New Compilation Languages Based on Restricted Weak Decomposability

Petr Illner

Department of Theoretical Computer Science and Mathematical Logic, Faculty of Mathematics and Physics
Charles University, Czech Republic
illner3@gmail.com

Abstract

This paper introduces two new compilation languages restricting weak decomposable negation normal form (wDNNF) circuits and integrates them into the knowledge compilation map. Positive (resp. negative) wDNNF circuits restrict wDNNF circuits so that each variable shared among the inputs of a conjunction node can only have positive (resp. negative) occurrences in that subcircuit. Unlike wDNNF circuits, pwDNNF (resp. nwDNNF) circuits satisfy the maximum (resp. minimum) cardinality query. We present a compiler for converting CNF formulae into pwDNNF and nwDNNF circuits by extending Bella — the state-of-the-art compiler for wDNNF circuits. We introduce a new caching scheme, called Cara, that exploits isomorphism. Using that scheme, we show a new compilation method based on copying subcircuits, which may significantly speed up compilations at the expense of increasing circuit sizes. Our experiments demonstrate that nwDNNF circuits are suitable for computing most probable explanations (MPEs) in two-layer Bayesian networks (BNs) with large domains.

Introduction

Knowledge representation is a research area that studies representations (also called languages) of propositional theories. *The knowledge compilation map* (Darwiche and Marquis 2002) consists of languages that have been studied from the following perspectives: (1) Universality: Can the language represent any Boolean function? (2) Succinctness (Gogic et al. 1995): How compactly can the language represent Boolean functions with respect to other languages? (3) Tractable operations: What operations (that is, queries and transformations) does the language satisfy?

This paper focuses only on a particular class of languages: Circuits based on negation normal form (NNF) circuits enriched with additional properties. The decomposability property (Darwiche 1999) or its variants (that is, weak decomposability (Akshay et al. 2018) or structured decomposability (Pipatsrisawat and Darwiche 2008)) are sufficient for the following queries: Consistency check, clausal entailment check, and model enumeration. Therefore, decomposable NNF (DNNF) circuits (Darwiche 1999), weak DNNF (wDNNF) circuits (Akshay et al. 2018; Illner and Kučera 2024), and structured DNNF (SDNNF) circuits (Pipatsrisawat and Darwiche 2008) are appropriate for such queries. These circuit types have been thoroughly studied in terms of the knowledge compilation map. The other circuit types that satisfy the mentioned queries (for example, deterministic DNNF (d-DNNF) circuits (Darwiche 2001b), decision-DNNF circuits (Huang and Darwiche 2007), SDDs (Darwiche 2011), or OBDDs (Bryant 1986)) also satisfy other unnecessarily more complex queries (for example, the model counting query) at the expense of succinctness, meaning that circuits can be exponentially bigger.

Knowledge compilation is concerned with transforming a given propositional theory (usually a CNF formula) into a circuit of an appropriate type. For this purpose, (knowledge) compilers are used. The existence of a compiler is a vital part of a language. For wDNNF circuits, there is a compiler called Bella (Illner and Kučera 2024). To our knowledge, there is no compiler that compiles (directly) into DNNF or SDNNF circuits. Hence, instead of compiling directly into DNNF circuits, decision-DNNF circuits, which are strictly less succinct, are used. The well-known compilers for decision-DNNF circuits are C2D (Darwiche 2004), D4 (Lagniez and Marquis 2017), DSHARP (Muise et al. 2010), and SharpSAT-TD (Kiesel and Eiter 2023).

Besides theoretical aspects, circuits can also be employed in applications. The main idea is divided into two phases: Offline and online. Let us assume we have a CNF formula that represents the problem we wish to solve. The CNF formula is compiled into a circuit of an appropriate type in the offline phase. This phase is time-consuming but is done only once. In the online phase, we solve multiple instances of that problem in polynomial time in the size of that circuit using an appropriate query that is tractable for that circuit type and is suitable for solving that problem. Such knowledge compilation-based approaches are often considered state-of-the-art, even if the circuit is used only once.

The maximum and minimum cardinality queries are known to be tractable for DNNF circuits (Darwiche 2001a), and we prove that they are intractable for wDNNF circuits. Therefore, if we are interested in any of these queries, which are essential for many applications, we must use decision-DNNF circuits. Addressing this issue, we introduce two new restricted variants of wDNNF circuits, namely pwDNNF and nwDNNF circuits, that satisfy these queries.

The main contributions of this paper are as follows:

- We introduce pwDNNF and nwDNNF circuits and study them in terms of the knowledge compilation map.
- We relate the new circuit types to wDNNF and DNNF circuits with respect to succinctness.
- We show a new caching scheme that exploits isomorphism and a new compilation method based on that caching scheme.
- We present our extended variant of Bella for pwDNNF and nwDNNF circuits.
- We demonstrate that nwDNNF circuits are suitable for computing MPEs in two-layer BNs with large domains.

Basic Definitions

In this section, we introduce notation and preliminaries. We assume that the reader is familiar with the following terminologies related to probabilistic reasoning: *Bayesian networks* (BNs), *conditional probability tables* (CPTs), and *variable instantiations*. For the definitions, see (Park 2002).

Propositional Logic

We mention only some crucial definitions. For more, see Section "Propositional Logic" in (Illner and Kučera 2024).

Let X be a set of *variables*. A *literal* is a variable $x \in X$ (a *positive literal*) or its complement $\neg x$ (a *negative literal*). A *clause* is a disjunction of literals. A *weighted clause* is a clause with an associated non-negative weight. We denote the weight of a weighted clause as a superscript (for example, $(x_i \vee \neg x_j)^{w_c}$). A (*weighted*) *conjunctive normal form* (CNF) *formula* is a conjunction of (weighted) clauses. A CNF formula φ is *positive* (resp. *negative*) if every clause of φ contains only positive (resp. negative) literals.

Negation Normal Form Circuits and Variants

A *negation normal form* (NNF) *circuit* is a rooted directed acyclic graph in which each inner node is a conjunction ($\wedge$) or disjunction ($\vee$), and each leaf is a literal, *True*, or *False*.

For a node α, we denote by $\mathrm{Vars}(\alpha)$ (resp. $|\mathrm{Vars}(\alpha)|$) the set (resp. the number) of variables that occur in the subcircuit rooted at α. A variable x is positive (resp. negative) in a subcircuit if $\neg x$ (resp. x) does not occur in that subcircuit. The size of a circuit Δ, denoted by $|\Delta|$, is measured by the number of edges in that circuit. Sometimes, we handle a node α of a circuit as the subcircuit rooted at α.

The *cardinality* of a model is the number of variables that are set to *True*. The *minimum* (resp. *maximum*) *cardinality* of an NNF circuit Δ, denoted by $\mathrm{MinCard}(\Delta)$ (resp. $\mathrm{MaxCard}(\Delta)$), is the minimum (resp. maximum) cardinality of all its models. If Δ has no model, the minimum (resp. maximum) cardinality is defined as ∞ (resp. $-\infty$).

Decomposable NNF Circuits A conjunction node α is *decomposable* (Darwiche 1999) if no variable is shared among its child nodes (that is, $\mathrm{Vars}(\mathrm{child}_i) \cap \mathrm{Vars}(\mathrm{child}_j) = \emptyset$ for any pair of different child nodes child_i and child_j of α). An NNF circuit in which all the conjunction nodes are decomposable is named a *decomposable NNF* (DNNF) *circuit* (Darwiche 1999).

Weak DNNF Circuits A conjunction node α is *weak decomposable* (Akshay et al. 2018) if each variable $x \in \mathrm{Vars}(\alpha)$ that is shared by at least two child nodes of α is either positive or negative in the subcircuit rooted at α. An NNF circuit in which all the conjunction nodes are weak decomposable is named a *weak DNNF* (wDNNF) *circuit* (Akshay et al. 2018).

Problems

The *weighted MaxSAT* problem is defined as follows: Given a weighted CNF formula, find an assignment that maximises the sum of the weights of the satisfied clauses. Sometimes, the clauses of that formula are divided into two groups: *Hard clauses*, which must always be satisfied, and *soft clauses*.

The *weighted minimum cardinality* ($\mathrm{wMinCard}$) *problem* is defined as follows: Given an NNF circuit Δ and a non-negative weight for each variable, find a model of Δ that minimises the sum of the weights of the variables that are set to *True*. The weighted minimum cardinality problem on CNF formulae is called the *minimum-cost satisfiability* (MinCostSat) *problem* (Li 2004). In the *weighted maximum cardinality* ($\mathrm{wMaxCard}$) *problem*, we wish to maximise the sum of the weights of the variables that are set to *True*.

The *most probable explanation* (MPE) *problem* is defined as follows: Given a Bayesian network and some evidence, find a variable instantiation of the remaining variables (that is, an *MPE instantiation*) with the highest probability given that evidence (that is, the *MPE probability*). The decision version of this problem is $\mathcal{NP}$-complete (Littman 1999).

Positive and Negative wDNNF Circuits

In this section, we define two new restricted variants of weak decomposability (that is, positive and negative weak decomposability) and the new circuit types, namely pwDNNF and nwDNNF circuits, which are based on them.

Definition 1. *A conjunction node α is* positive weak decomposable *(resp.* negative weak decomposable*) if every variable that is shared by at least two child nodes of α is positive (resp. negative) in the subcircuit rooted at α.*

Definition 2. *An NNF circuit in which all the conjunction nodes are positive (resp. negative) weak decomposable is named a* positive wDNNF *(pwDNNF)* circuit *(resp.* negative wDNNF *(nwDNNF)* circuit*).*

The following is a direct corollary of the definitions.

Corollary 1. *Positive (resp. negative) CNF formulae form a proper subset of pwDNNF (resp. nwDNNF) circuits.*

Corollary 2. *The following is true based on the definitions of decomposability and its generalised variants:*

(i) DNNF circuits are a proper subset of pwDNNF and nwDNNF circuits (for example, $(x_1 \vee x_2) \wedge (x_2 \vee \neg x_3)$ is a pwDNNF circuit but not a DNNF circuit).

(ii) The new circuit types form a proper subset of wDNNF circuits (for example, $(x_1 \vee x_2) \wedge (x_2 \vee \neg x_3) \wedge (\neg x_3 \vee x_4)$ is a wDNNF circuit, but it is neither a pwDNNF circuit nor an nwDNNF circuit).

The new circuit types are universal (that is, they can represent any Boolean function) since they include DNNF circuits, which are universal (Darwiche 2001a).

Operations

This section examines the new circuit types concerning the operations presented in the knowledge compilation map. Operations are composed of queries (that is, circuit questions) and transformations (that is, circuit modifications).

In addition to the queries presented in the knowledge compilation map, we incorporate two cardinality queries: The maximum cardinality query, which was studied for DNNF circuits (see Darwiche 2001a), and the minimum cardinality query. Note that our definition of the maximum cardinality query follows (Pipatsrisawat and Darwiche 2007), whilst it is called the minimum cardinality query in (Darwiche 2001a). Similarly, we consider two additional transformations dealing with cardinality: The maximising (Darwiche 2001a) and minimising transformations.

Queries

A query is called *tractable* for a circuit type (that is, the *circuit type satisfies that query*) if and only if there is a polynomial time algorithm for answering that query on the circuits of that type. Otherwise, the query is called *intractable*. The following queries are considered:

- *Consistency check* (CO): Is the circuit consistent?
- *Validity check* (VA): Is the circuit valid?
- *Clausal entailment check* (CE): Does the circuit entail a given clause?
- *Implicant check* (IM): Does a given term imply the circuit?
- *Model counting* (CT): How many models does the circuit have?
- *Model enumeration* (ME): Enumerate the circuit models with polynomial delay.
- *Sentential entailment check* (SE): Does one circuit entail another circuit?
- *Equivalence check* (EQ): Are two circuits equivalent?
- *Minimum cardinality* (MinCard): What is the minimum cardinality of the circuit?
- *Maximum cardinality* (MaxCard): What is the maximum cardinality of the circuit?

Table 1 presents the tractable and intractable queries for pwDNNF and nwDNNF circuits. DNNF circuits (Darwiche and Marquis 2002) and wDNNF circuits (Illner and Kučera 2024) are shown for comparison.

Proposition 1. *The results for the queries in Table 1 hold.*

Considering the knowledge compilation map queries, wDNNF, pwDNNF, nwDNNF and DNNF circuits are indistinguishable. The new circuit types satisfy CO, CE, and ME since they are also satisfied by wDNNF circuits (Illner and Kučera 2024), which form a proper superset (see Corollary 2(ii)). We note that the CO_p predicate presented in (Illner and Kučera 2024) can also be used to check the consistency of a pwDNNF and nwDNNF circuit. Corollary 2(i) and the fact that VA, IM, CT, SE, and EQ are not tractable for DNNF circuits (Darwiche and Marquis 2002) make these queries intractable for pwDNNF and nwDNNF circuits.

The following procedure can be used to show that nwDNNF circuits satisfy MinCard.

Definition 3. *For a node α, the value of the $\mathrm{MinCard}_\mathrm{p}(\alpha)$ procedure is recursively computed as follows:*

(a) $\mathrm{MinCard}_\mathrm{p}(\alpha = \mathit{True}) = 0$.
(b) $\mathrm{MinCard}_\mathrm{p}(\alpha = \mathit{False}) = \infty$.
(c) $\mathrm{MinCard}_\mathrm{p}(\alpha) = \mathbf{1}$ *if α is a positive literal.*
(d) $\mathrm{MinCard}_\mathrm{p}(\alpha) = \mathbf{0}$ *if α is a negative literal.*
(e) $\mathrm{MinCard}_\mathrm{p}(\alpha = \bigvee_i \alpha_i) = \min_i \mathrm{MinCard}_\mathrm{p}(\alpha_i)$.
(f) $\mathrm{MinCard}_\mathrm{p}(\alpha = \bigwedge_i \alpha_i) = \sum_i \mathrm{MinCard}_\mathrm{p}(\alpha_i)$.

$\mathrm{MinCard}_\mathrm{p}(\alpha)$ can obviously be computed in linear time in the size of the circuit rooted at α.

Theorem 1. *The $\mathrm{MinCard}_\mathrm{p}$ procedure computes the minimum cardinality of an nwDNNF circuit.*

Darwiche (2001a) showed that the maximum cardinality query is tractable for DNNF circuits. By Theorem 1 and Corollary 2(i), DNNF circuits also satisfy the minimum cardinality query.

Symmetrically, we can define the $\mathrm{MaxCard}_\mathrm{p}$ procedure that originates from the $\mathrm{MinCard}_\mathrm{p}$ procedure by switching the roles of the constants $\mathbf{0}$ and $\mathbf{1}$. We can use this procedure to compute MaxCard as follows.

Theorem 2. *The maximum cardinality of a pwDNNF circuit with the root α is equal to $|\mathrm{Vars}(\alpha)| - \mathrm{MaxCard}_\mathrm{p}(\alpha)$.*

The $\mathrm{MinCard}_\mathrm{p}$ (resp. $\mathrm{MaxCard}_\mathrm{p}$) procedure can be trivially extended to return a model with the minimum (resp. maximum) cardinality.

Let us modify $\mathrm{MinCard}_\mathrm{p}$ (resp. $\mathrm{MaxCard}_\mathrm{p}$) so that the weight w_x of a variable x replaces the constant $\mathbf{1}$. We call this modified procedure $\mathrm{wMinCard}_\mathrm{p}$ (resp. $\mathrm{wMaxCard}_\mathrm{p}$).

Using the $\mathrm{wMinCard}_\mathrm{p}$ (resp. $\mathrm{wMaxCard}_\mathrm{p}$) procedure, we find that the weighted minimum (resp. maximum) cardinality problem can be solved in linear time on an nwDNNF (resp. pwDNNF) circuit.

Theorem 3. *Let Δ be a circuit with the root α and assume a non-negative weight w_x for each variable $x \in \mathrm{Vars}(\alpha)$.*

(i) If Δ is an nwDNNF circuit, then $\mathrm{wMinCard}_\mathrm{p}(\alpha)$ solves the wMinCard *problem of Δ.*
(ii) If Δ is a pwDNNF circuit, then the wMaxCard *problem on Δ is solved using $\left(\sum_{x \in \mathrm{Vars}(\alpha)} w_x\right) - \mathrm{wMaxCard}_\mathrm{p}(\alpha)$.*

Transformations

A transformation can be viewed as a function from a circuit (resp. a set of circuits of identical type) to an appropriate circuit of the same type. A transformation is called *tractable* for a circuit type (that is, the *circuit type satisfies that transformation*) if and only if a polynomial time algorithm exists that carries out that transformation on the circuits of the given type. Otherwise, the transformation is called *intractable*. We regard the following transformations:

Circuit type	CO	VA	CE	IM	CT	ME	SE	EQ	**MinCard**	**MaxCard**	CD	FO	$\wedge$BC	$\vee$BC	$\neg$C	**Mini**	**Maxi**
wDNNF	✓	○	✓	○	○	✓	○	○	○	○	✓	✓	○	✓	○	○	○
pwDNNF	✓	○	✓	○	○	✓	○	○	○	✓	✓	✓	○	✓	○	○	✓
nwDNNF	✓	○	✓	○	○	✓	○	○	✓	○	✓	✓	○	✓	○	✓	○
DNNF	✓	○	✓	○	○	✓	○	○	✓	✓	✓	✓	○	✓	○	✓	✓

Table 1: ✓ denotes that the operation is tractable, and ○ means that the operation is intractable unless $\mathcal{P} = \mathcal{NP}$.

- *Conditioning* (CD): Given a circuit Δ and a consistent term τ, construct a circuit representing $\Delta|\tau$.
- *Forgetting* (FO): Given a circuit Δ and a set of variables Y, construct a circuit equivalent to $\exists Y.\Delta$.
- *Bounded conjunction* ($\wedge$BC): Given Δ_1 and Δ_2 of the same type, construct a circuit representing $\Delta_1 \wedge \Delta_2$.
- *Bounded disjunction* ($\vee$BC): Given Δ_1 and Δ_2 of the same type, construct a circuit representing $\Delta_1 \vee \Delta_2$.
- *Negation* ($\neg$C): Given a circuit Δ, construct a circuit equivalent to $\neg\Delta$.
- *Minimising* (Mini): Given a circuit Δ, construct a circuit whose models are exactly the minimum cardinality models of Δ. Such a circuit is called a *minimisation of* Δ.
- *Maximising* (Maxi): Given a circuit Δ, construct a circuit whose models are exactly the maximum cardinality models of Δ. Such a circuit is called a *maximisation of* Δ.

Table 1 summarises the tractable and intractable transformations for pwDNNF and nwDNNF circuits. DNNF circuits (Darwiche and Marquis 2002) and wDNNF circuits (Illner and Kučera 2024) are presented for comparison.

Proposition 2. *The results in Table 1 hold.*

Similar to the queries, wDNNF, pwDNNF, nwDNNF and DNNF circuits satisfy the same transformations except those dealing with cardinality.

The proof of Proposition 5.1 in (Darwiche and Marquis 2002) can be reused to show that $\wedge$BC, and $\neg$C are hard for pwDNNF and nwDNNF circuits. The tractability of FO follows directly from the proof used for wDNNF circuits (Illner and Kučera 2024) since the new circuit types are special cases of wDNNF circuits.

The following procedure can be used to create a minimisation of an nwDNNF circuit.

Definition 4. *For a node α, let* $\mathrm{Minimise_p}(\alpha)$ *be defined as follows:*

(a) $\mathrm{Minimise_p}(\alpha) = \alpha$ *if α is a literal, True, or False.*

(b) $\mathrm{Minimise_p}(\alpha = \bigvee_i \alpha_i) = \bigvee_{\substack{\mathrm{MinCard}(\alpha_i) \\ =\mathrm{MinCard}(\alpha)}} \mathrm{Minimise_p}(\alpha_i).$

(c) $\mathrm{Minimise_p}(\alpha = \bigwedge_i \alpha_i) = \bigwedge_i \mathrm{Minimise_p}(\alpha_i).$

In case (b), if $\mathrm{MaxCard}(\alpha_i) = \mathrm{MaxCard}(\alpha)$ replaces $\mathrm{MinCard}(\alpha_i) = \mathrm{MinCard}(\alpha)$, we get the $\mathrm{Maximise_p}$ procedure that can be used to create a maximisation of a pwDNNF circuit.

Theorem 4. *For an nwDNNF (resp. pwDNNF) circuit with the root α, the result of* $\mathrm{Minimise_p}(\alpha)$ *(resp.* $\mathrm{Maximise_p}(\alpha)$*) is a minimisation (resp. maximisation) of that circuit.*

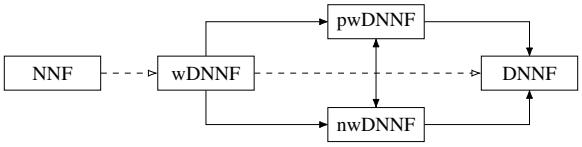

Figure 1: An edge $C_1 \to C_2$ means that $C_1 < C_2$. An edge $C_1 \longleftrightarrow C_2$ indicates that C_1 and C_2 are not comparable. The dashed edges show relations that are already known.

It should be evident that all of the above-mentioned procedures run in linear time in the size of the circuit.

Succinctness

This section relates the new circuit types to wDNNF and DNNF circuits in terms of succinctness (Gogic et al. 1995).

Definition 5. *Let C_1 and C_2 be two different circuit types. C_1 is said to be* at least as succinct as *C_2, denoted by $C_1 \leq C_2$, if and only if for every circuit Δ_2 of type C_2, there exists an equivalent circuit of type C_1 whose size is polynomial in the size of Δ_2. C_1 is said to be* strictly more succinct *than C_2, denoted by $C_1 < C_2$, if $C_1 \leq C_2$ and $C_2 \not\leq C_1$. If $C_1 \not\leq C_2$ and $C_2 \not\leq C_1$, then C_1 and C_2* are not comparable.

It was shown (de Colnet and Mengel 2021) that wDNNF circuits are strictly more succinct than DNNF circuits. The idea of the proof can be reused to show that the new circuit types are also strictly more succinct than DNNF circuits.

Bova et al. (2014) introduced a class $\mathbb{C}_{\mathrm{B}}^{2+}$ of positive 2-CNF formulae that provides an exponential separation between CNF formulae and DNNF circuits. Let us define $\mathbb{C}_{\mathrm{B}}^{2-} = \{\bar{\varphi} \mid \varphi \in \mathbb{C}_{\mathrm{B}}^{2+}\}$ where $\bar{\varphi}$ originates from φ by negating all the literals in φ. Then $\mathbb{C}_{\mathrm{B}}^{2-}$ is a class of negative 2-CNF formulae that also provides an exponential separation between CNF formulae and DNNF circuits.

Corollary 3. *By Corollary 1, pwDNNF and nwDNNF circuits are strictly more succinct than DNNF circuits.*

The following theorem gives the rest of the relations.

Theorem 5. *The following relations hold:*

(i) pwDNNF and nwDNNF circuits are not comparable.
(ii) wDNNF circuits are strictly more succinct than pwDNNF and nwDNNF circuits.

Figure 1 depicts the relations.

A Knowledge Compiler: Bella

Since pwDNNF and nwDNNF circuits are just special cases of wDNNF circuits, it is natural to use some existing com-

piler for wDNNF circuits and restrict that compiler so that only positive (resp. negative) variables can be shared to create a compiler for pwDNNF (resp. nwDNNF) circuits. Based on this idea, we extended[1] Bella (Illner and Kučera 2024), the only known compiler for wDNNF circuits.

We describe only our modifications, namely how connected components are computed, and how dual hypergraphs are created. We refer the curious reader to (Illner and Kučera 2024) for a more detailed description of Bella and the pseudocode, where the methods $\mathrm{computeComponents}$ and $\mathrm{computeNewCut}$ were changed as described below.

For pwDNNF (resp. nwDNNF) circuits, disjoint components can only share positive (resp. negative) variables. This is in contrast to wDNNF circuits for which disjoint components can share both positive and negative variables.

Example 1. *Let* $\varphi = (x_1 \vee x_2) \wedge (x_1 \vee \neg x_2) \wedge (x_1 \vee \neg x_3) \wedge (\neg x_3 \vee x_4) \wedge (\neg x_3 \vee \neg x_4) \wedge (\neg x_3 \vee x_5) \wedge (\neg x_3 \vee \neg x_5)$. *Considering pwDNNF circuits,* φ *has only two components:* $(x_1 \vee x_2) \wedge (x_1 \vee \neg x_2)$*, and* $(x_1 \vee \neg x_3) \wedge (\neg x_3 \vee x_4) \wedge (\neg x_3 \vee \neg x_4) \wedge (\neg x_3 \vee x_5) \wedge (\neg x_3 \vee \neg x_5)$. *Unlike wDNNF circuits, the last four clauses must be in the same component because the shared variable* x_3 *is negative in* φ.

The last thing that must be altered is the creation of a dual hypergraph, based on which a cut is computed representing a set of variables such that if the variables are assigned (regardless of how they are assigned), the simplified formula is split into more disjoint components. For wDNNF circuits, both positive and negative variables are left out in dual hypergraphs. Clearly, only positive (resp. negative) variables can be ignored for pwDNNF (resp. nwDNNF) circuits.

Example 2. *Let* $\varphi = (\neg x_1 \vee x_3 \vee x_5) \wedge (x_2 \vee x_4 \vee x_5) \wedge (\neg x_2 \vee \neg x_3 \vee x_5) \wedge (\neg x_1 \vee \neg x_4 \vee x_5)$ *be a residual CNF formula, which cannot be split into multiple components. Considering pwDNNF circuits, the variable* x_5 *is positive in* φ*, meaning this variable will not occur in the dual hypergraph. The dual hypergraph of* φ *is* $H(\varphi) = (\{c_1, c_2, c_3, c_4\}, \{\{c_1, c_4\}, \{c_2, c_3\}, \{c_1, c_3\}, \{c_2, c_4\}\})$ *where* c_i *represents the* i*-th clause and the* i*-th hyperedge corresponds to the variable* x_i*. The minimum cuts are* $\{x_1, x_2\}$ *and* $\{x_3, x_4\}$.

A Caching Scheme: Cara

During a compilation, the same residual CNF formula (that is, the input CNF formula under the current partial assignment) may be encountered multiple times. *Component caching* prevents repetitive compilations of these formulae. It is based on a cache, implemented as a hash table mapping residual CNF formulae to the identifiers of the nodes representing those formulae. Before a residual CNF formula (after unit propagation and decomposition) is compiled, it is checked if it has already been compiled. If so, the node from the cache is reused. If not, the residual CNF formula is compiled, and the result is stored in the cache. A *caching scheme* maps a residual CNF formula to a representation, typically a string of numbers, which is used for hashing.

We introduce a caching scheme, called Cara, that exploits isomorphism. The main idea is that even if a residual CNF formula has not been compiled yet, a syntactically identical formula up to renaming its variables (resp. literals) may have been compiled. If so, this information can be exploited to create a subcircuit that represents the residual CNF formula. Since deciding whether two formulae are isomorphic is expected not to be in $\mathcal{P}$, some approximation needs to be used. We took inspiration from sample moments in statistics.

The big picture is as follows. First, sample moments are computed for each variable V' appearing in a residual CNF formula. Second, the variables are sorted based on the sample moments. This sorting naturally provides a mapping from the variables to $1, \ldots, |V'|$. Last, based on the mapping, the variables (resp. literals) are renamed accordingly.

Let φ be a CNF formula. We assume a strict total ordering over the variables (that is, $x_1 \prec \ldots \prec x_{n'}$). By the index of x_i (resp. $\neg x_i$), we mean i (resp. $-i$). For a literal ℓ, C_ℓ (resp. μ_ℓ) denotes the number (resp. the average size) of clauses with ℓ. For each variable x_i, we compute the following (ordered) tuple: $(C_{x_i}, C_{\neg x_i}, \mu_{x_i}, \mu_{\neg x_i}, i)$. We create a mapping function $\mathcal{M}_\varphi$ from the variables to $1, \ldots, n'$ based on the lexicographic ordering of the tuples. Let $\varphi_\mathcal{M}$ originate from φ by renaming the variables according to $\mathcal{M}_\varphi$. The representation r_φ is described by the sorted formula $\varphi_\mathcal{M}$, where each clause is represented as a string of sorted indices of its literals, terminated by zero. For multi-occurrent clauses, only one occurrence is kept. $\mathcal{M}_\varphi$, which is represented as a string of indices, is stored as part of the value in the cache.

Example 3. *Let* $\varphi = (x_5 \vee \neg x_6) \wedge (x_6 \vee \neg x_7)$. *The ordered tuples for* x_7*,* x_5*, and* x_6 *are* $(0, 1, 0, 2, 7)$*,* $(1, 0, 2, 0, 5)$*, and* $(1, 1, 2, 2, 6)$*, respectively.* $\mathcal{M}_\varphi$ *is "7, 5, 6",* $\varphi_\mathcal{M} = (x_2 \vee \neg x_3) \wedge (x_3 \vee \neg x_1)$*, and* r_φ *is "*$-1, 3, 0, 2, -3, 0$*".*

Let us mention several observations. First, the creation of representations is deterministic because of the indices in tuples. Second, two syntactically identical (up to the ordering) CNF formulae have the same representation and mapping function. Third, more moments can be used in tuples (in Example 3, only the first moment μ is used), which may improve isomorphism detection at the expense of slowing down. Fourth, mapping functions can also take signs into account. For example, if $C_{x_i} < C_{\neg x_i}$, then the sign is changed. In Example 3, $\mathcal{M}_\varphi$ (resp. r_φ) would be "-7, 5, 6" (resp. "$1, 3, 0, 2, -3, 0$"). However, this is reasonable only for compiling into wDNNF and decision-DNNF circuits.

Ignoring isomorphism, based on the second observation, Cara and the caching scheme i (Lagniez and Marquis 2020), which is used in D4, have the same quality of equivalence detection. Since untouched clauses are ignored in i, the representation sizes are smaller than those created by Cara. On the other hand, having all clauses in a representation, which makes the correctness of Cara easy to see, is necessary for correct isomorphism detection. So, it is a quid pro quo.

We refer the reader to (Lagniez and Marquis 2020) for more caching schemes that do not exploit isomorphism and are used in other knowledge compilers or #SAT solvers.

A Method Based on Copying Subcircuits

In the following, we assume that Cara is used. Suppose we wish to compile a residual CNF formula ψ, and there is an

[1] https://github.com/Illner/BellaCompiler

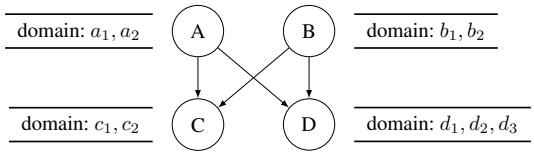

Figure 2: A fully dense two-layer BN. The CPTs are left out because they are unnecessary; only the domains are shown.

entry $\langle r_\varphi, (M_\varphi, id_\varphi)\rangle$ in the cache such that $r_\psi = r_\varphi$. The formula does not have to be compiled. However, depending on whether isomorphism was exploited, two situations can occur. The trivial case is if M_ψ and M_φ are the same, then the subcircuit rooted at id_φ is simply shared. Otherwise, we copy that subcircuit in a bottom-up fashion while renaming the literals according to $M_\varphi \circ M_\psi^{-1}$. We point out that *the unique-node technique* (see Brace, Rudell, and Bryant 1991) must be used to prevent possible exponential blow-ups.

Example 4. *Continuing with Example 3, suppose that φ was successfully compiled, and $\langle r_\varphi, (M_\varphi, id_\varphi)\rangle$ was stored in the cache. During the compilation, we encounter $\psi = (x_4 \vee \neg x_6) \wedge (x_6 \vee \neg x_8)$. $\mathcal{M}_\psi$ is "8, 4, 6", and r_ψ is "$-1, 3, 0, 2, -3, 0$". Since $r_\psi = r_\varphi$, we get a cache hit. $M_\varphi \circ M_\psi^{-1} = \{7 \mapsto 8, 5 \mapsto 4, 6 \mapsto 6\}$. The subcircuit rooted at id_φ is copied while respecting the given renaming.*

Reducing MPE to MinCostSat

Park (2002) presented an intuitive reduction from MPE to weighted MaxSAT, and Pipatsrisawat and Darwiche (2007) showed a reduction from weighted MaxSAT to MinCostSat. In the following two subsections, we outline these two known reductions, which are then combined.

Considering BNs, there are two main classes of reductions. The first exploits the global structure (that is, the topology) of a BN, and the second additionally exploits the local structure (that is, the specific values of probabilities). Each of these classes has different applicability. On the one hand, the use of local structures (for example, determinism (Chavira and Darwiche 2005), and context-specific independence (Boutilier et al. 1996)) can greatly simplify CNF formulae. On the other hand, robustness to changes in probabilities is lost. We focus on the former class.

Reducing MPE to Weighted MaxSAT

We delineate how to convert a discrete Bayesian network into a weighted CNF formula so that solving a weighted MaxSAT problem computes the corresponding MPE.

Since the domain of a network variable can have more than two values, we introduce an *indicator variable* for each value of that variable. For example, the network variable D in Figure 2 has three indicator variables: I_{d_1}, I_{d_2}, and I_{d_3}.

For each CPT entry, we introduce a weighted clause (called the *parameter clause*) containing the negation of the corresponding indicator variable for each network variable in that variable instantiation. The weight of that clause is the negative log of the corresponding conditional probability. If a CPT entry has zero probability, we treat the corresponding parameter clause as a hard clause. The parameter clause representing the CPT entry $\Pr(d_3 \mid a_1, b_2)$ in Figure 2 is $(\neg I_{a_1} \vee \neg I_{b_2} \vee \neg I_{d_3})^{-\log(\Pr(d_3 \mid a_1, b_2))}$.

For each network variable, it must be guaranteed that precisely one of the corresponding indicator variables must be true. For this purpose, an exactly-one constraint is used, and the clauses encoding such a constraint are called the *indicator clauses*. For example, the indicator clauses for the network variable D in Figure 2 are $(I_{d_1} \vee I_{d_2} \vee I_{d_3})$, $(\neg I_{d_1} \vee \neg I_{d_2})$, $(\neg I_{d_1} \vee \neg I_{d_3})$, and $(\neg I_{d_2} \vee \neg I_{d_3})$.

Since indicator clauses must always be satisfied to ensure valid variable instantiations, we treat them as hard clauses.

Example 5. *We show a part of the weighted CNF formula that encodes the Bayesian network depicted in Figure 2. The hard clauses (that is, indicator clauses) do not have weights.*

The part that encodes A is $(I_{a_1} \vee I_{a_2}) \wedge (\neg I_{a_1} \vee \neg I_{a_2}) \wedge (\neg I_{a_1})^{-\log(\Pr(a_1))} \wedge (\neg I_{a_2})^{-\log(\Pr(a_2))}$.

The part that encodes C is $(I_{c_1} \vee I_{c_2}) \wedge (\neg I_{c_1} \vee \neg I_{c_2}) \wedge$
$(\neg I_{a_1} \vee \neg I_{b_1} \vee \neg I_{c_1})^{-\log(\Pr(c_1 \mid a_1, b_1))} \wedge$
$(\neg I_{a_1} \vee \neg I_{b_1} \vee \neg I_{c_2})^{-\log(\Pr(c_2 \mid a_1, b_1))} \wedge$
$(\neg I_{a_1} \vee \neg I_{b_2} \vee \neg I_{c_1})^{-\log(\Pr(c_1 \mid a_1, b_2))} \wedge$
$(\neg I_{a_1} \vee \neg I_{b_2} \vee \neg I_{c_2})^{-\log(\Pr(c_2 \mid a_1, b_2))} \wedge$
$(\neg I_{a_2} \vee \neg I_{b_1} \vee \neg I_{c_1})^{-\log(\Pr(c_1 \mid a_2, b_1))} \wedge$
$(\neg I_{a_2} \vee \neg I_{b_1} \vee \neg I_{c_2})^{-\log(\Pr(c_2 \mid a_2, b_1))} \wedge$
$(\neg I_{a_2} \vee \neg I_{b_2} \vee \neg I_{c_1})^{-\log(\Pr(c_1 \mid a_2, b_2))} \wedge$
$(\neg I_{a_2} \vee \neg I_{b_2} \vee \neg I_{c_2})^{-\log(\Pr(c_2 \mid a_2, b_2))}$.

Evidence is treated as conditioning. The indicator variables are assigned appropriately for each observed network variable. The correctness is given in (Park 2002).

Reducing Weighted MaxSAT to MinCostSat

We show the main idea of the reduction from weighted MaxSAT to MinCostSat based on selector variables.

Let $\varphi = C_1^{w_1} \wedge \cdots \wedge C_m^{w_m}$ be a weighted CNF formula. We introduce a *selector variable* s_i for each clause C_i. Let ψ denote the CNF formula obtained from φ by augmenting each C_i with s_i. That is, $\psi = (C_1 \vee s_1) \wedge \cdots \wedge (C_m \vee s_m)$. Let the weight of s_i be w_i, and let the original variables have zero weights. Pipatsrisawat and Darwiche (2007) showed that solving the MinCostSat problem on ψ with these weights is equivalent to solving the weighted MaxSAT problem on φ.

Following the reduction, each hard clause should have a selector variable with a sufficiently large weight. Clearly, this is superfluous, and no selector variable is needed at all.

Bella and nwDNNF Circuits

Based on how Bella compiles into nwDNNF circuits, we claim that leaf network variables (LNVs) do not require their indicator clauses. We can notice that a positive literal of an indicator variable can occur only in indicator clauses. Thus, all the occurrences of all the indicator variables of an LNV are negative. Therefore, these indicator variables will never appear in dual hypergraphs (so not even in hypergraph cuts) and will thus never be chosen as decision variables. For an LNV, after assigning an indicator variable for each of its parents to *True*, the corresponding simplified parameter clauses are trivially represented as an nwDNNF circuit.

Evidence is treated as follows: The indicator variables are assigned appropriately for each observed network variable. Before the minimum cardinality query is computed to obtain an MPE instantiation, all the indicator variables for all the LNVs must be assigned as follows. The path from the root to such a node gives the values to all its parents. Suppose that the value of the LNV corresponding to that indicator variable has the highest probability in the related CPT. In that case, the node is set to *False*. Otherwise, it is set to *True*.

Experimental Results

For completeness, we mention that there is another encoding (Chavira and Darwiche 2008) not described in this paper that requires smooth d(ecision)-DNNF circuits (Choi and Darwiche 2017) and the weighted model counting query. This encoding was also incorporated into the experiments.

We focused solely on two-layer BNs with large domains. Such two-layer topologies are used in practice (see, for example, Shwe et al. 1992; de Campos, Fernández-Luna, and Huete 2002), and large domains are common for real-world problems. We highlight that such topologies are used for medical diagnosis. In such a BN, the top (resp. bottom) layer represents diseases (resp. symptoms) with large domains. Moreover, diseases typically change their probabilities (for example, an epidemic outbreak). If a disease causes a symptom, there is an edge from the disease to the symptom. An example of such a network is the *QMR-DT* network (Shwe et al. 1992) that consists of 600 diseases and 4000 symptoms. This network is typically not used in experiments because of its huge size and density. Thus, abstract variants called *DQMR* networks, which are simplified, significantly smaller, and randomly generated, are used instead.

We implemented Bels[2] to (randomly) generate and convert such BNs into CNF formulae using the encodings mentioned in this paper. We evaluated both equally and differently sized top and bottom layers. We also considered different BN densities (that is, how many edges were randomly generated). For example, a density of 60% means that a BN has exactly 60% of all the possible edges.

The experiments[3] were performed on a Linux machine (Debian 11) using an AMD EPYC™ 7543 2.8GHz processor and 512 GiB of RAM. The time-out (resp. memory-out) was set to two/six hours (resp. 16 GB). The following compilers were considered: Bella, D4[4] (the randomised variant introduced by Illner and Kučera (2024) was used), C2D[5], and SharpSAT-TD[6]. Ten instances were created for each density. Since the compilers are randomised, each instance was compiled three times, and the given results are averages. Cara exploits two moments — the first and second moments.

Table 2 presents the compilation times (in seconds), and the circuit sizes for fully dense BNs, which are the most challenging due to the number of parameters. First, the encoding requiring smooth d(ecision)-DNNF circuits is poor. Second, nwDNNF circuits have smaller sizes than decision-DNNF circuits. Third, considering nwDNNF circuits and decision-DNNF circuits (D4), as the domain sizes increase, so do the time differences (column "improv."). This improvement is maintained as the number of nodes increases. Last, more instances were compiled into nwDNNF circuits.

Table 3 presents the compilation times (in seconds), the circuit sizes, and the number of successfully compiled instances (maximum is ten) for 60% and 80% densities. To perform experiments for decision-DNNF circuits and Cara (signed variant), we extended Bella to compile into decision-DNNF circuits just like D4. First, focusing on the caching scheme i, Bella is faster than D4. Second, Cara noticeably speeds up the compilation times at the expense of increasing circuit sizes. However, this increase is tolerable for nwDNNF circuits, whilst significant for decision-DNNF circuits. Third, the more sparse a BN is, the better Cara performs. Last, significantly more instances were compiled into nwDNNF circuits using Cara.

Based on our experiments dealing with BNs with more complex topologies, we observed that deciding on LNVs during compilations is greatly beneficial. However, LNVs will never be used as decision variables for the encoding used for nwDNNF circuits, resulting in poorer performance.

We do not consider the data set used in (Lagniez and Marquis 2017) here because the results for the new circuit types are similar to those for wDNNF circuits presented in (Illner and Kučera 2024). We also mention that DQMR networks, which are encoded by the encoding that requires smooth d(ecision)-DNNF circuits, are also part of the data set.

Conclusion

We have introduced pwDNNF and nwDNNF circuits and integrated them into the knowledge compilation map. We have distinguished wDNNF, pwDNNF, nwDNNF and DNNF circuits in terms of operations. We have presented our extended variant of Bella for pwDNNF and nwDNNF circuits. We have shown a new compilation method based on copying subcircuits, which may significantly speed up compilations. We have demonstrated that nwDNNF circuits are suitable for computing MPEs in two-layer BNs with large domains.

Proofs

Proof of Theorem 1. We proceed by induction on the structure of the circuit. Cases (a) to (d) follow directly. Case (e): The minimum cardinality of $\bigvee_i \alpha_i$ is the smallest minimum cardinality among all α_i. Case (f): Let $\alpha = \bigwedge_i \alpha_i$ be negative weak decomposable. All the variables shared by at least two child nodes of α can be set to *False* without increasing the minimum cardinality of α. Thus, the minimum cardinality of α is the sum of the minimum cardinalities of α_i. □

Proof of Theorem 2. Maximising the number of variables that are set to *True* can be reformulated as minimising the number of variables set to *False*. The proof of the correctness of $\mathrm{MaxCard_p}$ is analogous to the case of $\mathrm{MinCard_p}$. Because $\mathrm{MaxCard_p}$ computes the minimum number of variables set to *False*, we must subtract this result from the total number of variables to obtain the maximum cardinality. □

[2]https://github.com/Illner/Bels

[3]https://github.com/Illner/BellaExperimentalResults-AAAI25

[4]https://github.com/Illner/BellaModifiedD4-AAAI24

[5]http://reasoning.cs.ucla.edu/c2d/

[6]https://github.com/raki123/sharpsat-td

#nodes	domain size	**nwDNNF circuits**			decision-DNNF circuits				smooth decision-DNNF circuits	
		Bella (i)			D4		C2D		D4	
		time (s)	size	improv.	time (s)	size	time (s)	size	time (s)	size
5 : 5	4	0.614	70 503.67	1.64	1.007	196 255.00	95.105	2 533 011.67	13.704	485 192.00
	7	107.525	1 909 131.67	5.48	589.727	6 770 407.00	—	—	19 345.839	20 459 704.00
	8	459.405	4 209 175.00	7.17	3 295.746	15 836 293.00	—	—	—	—
	9	1 883.894	8 471 321.33	>11.46	—	—	—	—	—	—
6 : 6	3	0.415	45 995.00	1.27	0.525	113 348.00	39.237	1 093 022.67	8.079	268 258.00
	5	121.543	1 539 814.67	3.23	391.891	4 781 238.00	—	—	20 991.073	14 023 422.00
	6	1 455.182	5 436 525.33	>14.84	—	—	—	—	—	—

Table 2: The compilation times (in seconds), and the circuit sizes for fully dense BNs. The time-out was set to six hours.

#nodes	density	domain size	**nwDNNF circuits**						decision-DNNF circuits					
			Bella (i)			**Bella (Cara)**			D4 (i)			Bella (Cara - signed variant)		
			time (s)	size	#	time (s)	size	#	time (s)	size	#	time (s)	size	#
5 : 5	80%	13	664	19 509 851	10	437	21 705 539	10	6 625	34 465 982	1	621	215 247 313	8
		14	1 492	32 614 471	10	867	36 403 555	10	—	—	0	1 091	274 720 896	2
		15	3 255	49 867 671	10	1 670	56 643 910	10	—	—	0	—	—	0
	60%	20	856	29 159 870	10	121	36 543 985	10	3 255	26 635 851	8	168	215 716 865	8
		25	2 027	34 843 482	10	577	45 881 028	10	—	—	0	576	352 399 770	4
		28	3 796	52 154 815	10	1 265	78 709 459	10	—	—	0	—	—	0
7 : 7	80%	7	499	26 198 478	10	138	29 087 302	10	2 072	23 058 910	10	207	166 739 933	8
		8	2 150	79 957 669	10	706	87 660 004	10	—	—	0	716	189 548 823	1
		9	4 921	79 666 358	5	2 951	160 648 695	10	—	—	0	—	—	0
	60%	10	741	43 397 886	10	98	57 991 430	10	2 526	42 752 348	9	154	267 145 930	2
		11	1 934	83 373 914	9	266	114 309 981	9	5 254	48 384 725	6	—	—	0
		12	3 653	132 864 751	8	551	209 529 685	8	—	—	0	—	—	0

Table 3: The compilation times (in seconds), the circuit sizes, and the number of successfully compiled instances (maximum is ten) for 60% and 80% densities. The time-out was set to two hours. The caching schemes i and Cara were considered.

Proof of Proposition 1. Most of the proof follows from Theorems 1 and 2, and the discussion below the proposition statement. It remains to show the hardness of $\mathrm{MinCard}$ (resp. $\mathrm{MaxCard}$) for pwDNNF (resp. nwDNNF) circuits. The *minimum vertex cover* problem is defined as follows: Given a graph, find the size of a minimum set of vertices $\mathbb{X}$ such that for each edge e, there is at least one incident vertex of e in $\mathbb{X}$. This problem is $\mathcal{NP}$-hard (Dinur and Safra 2005). We show a sketch of a polynomial reduction from vertex cover to $\mathrm{MinCard}$. Let $\mathbb{G}$ be a graph. We construct a positive formula φ such that for every edge $e = (v_i, v_j)$, we have a clause $(v_i \vee v_j)$ in φ. By Corollary 1, a pwDNNF circuit can compactly represent φ. $\mathrm{MinCard}$ of that circuit can solve the minimum vertex cover problem of $\mathbb{G}$. To show the hardness of $\mathrm{MaxCard}$ for nwDNNF circuits, we modify the reduction using clause $(\neg v_i \vee \neg v_j)$. By Corollary 2(ii), the cardinality queries are $\mathcal{NP}$-hard for wDNNF circuits. □

Proof of Theorem 4. We proceed by induction on the structure of the circuit. Case (a) is straightforward. Case (b): It follows that minimising a disjunction can be replaced by minimising each of its children and then removing those minimised children with a higher minimum cardinality. Case (c): Let $\alpha = \bigwedge_i \alpha_i$ be negative weak decomposable. All the variables shared by at least two child nodes of α can be set to *False* without increasing the minimum cardinality of α. In addition, the minimum cardinality of α is the sum of the minimum cardinalities of α_i. Thus, minimising α can be done by minimising each of its children α_i. The proof of the correctness of $\mathrm{Maximise_p}$ is symmetrical. □

Proof of Proposition 2. Most of the proof follows from Theorem 4 and the discussion below the proposition statement. CD: Conditioning means replacing some literals by *True* or *False*, but this cannot violate positive (resp. negative) weak decomposability. $\vee$BC: Positive (resp. negative) weak decomposability restricts only conjunction nodes. $\mathrm{Minimising}$ and $\mathrm{Maximising}$: The hardness of $\mathrm{Minimising}$ (resp. $\mathrm{Maximising}$) for pwDNNF (resp. nwDNNF) circuits is a result of the hardness of $\mathrm{MinCard}$ (resp. $\mathrm{MaxCard}$). □

Proof of Theorem 5. (i) Let φ be a positive 2-CNF formula from $\mathbb{C}^{2+}_{\mathrm{B}}$. Assume an nwDNNF circuit Δ representing φ. Let Δ' originate from Δ by replacing every negative literal by *True*. By Lemmata 1 and 2 in (Illner and Kučera 2024), it holds that $\Delta \equiv \Delta'$ and $|\Delta| \geq |\Delta'|$. Since Δ' does not contain any negative literal, it is clear that Δ' is a DNNF circuit. Therefore, the size of Δ' is exponential in the size of φ. By Corollary 1, a pwDNNF circuit can compactly represent φ. Hence, nwDNNF circuits are not at least as succinct as pwDNNF circuits. By a symmetric argument using $\mathbb{C}^{2-}_{\mathrm{B}}$, we get that pwDNNF circuits are not at least as succinct as nwDNNF circuits. (ii) This follows from the incomparability of pwDNNF and nwDNNF circuits (case (i)). □

Acknowledgments

The author would like to thank Petr Kučera for the fruitful discussion about this paper.

This research was partially supported by SVV project number 260 698 and by Center for Foundations of Modern Computer Science (Charles Univ. project UNCE 24/SCI/008). Computational resources were provided by the e-INFRA CZ project (ID:90254), supported by the Ministry of Education, Youth and Sports of the Czech Republic.

References

Akshay, S.; Chakraborty, S.; Goel, S.; Kulal, S.; and Shah, S. 2018. What's Hard About Boolean Functional Synthesis? In Chockler, H.; and Weissenbacher, G., eds., *Computer Aided Verification*, 251–269. Cham: Springer International Publishing. ISBN 978-3-319-96145-3.

Boutilier, C.; Friedman, N.; Goldszmidt, M.; and Koller, D. 1996. Context-Specific Independence in Bayesian Networks. In *Proceedings of the Twelfth International Conference on Uncertainty in Artificial Intelligence*, UAI'96, 115–123. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc. ISBN 155860412X.

Bova, S.; Capelli, F.; Mengel, S.; and Slivovsky, F. 2014. A strongly exponential separation of DNNFs from CNF formulas. *arXiv preprint arXiv:1411.1995*.

Brace, K. S.; Rudell, R. L.; and Bryant, R. E. 1991. Efficient Implementation of a BDD Package. In *Proceedings of the 27th ACM/IEEE Design Automation Conference*, DAC '90, 40–45. New York, NY, USA: Association for Computing Machinery. ISBN 0897913639.

Bryant, R. E. 1986. Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Transactions on Computers*, C-35(8): 677–691.

Chavira, M.; and Darwiche, A. 2005. Compiling Bayesian Networks with Local Structure. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence*, IJCAI'05, 1306–1312. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Chavira, M.; and Darwiche, A. 2008. On probabilistic inference by weighted model counting. *Artif. Intell.*, 172: 772–799.

Choi, A.; and Darwiche, A. 2017. On Relaxing Determinism in Arithmetic Circuits. In Precup, D.; and Teh, Y. W., eds., *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, 825–833. PMLR.

Darwiche, A. 1999. Compiling Knowledge into Decomposable Negation Normal Form. In *Proceedings of the 16th International Joint Conference on Artifical Intelligence - Volume 1*, IJCAI'99, 284–289. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Darwiche, A. 2001a. Decomposable negation normal form. *Journal of the ACM (JACM)*, 48(4): 608–647.

Darwiche, A. 2001b. On the tractable counting of theory models and its application to truth maintenance and belief revision. *Journal of Applied Non-Classical Logics*, 11(1-2): 11–34.

Darwiche, A. 2004. New Advances in Compiling CNF to Decomposable Negation Normal Form. In *Proceedings of the 16th European Conference on Artificial Intelligence*, ECAI'04, 318–322. Amsterdam, The Netherlands: IOS Press. ISBN 978-1-58603-452-8.

Darwiche, A. 2011. SDD: A New Canonical Representation of Propositional Knowledge Bases. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence - Volume Volume Two*, IJCAI'11, 819–826. AAAI Press. ISBN 978-1-57735-514-4.

Darwiche, A.; and Marquis, P. 2002. A knowledge compilation map. *Journal of Artificial Intelligence Research*, 17: 229–264.

de Campos, L. M.; Fernández-Luna, J. M.; and Huete, J. F. 2002. A Layered Bayesian Network Model for Document Retrieval. In Crestani, F.; Girolami, M.; and van Rijsbergen, C. J., eds., *Advances in Information Retrieval*, 169–182. Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN 978-3-540-45886-9.

de Colnet, A.; and Mengel, S. 2021. A Compilation of Succinctness Results for Arithmetic Circuits. In *Proceedings of the 18th International Conference on Principles of Knowledge Representation and Reasoning*, 205–215.

Dinur, I.; and Safra, S. 2005. On the Hardness of Approximating Minimum Vertex Cover. *Annals of Mathematics*, 162(1): 439–485.

Gogic, G.; Kautz, H.; Papadimitriou, C.; and Selman, B. 1995. The Comparative Linguistics of Knowledge Representation. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 1*, IJCAI'95, 862–869. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc. ISBN 1558603638.

Huang, J.; and Darwiche, A. 2007. The Language of Search. *J. Artif. Int. Res.*, 29(1): 191–219.

Illner, P.; and Kučera, P. 2024. A Compiler for Weak Decomposable Negation Normal Form. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(9): 10562–10570.

Kiesel, R.; and Eiter, T. 2023. Knowledge Compilation and More with SharpSAT-TD. In *Proceedings of the 20th International Conference on Principles of Knowledge Representation and Reasoning*, 406–416.

Lagniez, J.-M.; and Marquis, P. 2017. An Improved Decision-DNNF Compiler. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*, 667–673.

Lagniez, J.-M.; and Marquis, P. 2020. Enhanced Caching for #SAT Solving. Working paper or preprint.

Li, X. Y. 2004. *Optimization Algorithms for the Minimum-Cost Satisfiability Problem*. Ph.D. thesis, North Carolina State University.

Littman, M. L. 1999. Initial Experiments in Stochastic Satisfiability. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence and the Eleventh Innovative Applications of Artificial Intelligence Conference Innovative Applications of Artificial Intelligence*, AAAI '99/IAAI '99, 667–672. USA: American Association for Artificial Intelligence. ISBN 0262511061.

Muise, C.; McIlraith, S.; Beck, J. C.; and Hsu, E. 2010. Fast d-DNNF Compilation with sharpSAT. In *Workshops at the twenty-fourth AAAI conference on artificial intelligence*.

Park, J. D. 2002. Using Weighted MAX-SAT Engines to Solve MPE. In *Eighteenth National Conference on Artificial Intelligence*, 682–687. USA: American Association for Artificial Intelligence. ISBN 0262511290.

Pipatsrisawat, K.; and Darwiche, A. 2007. Clone: Solving Weighted Max-SAT in a Reduced Search Space. In Orgun, M. A.; and Thornton, J., eds., *AI 2007: Advances in Artificial Intelligence*, 223–233. Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN 978-3-540-76928-6.

Pipatsrisawat, K.; and Darwiche, A. 2008. New Compilation Languages Based on Structured Decomposability. In *Proceedings of the 23rd National Conference on Artificial Intelligence - Volume 1*, AAAI'08, 517–522. AAAI Press. ISBN 9781577353683.

Shwe, M.; Middleton, B.; Heckerman, D.; Henrion, M.; Horvitz, E.; Lehmann, H.; and Cooper, G. 1992. Probabilistic Diagnosis Using a Reformulation of the INTERNIST-1/QMR Knowledge Base: I. The Probabilistic Model and Inference Algorithms. *Methods of information in medicine*, 30.