Knowledge Compilation and More with SharpSAT-TD

Rafael Kiesel , Thomas Eiter
TU Wien, Vienna, Austria
rafael.kiesel@web.de, thomas.eiter@tuwien.ac.at

Abstract

SharpSAT-TD is a recently published exact model counter that performed exceptionally well in the recent editions of the Model Counting Competition (https://mccompetition.org/). Notably, it also features *weighted* model counting capabilities over commutative semirings. In this work, we show how to exploit this fact to use SharpSAT-TD as a knowledge compiler to the class of sd-DNNF circuits. Our experimental evaluation shows that the efficiency of SharpSAT-TD for (weighted) model counting transfers to knowledge compilation since it outperforms other state of the art knowledge compilers on standard benchmark sets. In addition, we generalized SharpSAT-TD's preprocessing to support arbitrary semirings and consider the utility of auxiliary variables in this setting.

1 Introduction

The idea behind knowledge compilation is to transform a logical formula ϕ into a more efficient representation C such that (i) ϕ is equivalent to C and (ii) we can perform a task that is intractable on ϕ instead tractably on C (Darwiche and Marquis 2002). A typical such representation are (s)d-DNNF circuits, which are successfully applied in a manifold of applications, ranging from probabilistic inference for Bayesian networks (Chavira and Darwiche 2008; Bart et al. 2016; Van den Broeck and Suciu 2017) or probabilistic logic programs (De Raedt, Kimmig, and Toivonen 2007; Eiter, Hecher, and Kiesel 2021; Riguzzi and Swift 2011) to QBF-solving (Capelli and Mengel 2019), model enumeration in database theory (Amarilli et al. 2017), and uniform sampling (Sharma et al. 2018), to name only a few of them.

However, while powerful if possible, knowledge compilation is well-known to be a highly costly problem (Amarilli et al. 2020; Beame and Liew 2015). Thus, many of the application areas are limited by the performance of compilation.

In this work, we aim to push the limits of knowledge compilation by enabling the use of SharpSAT-TD (Korhonen and Järvisalo 2021) as a knowledge compiler. SharpSAT-TD is a recent exact model counter for propositional formulas that performed exceptionally well in the latest model counting competitions,[1] achieving first place in 2021 and second place in 2022 on the exact counting track. Additionally, the winning submission in 2022 was also based on SharpSAT-TD but used a different preprocessor.

It is well-known that compilation to (s)d-DNNF is possible by recording the trace of DPLL-based model counters (Huang and Darwiche 2005). SharpSAT-TD falls into this category of solvers, meaning we can transform it into a knowledge compiler in principle. Even further than that, SharpSAT-TD's well-thought-out design allows not only model counting and weighted model counting but also algebraic model counting (AMC), i.e., weighted model counting over any semiring. This gives us the possibility to perform compilation with SharpSAT-TD via a knowledge compilation semiring similar to that of Kimmig, Van den Broeck, and De Raedt (2017). Importantly, this means that we can leave the underlying algorithmic details of SharpSAT-TD untouched and perform knowledge compilation by recording its trace using an appropriate semiring.

Apart from that, we consider two minor but useful techniques. First, we allow a subset X of the variables in a given formula ϕ to be projected off during compilation, meaning that the resulting circuit is equivalent to $\exists X \phi$ instead of ϕ. Oztok and Darwiche (2017) showed that this technique could be used to speed up compilation to DNNF, a tractable circuit class with weaker requirements than sd-DNNF. Moreover, we show that if these variables are *defined*, i.e., functionally determined by the other variables in ϕ, we even obtain an sd-DNNF when applying this technique.[2] This can lead to smaller circuits if the variables in X are irrelevant, e.g., if they were only used as auxiliary variables to encode a logical formula into CNF. Notably, in probabilistic logic programming, the number of auxiliary variables often vastly exceeds the number of relevant variables (Fierens et al. 2015), which opens up a promising application perspective.

Second, we extend the preprocessor of SharpSAT-TD with specific focus on the weighted model counting/compilation setting. Mostly, we focus on SharpSAT-TD's approach for variable elimination here. While we can only use it in a limited manner in this general setting, it may help nevertheless in special cases.

Summarizing, our main contributions are the following:

- We provide a knowledge compilation version of

[1] https://mccompetition.org/

[2] In general, the result is only guaranteed to be a DNNF.

SharpSAT-TD. Interestingly, our proof of correctness shows that we can perform knowledge compilation using any AMC solver that performs the same steps independently of the semiring and weights.

- Our empirical evaluation shows that the performance of SharpSAT-TD transfers from model counting to knowledge compilation, solving more instances than c2d (Darwiche 2004) and d4 (Lagniez and Marquis 2017), two state of the art knowledge compilers.
- Additionally, we allow variables to be projected away during compilation, leading to smaller circuits and an interesting use case in DNNF compilation.
- Our changes to SharpSAT-TD's preprocessing generalize it to weighted model counting over arbitrary semirings.

By combining a variety of results and ideas from other works (Korhonen and Järvisalo 2021; Oztok and Darwiche 2017; Ritter 2022; Kimmig, Van den Broeck, and De Raedt 2017; Huang and Darwiche 2005), we can present a knowledge compilation version of SharpSAT-TD that is not only faster than current state of the art compilers but also comes with additional interesting features for some settings.

In the rest of the paper, we first introduce preliminaries in Section 2 before we introduce our approach to knowledge compilation with SharpSAT-TD in Section 3. Next, we discuss in Sections 4 and 5 two additional features regarding the existential quantification of variables during compilation and our modifications to SharpSAT-TD's preprocessing. Then, in Section 6 we provide an empirical evaluation, where we compare to other knowledge compilers and assess the effect of the additional advancements. We conclude in Section 7.

2 Preliminaries

We briefly introduce propositional logic, tractable circuit classes, and semirings and refer the interested reader to (Biere et al. 2021; Darwiche and Marquis 2002; Kimmig, Van den Broeck, and De Raedt 2017) for more details.

We use propositional formulas in Conjunctive Normal Form (CNF). A CNF $\mathcal{C}$, defined for a set V of variables, is a finite conjunction of *clauses* C_i, where each clause consists of a finite disjunction of *literals* $\ell \in \{v, \neg v\}$ for some $v \in V$. For simplicity, we use the convention that $\neg\neg v = v$.

We represent truth assignments as a subset of $\mathcal{I} \subseteq V$. Here, if $v \in \mathcal{I}$, then the literal v is satisfied. Otherwise, the literal $\neg v$ is satisfied. As usual, a clause is satisfied by $\mathcal{I}$ if one of its literals is satisfied, and a CNF is satisfied if all of its clauses are satisfied. We call an assignment that satisfies a CNF a *model*.

Additionally, we consider tractable circuit classes based on special negation normal forms (NNFs). An NNF (Darwiche 2004) is a rooted directed acyclic graph in which each leaf node is labeled with a literal, true, or false, and each internal node is labeled with a conjunction $\wedge$ or disjunction $\vee$. For any node n in an NNF graph, $\mathit{Vars}(n)$ denotes all variables in the subgraph rooted at n. By abuse of notation, we use n also to refer to the formula represented by the graph n. Then sd-DNNFs are NNFs that satisfy the following additional properties:

$$\begin{tikzcd}
 & & & & & \vee \arrow[dllll] \arrow[drrrr] & & & & & \\
 & \wedge \arrow[dl] \arrow[drr] & & & & & & & & \wedge \arrow[dl] \arrow[d] \arrow[dr] & \\
a & & & \vee \arrow[dll] \arrow[drr] & & & & & \neg a & \neg b & \neg c \\
 & \wedge \arrow[dl] \arrow[dr] & & & & \wedge \arrow[dl] \arrow[dr] & & & & & \\
b & & \vee \arrow[dl] \arrow[dr] & & \neg b & & c & & & & \\
 & c & & \neg c & & & & & & &
\end{tikzcd}$$

Figure 1: An sd-DNNF n_{run} over the variables $\{a, b, c\}$.

Decomposability (D): $\mathit{Vars}(n_i) \cap \mathit{Vars}(n_j) = \emptyset$ for any two children n_i and n_j of an and-node.

Determinism (d): $n_i \wedge n_j$ is logically inconsistent for any two children n_i and n_j of an or-node.

Smoothness (s): $\mathit{Vars}(n_i) = \mathit{Vars}(n_j)$ for any two children n_i and n_j of an or-node.

Accordingly, a DNNF is an NNF that satisfies decomposability.

Example 1 (Running). *Consider the CNF $\mathcal{C}_{run} = \{\neg a \vee b \vee c, a \vee \neg b, a \vee \neg c\}$. The sd-DNNF shown in Figure 1 is equivalent to $\mathcal{C}$, since both model that a holds iff $b \vee c$ holds.*

We want to enable compilation of CNF to sd-DNNF by exploiting SharpSAT-TD's capabilities of weighted model counting over *semirings*, where algebraic expressions are evaluated. Recall that semirings are as follows:

Definition 1 (Semiring). *A commutative* semiring $\mathcal{S} = (S, \oplus, \otimes, e_\oplus, e_\otimes)$ *is an algebraic structure with binary infix operations $\oplus, \otimes$ such that*

1. *$\oplus$ and $\otimes$ are associative,*
2. *$\oplus$ and $\otimes$ are commutative,*
3. *$\otimes$ right and left distributes over $\oplus$,*
4. *$e_\oplus$ (resp. $e_\otimes$) is a neutral element for $\oplus$ (resp. $\otimes$), and*
5. *$e_\oplus$ annihilates S, i.e., $\forall s \in S : s \otimes e_\oplus = e_\oplus = e_\oplus \otimes s$.*

Examples of well-known commutative semirings are

- $\mathbb{F} = (\mathbb{F}, +, \cdot, 0, 1)$, where $\mathbb{F} \in \{\mathbb{N}, \mathbb{Z}, \mathbb{Q}, \mathbb{R}\}$, the semiring over the numbers in $\mathbb{F}$ with addition and multiplication,
- $\mathcal{P} = ([0, 1], +, \cdot, 0, 1)$, the probability semiring,
- $\mathcal{S}_{\max,+} = (\mathbb{R} \cup \{-\infty\}, \max, +, -\infty, 0)$, the max-plus semiring.

3 Knowledge Compilation with SharpSAT-TD

Our main aim is to enable knowledge compilation of CNFs to sd-DNNF with SharpSAT-TD. It is a well-known result by Huang and Darwiche (2005) that this is possible since SharpSAT-TD is based on DPLL (combined with clause learning, component analysis, and caching). The idea here is that we can record SharpSAT-TD's trace, i.e., the way that the solver traverses the search tree, in such a way that the resulting recording corresponds to a d-DNNF. While we implicitly use this idea, we can obtain the same result in a much simpler manner by making use of SharpSAT-TD's capabilities for weighted model counting over semirings, a.k.a. Algebraic Model Counting (AMC):

Definition 2 (AMC (Kimmig, Van den Broeck, and De Raedt 2017))**.** *A triple $A = (\mathcal{C}, \mathcal{S}, \alpha)$, where $\mathcal{C}$ is a CNF over variables V, $\mathcal{S}$ is a commutative semiring, and α is a labeling function that assigns every literal ℓ from V a weight $\alpha(\ell)$ in $\mathcal{S}$, is called an Algebraic Model Counting (AMC) instance. Solving A is to compute*

$$AMC(A) = \bigoplus_{\mathcal{I} \subseteq V, \mathcal{I} \models \mathcal{C}} \bigotimes_{v \in \mathcal{I}} \alpha(v) \otimes \bigotimes_{v \in V \setminus \mathcal{I}} \alpha(\neg v).$$

If $\mathcal{S} = \mathbb{N}$ and $\alpha(\ell) = 1$ for all literals ℓ, then AMC corresponds to counting the models of the CNF $\mathcal{C}$. If $\mathcal{S} = \mathcal{S}_{\max,+}$, then AMC corresponds to maximum satisfiability, i.e., finding the optimal weight of any model of $\mathcal{C}$. Similarly, typical weighted model counting uses $\mathcal{S} = \mathbb{R}$, with an arbitrary labeling function α. In the special case of probabilistic reasoning, the labels can be used for probabilities $\alpha(\ell) \in [0, 1]$.

We define a knowledge compilation semiring similar to the ones of (Kimmig, Van den Broeck, and De Raedt 2017).

Definition 3 (Knowledge Compilation Semiring)**.** *Let V be a finite set of propositional variables. Then $KC(V) = (NNF(V), \vee, \wedge, \bot, \top)$ is the knowledge compilation semiring over V, where, $NNF(V)$ is the set of NNF circuits over the variables in V, and for $n_1, n_2 \in NNF(V)$ we let*

$$n_1 \vee n_2 = \begin{cases} n_1 & \text{if } n_2 = \bot \\ n_2 & \text{if } n_1 = \bot \\ n_1 \vee n_2 & \text{otherwise} \end{cases}$$

and

$$n_1 \wedge n_2 = \begin{cases} n_1 & \text{if } n_2 = \top \\ n_2 & \text{if } n_1 = \top \\ n_1 \wedge n_2 & \text{otherwise} \end{cases}$$

where the "otherwise" case means the NNF obtained by adding a new root node n with label $\vee$ (resp. $\wedge$) and children n_1, n_2. The symbols $\bot$ and $\top$ correspond to NNF with a single leaf node labeled with false and true, respectively.

The equality $n_1 = n_2$ holds iff n_1 and n_2 are logically equivalent.

Clearly, if we used syntactic instead of semantic equality, $KC(V)$ would not be a semiring since already $n_1 \vee n_2$ is unequal to $n_2 \vee n_1$ syntactically.

Corollary 4. *For each set of variables, V, $KC(V)$ is a commutative semiring.*

We see that the idea behind the knowledge compilation semiring is simple: if we use it with SharpSAT-TD, then whenever an addition/multiplication is performed, we record a new internal node that represents the disjunction/conjunction of the inputs. Thus, in order to compile a CNF $\mathcal{C}$ over variables V to an sd-DNNF n, we use SharpSAT-TD to solve the AMC instance $KC(\mathcal{C}) = (\mathcal{C}, KC(V), \alpha_V)$, where $\alpha_V(\ell) = \ell$ for each literal ℓ.

Example 2 (cont'd.)**.** *For $\mathcal{C}_{run} = \{\neg a \vee b \vee c, a \vee \neg b, a \vee \neg c\}$, we consider how a standard DPLL-based algorithm for AMC would solve $KC(\mathcal{C}_{run})$. We can compute $AMC(KC(\mathcal{C}_{run}))$ as the sum (using the addition $\vee$ from the semiring) of the solutions given that a is true and false, respectively:*

$$\begin{aligned} &AMC(KC(\mathcal{C}_{run} \cup \{a\})) \vee AMC(KC(\mathcal{C}_{run} \cup \{\neg a\})) \\ &= (a \wedge AMC(KC(\{b \vee c\}))) \vee (\neg a \wedge \neg b \wedge \neg c)) \end{aligned}$$

If a is false, then b and c must also be false. In particular, the second line results from unit propagation. Thus, we obtain $\neg a \wedge \neg b \wedge \neg c$ here. Compare this to the sd-DNNF in Figure 1. Here, we also have two branches, one where a is true and $b \vee c$ holds and one where $\neg a \wedge \neg v \wedge \neg c$ holds. If we further evaluate $AMC(KC(\{b \vee c\}))$ in this manner, we can obtain the same sd-DNNF as in the figure.

Algorithm 1 $\text{EVAL}(A, n)$

Input An NNF n and an AMC instance $A = (\mathcal{C}, \mathcal{S}, \alpha)$.

```
1: switch $n$ do
2:     case $\ell$: return $\alpha(\ell)$
3:     case $n_1 \vee n_2$: return $\text{EVAL}(A, n_1) \oplus \text{EVAL}(A, n_2)$
4:     case $n_1 \wedge n_2$: return $\text{EVAL}(A, n_1) \otimes \text{EVAL}(A, n_2)$
5:     case $\bot$: return $e_\oplus$
6:     case $\top$: return $e_\otimes$
```

Consider the following procedure $\text{EVAL}(A, n)$.

If we take the syntactic result $n = AMC(KC(\mathcal{C}))$ produced by SharpSAT-TD and compute $\text{EVAL}(A, n)$ for an arbitrary AMC instance A, then we perform the exact additions/multiplications that SharpSAT-TD would have performed in the first place had we solved A instead of $KC(\mathcal{C})$. Since SharpSAT-TD is correct, $\text{EVAL}(A, n) = AMC(A)$.[3]

Interestingly, this already suffices to prove that the syntactic result of $AMC(KC(\mathcal{C}))$ is an sd-DNNF.

Theorem 5. *Suppose n is an NNF such that*

- *n is logically equivalent to a CNF $\mathcal{C}$ and*
- *n contains no leaf nodes labeled false or true.*

Then the following are equivalent:

(i) *n is an sd-DNNF*

(ii) *for every AMC instance $A = (\mathcal{C}, \mathcal{S}, \alpha)$ it holds that $\text{EVAL}(A, n) = AMC(A)$,*

(iii) *for every AMC instance $A = (\mathcal{C}, \mathbb{R}, \alpha)$ it holds that $\text{EVAL}(A, n) = AMC(A)$,*

Note that due to the definition of $\vee$ and $\wedge$, the syntactic result of $AMC(KC(\mathcal{C}))$ (a) contains a leaf node labeled false iff $\mathcal{C}$ is contradictory, and then $AMC(KC(\mathcal{C})) = \bot$, and (b) contains a leaf node labeled true iff $\mathcal{C}$ is the empty set of clauses over the empty set of variables, and then $AMC(KC(\mathcal{C})) = \top$. Thus, either $AMC(KC(\mathcal{C}))$ is a trivial NNF, or it does not contain leaf nodes labeled false or true, and we can apply the theorem to obtain that it is an sd-DNNF.

Proof of Theorem 5. (i) to (ii) is known (Kimmig, Van den Broeck, and De Raedt 2017, Theorem 2).

(ii) to (iii) is trivially true.

[3]Note that it is important n is the syntactic result rather than another NNF, since already $\mathcal{C}$ is semantically equivalent to n.

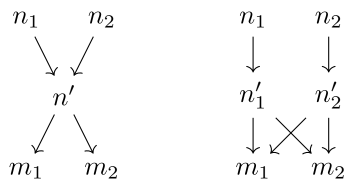

Figure 2: One step of treeification on n'.

For (iii) to (i), we proceed as follows. First, we "treeify" n, that is, we replace every node n' of n that has multiple incoming arcs by one copy n'_i for each arc (n_i, n'), add the arc (n_i, n'_i) and keep the children of n' as children for n'_i. We do this exhaustively until the resulting NNF n_{tree} is a (directed) tree. Note that $\textsc{Eval}(A, n) = \textsc{Eval}(A, n_{tree})$.

Second, we exhaustively apply the distributive law replacing nodes of the form $(n_1 \vee n_2) \wedge n_3$ (resp. $n_3 \wedge (n_1 \vee n_2)$) by $(n_1 \wedge n_3) \vee (n_2 \wedge n_3)$ (resp. $(n_3 \wedge n_1) \vee (n_3 \wedge n_1)$). This leads to an NNF of the form

$$n_{mods} = \bigvee_{i=1}^{m} \bigwedge_{j=1}^{o_i} \ell_{i,j},$$

where $\ell_{i,j}$ are literals over the variables of the CNF $\mathcal{C}$, since all leaves must be labelled by a literal. Again, $\textsc{Eval}(A, n) = \textsc{Eval}(A, n_{mods})$ holds since $\mathbb{R}$ is a semiring and thus satisfies the distributive law.

Note that this expression is of the same form as the original definition of $AMC(A)$, i.e.,

$$\bigoplus_{\mathcal{I} \subseteq V, \mathcal{I} \models \mathcal{C}} \bigotimes_{v \in \mathcal{I}} \alpha(v) \otimes \bigotimes_{v \in V \setminus \mathcal{I}} \alpha(\neg v).$$

Note that computing $\textsc{Eval}(A, \mathbb{R}, \alpha)$ corresponds to replacing $\ell_{i,j}$ by $\alpha(\ell_{i,j})$, $\bigvee$ by $\sum$, and $\bigwedge$ by $\prod$, and evaluating the resulting expression. That is, both the definition of $AMC(A)$ and n_{mods} can be seen as a polynomial in multiple variables over $\mathbb{R}$ that have a unique representation as a sum of monomials

$$\alpha(v_1)^{k_1} \cdot \dots \cdot \alpha(v_{|V|})^{k_{|V|}} \cdot \alpha(\neg v_1)^{k_{1+|V|}} \cdot \dots \cdot \alpha(\neg v_{|V|})^{k_{2|V|}}.$$

Since the polynomials are equal, they have the same monomials, which implies the following:

1. Every monomial occurs at most once in n_{mods}.
2. Every monomial of n_{mods} has either a factor v or $\neg v$ for every $v \in V$.
3. The exponents k_i of every factor in every monomial of n_{mods} are either zero or one.

Thus, n must be an sd-DNNF since 1., 2., and 3. correspond to (d), (s), and (D), exactly:

- If there is an internal node $n_1 \vee n_2$ such that $n_1 \wedge n_2$ are logically consistent, there is (under the assumption that 2. holds) a duplicate monomial, contradicting 1.
- If there is an internal node $n_1 \vee n_2$ such that $Vars(n_1) \neq Vars(n_2)$, there are (under the assumption that 3. holds) two monomials with a different set of variables mentioned in it, contradicting 2.
- If there is an internal node $n_1 \wedge n_2$ such that $Vars(n_1) \cap Vars(n_2) \neq \emptyset$, there is (under the assumption that 2. holds) a factor with exponent ≥ 2, contradicting 3. □

Note that this result implies two interesting facts for AMC solvers that are "algebraically agnostic", i.e., whose evaluation strategy does not depend on the given semiring and labeling function. First, we see that any such solver can be modified to compile sd-DNNFs, and second, it follows that its runtime is at least the size of the smallest sd-DNNF for the CNF of the AMC instance. This means that to achieve better efficiency than would be possible with knowledge compilation, solvers must somehow exploit the semiring and labeling function of the instance.

3.1 Implementation

We saw that we can produce an sd-DNNF for a CNF $\mathcal{C}$ by recording a new $\wedge$ node whenever a multiplication is performed and by recording a new $\vee$ node whenever an addition is performed. In our implementation, we do exactly this apart from one minor specialization. Namely, when SharpSAT-TD takes a decision that sets a literal ℓ to true and derives $\ell_1, \dots, \ell_n$ by unit propagation, then we do not create n new $\wedge$ nodes with two children each but create only one $\wedge$ node with the $n + 1$ children $\ell, \ell_1, \dots, \ell_n$. This leads to smaller sd-DNNFs and does not seem to impact performance negatively.

To produce the file containing the final sd-DNNF, we offer two options. Either (a) we immediately write the recorded nodes to the file, or (b) we keep the recording and write the whole sd-DNNF after compilation. Option (a) can lead to performance drops due to the heavy use of blocking I/O operations but comes with a lower memory footprint since we do not need to keep a representation of the sd-DNNF in memory. Accordingly, option (b) requires more memory but does not suffer more than necessary from I/O limitations. In our experience, when running one instance of SharpSAT-TD on a machine with a solid-state disk, there is no noticeable difference in performance between the two methods. However, in a benchmarking setting, where we run multiple instances simultaneously on one machine, there is a notable performance drop when using option (a).

4 Utilizing Existentially Quantified Variables

In this section, we consider and extend upon an idea by Oztok and Darwiche (2017). They showed that there are logical formulas ϕ over propositional variables V such that (i) it is impossible to construct a CNF $\mathcal{C}$ over V such that current knowledge compilers can compile a d-DNNF based on $\mathcal{C}$ that is equivalent to ϕ and (ii) they showed that it is possible to construct a CNF $\mathcal{C}'$ over $V \cup X$ such that current knowledge compilers *can* compile a d-DNNF n based on $\mathcal{C}'$ such that $\exists X n$ is equivalent to ϕ.

On the one hand, this is very convenient since computing $\exists X n$ for a d-DNNF n is possible in linear time in n by simply replacing every literal label $v, \neg v$ such that $v \in X$ by the label true. We denote the result of this procedure by $\textsc{Exists}(n, X)$. On the other hand, it is well-known that $\textsc{Exists}(n, X)$ is only guaranteed to be a DNNF in general.

Example 3 (cont'd.). *Recall the sd-DNNF n_{run} in Figure 1. We consider $\textsc{Exists}(n_{run}, \{b\})$ and observe that the partial evaluation and simplification of $(b \wedge (c \vee \neg c)) \vee (\neg b \wedge c)$*

results in $(c \vee \neg c) \vee c$. *Since the conjunction of the disjuncts is not a contradiction,* $\textsc{Exists}(n_{run}, \{b\})$ *is not deterministic.*

Thus, it may initially seem like we can only apply this strategy to speed up DNNF compilation. However, we can show that we can use the same strategy to speed up sd-DNNF compilation, if we can ensure that the variables that are existentially quantified are defined in terms of the remaining variables.

Definition 6 (Definability (Lagniez, Lonca, and Marquis 2016))**.** *A variable x is* defined *by a set of variables V with respect to a CNF $\mathcal{C}$ over variables $V \cup X$ if for every assignment $\mathcal{I}$ of V it holds that the variable v is either included in* every *model $\mathcal{I}'$ of $\mathcal{C}$ such that $V \cap \mathcal{I}' = \mathcal{I}$ or in none of them.*

Intuitively, this means that the truth of a defined variable x is functionally determined by the variables in V in every model of a CNF.

Using definability, we obtain the following result:

Lemma 7. *Let $\mathcal{C}$ be a CNF over variables $V \cup X$ such that every variable $x \in X$ is defined by V with respect to $\mathcal{C}$ and let n be an sd-DNNF that is equivalent to $\mathcal{C}$. Then* $\textsc{Exists}(n, X)$ *is an sd-DNNF that is equivalent to* $\exists X \mathcal{C}$.

Decomposability of $\textsc{Exists}(n, X)$ and equivalence to $\exists X \mathcal{C}$ are known (Oztok and Darwiche 2017). Also smoothness is not hard to prove, since for all NNFs n, n' it holds that $Vars(n) = Vars(n')$ implies $Vars(n) \setminus X = Vars(n') \setminus X$. However, the fact that $\textsc{Exists}(n, X)$ is *deterministic* is, to the best of our knowledge, new.

Proof (Sketch). We use that if a variable x is defined in terms of V and $\mathcal{C}$, then the sets $\{\mathcal{I} \cap V \mid \mathcal{I} \models \mathcal{C}, x \in \mathcal{I}\}$ and $\{\mathcal{I} \cap V \mid \mathcal{I} \models \mathcal{C}, x \notin \mathcal{I}\}$ are disjoint. Thus, if for an or-node $n_1 \vee n_1$ the conjunction $n_1 \wedge n_2$ is logically inconsistent due to different values of x, then the conjunction $\textsc{Exists}(n_1, \{x\}) \wedge \textsc{Exists}(n_2, \{x\})$ must still be logically inconsistent.

It follows that determinism still holds for $\textsc{Exists}(n, X)$. $\square$

4.1 Applications

This shows that it is, in principle, possible to use auxiliary variables in a CNF in order to speed up compilation while maintaining the ability to obtain a d-DNNF from the compilation result by applying $\textsc{Exists}(.)$. However, this is not only of theoretical interest. In fact, it is well-known that there are propositional formulas ϕ such that every CNF $\mathcal{C}$ over the same variables has exponential size in ϕ. The standard Tseitin transformation (Biere et al. 2021, Chapter 2) avoids this by adding auxiliary variables that represent the truth of subformulas, meaning they are defined.

Another less obvious example are the works of (Hecher 2022; Eiter, Hecher, and Kiesel 2021) that consider translations of (probabilistic) logic programs to CNF that use auxiliary variables to ensure low treewidth of the CNF. Since low treewidth is known to lead to efficient knowledge compilation (Darwiche 2004; Korhonen and Järvisalo 2021), this immediately provides a use case for the existential quantification of auxiliary variables. Namely, it allows us to compile DNNFs (resp. d-DNNFs) for logic programs using (Hecher 2022) (resp. (Eiter, Hecher, and Kiesel 2021)).

The resulting d-DNNFs can not only be simplified to smaller ones than those containing the original variables but are also of interest because they represent the original logical theory in the true sense as they only refer to the original variables.

4.2 Implementation

In our implementation, we do not first compile an sd-DNNF and apply $\textsc{Exists}(.)$ afterwards. Instead, we can revisit the definition of the AMC instance $KC(\mathcal{C})$ that we use to compile $\mathcal{C}$. Here, if we aim to existentially quantify the variables in the set X and compile a CNF $\mathcal{C}$ over variables V, we instead use the AMC instance $KC(\mathcal{C}, X) = (\mathcal{C}, KC(V \setminus X), \alpha_{V \setminus X})$, where $KC(V \setminus X)$ is as before and for a literal ℓ over V the label $\alpha_{V \setminus X}(\ell) = \ell$, if $\ell = v, \neg v$ for $v \in V \setminus X$ and $\alpha_{V \setminus X}(\ell) = \top$, otherwise.

5 Semiring-enabled Preprocessing

Our last but not least addition to SharpSAT-TD concerns its preprocessing. The general idea of preprocessing is to modify the input CNF so that it is easier to solve afterwards but leads to the same result.

Here, SharpSAT-TD uses five main techniques interleaved with regular unit propagation and clause subsumption:

1. Failed Literal Probing (Lynce and Silva 2003),
2. Vivification (Piette, Hamadi, and Sais 2008),
3. Sparsification, i.e., removal of entailed clauses,
4. Equivalent Literal Substitution (Bacchus 2002), and
5. Variable Elimination (Lagniez, Lonca, and Marquis 2016).

Techniques 1. to 3. are applicable in the general setting of weighted model counting over semirings, and Korhonen and Järvisalo modified 4. in such a manner that the same holds. However, 5. only applies to the unweighted case without changing the result.

We leave 1. to 3. as they are and consider changes to 4. and 5.

5.1 Changes to Equivalent Literal Substitution

The idea behind here is the following: if there are two variables $v_1, v_2 \in V$ in a CNF $\mathcal{C}$ over V such that for one of the literals $\ell = v_2, \neg v_2$ it holds that for every model $\mathcal{I}$ of $\mathcal{C}$ we have $v_1 \in \mathcal{I}$ iff $\ell \in \mathcal{I}$, then we can replace every occurrence of v_1 in $\mathcal{C}$ by ℓ and every occurrence of $\neg v_1$ in $\mathcal{C}$ by $\neg \ell$.

The resulting CNF $\mathcal{C}[v_1 \leftarrow \ell]$ over $V \setminus \{v_1\}$ has the same number of models and can be easier to solve.[4] However, when we perform *weighted* model counting, this standard strategy usually leads to a different result since we lose the information about the weight of the removed variable v_1.

Therefore, Korhonen and Järvisalo do not use the CNF $\mathcal{C}[v_1 \leftarrow \ell]$ over $V \setminus \{v_1\}$ but the CNF $\mathcal{C}[v_1 \leftarrow \ell] \cup \{v_1 \vee$

[4]To ensure easier solving, Korhonen and Järvisalo only apply this technique to variables that occur together in some clause, as this is guaranteed to prevent an increase of the treewidth.

$\neg\ell, \neg v_1 \vee \ell\}$ over V. This ensures that (i) the information that v_1 and ℓ are equivalent is easily available to the solver and (ii) the weighted model count remains unchanged regardless of the semiring.

We slightly modify this approach by using preprocessing in a weight-aware manner.

Lemma 8. *Let $A = (\mathcal{C}, \mathcal{S}, \alpha)$ be an AMC instance, where $\mathcal{C}$ is a CNF over V. If for $v_1, v_2 \in V$ it holds that v_1 is equivalent to $l \in \{v_2, \neg v_2\}$, then for $A[v_1 \leftarrow \ell] = (\mathcal{C}[v_1 \leftarrow \ell], \mathcal{C}, \alpha[v_1 \leftarrow \ell])$, where*

$$\alpha[v_1 \leftarrow \ell](\ell^*) = \begin{cases} \alpha(v_1)\otimes\alpha(\ell) & \text{if } \ell^* = \ell \\ \alpha(\neg v_1)\otimes\alpha(\neg\ell) & \text{if } \ell^* = \neg\ell \\ \alpha(\ell^*) & \text{otherwise} \end{cases}$$

it holds that $AMC(A) = AMC(A[v_1 \leftarrow \ell])$.

Thus, we can use the standard strategy for merging equivalent variables, as long as we update the weights accordingly.

We note that it would be unreasonable to expect a significant increase in the performance when using the weight-updating variant of equivalent literal substitution instead of SharpSAT-TD's original variant. This is because the size decrease of the CNF is very marginal. Reasoning does not become easier since deciding an equivalent variable leads to an assignment to the others by unit propagation, and also, the number of avoided multiplications is low.

However, while this change is inconsequential for solving performance, it has a positive effect when we use SharpSAT-TD for knowledge compilation. Consider the AMC instance $KC(\mathcal{C})[v_1 \leftarrow \ell]$. Here, the label of ℓ (resp. $\neg\ell$) after substitution is $v_1 \wedge \ell$ (resp. $\neg v_1 \wedge \neg\ell$). This reduces the size of the compiled sd-DNNF by reusing internal nodes instead of referring to the leaf nodes of all equivalent variables. Since we compile smooth circuits, we can avoid one edge in *every* branch of the circuit per equivalence.

5.2 Changes to Bounded Variable Elimination

Variable elimination is based on binary resolution. Given two clauses $C_1 = \{x, v_1, \dots, v_n\}$ and $C_2 = \{\neg x, w_1, \dots, w_m\}$, their resolvent (on x) is the clause $C_1 \odot C_2 = \{v_1, \dots, v_n, w_1, \dots, w_m\}$. Given a CNF $\mathcal{C}$, we denote by $\mathcal{C}_\ell$ the set of clauses that contain ℓ. Then we can define $\mathcal{C}_x^{res}$ as the set of all resolvents on x given a CNF $\mathcal{C}$ via

$$\mathcal{C}_x^{res} = \{C_1 \odot C_2 \mid C_1 \in \mathcal{C}_x, C_2 \in \mathcal{C}_{\neg x}\}$$

The general idea behind (bounded) variable elimination is the following (Eén and Biere 2005): let $v \in V$ be a variable in a CNF $\mathcal{C}$ over V. If it is expected that solving $\exists\{v\}\mathcal{C}$ over $V \setminus \{v\}$ is easier than solving $\mathcal{C}$, then we compute $\exists\{v\}\mathcal{C}$ as $\mathcal{C} \setminus (\mathcal{C}_x \cup \mathcal{C}_{\neg x}) \cup \mathcal{C}_x^{res}$. A usual strategy to heuristically check whether this makes solving easier is to bound the increase in the number of (non-tautological) clauses that are added this way.

While this is satisfiability preserving (Eén and Biere 2005), it is easy to see that, in general, it does not preserve the number of models.

Example 4. *The empty CNF $\mathcal{C} = \emptyset$ over the set of variables $V = \{v\}$ has 2 models, while $\exists\{v\}\mathcal{C}$ only has one model.*

However, Lagniez, Lonca, and Marquis showed that if a variable v is defined by V with respect to a CNF $\mathcal{C}$ over $V \cup \{v\}$, then $\mathcal{C}$ and $\exists\{v\}\mathcal{C}$ have the same number of models. This insight enabled (bounded) variable elimination also for model counting and was shown to lead to significant performance improvements (Lagniez, Lonca, and Marquis 2016).

Thus, using variable elimination also for *weighted* model counting would be desirable. Unfortunately, this is not generally possible even if the eliminated variable is defined.

Example 5 (cont'd.)**.** *Let $A = (\mathcal{C}_{run}, \mathbb{R}, \alpha)$ be an AMC instance, where $\alpha(a) = 0.2, \alpha(\neg a) = 0.8$, and $\alpha(\ell) = 1$, otherwise. Then $AMC(A) = 0.2 + 0.2 + 0.2 + 0.8 = 1.4$. But while a is defined by $\{b, c\}$ with respect to $\mathcal{C}$, there is no obvious way how we can modify the weight function α such that for $B = (\exists\{a\}\mathcal{C}_{run}, \mathbb{R}, \alpha)$ it holds that $AMC(B) = 1.4$.*

We use the following modified conditions to perform a variant of variable elimination that preserves the weighted model count.

Lemma 9. *Let $A = (\mathcal{C}, \mathcal{S}, \alpha)$ be an AMC instance over the variables in $V \cup \{x\}$ such that*

- *x is defined by V with respect to $\mathcal{C}$ or $\alpha(x)\oplus\alpha(\neg x) = \alpha(x)$ and*
- *$\alpha(x) = \alpha(\neg x)$.*

Then for $B = (\{x\} \cup \exists\{x\}\mathcal{C}, \mathcal{S}, \alpha)$ it holds that $AMC(A) = AMC(B)$.

Here, we do not completely eliminate the variable x but add it as a unit clause to ensure that its weight is included.

Proof. We consider the following three disjoint sets

$$M_b = \{\mathcal{I} \subseteq V \mid \mathcal{I} \cup \{x\} \models \mathcal{C} \text{ and } \mathcal{I} \models \mathcal{C}\}$$
$$M_x = \{\mathcal{I} \subseteq V \mid \mathcal{I} \cup \{x\} \models \mathcal{C} \text{ and } \mathcal{I} \not\models \mathcal{C}\}$$
$$M_{\neg x} = \{\mathcal{I} \subseteq V \mid \mathcal{I} \cup \{x\} \not\models \mathcal{C} \text{ and } \mathcal{I} \models \mathcal{C}\}$$

Then, $AMC(A)$ is equal to

$$\begin{aligned}
& \textstyle\bigoplus_{\mathcal{I} \subseteq V \cup \{x\}, \mathcal{I} \models \mathcal{C}} \bigotimes_{v \in \mathcal{I}} \alpha(v) \otimes \bigotimes_{v \in V \cup \{x\} \setminus \mathcal{I}} \alpha(\neg v) \\
= \; & \alpha(x) \otimes \textstyle\bigoplus_{\mathcal{I} \in M_x} \bigotimes_{v \in \mathcal{I}} \alpha(v) \otimes \bigotimes_{v \in V \setminus \mathcal{I}} \alpha(\neg v) \\
& \oplus \alpha(\neg x) \otimes \textstyle\bigoplus_{\mathcal{I} \in M_{\neg x}} \bigotimes_{v \in \mathcal{I}} \alpha(v) \otimes \bigotimes_{v \in V \setminus \mathcal{I}} \alpha(\neg v) \\
& \oplus (\alpha(x) \oplus \alpha(\neg x)) \otimes \textstyle\bigoplus_{\mathcal{I} \in M_b} \bigotimes_{v \in \mathcal{I}} \alpha(v) \otimes \bigotimes_{v \in V \setminus \mathcal{I}} \alpha(\neg v) \\
= \; & \alpha(x) \otimes \textstyle\bigoplus_{\mathcal{I} \in M_x} \bigotimes_{v \in \mathcal{I}} \alpha(v) \otimes \bigotimes_{v \in V \setminus \mathcal{I}} \alpha(\neg v) \\
& \oplus \alpha(x) \otimes \textstyle\bigoplus_{\mathcal{I} \in M_{\neg x}} \bigotimes_{v \in \mathcal{I}} \alpha(v) \otimes \bigotimes_{v \in V \setminus \mathcal{I}} \alpha(\neg v) \\
& \oplus \alpha(x) \otimes \textstyle\bigoplus_{\mathcal{I} \in M_b} \bigotimes_{v \in \mathcal{I}} \alpha(v) \otimes \bigotimes_{v \in V \setminus \mathcal{I}} \alpha(\neg v) \\
= \; & \alpha(x) \otimes \textstyle\bigoplus_{\mathcal{I} \models \exists\{x\}\mathcal{C}} \bigotimes_{v \in \mathcal{I}} \alpha(v) \otimes \bigotimes_{v \in V \setminus \mathcal{I}} \alpha(\neg v)
\end{aligned}$$

Here, the first equality holds by definition, the second since $\alpha(x) = \alpha(\neg x)$ and (i) x is defined and M_b is empty or (ii) $\alpha(x)\oplus\alpha(\neg x) = \alpha(x)$. The last equation holds because the set of models of $\exists\{x\}\mathcal{C}$ is the disjoint union of $M_b, M_x, M_{\neg x}$.

Since the last expression equals $AMC(B)$, we are done. □

We see that when there are variables v such that their positive and negative literals have the same weight, we have a chance of applying variable elimination. Clearly, if the weights of literals are chosen randomly, this is unlikely. However, recall our discussion of existentially quantified variables in Section 4. Here, we noted that many logical formalisms require auxiliary and notably *defined* variables to avoid an exponential size increase during the translation to CNF/AMC, as in the case of Tseitin's transformation. It is well-known that in order to preserve the value of weighted model counting, we need to assign these auxiliary variables weight one for both the positive and negative literal.[5]

This insight opens up new possibilities to apply variable elimination as a preprocessing in AMC. Notably, it even means that if there are existentially quantified variables in a knowledge compilation instance, we may always eliminate them during preprocessing without changing the correctness of the result. These variables x have $\alpha(x) = \alpha(\neg x) = \top$ and thus satisfy $\alpha(x) \oplus \alpha(\neg x) = \top = \alpha(x)$, regardless of whether they are defined.

It is important to note that variable elimination is not always beneficial to solving performance. Recall, for example that the aim of Tseitin transformation is precisely the opposite, i.e., it adds auxiliary variables to ensure small CNF size and consequently (likely) faster solving. In our implementation, we therefore only add the conditions of Lemma 9 to allow for variable elimination in the general setting of AMC but rely on SharpSAT-TD's heuristic for choosing variables to eliminate. Notably, this heuristic is guaranteed not to lead to an increase in the treewidth.

6 Experimental Evaluation

Here, we evaluate the utility of using our modified version of SharpSAT-TD for knowledge compilation.[6] That is, we aim to answer the following two questions:

Q1 How does knowledge compilation with SharpSAT-TD compare to other state of the art knowledge compilers?

Q2 What is the impact of enabling variable elimination for knowledge compilation with existentially quantified variables?

For **Q1**, we expect that the performance of SharpSAT-TD transfers from (weighted) model counting, where it outperformed other knowledge compilation based solvers in recent editions of the Model Counting Competition. Additionally, it is interesting to see how large the circuits compiled with SharpSAT-TD are compared to that of other solvers.

Regarding **Q2**, it is not clear whether the utility of variable elimination for model counting transfers to knowledge compilation with existentially quantified variables. In fact, eliminating too many variables may even be detrimental since, as we discussed previously, these auxiliary variables may have been added to make solving easier.

6.1 Setup

To answer our questions, we used the following setup.

[5]For knowledge compilation, the weight one is $\top$, corresponding to existential quantification.

[6]Most data is online: github.com/raki123/KC-benchmarking.

Benchmark Platform We ran all solvers on a cluster consisting of 12 nodes. Each of them is equipped with two Intel Xeon E5-2650 CPUs of 2.2 GHz clock speed and access to 256 GB shared RAM under Ubuntu 16.04.1 LTS powered on kernel 4.4.0-139 with no hyperthreading. Per instance, we always use a memory limit of 32GB and a time limit of 1800 seconds on a single core.

Benchmark Instances We rely on two benchmark sets. For **Q1**, we used a standard set of CNFs for knowledge compilation[7]. For **Q2**, we need to associate a realistic subset of existentially quantified variables with the CNF. Therefore, we introduce a new set of benchmarks using two tools to translate (probabilistic) logic programs to CNFs (Janhunen and Niemelä 2011; Eiter, Hecher, and Kiesel 2021) on standard benchmarks from probabilistic logic programming. For the resulting CNFs, we know (i) which of the variables are auxiliary and can be existentially quantified and (ii) know that the auxiliary variables are defined in terms of the remaining ones since the translations are (weighted) model count preserving.

Solver Configurations In terms of knowledge compilers, we use the following configurations:

- d4 (Lagniez and Marquis 2017), with default options but minimal modifications to ensure the compilation of smooth circuits in order to have a fair circuit size comparison.[8]
- c2d (Darwiche 2004) version 2.20, with options "-smooth_all", "-reduce", "-cache_size 32000", and "-dt_in dtreefile" corresponding to smooth circuits, reduced size circuits, 32 Gb cache size, and a custom dtree, respectively. We generated the dtree from a tree decomposition using Korhonen and Järvisalo's methodology, which they showed to improve the performance of c2d.
- The knowledge compilation-enabled version of SharpSAT-TD, with options "-decot 10", "-decow 10000", and "-cs 16000" corresponding to 10 seconds of time to generate a tree decomposition, a decomposition weight of 10000 and a cache size of 16 Gb.[9]

On **Q1**, we ran all solvers and compared the time needed for compilation as well as the size of the circuits. On **Q2**, we did not use d4, since it does not allow for existential quantification. Both SharpSAT-TD and c2d do, so we compared standard compilation and compilation with existentially quantified variables, with adapted variable elimination in the case of SharpSAT-TD. For c2d, we replaced option "-smooth_all" with "-smooth" and added option "-exist existfile" to ensure that the auxiliary variables are existentially quantified.

[7]https://www.cril.univ-artois.fr/KC/benchmarks.html without the "Handmade" instances, which all contained empty clauses.

[8]The modifications are restricted to writing the circuit to file and can be found at https://github.com/raki123/d4/

[9]The authors recommend specifying half the available memory.

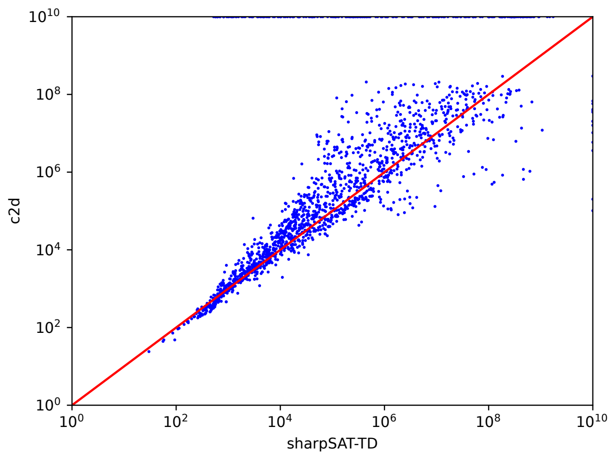

Figure 3a. Q1: Scatter plot comparing the number of edges in the compiled sd-DNNFs produced by SharpSAT-TD and c2d.

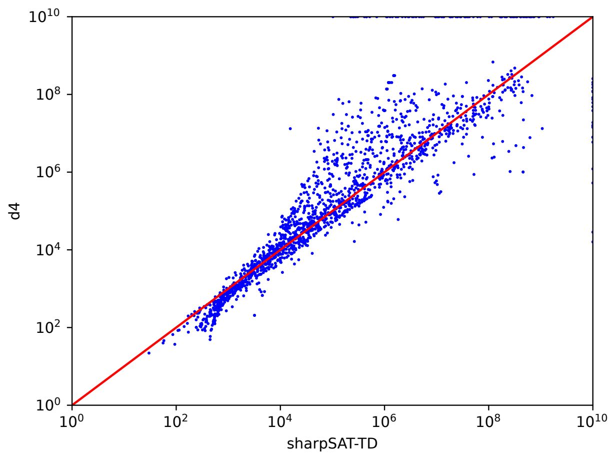

Figure 3b. Q1: Scatter plot comparing the number of edges in the compiled sd-DNNFs produced by SharpSAT-TD and d4.

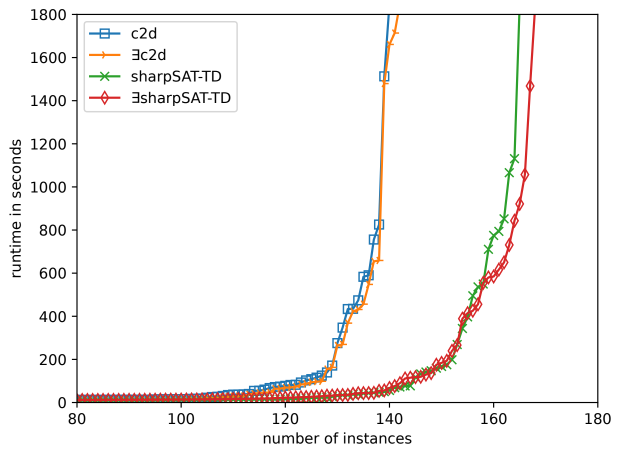

Figure 3c. Q2: Runtime on all instances, for (∃)c2d and (∃)SharpSAT-TD.

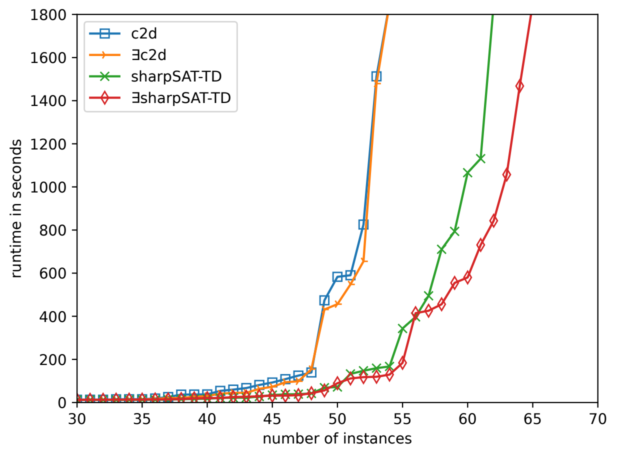

Figure 3d. Q2: Runtime on instances encoded via Janhunen and Niemelä (2011), for (∃)c2d and (∃)SharpSAT-TD.

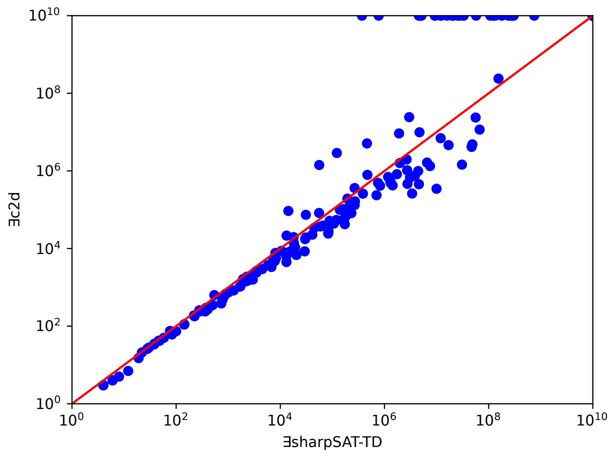

Figure 3e. Q2: Number of edges in sd-DNNFs produced by ∃SharpSAT-TD and ∃c2d.

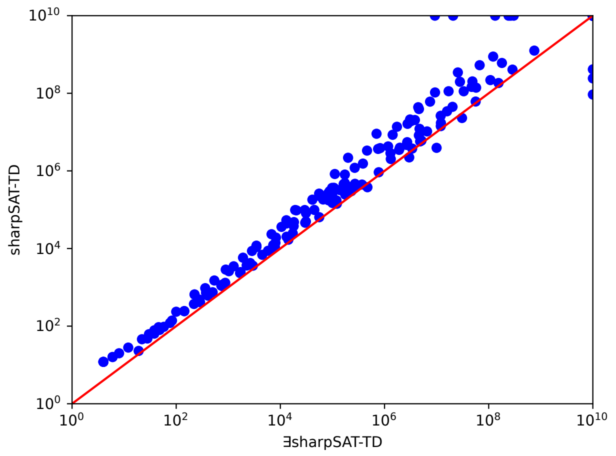

Figure 3f. Q2: Number of edges in sd-DNNFs produced by ∃SharpSAT-TD and SharpSAT-TD.

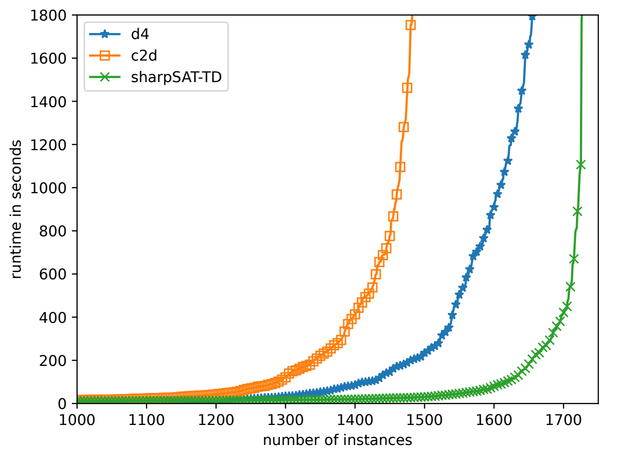

Figure 4: Q1: Cactus plot showing the number of solved instances after a given runtime for c2d, d4, and SharpSAT-TD. Only every fifth point is marked.

6.2 Results

For the results, we assigned instances a runtime of 1800 seconds, and a circuit size of 10^{10} edges if they did not finish within the time or memory limit.

The results for **Q1** are shown in Figures 4, 3a, and 3b. In Figure 4, we see that SharpSAT-TD solves the most instances (1727), followed by d4 (1656) and c2d (1483). This is in line with the results of Lagniez and Marquis (2017) and the performance of the respective solvers for (weighted) model counting in the Model Counting Competitions.

Additionally, we compared the number of edges in the sd-DNNF circuits produced by the different solvers. Figures 3a and 3b show similar behavior for d4 and c2d. While there are also a lot of instances where the circuits produced by SharpSAT-TD are slightly larger, there are only a few instances with significantly larger circuits. Additionally, there are many instances where compilation with SharpSAT-TD leads to significantly smaller circuits. This shows that SharpSAT-TD's performance is not only based on good engineering but also due to its tree decomposition-guided heuristic for variable selection.

The results for **Q2** are not as clear cut. In 3c, we see the runtime comparison, where ∃SharpSAT-TD and ∃c2d denote the runs of SharpSAT-TD and c2d with existentially quantified variables. As expected (∃)SharpSAT-TD is faster than ∃c2d. However, the benefit of ∃ is only moderate. For ∃c2d, this is expected as it only affects the circuit size that needs to be written to a file and smoothed. But for ∃SharpSAT-TD, it also enables variable elimination, which can lead to performance improvements.

We took a closer look (see Figure 3d) and found that while variable elimination does not have any effect on the CNFs generated according to Eiter, Hecher, and Kiesel (2021), it leads to a notable runtime decrease and more solved instances on the CNFs generated according to Janhunen and Niemelä (2011). This makes sense since the encoding of Eiter, Hecher, and Kiesel (2021) adds auxiliary variables to achieve low treewidth, whereas the encoding by Janhunen and Niemelä (2011) adds auxiliary variables to achieve small CNF sizes. Since SharpSAT-TD's preprocessing aims to eliminate variables in order to decrease the treewidth, it is, therefore, likely to have more success on encodings that are not optimized for low treewidth already. This shows that while, unsurprisingly, variable elimination does not always lead to improved performance, its utility at least partially transfers to the setting of knowledge compilation with auxiliary variables.

Last but not least, we see in Figures 3e and 3f that, as expected, the existential quantification of auxiliary variables leads to smaller circuits. Interestingly, in this setting, the circuits produced by ∃c2d are often slightly smaller than those produced by ∃SharpSAT-TD.

7 Conclusion

We showed how SharpSAT-TD can be used as a knowledge compiler, integrated the use of existentially quantified variables, and adapted its preprocessor to enable its simplification capabilities over general semirings and specifically to make use of variable elimination for knowledge compilation with existentially quantified variables.

Using SharpSAT-TD's capabilities for algebraic model counting significantly simplified our work but also led to an interesting theoretical side result. Namely, it shows that (i) we can use our strategy for knowledge compilation on any algebraically agnostic AMC solver and (ii) the runtime of any algebraically agnostic AMC solver is lower bounded by sd-DNNF size.

In our experimental evaluation, we saw that the high performance of SharpSAT-TD for (weighted) model counting transfers to the knowledge compilation setting. Additionally, it can often lead to smaller sd-DNNFs, although not in all cases. While enabling SharpSAT-TD's preprocessing for knowledge compilation with existentially quantified variables does not always improve the performance, it also does not decrease it. It can have a notable positive effect when the encoding is not already optimized for low treewidth.

Thus, our changes to SharpSAT-TD provide the community with a tool that is not only useful for efficient knowledge compilation but also comes with additional features that can improve the compilation performance for logical theories that first need to be translated to CNF.

Acknowledgements

This work has been supported by the Austrian Science Fund (FWF) grant W1255-N23.

References

Amarilli, A.; Bourhis, P.; Jachiet, L.; and Mengel, S. 2017. A circuit-based approach to efficient enumeration. In Chatzigiannakis, I.; Indyk, P.; Kuhn, F.; and Muscholl, A., eds., *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, volume 80 of *LIPIcs*, 111:1–111:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

Amarilli, A.; Capelli, F.; Monet, M.; and Senellart, P. 2020. Connecting knowledge compilation classes and width parameters. *Theory Comput. Syst.* 64(5):861–914.

Bacchus, F. 2002. Enhancing davis putnam with extended binary clause reasoning. In Dechter, R.; Kearns, M. J.; and Sutton, R. S., eds., *Proceedings of the Eighteenth National Conference on Artificial Intelligence and Fourteenth Conference on Innovative Applications of Artificial Intelligence, July 28 - August 1, 2002, Edmonton, Alberta, Canada*, 613–619. AAAI Press / The MIT Press.

Bart, A.; Koriche, F.; Lagniez, J.; and Marquis, P. 2016. An improved CNF encoding scheme for probabilistic inference. In Kaminka, G. A.; Fox, M.; Bouquet, P.; Hüllermeier, E.; Dignum, V.; Dignum, F.; and van Harmelen, F., eds., *ECAI 2016 - 22nd European Conference on Artificial Intelligence, 29 August-2 September 2016, The Hague, The Netherlands - Including Prestigious Applications of Artificial Intelligence (PAIS 2016)*, volume 285 of *Frontiers in Artificial Intelligence and Applications*, 613–621. IOS Press.

Beame, P., and Liew, V. 2015. New limits for knowledge compilation and applications to exact model counting. In Meila, M., and Heskes, T., eds., *Proceedings of the Thirty-First Conference on Uncertainty in Artificial Intelligence, UAI 2015, July 12-16, 2015, Amsterdam, The Netherlands*, 131–140. AUAI Press.

Biere, A.; Heule, M.; van Maaren, H.; and Walsh, T., eds. 2021. *Handbook of Satisfiability - Second Edition*, volume 336 of *Frontiers in Artificial Intelligence and Applications*. IOS Press.

Capelli, F., and Mengel, S. 2019. Tractable QBF by knowledge compilation. In Niedermeier, R., and Paul, C., eds., *36th International Symposium on Theoretical Aspects of Computer Science, STACS 2019, March 13-16, 2019, Berlin, Germany*, volume 126 of *LIPIcs*, 18:1–18:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

Chavira, M., and Darwiche, A. 2008. On probabilistic inference by weighted model counting. *Artif. Intell.* 172(6-7):772–799.

Darwiche, A., and Marquis, P. 2002. A knowledge compilation map. *J. Artif. Intell. Res.* 17:229–264.

Darwiche, A. 2004. New advances in compiling CNF into decomposable negation normal form. In de Mántaras, R. L., and Saitta, L., eds., *Proceedings of the 16th Eureopean Conference on Artificial Intelligence, ECAI'2004, including Prestigious Applicants of Intelligent Systems, PAIS 2004, Valencia, Spain, August 22-27, 2004*, 328–332. IOS Press.

De Raedt, L.; Kimmig, A.; and Toivonen, H. 2007. Problog: A probabilistic prolog and its application in link discovery. In Veloso, M. M., ed., *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007*, 2462–2467.

Eén, N., and Biere, A. 2005. Effective preprocessing in SAT through variable and clause elimination. In Bacchus, F., and Walsh, T., eds., *Theory and Applications of Satisfiability Testing, 8th International Conference, SAT 2005, St. Andrews, UK, June 19-23, 2005, Proceedings*, volume 3569 of *Lecture Notes in Computer Science*, 61–75. Springer.

Eiter, T.; Hecher, M.; and Kiesel, R. 2021. Treewidth-aware cycle breaking for algebraic answer set counting. In Bienvenu, M.; Lakemeyer, G.; and Erdem, E., eds., *Proceedings of the 18th International Conference on Principles of Knowledge Representation and Reasoning, KR 2021, Online event, November 3-12, 2021*, 269–279.

Fierens, D.; Van den Broeck, G.; Renkens, J.; Shterionov, D. S.; Gutmann, B.; Thon, I.; Janssens, G.; and De Raedt, L. 2015. Inference and learning in probabilistic logic programs using weighted boolean formulas. *Theory Pract. Log. Program.* 15(3):358–401.

Hecher, M. 2022. Treewidth-aware reductions of normal ASP to SAT - is normal ASP harder than SAT after all? *Artif. Intell.* 304:103651.

Huang, J., and Darwiche, A. 2005. DPLL with a trace: From SAT to knowledge compilation. In Kaelbling, L. P., and Saffiotti, A., eds., *IJCAI-05, Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence, Edinburgh, Scotland, UK, July 30 - August 5, 2005*, 156–162. Professional Book Center.

Janhunen, T., and Niemelä, I. 2011. Compact translations of non-disjunctive answer set programs to propositional clauses. In Balduccini, M., and Son, T. C., eds., *Logic Programming, Knowledge Representation, and Nonmonotonic Reasoning - Essays Dedicated to Michael Gelfond on the Occasion of His 65th Birthday*, volume 6565 of *Lecture Notes in Computer Science*, 111–130. Springer.

Kimmig, A.; Van den Broeck, G.; and De Raedt, L. 2017. Algebraic model counting. *J. Appl. Log.* 22:46–62.

Korhonen, T., and Järvisalo, M. 2021. Integrating tree decompositions into decision heuristics of propositional model counters. In *27th International Conference on Principles and Practice of Constraint Programming (CP 2021)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik.

Lagniez, J., and Marquis, P. 2017. An improved decision-dnnf compiler. In Sierra, C., ed., *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, 667–673. ijcai.org.

Lagniez, J.-M.; Lonca, E.; and Marquis, P. 2016. Improving model counting by leveraging definability. In *IJCAI*, 751–757.

Lynce, I., and Silva, J. P. M. 2003. Probing-based preprocessing techniques for propositional satisfiability. In *15th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2003), 3-5 November 2003, Sacramento, California, USA*, 105. IEEE Computer Society.

Oztok, U., and Darwiche, A. 2017. On compiling dnnfs without determinism. *CoRR* abs/1709.07092.

Piette, C.; Hamadi, Y.; and Sais, L. 2008. Vivifying propositional clausal formulae. In Ghallab, M.; Spyropoulos, C. D.; Fakotakis, N.; and Avouris, N. M., eds., *ECAI 2008 - 18th European Conference on Artificial Intelligence, Patras, Greece, July 21-25, 2008, Proceedings*, volume 178 of *Frontiers in Artificial Intelligence and Applications*, 525–529. IOS Press.

Riguzzi, F., and Swift, T. 2011. The PITA system for logical-probabilistic inference. In Muggleton, S. H., and Watanabe, H., eds., *Latest Advances in Inductive Logic Programming, ILP 2011, Late Breaking Papers, Windsor Great Park, UK, July 31 - August 3, 2011*, 79–86. Imperial College Press / World Scientific.

Ritter, M. 2022. Evaluation techniques for algebraic answer set counting over idempotent semirings. In *Diploma Thesis, Technische Universität Wien*. reposiTUm.

Sharma, S.; Gupta, R.; Roy, S.; and Meel, K. S. 2018. Knowledge compilation meets uniform sampling. In Barthe, G.; Sutcliffe, G.; and Veanes, M., eds., *LPAR-22. 22nd International Conference on Logic for Programming, Artificial Intelligence and Reasoning, Awassa, Ethiopia, 16-21 November 2018*, volume 57 of *EPiC Series in Computing*, 620–636. EasyChair.

Van den Broeck, G., and Suciu, D. 2017. Query processing on probabilistic data: A survey. *Found. Trends Databases* 7(3-4):197–341.