New Algorithm for Weighted Model Counting using Nice Tree Decomposition

Anton Lykov

May 20, 2019

Abstract

We propose an algorithm to perform the Weighted Model Counting on CNFs, with time complexity being exponential only in the treewidth of the primal graph of the CNF. Our work is a generalization of an existing Model Counting technique.

1 Introduction

A family of 3 algorithms for solving $\#SAT$ was proposed in [3]. These algorithms are bounded by the treewidth of 3 types of trees – primal, dual, and incidence trees. In this write up, we generalize one of these techniques to perform Weighted Model Counting, with the same upper bound. Here, we provide a detailed descritpion of an algorithm based on the primal tree of the CNF. The claimed complexity for the model counting using primal graph in [3] is given by

$$O(2^{k_1} k_1 d N_1)$$

where k_1, N_1 denote the width & number of nodes of the tree decomposition, and d denotes the 'maximum number of occurrences over all variables' [3]. Our time-complexity stays the same asymptotically.

2 Preliminaries

We will breifly go throguh the necessary definitions. First, we will use the term *tree decomposition* in its regular sense. We will also need the *nice tree decomposition*, which is defined as follows. For a graph G, a triple (T, χ, r) is a *nice tree decomposition* if (T, χ) is a tree decomposition, tree T is rooted at node r, and the following three conditions hold [2]:

1. Every node of T has at most 2 children.

2. If a node t of T has 2 children t_1, t_2, then $\chi(t_1) = \chi(t_2) = \chi(t)$ [1]. We call t a *join node* in this case.

3. If a node t has exactly one child t', then exactly one of the following condition holds:

   - $|\chi(t)| = |\chi(t')| + 1$ and $\chi(t') \subset \chi(t)$; in this case we call t an *introduce node*
   - $|\chi(t)| = |\chi(t')| - 1$ and $\chi(t) \subset \chi(t')$; in this case we call t a *forget node*

It is known that we can efficiently transform any tree decomposition of treewidth k and n nodes into a nice tree decomposition of treewidth at most k and at most $4n$ nodes [2].

3 The algorithm

3.1 Description and proofs

Suppose we have a CNF F, its primal graph G, and a nice tree decomposition of G - (T, χ, r). The trick is to use dynamic programming approach and accumulate knowledge about model count 'so far' in the special tables accosiated with each tree node. In particular, they give the following definition.
Pick a node t from a nice tree decomposition. For each truth assignment $\alpha : \chi(t) \to \{0, 1\}$, we define $N(t, \alpha)$ as the set of truth assignments[2] $\tau : V_t \to \{0, 1\}$, 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 falsified by τ.

The paper denotes $|N(t, \alpha)|$ as $n(t, \alpha)$. Thus, for each node t, we can create a table M_t with $|\chi(t)|+1$ columns and $2^{|\chi(t)|}$ rows, each row representing an truth assignment for variables in t.

In our work, we adjust these definitions. In particular, we denote by $n(t, \alpha)$ the sum of weights $W(t)$ of all truth assignments that satisfy the two properties above, *only* considering the variables in V_t. Also, since F is a WBF, we assign the weights to the literals.

We will follow an example throughout the paper. Let's $F = C_1 \wedge C_2 \wedge C_3 \wedge C_4$, where:

- $C_1 = \bar{x} \vee y \vee z$
- $C_2 = \bar{y} \vee \bar{z}$
- $C_3 = y \vee s$
- $C_4 = z \vee p$

[1] For a node t in a tree decomposition of a graph G, $\chi(t)$ denotes the set of vertices in G corresponding to t

[2] V_t denotes the set of vertices of G that appear in the subtree of T rooted at t.

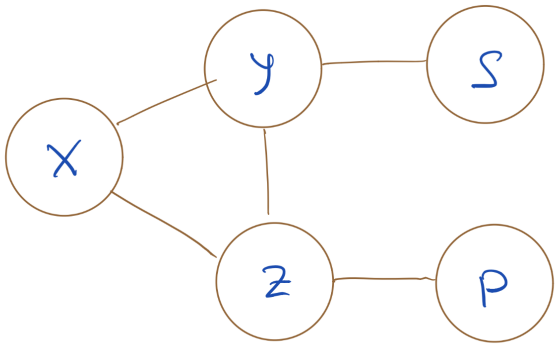

Figure 1: Primal graph of F

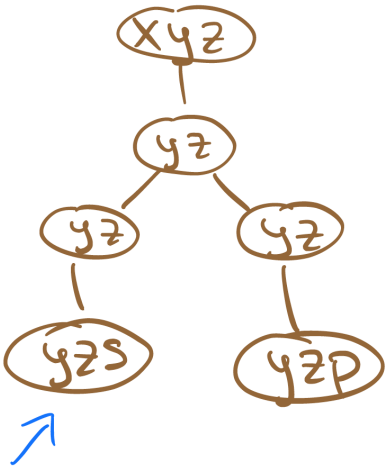

Figure 2: Nice tree decomposition of the primal graph for F

We define the weights as follows:

$$w(x) = 0.1, w(y) = 0.2, w(z) = 0.3, w(s) = 0.4, w(p) = 0.5$$

Correspondingly,

$$w(\bar{x}) = 0.9, w(\bar{y}) = 0.8, w(\bar{z}) = 0.7, w(\bar{s}) = 0.6, w(\bar{p}) = 0.5$$

We will consider a nice tree decomposition for the primal graph of F, as in Figures 1 and 2.

For example, consider the leaf node $A = \{y, z, s\}$ (bottom left). Since it is a leaf node, then for each truth assignment α, $N(A, \alpha)$ is either 0 or 1. For those assignments that do not violate any clauses of F, we set $n(A, \alpha)$ to the product of corresponding weights of variables y, z, s. In our example case, we can draw the table M_A, as in Figure 3.

As we can see, it is easy to fill in the tables M_t for the leaf nodes. Indeed, it takes time exponential in k, where k is the treewidth of our decomposition T. The idea is to propagate

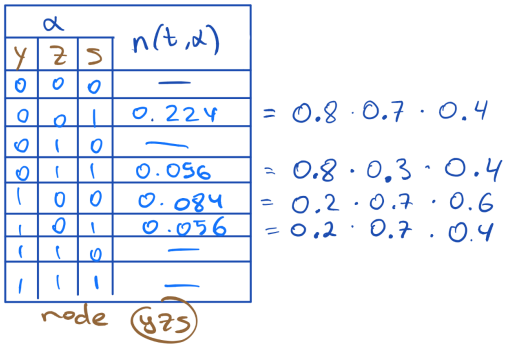

Figure 3: Table for the leaf node {y,z,s}

the knowledge up through this tree, and it turns out this can be done efficiently not only for model counting, but for weighted model counting as well. Once we have filled the table corresponding to the root in T, simply summing up all values $n(r, \alpha)$, for all α, will give us the desired WMC.

To propagate the tree upwards, we prove three lemmas. Namely we show how to fill in the values in the tables of join, forget, and introduce nodes. The first lemma is the most important one.

Lemma 1. *Suppose t is a join node with children t_1 and t_2. Then, for each truth assignment α on vertices in $\chi(t)$, the following holds:*

$$n(t, \alpha) = \frac{n(t_1, \alpha) n(t_2, \alpha)}{\prod_{v \in \chi(t)} w(\alpha(v))}$$

Proof. The main chunk of the proof is done in [3], so we'll not repeat it. The takeaway from the proof is that, 'merging' *any* valid assignment τ_1 from $N(t_1, \alpha)$, *any* valid assignment τ_2 from $N(t_2, \alpha)$ yields a valid assignment $\tau \in N(t, \alpha)$, and all valid assignments $\tau \in N(t, \alpha)$ are obtained precisely this way. Denote $P_{t,\tau} = \prod_{v \in V_t \setminus \chi(t)} w(\tau(v))$.

By definition, for any α[3] :

$$n(t,\alpha) = \sum_{\tau \in N(t,\alpha)} \prod_{v \in V_t} w(\tau(v)) \tag{1}$$

$$= \sum_{\tau \in N(t,\alpha)} \prod_{v \in \chi(t)} w(\tau(v)) P_{t,\tau} \tag{2}$$

$$= \sum_{\tau \in N(t,\alpha)} \prod_{v \in \chi(t)} w(\alpha(v)) P_{t,\tau} \tag{3}$$

$$= \prod_{v \in \chi(t)} w(\alpha(v)) \sum_{\tau \in N(t,\alpha)} P_{t,\tau} \tag{4}$$

Similarly,

$$n(t_1,\alpha) = \prod_{v \in \chi(t_1)} w(\alpha(v)) \sum_{\tau \in N(t_1,\alpha)} P_{t_1,\tau} \tag{5}$$

$$n(t_2,\alpha) = \prod_{v \in \chi(t_2)} w(\alpha(v)) \sum_{\tau \in N(t_2,\alpha)} P_{t_2,\tau} \tag{6}$$

and

$$\frac{n(t_1,\alpha)n(t_2,\alpha)}{\prod_{v \in \chi(t)} w(\alpha(v))} = \prod_{v \in \chi(t_2)} w(\alpha(v)) \sum_{\tau \in N(t_2,\alpha)} P_{t_2,\tau} \sum_{\tau \in N(t_1,\alpha)} P_{t_1,\tau} \tag{7}$$

Now, the key observation is that

$$\sum_{\tau \in N(t,\alpha)} P_{t,\tau} = \sum_{\tau \in N(t_2,\alpha)} P_{t_2,\tau} \sum_{\tau \in N(t_1,\alpha)} P_{t_1,\tau} \tag{8}$$

which, after plugging in (4), completes the proof. □

Lemma 2. *Suppose t is a forget node with child t_1, and v is the 'forgotten' variable. Then, for each truth assignment α:*

$$n(t,\alpha) = n(t_1, \alpha \cup [v \to 0]) + n(t_1, \alpha \cup [v \to 1])$$

The proof is almost exactly the same as in [3], and so we'll not repeat it for the sake of space.

Lemma 3. *Suppose t is an introduce node with child t_1, and v is the 'introduced' variable. Then, for each truth assignment α:*

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

Proof. This lemma is proved exacly as in the original paper. The idea is that introducing a variable should not falsify the 'old' assignment α. If it does not, we multiply our current $n(t_1,\alpha)$ by the weight of the introduced literal. If it does, we cannot consider this τ anymore. □

[3]In the equations below, we overload the definition of α and τ to also mean the corresponding literal over the variable

3.2 Complexity

Note that we only have to compute product of variables in the join nodes as we propagate upwards, and that's the only distinction with the original algorithm in terms of speed. Such overhead is neglectible in our setting, since w can be consulted in constant time; we also have additional multiplication of at most k terms while computing each row of the table for a join node.
Thus we claim that the complexity stays the same, namely linear in number of nodes, and exponential only in the primal treewidth.

4 Example

Refer to an example worked out below, see Figure 4 (next page). We fill the tables for each of the nodes of our nice tree decomposition according to the algorithm above.
After the table of the root is filled, we can sum all $n(t, \alpha)$ entries in this table, and this will preciely be the WMC of F.

5 Future work

This technique looks promising for studying more general objects, not necessarily CNFs. In particular, we will work on applying these methods to WBF compiled using methods in [1]. Also, we will see if similar results for dual graph and incidence graph are possible.
We may also study [2] in more detail to see which nice tree decompositions yield better bounds in terms of constant factors in the funtime.

References

[1] Holtzen, S. Symbolic exact inference for discrete probabilistic programs.

[2] Kloks, T. *Treewidth: computations and approximations*, vol. 842. Springer Science & Business Media, 1994.

[3] Samer, M., and Szeider, S. Journal of discrete algorithms.

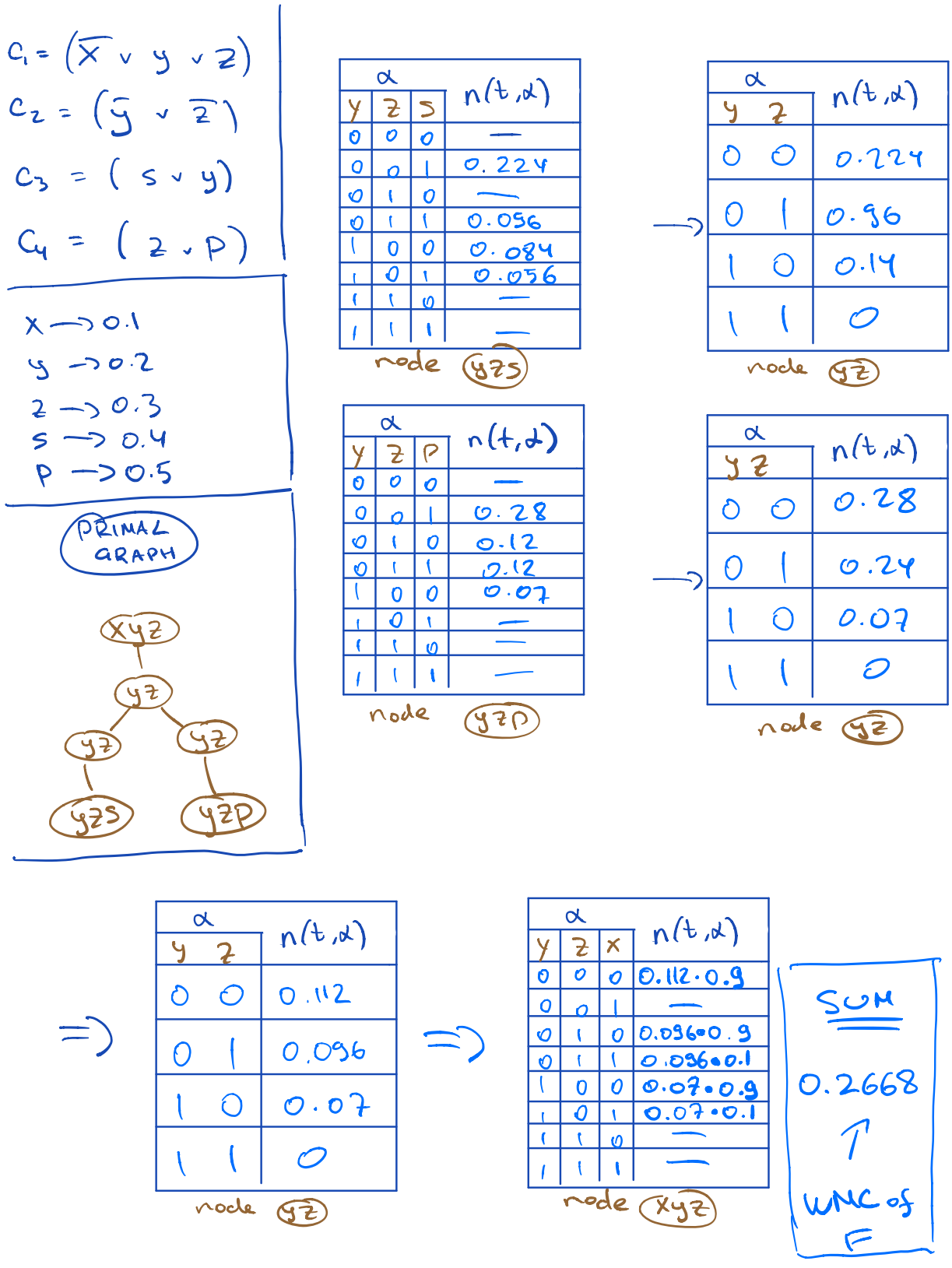

Figure 4: Algorithm written out for a particular F