(C) \cap var(C') \neq \emptyset$, which contradicts the definition of a dual graph or a tree decomposition. In particular, this implies that if $\tau \in N(t', A \setminus \{C\})$ then also $\tau' \in N(t', A \setminus \{C\})$ if τ' differs from τ only on variables in $var(C) \cap (var(V_{t'}) \setminus var(A \setminus \{C\})) = var(C) \cap (var(\chi(t')) \setminus var(A \setminus \{C\}))$. Thus, we have $|M_2| = |N(t', A \setminus \{C\})| / 2^{|var(C) \cap (var(\chi(t')) \setminus var(A \setminus \{C\}))|}$. ☐

Lemma 8. *Let t be a* forget node *with child t' and $\chi(t) = \chi(t') \setminus \{C\}$ for a clause C. Then, for each set $A \subseteq \chi(t)$, we have*

$$n(t, A) \quad = \quad n(t', A) - n(t', A \cup \{C\}).$$

Proof. Let $M = N(t', A) \setminus N(t', A \cup \{C\})$ be the set of truth assignments $\tau : var(V_{t'}) \rightarrow \{0, 1\}$ such that every clause in A is falsified by τ and every clause in $(V_{t'} \setminus \chi(t')) \cup \{C\} = V_{t'} \setminus \chi(t)$ is satisfied by τ. Since $V_t = V_{t'}$, it is easy to see that $N(t, A) = M$. Moreover, since always $N(t', A \cup \{C\}) \subseteq N(t', A)$, the above equality follows immediately. ☐

Lemma 9. *Let t be a* leaf node. *Then, for each set $A \subseteq \chi(t)$, we have*

$$n(t, A) \quad = \quad \begin{cases} 0 & \textit{if } A \textit{ is unfalsifiable;} \\ 2^{|var(\chi(t)) \setminus var(A)|} & \textit{otherwise.} \end{cases}$$

Proof. Note that, by definition, $N(t, A) = \emptyset$ if there is no truth assignment $\tau : var(V_t) \rightarrow \{0, 1\}$ that falsifies A. Thus, let us assume that A is falsifiable. It is easy to see that there are exactly $2^{|var(V_t) \setminus var(A)|}$ truth assignments $\tau : var(V_t) \rightarrow \{0, 1\}$ such that every clause in A is falsified by τ. Hence, since $V_t = \chi(t)$ for every leaf node t, the above equality follows immediately. ☐

By using these equalities, we can now construct the tables M_t from the leaves to the root according to the following lemma.

Lemma 10. *Let t be a node of T. Given the tables of the children of t, we can compute the table M_t in time $\mathcal{O}(2^k(kl + \delta))$, where l is the size of a largest clause.*

Proof. To check the runtime of computing M_t, let $q = |\chi(t)|$; since we assume that the width of the tree decomposition under consideration is k, we have $q \leq k + 1$. Now let us distinguish between the different kinds of nodes.

(i) Let t be a *join node* with children t_1, t_2. We compute the table M_t from the tables M_{t_1} and M_{t_2} according to Lemma 6 as follows: For each of the 2^q choices of A in table M_t, we go simultaneously through the corresponding rows in the tables M_{t_1} and M_{t_2} and compute the cardinality of $var(\chi(t)) \setminus var(A)$, which can be accomplished in time $\mathcal{O}(ql)$. We set the last entry of row $M_t(A)$ to $n(t_1, A) \cdot n(t_2, A)$ and perform a shift of $|var(\chi(t)) \setminus var(A)|$ bits to the right. Hence, we can compute M_t in time $\mathcal{O}(2^q(ql + \delta)) \subseteq \mathcal{O}(2^k(kl + \delta))$.

(ii) Let t be an *introduce node* with child t'. We compute the table M_t from table $M_{t'}$ according to Lemma 7 as follows: For each of the 2^q choices of A in table M_t, we search for row $M_{t'}(A \setminus \{C\})$, which can be accomplished in time $\mathcal{O}(q)$. Then we check whether A is falsifiable and, if so, whether $C \in A$, which can be accomplished in time $\mathcal{O}(ql)$. If $C \notin A$, we compute the cardinality of $var(C) \setminus var(\chi(t'))$ and, if $C \in A$, we compute the cardinality of $var(C) \cap (var(\chi(t')) \setminus var(A \setminus \{C\}))$, which can also be accomplished in time $\mathcal{O}(ql)$. In the first case we set the last entry of row $M(t, A)$ to $n(t', A)$ and perform a shift of $|var(C) \setminus var(\chi(t'))|$ bits to the left, and in the second case we set the last entry of row $M(t, A)$ to $n(t', A \setminus \{C\})$ and perform a shift of $|var(C) \cap (var(\chi(t')) \setminus var(A \setminus \{C\}))|$ bits to the right. Hence, we can compute M_t in time $\mathcal{O}(2^q(ql + \delta)) \subseteq \mathcal{O}(2^k(kl + \delta))$.

(iii) Let t be a *forget node* with child t'. We compute the table M_t from table $M_{t'}$ according to Lemma 8 as follows: For each of the 2^q choices of A in table M_t, we search for the rows $M_{t'}(A)$ and $M_{t'}(A \cup \{C\})$, which can be accomplished in time $\mathcal{O}(q)$. We set the last entry of row $M_t(A)$ to $n(t', A) - n(t', A \cup \{C\})$. Hence, we can compute M_t in time $\mathcal{O}(2^q(q + \delta)) \subseteq \mathcal{O}(2^k(kl + \delta))$.

(iv) Let t be a *leaf node*. We compute the table M_t according to Lemma 9 as follows: For each of the 2^q choices of A in table M_t, we check whether A is falsifiable and, if so, we compute the cardinality of $var(\chi(t)) \setminus var(A)$, which can be accomplished in time $\mathcal{O}(ql)$. If A is not falsifiable, we set the last entry of row $M(t, A)$ to 0; otherwise, we set the last entry of row $M(t, A)$ to 1 and perform a shift of $|var(\chi(t)) \setminus var(A)|$ bits to the left. Hence, we can compute M_t in time $\mathcal{O}(2^q(ql + \delta)) \subseteq \mathcal{O}(2^k(kl + \delta))$. □

Theorem 2. *Given a nice tree decomposition of the dual graph of a CNF formula F, we can compute $\#(F)$ in time $\mathcal{O}(2^k(kl + \delta)\, N)$; l denotes the size of a largest clause, k denotes the width and N the number of nodes of the tree decomposition.*

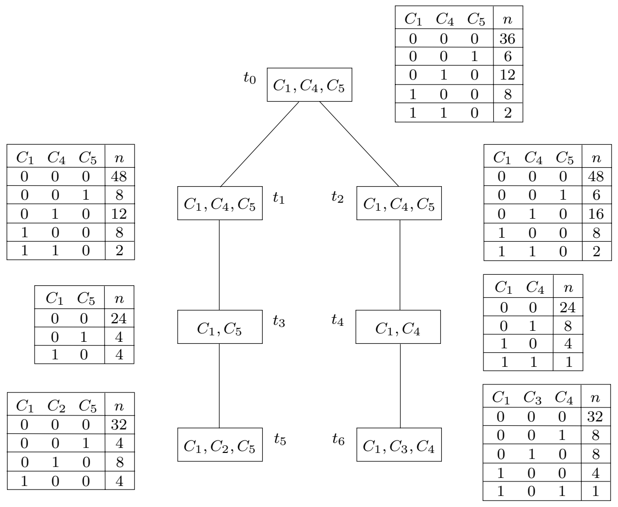
Proof. Let (T, χ, r) be a nice tree decomposition of the dual graph of F; let k and N be the width and number of nodes of (T, χ, r) respectively. Starting from the leaf nodes of T we compute the tables M_t for all nodes t of T in a bottom-up ordering. Each table can be computed by Lemma 10 in time $\mathcal{O}(2^k(kl + \delta))$. Now we show how to compute $\#(F)$ from table M_r at the root r. To this aim, recall that $n(r, A)$ is the number of truth assignments $\tau : var(F) \to \{0, 1\}$ such that every clause in A is falsified by τ and every clause in $F \setminus \chi(r)$ is satisfied by τ. Thus, by the inclusion-exclusion principle, it follows immediately that we can compute the number of models of F from the entries of M_r in the following way:

$$\#(F) = \sum_{i=0}^{|\chi(r)|} \big((-1)^i \sum_{A \subseteq \chi(r),\, |A|=i} n(r, A)\big)$$

We can do this by going through all at most 2^{k+1} choices of $A \subseteq \chi(r)$: Starting with an initial value of 0, we add or subtract $n(r, A)$, depending on whether the cardinality of A is even or odd. This can be done in time $\mathcal{O}(2^k(k + \delta))$. □

An example of this algorithm on the tree decomposition of the dual graph in Figure 1 is shown in Figure 3. Note that, for simplicity, we have omitted those rows from the tables where $n(t, A) = 0$. From table M_{t_0} we can read off that there are exactly $36 - 6 - 12 - 8 + 2 = 12$ models of the corresponding CNF formula.

Note that, in contrast to join-tree algorithms for constraint satisfaction, edge removals from the dual graph based on the *running intersection property* [11] can yield incorrect



C_1	C_4	C_5	n
0	0	0	36
0	0	1	6
0	1	0	12
1	0	0	8
1	1	0	2

C_1	C_4	C_5	n
0	0	0	48
0	0	1	8
0	1	0	12
1	0	0	8
1	1	0	2

C_1	C_4	C_5	n
0	0	0	48
0	0	1	6
0	1	0	16
1	0	0	8
1	1	0	2

C_1	C_5	n
0	0	24
0	1	4
1	0	4

C_1	C_4	n
0	0	24
0	1	8
1	0	4
1	1	1

C_1	C_2	C_5	n
0	0	0	32
0	0	1	4
0	1	0	8
1	0	0	4

C_1	C_3	C_4	n
0	0	0	32
0	0	1	8
0	1	0	8
1	0	0	4
1	0	1	1

Figure 3: Solving #SAT on a nice tree decomposition of the dual graph

results in our setting. For example, in the CNF formula $\{C_1, C_2, C_3\}$, where $C_1 = \{\neg x, y\}$, $C_2 = \{\neg y, z\}$, and $C_3 = \{\neg y, \neg z\}$, we cannot remove the edge between C_1 and C_3 in the dual graph, since we would loose the information that C_1 and C_3 have a variable in common. In that case, however, our algorithm would not be able to detect that C_1 and C_3 contain complementary literals and ignoring this relation would give us a wrong result.

3.3. Incidence Treewidth

For this section, let (T, χ, r) be a nice tree decomposition of the incidence graph $G^*(F)$ of a CNF formula F. Let k denote the width of (T, χ, r).

For each node t of T, let F_t denote the set consisting of all the clauses in V_t, and let X_t denote the set of all variables in V_t, i.e., $F_t = V_t \cap F$ and $X_t = V_t \cap var(F)$. We also use the shorthands $\chi_c(t) = \chi(t) \cap F$ and $\chi_v(t) = \chi(t) \cap var(F)$ for the set of variables and the set of clauses in $\chi(t)$, respectively.

Let t be a node of T. For each truth assignment $\alpha : \chi_v(t) \to \{0, 1\}$ and subset $A \subseteq \chi_c(t)$ we define $N(t, \alpha, A)$ as the set of truth assignments $\tau : X_t \to \{0, 1\}$ for which the following two conditions hold:

1. $\tau(x) = \alpha(x)$ for all variables $x \in \chi_v(t)$.
2. A is exactly the set of clauses in F_t that are not satisfied by τ.

We represent the values of $n(t, \alpha, A) = |N(t, \alpha, A)|$ for all $\alpha : \chi_v(t) \to \{0, 1\}$ and $A \subseteq \chi_c(t)$ by a table M_t with $|\chi(t)| + 1$ columns and $2^{|\chi(t)|}$ rows. The first $|\chi(t)|$ columns of M_t contain Boolean values encoding $\alpha(x)$ for variables $x \in \chi_v(t)$, and membership of C in A for clauses $C \in \chi_c(t)$. The last entry of each row contains the integer $n(t, \alpha, A)$.

Lemma 11. *Let t be a* join node *of T with children t_1, t_2. Then, for each truth assignment $\alpha : \chi_v(t) \to \{0, 1\}$ and set $A \subseteq \chi_c(t)$, we have*

$$n(t, \alpha, A) \quad = \sum_{A_1, A_2 \subseteq \chi_c(t),\, A_1 \cap A_2 = A} n(t_1, \alpha, A_1) \cdot n(t_2, \alpha, A_2).$$

Proof. In the following, we will show that the mapping $f : \tau \mapsto (\tau|_{X_{t_1}}, \tau|_{X_{t_2}})$ is a bijection from the set $N(t, \alpha, A)$ into the set $M = \{(\tau_1, \tau_2) \mid \text{there exists } A_1 \subseteq \chi_c(t_1) \text{ and } A_2 \subseteq \chi_c(t_2) \text{ with } A_1 \cap A_2 = A \text{ such that } \tau_1 \in N(t_1, \alpha, A_1) \text{ and } \tau_2 \in N(t_2, \alpha, A_2)\}$. The above equality follows then immediately.

First, let us show that f is a mapping from $N(t, \alpha, A)$ into M. To this aim, let $\tau \in N(t, \alpha, A)$ and $f(\tau) = (\tau_1, \tau_2)$. Now let A_1 and A_2 be exactly the sets of clauses of F_{t_1} and F_{t_2} that are not satisfied by τ_1 and τ_2 respectively. Since $X_t = X_{t_1} \cup X_{t_2}$, we know that a clause is satisfied by τ if and only if it is satisfied by τ_1 or τ_2. Thus, since $F_t = F_{t_1} \cup F_{t_2}$, we have $A \subseteq A_1 \cap A_2$ and $A_1 \cap A_2 \subseteq A$, that is, $A_1 \cap A_2 = A$. In addition, we know that $A_1 \subseteq \chi_c(t_1)$. For the sake of contradiction, let us assume that there exists a clause $C \in F_{t_1} \setminus \chi_c(t_1) \subseteq F_t$ that is not satisfied by τ_1. If C is not satisfied by τ, then $C \in A \subseteq \chi_c(t) = \chi_c(t_1)$, which contradicts our assumption. Otherwise, if C is satisfied by τ, then C must also be satisfied by τ_2, since it is not satisfied by τ_1. Thus, there exists a variable $x \in X_{t_2}$ that occurs also in C and satisfies C under the truth assignment $\tau_2(x) = \tau(x)$. By the definition of an incidence graph and a tree decomposition, however, this implies that $x \in X_{t_1}$. So we have $x \in X_{t_1} \cap X_{t_2} = \chi_v(t) = \chi_v(t_1)$, which implies that $\tau(x) = \tau_1(x)$. Thus, C is satisfied by τ_1, which again contradicts our assumption. So we have $A_1 \subseteq \chi_c(t_1)$. It is now easy to see that $\tau_1 \in N(t_1, \alpha, A_1)$. The case of τ_2 is completely symmetric. Consequently, f is indeed a mapping from $N(t, \alpha, A)$ into M.

To show that f is injective, let $\tau, \sigma \in N(t, \alpha, A)$ such that $f(\tau) = f(\sigma)$. Then, since $\tau|_{X_{t_1}} = \sigma|_{X_{t_1}}$ and $\tau|_{X_{t_2}} = \sigma|_{X_{t_2}}$, we know that $\tau = \sigma$. To show that f is surjective, let $(\tau_1, \tau_2) \in M$. Now let us define the truth assignment $\tau : X_t \to \{0, 1\}$ by $\tau|_{X_{t_1}} = \tau_1$ and $\tau|_{X_{t_2}} = \tau_2$. Since $X_t = X_{t_1} \cup X_{t_2}$, we know that a clause is satisfied by τ if and only if it is satisfied by τ_1 or τ_2. Thus, we know that $A = A_1 \cap A_2$ is exactly the set of clauses of $F_t = F_{t_1} \cup F_{t_2}$ that are not satisfied by τ. It is now easy to see that $\tau \in N(t, \alpha, A)$. Consequently, f is indeed a bijection from $N(t, \alpha, A)$ into M. □

Lemma 12. *Let t be an* introduce node *with child t'.*
(a) *If $\chi(t) = \chi(t') \cup \{x\}$ for a variable x, then, for each truth assignment $\alpha : \chi_v(t') \to \{0, 1\}$ and set $A \subseteq \chi_c(t)$, we have*

$$n(t, \alpha \cup \{(x, 0)\}, A) = \begin{cases} 0 & \text{if } \neg x \in C \text{ for some clause } C \in A; \\ \sum_{B' \subseteq B} n(t', \alpha, A \cup B') & \text{otherwise, where } B = \{C \in \chi_c(t) \mid \neg x \in C\}; \end{cases}$$

$$n(t, \alpha \cup \{(x, 1)\}, A) = \begin{cases} 0 & \text{if } x \in C \text{ for some clause } C \in A; \\ \sum_{B' \subseteq B} n(t', \alpha, A \cup B') & \text{otherwise, where } B = \{C \in \chi_c(t) \mid x \in C\}. \end{cases}$$

(b) *If $\chi(t) = \chi(t') \cup \{C\}$ for a clause C, then, for each truth assignment $\alpha : \chi_v(t) \to \{0, 1\}$ and set $A \subseteq \chi_c(t)$, we have*

$$n(t, \alpha, A) = \begin{cases} n(t', \alpha, A) & \text{if } C \notin A \text{ and } \alpha \text{ satisfies } C; \\ n(t', \alpha, A \setminus \{C\}) & \text{if } C \in A \text{ and } \alpha \text{ does not satisfy } C; \\ 0 & \text{otherwise.} \end{cases}$$

Proof. *(a)* Let us consider the case of $N(t, \alpha \cup \{(x, 0)\}, A)$; the case of $N(t, \alpha \cup \{(x, 1)\}, A)$ is completely symmetric. Note that, by definition, $N(t, \alpha \cup \{(x, 0)\}, A) = \emptyset$ if there is some clause C in A such that C contains $\neg x$. Thus, let us assume that no clause in A contains $\neg x$. Moreover, let $B = \{C \in \chi_c(t) \mid \neg x \in C\}$. In the following, we will show that the mapping $f : \tau \mapsto \tau|_{X_{t'}}$ is a bijection from the set $N(t, \alpha \cup \{(x, 0)\}, A)$ into the set $\bigcup_{B' \subseteq B} N(t', \alpha, A \cup B')$. Note that always $N(t', \alpha, A \cup B') \cap N(t', \alpha, A \cup B'') = \emptyset$ for $B' \neq B''$. The above equality follows then immediately.

For any $\tau \in N(t, \alpha \cup \{(x, 0)\}, A)$, let $f(\tau) = \tau'$. It is then easy to see that $\tau' \in N(t', \alpha, A \cup B')$ for some $B' \subseteq B$. To show that f is injective, let $\tau, \sigma \in N(t, \alpha \cup \{(x, 0)\}, A)$ such that $f(\tau) = f(\sigma)$. Then, since $\tau|_{X_{t'}} = \sigma|_{X_{t'}}$ and $\tau(x) = \sigma(x) = 0$ for the single variable $x \in X_t \setminus X_{t'}$, we know that $\tau = \sigma$. To show that f is surjective, let $\tau' \in N(t', \alpha, A \cup B')$ for some $B' \subseteq B$. Now we define the truth assignment $\tau : X_t \to \{0, 1\}$ by $\tau|_{X_{t'}} = \tau'$ and $\tau(x) = 0$. It is then easy to see that $\tau \in N(t, \alpha \cup \{(x, 0)\}, A)$. Consequently, f is indeed a bijection from $N(t, \alpha \cup \{(x, 0)\}, A)$ into $\bigcup_{B' \subseteq B} N(t', \alpha, A \cup B')$.

(b) Note that, by definition, $N(t, \alpha, A) = \emptyset$ if $C \in A$ and α satisfies C or $C \notin A$ and α does not satisfy C for the single clause $C \in \chi_c(t) \setminus \chi_c(t')$. Thus, let us assume that (i) $C \notin A$ and α satisfies C or (ii) $C \in A$ and α does not satisfy C. In the following, we will show that the mapping $f : \tau \mapsto \tau$ is a bijection from the set $N(t, \alpha, A)$ into the set $N(t', \alpha, A \setminus \{C\})$. The above equalities follow then immediately.

For any $\tau \in N(t, \alpha, A)$, it is easy to see that $f(\tau) = \tau \in N(t', \alpha, A \setminus \{C\})$. Moreover, since $f(\tau) = \tau$, it follows trivially that f is injective. To show that f is surjective, let $\tau \in N(t', \alpha, A \setminus \{C\})$. Under the assumption of case (i) resp. case (ii), it is then easy to see that $\tau \in N(t, \alpha, A)$. Consequently, f is indeed a bijection from $N(t, \alpha, A)$ into $N(t', \alpha, A \setminus \{C\})$. □

Lemma 13. *Let t be a* forget node *with child t'.*
(a) *If $\chi(t) = \chi(t') \setminus \{x\}$ for a variable x, then, for each truth assignment $\alpha : \chi_v(t) \to \{0, 1\}$ and set $A \subseteq \chi_c(t)$, we have*

$$n(t, \alpha, A) \quad = \quad n(t', \alpha \cup \{(x, 0)\}, A) + n(t', \alpha \cup \{(x, 1)\}, A).$$

(b) *If $\chi(t) = \chi(t') \setminus \{C\}$ for a clause C, then, for each truth assignment $\alpha : \chi_v(t) \to \{0, 1\}$ and set $A \subseteq \chi_c(t)$, we have*

$$n(t, \alpha, A) \quad = \quad n(t', \alpha, A).$$

Proof. It is easy to see that $N(t, \alpha, A) = N(t', \alpha \cup \{(x, 0)\}, A) \cup N(t', \alpha \cup \{(x, 1)\}, A)$ in case *(a)* and that $N(t, \alpha, A) = N(t', \alpha, A)$ in case *(b)*. The above equalities follow then immediately. □

Lemma 14. *Let t be a* leaf node. *Then, for each truth assignment $\alpha : \chi_v(t) \to \{0, 1\}$ and set $A \subseteq \chi_c(t)$, we have*

$$n(t, \alpha, A) \quad = \quad \begin{cases} 1 & \text{if } A = \{\, C \in \chi_c(t) \mid \alpha \text{ does not satisfy } C \,\}; \\ 0 & \text{otherwise.} \end{cases}$$

Proof. Since $X_t = \chi_v(t)$ and $F_t = \chi_c(t)$ for every leaf node t, we know that for each truth assignment $\tau : X_t \to \{0, 1\}$ there exists exactly one truth assignment $\alpha : \chi_v(t) \to \{0, 1\}$ (and vice versa) such that $\tau(x) = \alpha(x)$ for all variables $x \in X_t$. Hence, the above equality follows immediately. □

By using these equalities, we can now construct the tables M_t from the leaves to the root according to the following lemma.

Lemma 15. *Let t be a node of T. Given the tables of the children of t, we can compute the table M_t in time $\mathcal{O}(2^k(kl + 2^k(k + \delta)))$, where l is the size of a largest clause.*

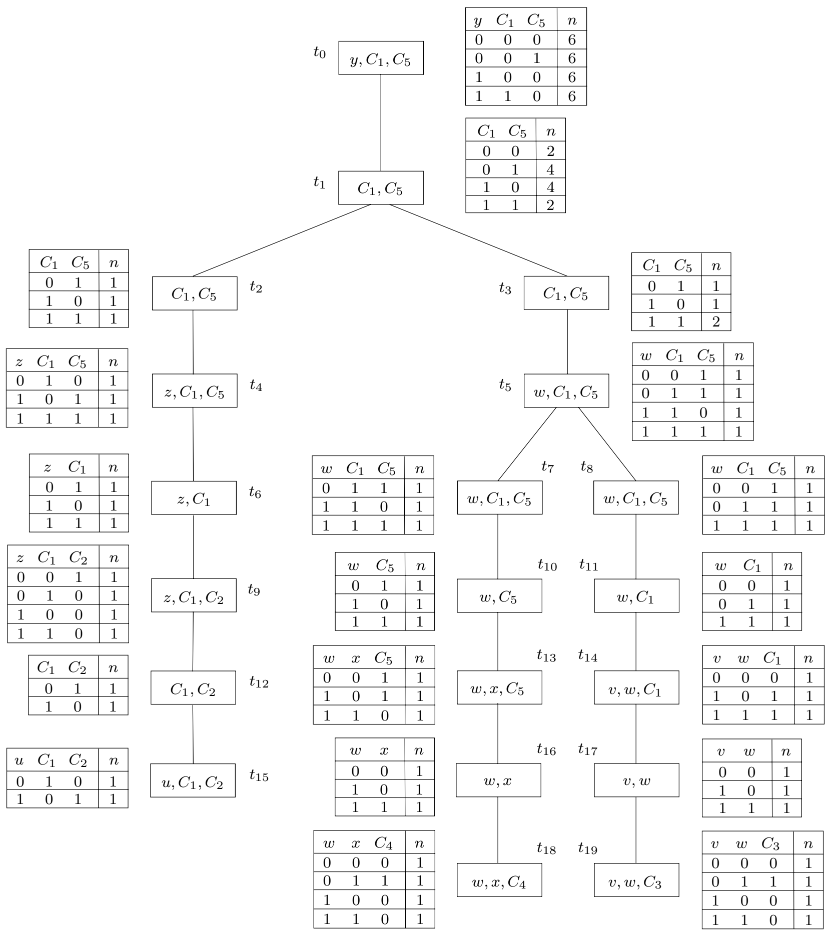
PROOF. To check the runtime of computing M_t, let $p = |\chi_v(t)|$ and $q = |\chi_c(t)|$; since we assume that the width of the tree decomposition under consideration is k, we have $p + q \leq k + 1$. Now let us distinguish between the different kinds of nodes.

(i) Let t be a *join node* with children t_1, t_2. We compute the table M_t from the tables M_{t_1} and M_{t_2} according to Lemma 11 as follows: First we initialize the last entry of all 2^{p+q} rows of table M_t with 0. For each of the 2^p choices of α in table M_t, we go simultaneously through the corresponding rows in the tables M_{t_1} and M_{t_2}. In each step, we consider all 2^q possibilities for A_1 in table M_{t_1} and all 2^q possibilities for A_2 in table M_{t_2}; we compute $A_1 \cap A_2$ and search for row $M_t(\alpha, A_1 \cap A_2)$, which can be accomplished in time $\mathcal{O}(q)$. We add $n(t_1, \alpha, A_1) \cdot n(t_2, \alpha, A_2)$ to the last entry of row $M_t(\alpha, A_1 \cap A_2)$. Hence, we can compute M_t in time $\mathcal{O}(2^p 2^q 2^q (q + \delta)) \subseteq \mathcal{O}(2^k(kl + 2^k(k + \delta)))$.

(ii) Let t be an *introduce node* with child t'. We compute the table M_t from table $M_{t'}$ according to Lemma 12 as follows: For each of the 2^p choices of α in table M_t, we consider all 2^q possibilities for A in table M_t. In case (a), we assume that the last entry of all 2^{p+q} rows of table M_t have been initialized with 0. Now we check whether $\alpha(x) = 0$ and, if so, whether for all $C \in A$ it holds that $\neg x \notin C$, which can be accomplished in time $\mathcal{O}(ql)$. If this is the case, we search for every row $M_{t'}(\alpha|_{X_{t'}}, A \cup B')$ with $B' \subseteq \{C \in \chi_c(t) \mid \neg x \in C\}$, which can be accomplished in time $\mathcal{O}(2^q(p + q))$. We add $n(t', \alpha|_{X_{t'}}, A \cup B')$ to the last entry of row $M_t(\alpha, A)$. The case $\alpha(x) = 1$ is completely symmetric. Hence, we can compute M_t in time $\mathcal{O}(2^p 2^q (ql + 2^q(p + q + \delta))) \subseteq \mathcal{O}(2^k(kl + 2^k(k + \delta)))$. In case (b), we set the last entry of row $M_t(\alpha, A)$ to 0 if either $C \in A$ and α satisfies C or $C \notin A$ and α does not satisfy C, which can be accomplished in time $\mathcal{O}(p + l)$. Otherwise, we search for row $M_{t'}(\alpha, A \setminus \{C\})$ and set the last entry of row $M_t(\alpha, A)$ to $n(t', \alpha, A \setminus \{C\})$, which can be accomplished in time $\mathcal{O}(p + q + \delta)$. Hence, we can compute M_t in time $\mathcal{O}(2^p 2^q (l + p + q + \delta)) \subseteq \mathcal{O}(2^k(kl + 2^k(k + \delta)))$.

(iii) Let t be a *forget node* with child t'. We compute the table M_t from table $M_{t'}$ according to Lemma 13 as follows: For each of the 2^p choices of α in table M_t, we consider all 2^q possibilities for A in table M_t. In case (a), we search for the rows $M_{t'}(\alpha \cup \{(x, 0)\}, A)$ and $M_{t'}(\alpha \cup \{(x, 1)\}, A)$, which can be accomplished in time $\mathcal{O}(p + q)$. We set the last entry of row $M_t(\alpha, A)$ to $n(t', \alpha \cup \{(x, 0)\}, A) + n(t', \alpha \cup \{(x, 1)\}, A)$. Hence, we can compute M_t in time $\mathcal{O}(2^p 2^q (p + q + \delta)) \subseteq \mathcal{O}(2^k(kl + 2^k(k + \delta)))$. In case (b), we search for row $M_{t'}(\alpha, A)$ and set the last entry of row $M_t(\alpha, A)$ to $n(t', \alpha, A)$, which can be accomplished in time $\mathcal{O}(p + q + \delta)$. Hence, we can compute M_t in time $\mathcal{O}(2^p 2^q (p + q + \delta)) \subseteq \mathcal{O}(2^k(kl + 2^k(k + \delta)))$.

(iv) Let t be a *leaf node*. We compute the table M_t according to Lemma 14 as follows: For each of the 2^p choices of α in table M_t, we consider all 2^q possibilities for A in table M_t. We set the last entry of row $M_t(\alpha, A)$ to 1 if $A = \{C \in \chi_c(t) \mid \alpha \text{ does not satisfy } C\}$ and to 0 otherwise, which can be accomplished in time $\mathcal{O}(p + ql)$. Hence, we can compute table M_t in time $\mathcal{O}(2^p 2^q (p + ql)) \subseteq \mathcal{O}(2^k(kl + 2^k(k + \delta)))$. ☐



y	C_1	C_5	n
0	0	0	6
0	0	1	6
1	0	0	6
1	1	0	6

C_1	C_5	n
0	0	2
0	1	4
1	0	4
1	1	2

C_1	C_5	n
0	1	1
1	0	1
1	1	1

C_1	C_5	n
0	1	1
1	0	1
1	1	2

z	C_1	C_5	n
0	1	0	1
1	0	1	1
1	1	1	1

w	C_1	C_5	n
0	0	1	1
0	1	1	1
1	1	0	1
1	1	1	1

z	C_1	n
0	1	1
1	0	1
1	1	1

w	C_1	C_5	n
0	1	1	1
1	1	0	1
1	1	1	1

w	C_1	C_5	n
0	0	1	1
0	1	1	1
1	1	1	1

z	C_1	C_2	n
0	0	1	1
0	1	0	1
1	0	0	1
1	1	0	1

w	C_5	n
0	1	1
1	0	1
1	1	1

w	C_1	n
0	0	1
0	1	1
1	1	1

C_1	C_2	n
0	1	1
1	0	1

w	x	C_5	n
0	0	1	1
1	0	1	1
1	1	0	1

v	w	C_1	n
0	0	0	1
1	0	1	1
1	1	1	1

u	C_1	C_2	n
0	1	0	1
1	0	1	1

w	x	n
0	0	1
1	0	1
1	1	1

v	w	n
0	0	1
1	0	1
1	1	1

w	x	C_4	n
0	0	0	1
0	1	1	1
1	0	0	1
1	1	0	1

v	w	C_3	n
0	0	0	1
0	1	1	1
1	0	0	1
1	1	0	1

Figure 4: Solving #SAT on a nice tree decomposition of the incidence graph

Theorem 3. *Given a nice tree decomposition of the incidence graph of a CNF formula F, we can compute $\#(F)$ in time $\mathcal{O}(2^k(kl + 2^k(k + \delta))\, N)$; l denotes the size of a largest clause, k denotes the width and N the number of nodes of the tree decomposition.*

Proof. Let (T, χ, r) be a nice tree decomposition of the incidence graph of F; let k and n be the width and number of nodes of (T, χ, r) respectively. Starting from the leaf nodes of T we compute the tables M_t for all nodes t of T in a bottom-up ordering. Each table can be computed by Lemma 15 in time $\mathcal{O}(2^k(kl + 2^k(k + \delta)))$. Since we have

$$\#(F) = \sum_{\alpha:\chi_v(r)\to\{0,1\}} n(r, \alpha, \emptyset),$$

we can read off $\#(F)$ from the table M_r at the root r. □

An example of this algorithm on the tree decomposition of the incidence graph in

Figure 1 is shown in Figure 4. Note that, for simplicity, we have omitted those rows from the tables where $n(t, \alpha, A) = 0$. From table M_{t_0} we can read off that there are exactly $6 + 6 = 12$ models of the corresponding CNF formula.

3.4. Space Requirements

When we perform dynamic programming on a nice tree decomposition we traverse the nodes of the tree in an arbitrary bottom-up ordering. When we compute the table of a node we can assume that the tables at its children are already computed and are currently kept in memory. Once the table of a node is computed, the tables of its children can be discarded. Thus, at some point, when the table of a node is computed, all tables of its children are simultaneously in memory; we will refer to this scheme of table computation as the "simultaneous updating scheme."

A variant of this scheme was considered by Aspvall, Proskurowski, and Telle [2], not requiring that the tables of children of a node are present simultaneously; the parent table is updated whenever a child table becomes available. We will refer to the scheme of Aspvall et al. as the "sequential updating scheme." In view of the updating functions for join nodes as defined in Lemmas 1, 6, and 11, respectively, one can use the sequential updating scheme for the primal and dual treewidth algorithms. The incidence treewidth algorithm, however, requires the simultaneous updating scheme.

The following algorithm carries out the simultaneous updating scheme on a nice tree decomposition (T, χ, r); the algorithm also computes for every node t the number $\rho(t)$ of tables required simultaneously to compute the table M_t. The algorithm is recursive, initially $t = r$.

1. Clearly $\rho(t) = 1$ if t is a leaf; M_t can be computed independently.
2. If t has only one child t', then recurse on the subtree $T_{t'}$ rooted at t' and compute the table $M_{t'}$ and the number $\rho(t')$. Now discard all tables of nodes below t' and compute the table M_t; then discard $M_{t'}$. This gives $\rho(t) = \max(2, \rho(t'))$.
3. If t has two children t' and t'', then compute $\rho(t')$ and $\rho(t'')$; w.l.o.g., assume $\rho(t') \geq \rho(t'')$. First recurse on $T_{t'}$ and compute the table $M_{t'}$; discard all tables below t' and keep $M_{t'}$ in memory. Next recurse on $T_{t''}$ to compute the table $M_{t''}$; discard all tables below t'' and keep $M_{t''}$ in memory. Now compute the table M_t using the tables $M_{t'}$ and $M_{t''}$; afterwards discard the tables $M_{t'}$ and $M_{t''}$. This gives $\rho(t) = \max(3, \rho(t'), \rho(t'') + 1)$.

Note that if the tree T is binary, then $\rho(r)$ is known as the Horton-Strahler number of T [32]. Aspvall et al. [2] show that if T has N nodes than the sequential updating scheme requires not more than $\lfloor \log_2 \frac{4}{3}(N + 1) \rfloor$ tables at any point of the computation. We use a similar reasoning to bound the space required by the simultaneous updating scheme (this holds in particular when the algorithm outlined above is applied).

Proposition 3. *The simultaneous updating scheme applied to a nice tree decomposition with N nodes requires not more than $\lfloor 1 + \log_2(N + 1) \rfloor$ tables at any point of the computation.*

Proof. We write $\rho(T) = \rho(r)$ if r is the root of tree T (thus $\rho(T)$ is the number of tables required by the simultaneous updating scheme on T). Let $\mathcal{C}$ denote the class of rooted trees where each node has at most two children, and let $\mathcal{C}_i = \{\, T \in \mathcal{C} : \rho(T) = i \,\}$.

We construct a sequence $T_1, T_2, \ldots$ of trees with $T_i \in \mathcal{C}_i$, where T_i belongs to $\mathcal{C}_i$ and has the smallest number of nodes among all trees in $\mathcal{C}_i$. For T_1 we clearly have to take the trivial one-node tree; for T_2 we take the tree consisting of the root and one leaf; T_3 has three nodes, the root and two leaves. For $i > 3$, we construct T_i by putting together a root r and two disjoint copies of T_{i-1} with their roots as the children of r. In view of case (3) above, it follows that indeed $\rho(T_i) = i$ and $|V(T_i)|$ is minimal. We have $|V(T_i)| = 2^{i-1} - 1$; taking logarithms yields $i = \lfloor 1 + \log_2(|V(T_i)| + 1) \rfloor$. $\square$

Thus the simultaneous updating scheme requires at most one more table than the sequential one. This result suggests the use of the simultaneous updating scheme for all three algorithms as it is slightly more convenient to implement without requiring significantly more space.

Acknowledgments

Research supported by the EPSRC, project EP/E001394/1. Material from the preliminary version [28] is reused with kind permission of Springer Science and Business Media.

References

[1] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms*, Chapter 1: Models of Computation, pages 1–41. Addison-Wesley, 1974.

[2] B. Aspvall, A. Proskurowski, and J. A. Telle. Memory requirements for table computations in partial k-tree algorithms. *Algorithmica*, 27(3-4):382–394, 2000.

[3] F. Bacchus, S. Dalmao, and T. Pitassi. Algorithms and complexity results for #SAT and Bayesian inference. In *Proc. of the 44th Annual IEEE Symposium on Foundations of Computer Science (FOCS'03)*, pages 340–351. IEEE Computer Society, 2003.

[4] H. L. Bodlaender. A tourist guide through treewidth. *Acta Cybernetica*, 11:1–21, 1993.

[5] H. L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal on Computing*, 25(6):1305–1317, 1996.

[6] H. L. Bodlaender. Discovering treewidth. In *Proc. of the 31st Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM'05)*, volume 3381 of LNCS, pages 1–16. Springer-Verlag, 2005.

[7] D. Cohen, P. Jeavons, and M. Gyssens. A unified theory of structural tractability for constraint satisfaction problems. *Journal of Computer and System Sciences*, 74(5):721–743, 2008.

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

[9] B. Courcelle, J. A. Makowsky, and U. Rotics. On the fixed parameter complexity of graph enumeration problems definable in monadic second-order logic. *Discrete Applied Mathematics*, 108(1-2):23–52, 2001.

[10] N. Creignou and M. Hermann. Complexity of generalized satisfiability counting problems. *Information and Computation*, 125(16):1–12, 1996.

[11] R. Dechter. *Constraint Processing*. Morgan Kaufmann, 2003.

[12] R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer-Verlag, 1999.

[13] E. Fischer, J. A. Makowsky, and E. R. Ravve. Counting truth assignments of formulas of bounded tree-width or clique-width. *Discrete Applied Mathematics*, 156(4):511–529, 2008.

[14] J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer-Verlag, 2006.

[15] M. Fürer. Faster integer multiplication. In *Proc. of the 39th Annual ACM Symposium on Theory of Computing (STOC'07)*, pages 57–66. ACM, 2007.

[16] G. Gottlob, N. Leone, and F. Scarcello. Hypertree decompositions and tractable queries. *Journal of Computer and System Sciences*, 64(3):579–627, 2002.

[17] G. Gottlob, F. Scarcello, and M. Sideri. Fixed-parameter complexity in AI and nonmonotonic reasoning. *Artificial Intelligence*, 138(1-2):55–86, 2002.

[18] M. Grohe and D. Marx. Constraint solving via fractional edge covers. In *Proc. of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'06)*, pages 289–298. ACM Press, 2006.

[19] P. Hlinený and S.-I. Oum. Finding branch-decompositions and rank-decompositions. *SIAM Journal of Computing*, 38(3):1012–1032, 2008.

[20] T. Kloks. *Treewidth: Computations and Approximations*. Springer-Verlag, 1994.

[21] D. E. Knuth. *The Art of Computer Programming*, volume 2: Seminumerical Algorithms, 3rd edition, chapter 4.3.3 How fast can we multiply?, pages 294–318. Addison-Wesley, 1998.

[22] P. G. Kolaitis and M. Y. Vardi. Conjunctive-query containment and constraint satisfaction. *Journal of Computer and System Sciences*, 61(2):302–332, 2000.

[23] A. M. C. A. Koster, H. L. Bodlaender, and S. P. M. van Hoesel. Treewidth: Computational experiments. *Electronic Notes in Discrete Mathematics*, 8:54–57, 2001. Extended version available as technical report ZIB 01-38, Konrad-Zuse-Zentrum für Informationstechnik Berlin, 2001.

[24] R. Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Oxford University Press, 2006.

[25] N. Nishimura, P. Ragde, and S. Szeider. Solving #SAT using vertex covers. *Acta Informatica*, 44(7-8):509–523, 2007.

[26] N. Robertson and P. D. Seymour. Graph minors X. Obstructions to tree-decomposition. *Journal of Combinatorial Theory, Series B*, 52(2):153–190, 1991.

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

[28] M. Samer and S. Szeider. Algorithms for propositional model counting. In *Proc. of the 14th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR'07)*, volume 4790 of LNCS, pages 484–498. Springer-Verlag, 2007.

[29] M. Samer and S. Szeider. Constraint satisfaction with bounded treewidth revisited. *Journal of Computer and System Sciences*, to appear.

[30] A. Schönhage and V. Strassen. Schnelle Multiplikation ganzer Zahlen. *Computing*, 7:281–292, 1971.

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

[32] X. G. Viennot. Trees. *Mots — Mélanges offerts à M.-P. Schützenberger*, pages 265–297. Hermes Science Publications, 1990.

[33] M. Yannakakis. Algorithms for acyclic database schemes. In *Proc. of the 7th International Conference on Very Large Data Bases (VLDB'81)*, pages 81–94. IEEE Press, 1981.