Integrating Tree Decompositions into Decision Heuristics of Propositional Model Counters

Tuukka Korhonen
HIIT, Department of Computer Science, University of Helsinki, Finland

Matti Järvisalo
HIIT, Department of Computer Science, University of Helsinki, Finland

Abstract

Propositional model counting (#SAT), the problem of determining the number of satisfying assignments of a propositional formula, is the archetypical #P-complete problem with a wide range of applications in AI. In this paper, we show that integrating tree decompositions of low width into the decision heuristics of a reference exact model counter (SharpSAT) significantly improves its runtime performance. In particular, our modifications to SharpSAT (and its derivant GANAK) lift the runtime efficiency of SharpSAT to the extent that it outperforms state-of-the-art exact model counters, including earlier-developed model counters that exploit tree decompositions.

2012 ACM Subject Classification Theory of computation → Constraint and logic programming

Keywords and phrases propositional model counting, decision heuristics, tree decompositions, empirical evaluation

Digital Object Identifier 10.4230/LIPIcs.CP.2021.8

Category Short Paper

Supplementary Material *Software (Source code and experimental data)*: `https://github.com/Laakeri/modelcounting-cp21`
archived at `swh:1:dir:fd1a6eaa9d3ba301b7151f077f51e1da29801ffe`

Funding Work financially supported by Academy of Finland under grants 322869 and 328718.

1 Introduction

Propositional model counting (#SAT), the problem of determining the number of satisfying assignments of a propositional formula, is the archetypical #P-complete problem [34]. Improving the scalability of state-of-the-art model counters is a challenging task, motivated by a wide range of applications in AI, including probabilistic reasoning, planning, quantified information flow analysis, differential cryptanalysis, and model checking [29, 5, 25, 20, 2].

Many current exact model counters rely heavily on search techniques adapted from Boolean satisfiability (SAT) solving and employ component caching to avoid repeatedly counting over the same residual formulas seen during the counting process. In particular, these techniques are applied both by "search-based" exact model counters (such as Cachet, SharpSAT and GANAK [28, 33, 30]) and "compilation-based" counters (such as c2d, minic2d, and D4 [7, 24, 21]) in which the compilation process is based on SAT solver traces. Hence improvements to decision heuristics in the underlying model counters have the promise of speeding up various state-of-the-art model counters.

In this work, we propose and evaluate the effects of integrating information on tree decompositions of CNF formulas to guide the decision heuristics in search-based exact propositional model counters. In theory, it is known that #SAT can be solved in time $poly(|\phi|)2^w$, where $|\phi|$ is the size of the formula and w the width of a given tree decomposition of the primal graph of the formula ϕ. If clause learning is not employed, search-based counters

© Tuukka Korhonen and Matti Järvisalo;
licensed under Creative Commons License CC-BY 4.0
27th International Conference on Principles and Practice of Constraint Programming (CP 2021).
Editor: Laurent D. Michel; Article No. 8; pp. 8:1–8:11
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

achieve this time complexity if they employ component caching and a variable selection algorithm based on the tree decomposition [1, 6, 9]. Tree decompositions have recently been employed in dynamic programming based model counters [12, 15, 17], and recent exact model counters have adapted alternative graph-based techniques, including heuristic graph partitioning algorithms [8, 21, 24] and graph centrality measures [3], for deciding variable orderings and decision heuristics. (For more discussion, see section on Related Work.) However, we are not aware of earlier work on integrating tree decompositions directly as a decision heuristic component in the context of search-based propositional model counters.

In this paper, we show that, in practice, exploiting tree decompositions of low width is easy and effective in speeding up state-of-the-art search-based exact model counters SharpSAT and GANAK on instances with treewidth as high as 150 (or even higher). In particular, motivating the approach through theoretical observations, we describe how to integrate tree decomposition guidance to the decision heuristics of these model counters. We show through extensive empirical evaluation that the tree decomposition guided modifications of SharpSAT and GANAK noticeably outperform other state-of-the-art exact model counters, including the counters themselves in their default settings. Beyond the empirical evidence provided in this paper, we note that our SharpSAT-based model counter `SharpSAT-TD`, implementing the ideas presented in this work, ranked first in tracks 1, 2, and 4 of Model Counting Competition 2021 (see `https://mccompetition.org/`).

2 Preliminaries

We consider the problem of counting the number of satisfying truth assignments (or models) of a conjunctive normal form (CNF) propositional formula, i.e., #SAT. A CNF formula is denoted by ϕ, its variables by $V(\phi)$, clauses by $cls(\phi)$, and variables of a clause c by $V(c)$. The size of a formula ϕ is $|\phi| = |V(\phi)| + |cls(\phi)|$. We denote by $\phi_{|x=1}$ the formula obtained from ϕ by assigning a variable $x \in V(\phi)$ to 1 (true), i.e., the formula ϕ with x removed from the variable set, each clause containing literal x removed, and each occurrence of $\neg x$ in any clause removed. The formula $\phi_{|x=0}$ is defined analogously. The formula obtained by applying unit propagation, i.e., setting $\phi \leftarrow \phi_{|x=0}$ whenever there is a clause $(\neg x)$ and $\phi \leftarrow \phi_{|x=1}$ whenever there is a clause (x), is denoted by UP(ϕ). The number of models of ϕ is $\#(\phi)$. For any variable x it holds that $\#(\phi) = \#(\phi_{|x=0}) + \#(\phi_{|x=1})$. Note also that $\#(\phi) = \#(UP(\phi))$. We denote the union of two formulas ϕ_1 and ϕ_2 with disjoint variable sets by $\phi_1 \sqcup \phi_2$. The fact that $\#(\phi_1 \sqcup \phi_2) = \#(\phi_1) \cdot \#(\phi_2)$ allows for separately counting the number of models in the variable-disjoint formulas ϕ_1 and ϕ_2 to obtain the model count of $\phi_1 \sqcup \phi_2$ [19].

We consider tree decompositions of primal graphs of CNF formulas (aka Gaifman graphs). A graph G has a set of vertices $V(G)$ and a set of edges $E(G)$. For a vertex set $X \subseteq V(G)$ we denote by X^2 the set of all possible edges within X. The primal graph $G(\phi)$ of a formula ϕ is a graph with $V(G(\phi)) = V(\phi)$ and $E(G(\phi)) = \bigcup_{c \in cls(\phi)} V(c)^2$. In words, the vertices of the primal graph are the variables and the edges are created by inducing a clique on the variables of each clause.

► **Example 1.** Consider the CNF formula ϕ with variables $V(\phi) = \{x_1, \ldots, x_6\}$ and clauses $cls(\phi)$ as shown in Figure 1 (left). The primal graph $G(\phi)$ is in Figure 1 (middle). The vertices of $G(\phi)$ are the variables of ϕ and the edges of $G(\phi)$ are defined by the clauses of ϕ. For example, $G(\phi)$ contains the edge $\{x_1, x_2\}$ because ϕ contains the clause $(x_1 \vee \neg x_2 \vee x_5)$.

A tree is a connected graph T with $|E(T)| = |V(T)| - 1$. A tree decomposition [26, 4] of a graph G is a tree T whose each node t corresponds to a bag $T[t] \subseteq V(G)$ containing vertices of G and which satisfies the properties

1. $V(G) \subseteq \bigcup_{t \in V(T)} T[t]$,
2. $E(G) \subseteq \bigcup_{t \in V(T)} T[t]^2$, and
3. for each $v \in V(G)$, the nodes $\{t \in V(T) \mid v \in T[t]\}$ form a connected subtree of T.

The width of a tree decomposition T is $w(T) = \max_{t \in V(T)} |T[t]| - 1$, and the treewidth of a graph G is the minimum width over all tree decompositions of G. We use the convention that one of the nodes of the tree decomposition is chosen as the root of the tree decomposition. The root can be chosen arbitrarily. We denote by $d_T(t)$ the distance from the root to the node t in the tree decomposition T, i.e., the depth of the node t.

▶ **Example 2.** Consider the CNF formula ϕ with variables $V(\phi) = \{x_1, \ldots, x_6\}$ and clauses $cls(\phi)$ as shown in Figure 1 (left). The primal graph $G(\phi)$ is shown in Figure 1 (middle), and a tree decomposition T of $G(\phi)$ in Figure 1 (right). The bags of T are $\{x_2, x_3, x_5\}$, $\{x_1, x_2, x_5\}$, $\{x_3, x_5, x_6\}$, and $\{x_1, x_4\}$. The width of T is 2 because the largest bag has size 3, and thus the treewidth of $G(\phi)$ is at most 2. Let t_1 denote the node of T with the bag $T[t_1] = \{x_2, x_3, x_5\}$ and t_2 the node with the bag $T[t_2] = \{x_1, x_4\}$. If t_1 is the root, then $d_T(t_1) = 0$ and $d_T(t_2) = 2$.

3 Tree Decomposition Guided Model Counting

Consider the basic DPLL-style algorithm with component caching for model counting [1] presented as Algorithm 1, consisting of unit propagation (Line 1), detection of disconnected components (Line 4), component caching (cache check on Line 6, caching on Line 9), and making decisions by selecting and assigning currently unassigned variables (Line 7).

Our focus in this work is on the decision heuristics, i.e., implementation of Line 7. Algorithm 2 specifies the tree decomposition guided variable selection algorithm. By using Algorithm 2 as the variable selection procedure in Algorithm 1, we obtain a DPLL-style tree decomposition guided model counter.

▶ **Example 3.** Consider the run of Algorithm 1 on the formula ϕ of Figure 1 (left) using Algorithm 2 with the tree decomposition T of Figure 1 (right), rooted on the node t_1 with the bag $T[t_1] = \{x_2, x_3, x_5\}$. In the first recursive call, the variable selected is x_2 because it is the lowest index variable in the bag of the root node. Consider a recursive call after variable decisions $x_2 = 1$, $x_3 = 1$ by unit propagation, and $x_5 = 1$. The remaining formula has variables x_1, x_4, and x_6, and only the clause $(x_1 \vee \neg x_4)$. On Line 4 it is partitioned to two formulas, one with variable set $\{x_1, x_4\}$, and one with variable set $\{x_6\}$. On a recursive call on the former formula, the variable x_1 is selected by Algorithm 2, because it is the only variable left in the lowest-depth bag $\{x_1, x_2, x_5\}$ intersecting the variable set of the formula.

The time complexity of Algorithm 1 equipped with Algorithm 2 for variable selection is $poly(|\phi|)2^{w(T)}$, where T is the tree decomposition given as input. This time complexity and similar observations have been already made earlier [1, 6, 9, 27].

$$\begin{aligned}\{&(\neg x_2 \vee x_3),\\ &(x_3 \vee \neg x_6),\\ &(x_5 \vee x_6),\\ &(x_1 \vee \neg x_2 \vee x_5),\\ &(x_1 \vee \neg x_4)\}\end{aligned}$$

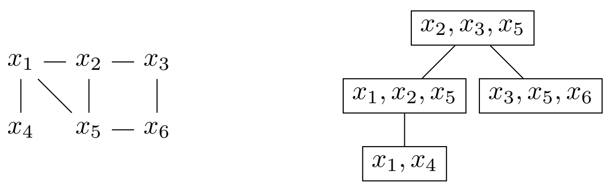

Figure 1 An example formula (left), its primal graph (middle), and one of tree decompositions of the primal graph (right).

Algorithm 1 DPLL-style model counter.

Input : Formula ϕ
Output: The number of satisfying assignments of ϕ

1 $\phi \leftarrow \mathrm{UP}(\phi)$
2 **if** $\emptyset \in cls(\phi)$ **then return** 0
3 **if** $V(\phi) = \emptyset$ **then return** 1
4 **if** $\phi = \phi_1 \sqcup \phi_2$ **then**
5 **return** $\mathrm{Count}(\phi_1) \cdot \mathrm{Count}(\phi_2)$
6 **if** ϕ *in cache* **then return** $\mathrm{cache}[\phi]$
7 $x \leftarrow \mathrm{VariableSelect}(\phi)$
8 $R \leftarrow \mathrm{Count}(\phi_{|x=0}) + \mathrm{Count}(\phi_{|x=1})$
9 $\mathrm{cache}[\phi] \leftarrow R$
10 **return** R

Algorithm 2 Tree decomposition guided variable selection.

Input : Formula ϕ and tree decomposition T of $G(\phi)$
Output: Variable $x \in V(\phi)$

1 $t \leftarrow$ The lowest depth node of T with $|T[t] \cap V(\phi)| \geq 1$
2 **return** The variable in $T[t] \cap V(\phi)$ with the lowest index

▶ **Proposition 4** ([6]). *If Algorithm 1 implements the variable selection of Algorithm 2, then the number of cache entries created during Algorithm 1 is at most* $|V(T)|(w(T)+1)2^{w(T)}$.

Proof. Suppose that the execution of Algorithm 1 is at Line 7. We show that there can be at most $2^{w(T)}$ different formulas ϕ for a fixed node t of T determined on Line 1 of Algorithm 2 and a fixed variable x returned by Algorithm 2. This implies the proposition because there are at most $|V(T)|$ choices for t and at most $(w(T)+1)$ choices for x.

Let p be the parent node of t in T. The formula ϕ can be obtained from the original input formula by assigning all variables in $T[p] \cap T[t]$ and the variables in $T[t]$ with lower index than x, then applying unit propagation, and then selecting the component containing x. There are at most $w(T)$ such variables, so the number of choices is $2^{w(T)}$. ◀

As each recursive call of Algorithm 1 is polynomial-time, time complexity $poly(|\phi|)2^{w(T)}$ follows from Proposition 4. Although Proposition 4 does not necessarily hold when equipping Algorithm 1 with clause learning, we will show that tree decomposition guidance provides significant performance improvements in practice also when clause learning is employed.

4 Integrating Tree Decompositions into Model Counters

In SharpSAT [33] and GANAK [30] (a SharpSAT derivative), variable selection is based on variable scores, maintained as an array `score` mapping variables to floating point numbers. The variable selection algorithm works by selecting the variable x with the highest $\texttt{score}(x)$. The score of each variable is based on two components: it is the sum of the frequency score of the variable and the activity score of the variable. The frequency score is the number of occurrences of the variable in the current formula, and an activity score similar to VSIDS in SAT solvers [23]. The resulting heuristic, $\texttt{score}(x) = \texttt{act}(x) + \texttt{freq}(x)$, with both frequency and activity is called VSADS. Further, GANAK makes use of another score called CacheScore for prioritizing variables whose components were not recently added to the cache. The resulting heuristic is called CSVSADS. We implement tree decomposition based variable selection by modifying the `score` array in both SharpSAT and GANAK. In principle, implementing tree decomposition based variable selection with the `score` array amounts to just setting the score of a variable x to $-\min_{\{t|x \in T[t]\}} d_T(t)$, where $d_T(t)$ is the distance from the root of T to the node t. However, as we show in our experiments it is sometimes beneficial to use hybrid scores, even though the theoretical bound will not hold in that case. In particular, we propose the following integration of tree decomposition guidance as a modification of VSADS into both SharpSAT and GANAK:

$$\texttt{score}(x) = \texttt{act}(x) + \texttt{freq}(x) - C \min_{\{t|x \in T[t]\}} d_T(t) \tag{1}$$

where C is a per-instance chosen positive constant and $d_T(t)$ is normalized to take values between 0 and 1. As default we use $C = 100 \exp(n/w)/n$, where n is the number of variables and w the width of the tree decomposition. We empirically justify this choice in Section 5.

For computing tree decompositions of low width in practice, we use FlowCutter [16, 32] FlowCutter was ranked second in the 2nd Parameterized Algorithms and Computational Experiments Challenge (PACE 2017) heuristic treewidth track, and was observed to outperform the winning implementation on large graphs [10]. It is also used in the recent DPMC model counter [12]. FlowCutter is an anytime algorithm, meaning that we can terminate it anytime to get the best tree decomposition computed thus far. As the root of the tree decomposition we choose a centroid node, i.e., a node t such that each component of $G(\phi) \setminus T[t]$ has at most $|V(G(\phi))|/2$ vertices. Before computing the tree decomposition we preprocess the formula with the standard techniques of unit propagation and failed literal elimination.

Finally, although not on the level of internal decision heuristics, we note that both c2d [7] and minic2d [24] can take as an input a structure to control the variable ordering inside the compiler. In particular, c2d can take a decision tree (dtree) as an input and minic2d a variable tree (vtree) as an input. Both of these structures can be constructed from a tree decomposition so that the variable selection algorithm of the compiler implements Algorithm 2. For empirically evaluating the impact of tree decompositions obtained with FlowCutter on c2d and minic2d, we construct both of these structures from a tree decomposition by placing all variables of the root bag to the top of the tree, and recursing to the subtrees.

5 Empirical Evaluation

We provide results from an extensive empirical evaluation, comparing the impact of integrating tree decomposition based heuristics on the runtime performance of SharpSAT, GANAK, c2d, and minic2d. We also compare to the extend possible the performance of these four model counters with tree decomposition heuristics to the performance of the recent model counters D4 [21], DPMC [12], gpusat [15] and NestHDB [17]; and SharpSAT with the recently proposed centrality-based heuristics [3]. DPMC has both a tensor and a decision diagram based implementation. We compare to the decision diagram based implementation, as it has been reported to perform significantly better [12]. The decision diagram based implementation has two versions, DPMC-LG which exploits tree decompositions and DPMC-HTB, and we compare to both of them. (DPMC-HTB is equivalent to ADDMC [11].)

As benchmarks we used 2424 instances from recent empirical evaluations of model counters. In particular, we merged an instance set of 1952 instances from `http://www.cril.univ-artois.fr/KC/benchmarks.html` used in e.g. [21, 3, 12, 14] with an instance set of 1619 instances from `https://github.com/dfremont/counting-benchmarks` used

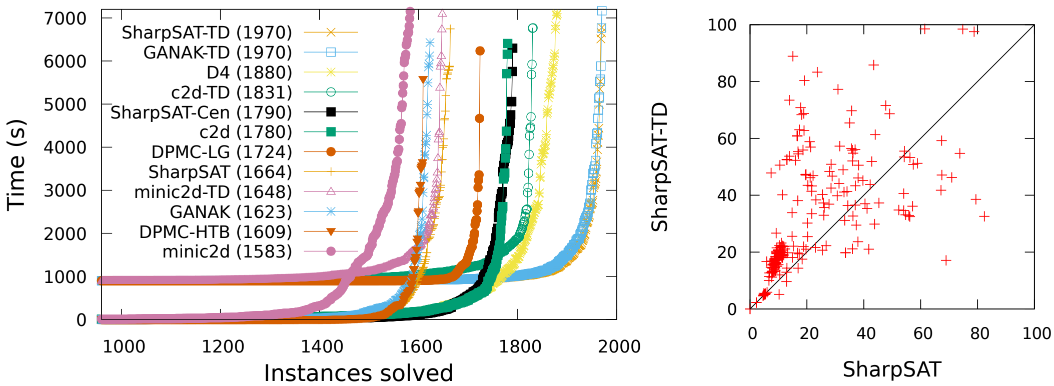

Figure 2 Left: Empirical runtime comparison of different model counters. Right: average number of variables in component cache hits, SharpSAT vs SharpSAT-TD.

Table 1 Pairwise comparison of original versions of SharpSAT, GANAK, c2d and minic2d against their tree decomposition guided versions.

Family	#Ins	VBS	VBS-O	VBS-TD	SharpSAT	SharpSAT-TD	GANAK	GANAK-TD	c2d	c2d-TD	minic2d	minic2d-TD
BN-Ratio	389	387	333	**387**	177	**385**	163	**386**	329	**345**	289	**330**
BN-DQMR	660	629	627	**629**	627	**629**	620	**629**	577	**593**	**598**	584
BN-Ace	31	22	22	22	22	22	22	22	22	22	**19**	16
BN-other	5	4	3	**4**	1	**2**	1	**2**	3	**4**	2	2
Plan recognition	11	11	11	11	11	11	11	11	10	**11**	10	**11**
Planning-pddl	529	458	**451**	447	417	**446**	405	**446**	426	**428**	**415**	390
Planning-other	17	16	16	16	15	**16**	15	**16**	13	13	11	**13**
Circuit-iscas	132	117	**117**	116	112	**116**	109	**116**	**110**	109	**109**	104
Circuit-other	17	10	10	10	10	10	10	10	9	9	**10**	9
BMC-symb-markov	130	118	112	**118**	52	**118**	53	**114**	94	**108**	20	20
BMC-other	18	14	**13**	11	**12**	11	**12**	11	7	**8**	3	**7**
Symbolic-sygus	138	22	**21**	17	**19**	15	**20**	16	**1**	0	0	0
QIF-maxcount-qif	127	12	11	**12**	11	**12**	6	**12**	10	10	4	4
QIF-other	7	5	4	**5**	4	**5**	4	**5**	4	**5**	2	**3**
Handmade	68	36	36	36	**36**	34	35	**36**	31	**33**	33	**35**
Configuration	35	35	35	35	35	35	34	**35**	**33**	32	21	21
Random	104	103	103	103	103	103	103	103	101	101	37	**99**
Scheduling	6	0	0	0	0	0	0	0	0	0	0	0
Total	2424	1999	1925	**1979**	1664	**1970**	1623	**1970**	1780	**1831**	1583	**1648**

in e.g. [15, 14, 17], removing duplicates and instances found unsatisfiable using a SAT solver. The benchmark set divides into 18 families from applications in e.g. probabilistic reasoning, planning, model checking, synthesis [29, 25, 21]. The experiments were run single-threaded on computers with 2.6-GHz Intel Xeon E5-2670 processors. A time limit of 2 hours and memory limit of 16 GB was used. Please consult `https://github.com/Laakeri/modelcounting-cp21` for source code and detailed data.

Figure 2(left) overviews the relative performance of the model counters (apart from gpusat and NestHDB). SharpSAT and GANAK using the tree decomposition heuristics (*-TD) solved the greatest number of instances (1970), resulting in state-of-the-art performance over all the considered counters. After SharpSAT-TD and GANAK-TD, the best-performing counters are D4, c2d-TD (i.e., the tree decomposition guided c2d), and SharpSAT using centrality-based heuristics, solving 1880, 1831, and 1790 instances, respectively. Note that here we allowed a fixed 900 seconds for tree decomposition computation using FlowCutter on each instance for SharpSAT-TD, GANAK-TD, c2d-TD, and minic2d-TD (as well as DPMC-LG; see Related Work section). This 900-second runtime is included in the results, as can be clearly seen in Figure 2(left). However, using this relatively high number of seconds is not necessary: when using 5, 60, 900, and 1800 seconds, respectively, the numbers of

Table 2 Pairwise comparison grouped by width of tree decompositions used by SharpSAT-TD.

Width	#Ins	VBS	VBS-O	VBS-TD	SharpSAT	SharpSAT-TD	GANAK	GANAK-TD	c2d	c2d-TD	minic2d	minic2d-TD
≤ 20	810	810	809	**810**	798	**810**	791	**810**	809	**810**	809	**810**
$21 \dots 30$	526	525	509	**525**	405	**524**	385	**524**	467	**489**	**483**	451
$31 \dots 50$	378	307	286	**303**	173	**302**	164	**302**	254	**266**	165	**185**
$51 \dots 100$	259	164	131	**155**	101	**152**	95	**153**	106	**117**	60	**71**
$101 \dots 150$	57	27	26	**27**	25	**26**	25	**27**	18	**23**	8	**13**
$151 \dots 200$	128	115	114	**115**	114	**115**	114	**115**	**112**	110	44	**108**
$201 \dots 300$	43	31	**31**	27	**31**	26	**31**	26	11	**13**	**11**	7
$301 \leq$	223	20	**19**	17	**17**	15	**18**	13	3	3	3	3
Total	2424	1999	1925	**1979**	1664	**1970**	1623	**1970**	1780	**1831**	1583	**1648**

Table 3 Comparison of gpusat, NestHDB, and SharpSAT-TD, grouped by width of the tree decomposition used by SharpSAT-TD.

Width	#Ins	VBS	gpusat	NestHDB	SharpSAT-TD
≤ 30	1232	1232	1232	1232	1232
$31 \dots 50$	21	14	1	10	**14**
$51 \dots 100$	15	10	0	7	**9**
$101 \dots 200$	18	16	0	16	16
$201 \dots 266$	21	11	0	8	**10**
$267 \leq$	187	0	0	0	0
Total	1494	1283	1233	1273	**1281**

instances solved by SharpSAT-TD are, respectively, 1962, 1971, 1970, and 1967. In particular, using the much lower time limit of 5 seconds would result in very much the same overall performance for SharpSAT-TD.

Table 1 gives a per benchmark family comparison of the impact of tree decomposition based heuristics on the number of instances solved by SharpSAT, GANAK, c2d and minic2d. SharpSAT-TD improves significantly on SharpSAT (1970 vs 1664 solved), and similarly GANAK-TD improves significantly on GANAK (1970 vs 1623). Furthermore, SharpSAT-TD and GANAK-TD solve only 9 instances less that the virtual best solver VBS-TD, which is considered to solve an instance if at least one of SharpSAT-TD, GANAK-TD, c2d-TD, and minic2d-TD solves the instance. VBS-TD also outperforms the virtual best solver VBS-O over the original four model counters which evidently are more different from each other than their modifications, each using the same tree decomposition to guide the counting process; Indeed, the difference between VBS-O and the best original model counter is 145 instances, in contrast to the difference of 9 instance between VBS-TD and SharpSAT-TD.

The number of instances solved, with instances grouped by the width of the tree decomposition found with FlowCutter in 900 seconds, is shown in Table 2. We observe to a great extent consistent performance improvement for each of the four model counters up to width 150 and at times even up to width 200. For instances of width ≤ 20, SharpSAT-TD, GANAK-TD, c2d-TD, and minic2d-TD each solve all instances, while the original SharpSAT, GANAK, c2d, and minic2d each fail to solve some instances.

Due to the techniques gpusat and NestHDB implement – gpusat relies on certain GPU hardware, and NestHDB relies on a database management system – we were unable to run them ourselves. Hence we are forced to resort to comparing our runtimes with the empirical results provided for gpusat and NestHDB in their respective papers [15, 17] using the benchmark instances used therein. For this indirect comparison, following [17], we enforced a per-instance time limit of 900 s, memory limit of 16 GB, and tree decomposition computation time limit of 60 s on SharpSAT-TD. Table 3 provides the indirect comparison with instances grouped by the width of the tree decomposition used by SharpSAT-TD. On this set of 1494 instances, gpusat solves 1233 instances and NestHDB solves 1273, while SharpSAT-TD solves 1281 instances. Note that in [17] NestHDB was found to be the best against a range of other model counters on these benchmarks, and minic2d second-best solving 1243 instances. Here SharpSAT-TD outperforms gpusat and NestHDB on all ranges of width apart from ≤ 30 and [101..200], where it solves the same instances as VBS.

Finally, we shortly overview further observation on the impact of the tree decomposition based heuristics in SharpSAT. We considered modifications of the variable selection heuristics (Equation 1) for SharpSAT-TD. Recall that SharpSAT-TD solved 1970 instances using the heuristic with default activity and frequency components and C determined as $C = 100\exp(n/w)/n$. When selecting C as 10^3, 10^7, and $100\exp(n/w)$, SharpSAT-TD solves 1922, 1964, and 1960 instances, respectively. We note that the choice 10^7 leads to the tree decomposition based component always dominating in the equation, with activity and frequency serving only as tiebreakers. When $C = 100\exp(n/w)/n$ and the activity component is removed, SharpSAT-TD solves 1965 instances, while when the frequency component is removed SharpSAT-TD solves 1962 instances. Hence the impact of each of these two components on their own, when including the tree decomposition component, is relatively small. However, when both the activity and the frequency component are removed, SharpSAT-TD solves only 1855 instances. Putting all of these observations together, we believe that the activity and frequency components act mainly as a secondary tiebreaking mechanism for choosing between variables in the same bag of the decomposition. Furthermore, the impact of the choice between using activity vs frequency as the tiebreaking mechanism appears to be small, and the primary heuristic component leading to the observed performance improvements is indeed the tree decomposition component.

The tree decomposition based heuristics appears to have a positive impact on average cache hit size, i.e., the number of variables of the components found to be in cache during checks to the component cache. Intuitively, the larger the cache hits, the earlier SharpSAT can determine the number of models in the current search branch, thereby saving time due to the component cache. Figure 2 (right) shows average cache hit sizes reported by SharpSAT and SharpSAT-TD on instances which both of them solved using at least 60 seconds on search (267 instances). The tree decomposition guided variable selection increases average cache hit size for most of the instances. (We did not observe clear effects on cache hit rates; cache hit rates do not distinguish hits on small components from hits on large component.)

6 Related Work

The idea of exploiting low-width tree decompositions in model counters has recently gained popularity with the model counters gpusat [15], NestHDB [17] and DPMC-LG [12] explicitly exploiting low-width tree decompositions. In contrast to our work, gpusat, NestHDB, and the tensor implementation of DPMC-LG exploit tree decompositions by manipulating dense dynamic programming tables. The model counters gpusat and the tensor implementation of DPMC-LG are "pure" dynamic programming implementations that suffer from best-case time complexity exponential in treewidth, while NestHDB also incorporates hybrid techniques, including falling back to SharpSAT in subproblems with high treewidth. The decision diagram implementation of DPMC-LG uses tree decompositions via project join trees to build an algebraic decision diagram using the CUDD package [31].

In the context of #CSP, tree decompositions have been exploited in the #BTD [13] and #EBTD[18] backtracking algorithms. The method of exploiting tree decompositions in #BTD and #EBTD is similar to SharpSAT-TD when selecting a high value of the constant C, although many techniques exploited in these counters are CSP-specific.

Instead of tree decompositions, heuristic graph partitioning is used in compilation-based model counters: D4 uses the PaToH graph partitioner [21], c2d uses Hmetis [8], and minic2d uses the min-fill heuristic for variable ordering [24]. GANAK introduced a variable selection heuristic CSVSADS aiming to increase the cache hit rate by discouraging branching from variables whose components were recently cached [30]. In the context of constraint networks, heuristics aiming to promote decomposition into components have been evaluated in [22].

7 Conclusion

We proposed a simple approach for integrating tree decomposition guidance into the decision heuristics of exact model counters. As a decision heuristic, the approach is directly applicable to both unweighted and weighted model counting. The empirical results suggest that tree decomposition guided SharpSAT dominates in performance standard exact model counters. and provides significant performance improvements in practice.

References

1 Fahiem Bacchus, Shannon Dalmao, and Toniann Pitassi. Algorithms and complexity results for #SAT and Bayesian inference. In *44th Symposium on Foundations of Computer Science, FOCS 2003*, pages 340–351. IEEE Computer Society, 2003. doi:10.1109/SFCS.2003.1238208.

2 Fabrizio Biondi, Michael A. Enescu, Annelie Heuser, Axel Legay, Kuldeep S. Meel, and Jean Quilbeuf. Scalable approximation of quantitative information flow in programs. In Isil Dillig and Jens Palsberg, editors, *Verification, Model Checking, and Abstract Interpretation – 19th International Conference, VMCAI 2018*, volume 10747 of *Lecture Notes in Computer Science*, pages 71–93. Springer, 2018. doi:10.1007/978-3-319-73721-8_4.

3 Bernhard Bliem and Matti Järvisalo. Centrality heuristics for exact model counting. In *31st IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2019*, pages 59–63. IEEE, 2019. doi:10.1109/ICTAI.2019.00017.

4 Hans L. Bodlaender. Discovering treewidth. In Peter Vojtás, Mária Bieliková, Bernadette Charron-Bost, and Ondrej Sýkora, editors, *SOFSEM 2005: Theory and Practice of Computer Science, 31st Conference on Current Trends in Theory and Practice of Computer Science*, volume 3381 of *Lecture Notes in Computer Science*, pages 1–16. Springer, 2005. doi:10.1007/978-3-540-30577-4_1.

5 Mark Chavira and Adnan Darwiche. On probabilistic inference by weighted model counting. *Artificial Intelligence*, 172(6-7):772–799, 2008. doi:10.1016/j.artint.2007.11.002.

6 Adnan Darwiche. Decomposable negation normal form. *Journal of the ACM*, 48(4):608–647, 2001. doi:10.1145/502090.502091.

7 Adnan Darwiche. New advances in compiling CNF into decomposable negation normal form. In Ramón López de Mántaras and Lorenza Saitta, editors, *Proceedings of the 16th European Conference on Artificial Intelligence, ECAI'2004*, pages 328–332. IOS Press, 2004.

8 Adnan Darwiche. The C2D compiler user manual. Technical Report D-147, UCLA Department of Computer Science, 2005.

9 Rina Dechter and Robert Mateescu. AND/OR search spaces for graphical models. *Artificial Intelligence*, 171(2-3):73–106, 2007. doi:10.1016/j.artint.2006.11.003.

10 Holger Dell, Christian Komusiewicz, Nimrod Talmon, and Mathias Weller. The PACE 2017 parameterized algorithms and computational experiments challenge: The second iteration. In Daniel Lokshtanov and Naomi Nishimura, editors, *12th International Symposium on Parameterized and Exact Computation, IPEC 2017*, volume 89 of *LIPIcs*, pages 30:1–30:12. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.IPEC.2017.30.

11 Jeffrey M. Dudek, Vu Phan, and Moshe Y. Vardi. ADDMC: Weighted model counting with algebraic decision diagrams. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020*, pages 1468–1476. AAAI Press, 2020. URL: https://aaai.org/ojs/index.php/AAAI/article/view/5505.

12 Jeffrey M. Dudek, Vu H. N. Phan, and Moshe Y. Vardi. DPMC: Weighted model counting by dynamic programming on project-join trees. In Helmut Simonis, editor, *Principles and Practice of Constraint Programming – 26th International Conference, CP 2020*, volume 12333 of *Lecture Notes in Computer Science*, pages 211–230. Springer, 2020. doi:10.1007/978-3-030-58475-7_13.

13 Aurélie Favier, Simon de Givry, and Philippe Jégou. Exploiting problem structure for solution counting. In Ian P. Gent, editor, *Principles and Practice of Constraint Programming – CP 2009, 15th International Conference, CP 2009*, volume 5732 of *Lecture Notes in Computer Science*, pages 335–343. Springer, 2009. doi:10.1007/978-3-642-04244-7_27.

14 Johannes K. Fichte, Markus Hecher, and Florim Hamiti. The Model Counting Competition 2020. *CoRR*, abs/2012.01323, 2020. arXiv:2012.01323.

15 Johannes Klaus Fichte, Markus Hecher, and Markus Zisser. An improved GPU-based SAT model counter. In Thomas Schiex and Simon de Givry, editors, *Principles and Practice of Constraint Programming – 25th International Conference, CP 2019*, volume 11802 of *Lecture Notes in Computer Science*, pages 491–509. Springer, 2019. doi:10.1007/978-3-030-30048-7_29.

16 Michael Hamann and Ben Strasser. Graph bisection with pareto optimization. *ACM Journal of Experimental Algorithmics*, 23, 2018. doi:10.1145/3173045.

17 Markus Hecher, Patrick Thier, and Stefan Woltran. Taming high treewidth with abstraction, nested dynamic programming, and database technology. In Luca Pulina and Martina Seidl, editors, *Theory and Applications of Satisfiability Testing – SAT 2020 – 23rd International Conference*, volume 12178 of *Lecture Notes in Computer Science*, pages 343–360. Springer, 2020. doi:10.1007/978-3-030-51825-7_25.

18 Philippe Jégou, Hanan Kanso, and Cyril Terrioux. Improving exact solution counting for decomposition methods. In *28th IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2016*, pages 327–334. IEEE Computer Society, 2016. doi:10.1109/ICTAI.2016.0057.

19 Roberto J. Bayardo Jr. and Joseph Daniel Pehoushek. Counting models using connected components. In Henry A. Kautz and Bruce W. Porter, editors, *Proceedings of the Seventeenth National Conference on Artificial Intelligence , AAAI 2000*, pages 157–162. AAAI Press / The MIT Press, 2000. URL: http://www.aaai.org/Library/AAAI/2000/aaai00-024.php.

20 Stefan Kölbl, Gregor Leander, and Tyge Tiessen. Observations on the SIMON block cipher family. In Rosario Gennaro and Matthew Robshaw, editors, *Advances in Cryptology – CRYPTO 2015 – 35th Annual Cryptology Conference*, volume 9215 of *Lecture Notes in Computer Science*, pages 161–185. Springer, 2015. doi:10.1007/978-3-662-47989-6_8.

21 Jean-Marie Lagniez and Pierre Marquis. An improved decision-DNNF compiler. In Carles Sierra, editor, *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017*, pages 667–673. ijcai.org, 2017. doi:10.24963/ijcai.2017/93.

22 Jean-Marie Lagniez, Pierre Marquis, and Anastasia Paparrizou. Defining and evaluating heuristics for the compilation of constraint networks. In J. Christopher Beck, editor, *Principles and Practice of Constraint Programming – 23rd International Conference, CP 2017*, volume 10416 of *Lecture Notes in Computer Science*, pages 172–188. Springer, 2017. doi:10.1007/978-3-319-66158-2_12.

23 Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient SAT solver. In *Proceedings of the 38th Design Automation Conference, DAC 2001*, pages 530–535. ACM, 2001. doi:10.1145/378239.379017.

24 Umut Oztok and Adnan Darwiche. An exhaustive DPLL algorithm for model counting. *Journal of Artificial Intelligence Research*, 62:1–32, 2018. doi:10.1613/jair.1.11201.

25 Markus N. Rabe, Christoph M. Wintersteiger, Hillel Kugler, Boyan Yordanov, and Youssef Hamadi. Symbolic approximation of the bounded reachability probability in large Markov chains. In Gethin Norman and William H. Sanders, editors, *Quantitative Evaluation of Systems – 11th International Conference, QEST 2014*, volume 8657 of *Lecture Notes in Computer Science*, pages 388–403. Springer, 2014. doi:10.1007/978-3-319-10696-0_30.

26 Neil Robertson and Paul D. Seymour. Graph minors. II. Algorithmic aspects of tree-width. *Journal of Algorithms*, 7(3):309–322, 1986. doi:10.1016/0196-6774(86)90023-4.

27 Marko Samer and Stefan Szeider. Algorithms for propositional model counting. *Journal of Discrete Algorithms*, 8(1):50–64, 2010. doi:10.1016/j.jda.2009.06.002.

28 Tian Sang, Fahiem Bacchus, Paul Beame, Henry A. Kautz, and Toniann Pitassi. Combining component caching and clause learning for effective model counting. In *SAT 2004 – The Seventh International Conference on Theory and Applications of Satisfiability Testing*, 2004. URL: http://www.satisfiability.org/SAT04/programme/21.pdf.

29 Tian Sang, Paul Beame, and Henry A. Kautz. Performing Bayesian inference by weighted model counting. In Manuela M. Veloso and Subbarao Kambhampati, editors, *The Twentieth National Conference on Artificial Intelligence, AAAI 2005*, pages 475–482. AAAI Press / The MIT Press, 2005. URL: http://www.aaai.org/Library/AAAI/2005/aaai05-075.php.

30 Shubham Sharma, Subhajit Roy, Mate Soos, and Kuldeep S. Meel. GANAK: A scalable probabilistic exact model counter. In Sarit Kraus, editor, *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019*, pages 1169–1176. ijcai.org, 2019. doi:10.24963/ijcai.2019/163.

31 Fabio Somenzi. CUDD: CU decision diagram package–release 3.0.0, 2015. URL: https://github.com/ivmai/cudd.

32 Ben Strasser. Computing tree decompositions with FlowCutter: PACE 2017 submission. *CoRR*, abs/1709.08949, 2017. arXiv:1709.08949.

33 Marc Thurley. sharpSAT – counting models with advanced component caching and implicit BCP. In Armin Biere and Carla P. Gomes, editors, *Theory and Applications of Satisfiability Testing – SAT 2006, 9th International Conference*, volume 4121 of *Lecture Notes in Computer Science*, pages 424–429. Springer, 2006. doi:10.1007/11814948_38.

34 Leslie G. Valiant. The complexity of enumeration and reliability problems. *SIAM Journal on Computing*, 8(3):410–421, 1979. doi:10.1137/0208032.