An And-Sum Circuit with Signed Edges That Is More Succinct than SDD

Ryoma Onaka, Kengo Nakamura, Masaaki Nishino, Norihito Yasuda

NTT Communication Science Laboratories, NTT Corporation, Kyoto, Japan
{ryoma.onaka,kengo.nakamura,masaaki.nishino,norihito.yasuda}@ntt.com

Abstract

Knowledge compilation is a method of transforming knowledge into a compressed and tractable form for permitting more efficient operations. For Boolean functions, numerous representations have been proposed that enhance succinctness and tractability. In this paper, we introduce a new representation named structured Decomposable And-Sum Circuit (st-DASC), which employs $\wedge$ and $+$ nodes with signed edges, in place of the standard $\wedge$ and $\vee$ nodes with unsigned edges. Notably, incorporating negative signs permits polytime logical negation. By following a knowledge compilation map, we show that st-DASCs are more succinct than Sentential Decision Diagrams (SDDs) while maintaining support for every operation on the knowledge compilation map that SDD supports. Furthermore, st-DASCs are even more succinct than structured d-DNNFs (st-d-DNNFs), which are more succinct than SDDs although they support fewer operations than SDDs. Accordingly, st-DASCs break the traditional trade-off between succinctness and tractability over SDDs and st-d-DNNFs.

Introduction

Knowledge compilation is a method for handling propositional reasoning at high speed (Darwiche 1999). In this method, propositional logic is compiled into a certain representation and various operations can be performed on it.

The many representations of Boolean functions are characterized by the trade-off relationship between their succinctness and tractability. Among them, an *Ordered Binary Decision Diagram (OBDD)* (Bryant 1986) is a popular language due to its high succinctness and traceability, and it has been used in a wide range of applications (Álvaro Torralba et al. 2017; Chaki and Gurfinkel 2018; Hardy, Lucet, and Limnios 2007). *Sentential Decision Diagrams (SDDs)* (Darwiche 2011) are more succinct than OBDDs, although they have identical tractable operations. Thanks to its succinctness and tractability, an SDD also have numerous applications including probabilistic inference by weighted model counting (Choi, Kisa, and Darwiche 2013), neuro-symbolic AI (Manhaeve et al. 2018; Ahmed et al. 2022), and learning neural network parameters (Xu et al. 2018). A remaining challenge is finding a representation that is not only more succinct than SDD but also supports similar operations. The one closest to this ideal solution is a structured d-DNNF (st-d-DNNF) (Pipatsrisawat and Darwiche 2008). St-d-DNNFs are more succinct than SDDs, but they do not support certain tractable operations, such as logical negation, bounded disjunction, and single forgetting (Vinall-Smeeth 2024), all of which are supported by SDDs. To the best of our knowledge, no generalization of an st-d-DNNF has produced a representation that supports these operations.

In this paper, we propose *structured Decomposable And-Sum Circuits (st-DASCs)*, which decompose Boolean functions with $\wedge$ and $+$ nodes; existing methods use $\wedge$ and $\vee$. Here, signed edges are applied to st-DASC to enable it to perform logical negation. St-DASC is a generalization of st-d-DNNF, where the $\vee$ nodes are replaced into $+$ nodes with binary edge signs. We show that st-DASCs are more succinct than st-d-DNNFs, meaning that they are also more succinct than SDDs. Moreover, we also show that st-DASCs support in polynomial time all of the operations that are on knowledge compilation map (Darwiche and Marquis 2002) and that SDDs support in polynomial time. In that sense, st-DASCs actually break the trade-off of succinctness and tractability over SDDs and st-d-DNNFs. More precisely, representation A breaks the trade-off over representation B when (I) A supports all the operations that B supports and is more succinct than B, or (II) A is at least as succinct as B and supports more transformations and the same queries compared to B. Within this definition, st-DASCs meet both (I) and (II) over st-d-DNNFs and meet (I) over SDDs.

The key capability is logical negation. By imposing edge signs, st-DASCs support logical negation, which st-d-DNNFs do not. Some operations that st-d-DNNFs do not support can be reduced to negation, which contributes to the tractability of st-DASCs. Such tractable negation also demonstrates a proof of succinctness. As a result, st-DASCs overcome the current trade-off because they are not only more succinct than st-d-DNNFs but also support more operations in polytime, at least concerning the knowledge compilation map.

Related Work

Various target languages of knowledge compilation have been proposed to represent Boolean functions. In addition to OBDD, SDD, and st-d-DNNF, NNF subsets include NNF,

DNNF, and d-DNNF. Each representation reaches a different trade-off between succinctness and tractability, and each of the different applications has its appropriate representation (Huang and Darwiche 2005; Bonet and Geffner 2006; Pipatsrisawat and Darwiche 2007). Among them, OBDD and SDD have been used in a wide range of applications (Hardy, Lucet, and Limnios 2007; Inoue et al. 2014; Maehara, Suzuki, and Ishihata 2017; Xu et al. 2018; Chavira and Darwiche 2008; Choi, Kisa, and Darwiche 2013) since they are flexible languages with the richest number of operations, such as logical conjunction and disjunction in polytime. St-DASC is the most flexible knowledge compilation language among those following this trend. Among these representations of Boolean functions, succinctness and tractability basically form a trade-off relationship. A knowledge compilation map (Darwiche and Marquis 2002) has been developed to analyze this trade-off. It facilitates the selection of appropriate representation for each application. There has also been research on adding new representations to the knowledge compilation map (Darwiche 2014).

The arithmetic circuit (Darwiche 2003) is a circuit-based representation of real-valued functions. In addition, a probabilistic circuit (Poon and Domingos 2011; Kisa et al. 2014) is a subclass of arithmetic circuits limited to the representation of probability distributions. Although previous studies have addressed the operations supported among probabilistic circuits (Choi, Vergari, and Van den Broeck 2020; Shen, Choi, and Darwiche 2016; Vergari et al. 2021), no study has focused on the ability of arithmetic circuits to represent Boolean functions.

Preliminaries

A *Boolean function* takes n binary variables of $\{0, 1\}$ as input and returns $\{0, 1\}$ as output. We consider Boolean functions to be a subset of real-valued functions and introduce numerical operations such as $+, \times, \min$ and $\max$. The assignment of input variables that makes the output of a Boolean function equal to 1 is called a *model*. A Boolean function is considered *consistent* if it has a model. The logical expression in the form $\bigvee_i \ell_i$ for literal ℓ_i is called a *clause*. Similarly, a logical expression in the form $\bigwedge_i \ell_i$ is called a *term*. Next we introduce the *conditioning* and *forgetting* operations for Boolean functions.

Definition 1. (Darwiche 1999) Let f be a Boolean function and let t be a consistent term. The conditioning of f on t, denoted as $f|t$, is a Boolean function obtained by assigning each variable x of f by 1 if x is a positive literal of t and by 0 if it is a negative literal of t.

Definition 2. (Darwiche and Marquis 2002) Let f be a Boolean function, and let X be a subset of variables. The *forgetting* of X from f, denoted as $\exists X.f$, is a Boolean function that satisfies $\exists X.f|x = \exists X.f|\neg x$ for any $x \in X$ and $f \models g \Leftrightarrow \exists X.f \models g$ for any Boolean function g that does not include any variable from X. Here, $f \models g$ means that f implies g, i.e., $g = f \wedge g$.

Negation Normal Form (NNF), which is a representation of a Boolean function, is defined as follows.

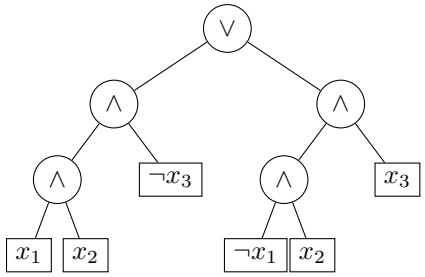

Figure 1: An st-d-DNNF representing $(x_1 \wedge x_2 \wedge \neg x_3) \vee (\neg x_1 \wedge x_2 \wedge x_3)$.

Definition 3. An NNF is a rooted DAG whose leaf nodes are labeled 1, 0, x, or $\neg x$. Non-leaf nodes are labeled either $\vee$ or $\wedge$.

Figure 1 shows an example of an NNF. Without loss of generality, we assume that each $\wedge$ node has only 2 children. The *size* of an NNF is the number of edges. For any node v in an NNF, we define its scope $\mathrm{Var}(v)$ as the set of variables that appear in the subgraph with v as the root. We define Boolean function f_v corresponding to node v as follows:

- When v is a leaf node, f_v equals the label of v.
- When v is a $\vee$ node, $f_v = \max_i f_{c_i}$, where $\{c_1, \ldots, c_k\}$ are the child nodes of v; here, the above equation means that they are equal for any assignment, i.e., $f_v(\mathbf{a}) = \max_i f_{c_i}(\mathbf{a})$ for any assignment $\mathbf{a}$.
- When v is a $\wedge$ node, $f_v = \min_i f_{c_i}$, which means that $f_v(\mathbf{a}) = \min_i f_{c_i}(\mathbf{a})$ for any assignment $\mathbf{a}$.

We denote a Boolean function corresponding to the OBDD with root r as f_r.

The subsets of NNFs are often considered by assuming various properties. The following are some of the main properties.

- *Decomposability*: An NNF is decomposable when no $\wedge$-node in it shares a variable. That is, for any $\wedge$ node with child nodes c_1, c_2, $\mathrm{Var}(c_1) \cap \mathrm{Var}(c_2) = \emptyset$.
- *Determinism*: An NNF is deterministic when at most one input becomes 1 for any $\vee$ node and any input assignment. In other words, an NNF is deterministic when $\sum_i f_{c_i}(\mathbf{a}) \leq 1$ for any $\vee$ node and any input assignment $\mathbf{a}$, where $\{c_1, \ldots, c_k\}$ are the child nodes of v.
- *Smoothness*: An NNF is smooth when the children of a $\vee$ node have the same scope. In other words, an NNF is smooth when $\mathrm{Var}(c_i) = \mathrm{Var}(c_j)$ for any $\vee$ node with child nodes $\{c_1, \ldots, c_k\}$ and any $i \neq j$.
- *Structured decomposability*: A decomposable NNF is structured decomposable when the scope of the variables follows a certain binary tree (*vtree*). A vtree for set of variables X is a binary tree in which each leaf node has a one-to-one correspondence to a variable; for each node v in the vtree, we define its scope $\mathrm{Var}(\mathsf{v})$ as the set of variables that appear in the subtree with v as its root. A decomposable NNF respects a vtree if for any $\wedge$ node with child nodes $\{c_1, c_2\}$, there exists a certain

node ϵ and its child nodes $\{\mathsf{c}_1, \mathsf{c}_2\}$ on the vtree such that $\mathrm{Var}(c_i) \subseteq \mathrm{Var}(\mathsf{c}_i)$ for $i \in \{1, 2\}$.

Various NNF subsets are defined by a combination of these properties, although the two discussed in this paper are st-d-DNNF and SDD. St-d-DNNFs are defined as structured decomposable and deterministic NNFs. SDDs can be regarded as st-d-DNNFs with additional constraints.

Definition 4. For two representation classes of Boolean functions, A and B, A is considered *at least as succinct as* B, denoted as $A \leq B$, if polynomial p exists such that, for every representation $\beta \in B$, there exists an equivalent representation $\alpha \in A$, where $|\alpha| \leq p(|\beta|)$. Here, $|\alpha|$ and $|\beta|$ are the sizes of α and β. We say representation class A is *more succinct than* B if A is at least as succinct as B and B is not at least as succinct as A.

The knowledge compilation map also analyses the possible operations in polynomial time for each representation class. There are two types of operations: queries and transformations. A query operation calculates a certain value without transforming a representation. A transformation operation constructs a representation that satisfies a condition. Table 1 shows all of the queries and transformations discussed in the knowledge compilation map paper (Darwiche and Marquis 2002). The richness of polytime queries and succinctness basically have a trade-off relationship. For example, st-d-DNNFs are more succinct than SDDs, although st-d-DNNFs do not support logical negation and certain other operations that SDDs support (Vinall-Smeeth 2024).

Structured Decomposable And-Sum Circuits (st-DASCs)

In this section, we define st-DASC and give some examples.

First, we explain the basic idea. St-d-DNNFs do not support logical negation and other operations that can be reduced to logical negation (Vinall-Smeeth 2024). Moreover, in st-d-DNNFs, there is a function whose size increases super-polynomially when negation is taken (Vinall-Smeeth 2024). Therefore, if a generalization of st-d-DNNFs could support negation without losing the ability to support the other operations, it would become a more tractable and succinct representation than st-d-DNNFs.

To achieve this, we introduce two approaches for generalizing st-d-DNNFs. First, we introduce $+$ nodes, instead of $\vee$ nodes that are used in st-d-DNNFs. A $+$ node decomposes a function into the sum of its child nodes. In st-d-DNNFs, a $\vee$ node never appears in cases where both inputs are 1 due to determinism. For the other inputs, the outputs are originally identical to $+$. Therefore, $+$ nodes are a generalization of $\vee$ nodes under determinism. Even if we replace deterministic $\vee$ nodes with $+$ nodes and impose a Boolean restriction instead of determinism, it supports the same operations as st-d-DNNF because it supports counting and conjunctions, as shown later. Furthermore, by using negative edge signs, it supports logical negation that st-d-DNNF does not. The tractable negation also contributes to demonstrating the proof of succinctness between st-d-DNNF and st-DASC.

Next we define ASCs, which are the representations decomposing Boolean function with $+$ and $\wedge$, as follows.

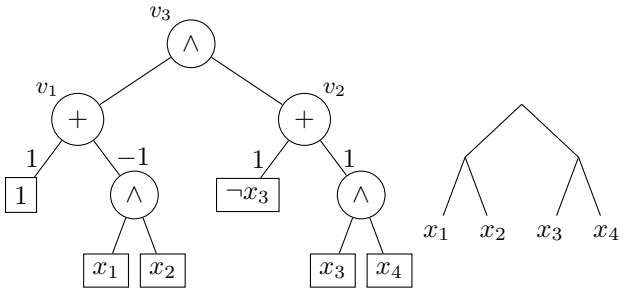

Figure 2: Example of st-DASC and its vtree.

Definition 5. An *And-Sum Circuit (ASC)* is a rooted DAG. Each leaf node is labeled as 1, 0, variable x, or its negation $\neg x$. Here, a 1 and a 0 nodes are called *constant nodes* and the other leaf nodes are called *literal nodes*. Each non-leaf node is labeled either $+$ or $\wedge$. Each edge between a $+$ node and its child node has sign $w \in \{-1, 1\}$.

For an ASC, function f_v, corresponding to each node v, is defined as follows. When v is a leaf node, f_v is defined in the same way as NNF. When v is a $+$ node with child nodes $\{c_1, \ldots, c_k\}$ and edge signs $\{w_1, \ldots, w_k\}$, $f_v = \sum_i w_i f_{c_i}$, denoting that $f_v(\mathbf{a}) = \sum_i w_i f_{c_i}(\mathbf{a})$ for any assignment $\mathbf{a}$. When v is a $\wedge$ node with child nodes c_i, $f_v = \min f_{c_i}$. [1] Let function f_r corresponding to root r be the function corresponding to the ASC.

Unlike NNF, an ASC's output is not guaranteed to be 0,1. Therefore, we assume that ASC satisfies the following property so that it represents a Boolean function. An ASC is called *Boolean* when for any node v, $f_v(\mathbf{a})$ is either 0 or 1 for any assignment $\mathbf{a}$. A Boolean ASC can be regarded as a representation of a Boolean function. Hereinafter, unless otherwise noted, ASCs are assumed to be Boolean. The *size* of an ASC is defined by the number of its edges. Since each edge sign uses only constant space, the size can be compared with reference to the number of edges, as in the existing decomposition using $\wedge, \vee$.

For ASCs, such properties as decomposability, smoothness and structured decomposability can be defined identically as for NNFs. Note that the $\vee$ nodes in the definition of smoothness are replaced with $+$ nodes. We denote a structured decomposable ASC by st-DASC.

For node v of st-DASC, *decomposition node* $\mathsf{d}(v)$ is defined as lowest node v on the vtree such that $\mathrm{Var}(v) \subseteq \mathrm{Var}(\mathsf{v})$. In the following, when we consider operations between two st-DASCs, we assume that they are respecting the same vtree. Note that this assumption is also ordinal and thus imposed in this paper for st-d-DNNFs.

Figure 2 shows an example of an st-DASC and its vtree. Node v_1 in the figure represents function $1 - (x_1 \wedge x_2)$ and corresponds to taking logical negation $\neg(x_1 \wedge x_2)$. Node v_2 represents function $\neg x_3 + x_3 \wedge x_4$, which simply corresponds to taking logical disjunction $\neg x_3 \vee (x_3 \wedge x_4)$. Therefore, the corresponding function to node v_3 is $\neg(x_1 \wedge x_2) \wedge (\neg x_3 \vee (x_3 \wedge x_4))$.

[1] If ASC is Boolean, min f_{c_i} equals $\wedge_i f_{c_i}$.

	Name	Explanation	Complexity
Query	CO	Answering whether f has a model	$O(m_1)$
	VA	Answering whether f is valid, i.e., f is evaluated as 1 for any assignment of variables	$O(m_1)$
	CE	Answering whether $f \models \gamma$ holds for clause γ	$O(m_1)$
	EQ	Answering whether f and g are equivalent	$O(m_1 m_2)$
	SE	Answering whether $f \models g$ holds	$O(m_1 m_2)$
	IM	Answering whether $t \models f$ holds for term t	$O(m_1)$
	CT	Counting the number of the models of f	$O(m_1)$
	ME	Enumerating the models of f	$O(n' m_1 \mathrm{CT}(f))$
Transformation	$\wedge$BC	Constructing st-DASC representing the logical conjunction of f, g	$O(m_1 m_2)$
	$\vee$BC	Constructing st-DASC representing the logical disjunction of f, g	$O(m_1 m_2)$
	$\neg$C	Constructing st-DASC representing the logical negation of f	$O(1)$
	CD	Constructing st-DASC representing $f\|t$, f conditioned by term t	$O(m_1)$
	SFO	Constructing st-DASC representing $\exists x.f$ for variable x	$O(m_1^2)$
	$\wedge$C	Constructing st-DASC representing the logical conjunction of $f_1, f_2, \ldots, f_k$	•
	$\vee$C	Constructing st-DASC representing the logical disjunction of $f_1, f_2, \ldots, f_k$	•
	FO	Constructing st-DASC representing $\exists \mathbf{X}.f$ for variable set $\mathbf{X}$	•

Table 1: Operations and their complexity on st-DASCs: We assume that f is represented by st-DASC A with size m_1 and that g is represented by st-DASC B with size m_2. The number of variables f mentions is denoted by n'. Symbol • means the operation cannot be performed in polytime on st-DASC regardless of whether P$\neq$ NP.

Main Results

Here we introduce two main theorems. The first involves operations supported by st-DASCs, while the second concerns succinctness of st-DASCs. The first theorem shows that st-DASCs support the same operations as SDDs. The second shows that st-DASCs are more succinct than st-d-DNNFs.

Theorem 1. CO, VA, EQ, CE, SE, IM, CT, $\wedge$BC, $\vee$BC, $\neg$C, CD and SFO can be performed in polynomial time on st-DASCs with respect to the sizes of st-DASCs. ME can be performed in polynomial time with respect to the size of st-DASC and the number of models. However, $\wedge$C, $\vee$C and FO cannot be performed in polynomial time on st-DASCs.

Theorem 2. St-DASCs are more succinct than st-d-DNNFs.

Theorem 2 also means that st-DASCs are more succinct than SDDs since st-d-DNNFs are more succinct than SDDs (Vinall-Smeeth 2024).

Proof of Theorem 1

Most operations can be performed by a combination of $\wedge$BC, CT, $\neg$C, and CD. CD can be performed by almost the same method as in the case of NNF. Therefore, we first show that $\wedge$BC, CT, and $\neg$C are possible in polytime. Then we show that other operations can be performed in polytime.

Let f, g be Boolean functions and γ, t be a clause and a term. We assume that f is represented by an st-DASC with size m_1, and g is represented by an st-DASC with size m_2. We denote the number of literals of γ, t as $|\gamma|, |t|$. Table 1 summarizes the complexities of the operations.

$\wedge$BC For two st-DASCs respecting the same vtree, the st-DASC corresponding to their logical conjunction can be constructed by Algorithm 1. It resembles the algorithm of conjoining st-d-DNNFs (Pipatsrisawat and Darwiche 2008); the difference is the existence of signs on the edges.

The following is the specific idea behind this approach. Considering that outputs of st-DASCs are only $\{0, 1\}$, the logical conjunction can be identified by multiplication. For st-DASCs A, B, and their roots u, v, multiplication $f_u f_v$ can be decomposed recursively. From Lines 4 to 6, we deal with a case where at least one of the nodes is a 0 or a 1 node. The part under Line 7 is the other case, where $lca(\mathsf{s}, \mathsf{t})$ is the lowest common ancestor of vtree nodes s and t. When u and v do not share variables, they can be simply connected with a $\wedge$ node (Lines 9-10). If u and v are both literal nodes, their product is returned, i.e., return 0 if one of the labels of u and v is x and the other is $\neg x$, or return u otherwise (Lines 11-15). In Lines 16-17, if u is a $+$ node, it decomposes into the weighted sum of child nodes. If both u and v are $\wedge$ nodes, they decompose into the conjunction (Lines 18-21). In this part, $\mathsf{L} = \mathsf{anc}^l, \mathsf{R} = \mathsf{anc}^r$ are the left child and the right child of anc. Node u^{L} is u's child c where $\mathrm{Var}(c) \subseteq \mathrm{Var}(\mathsf{L})$ if $\mathsf{anc} = \mathsf{d}(u)$; otherwise, u^{L} is u itself. Node u^{R} is u's child c where $\mathrm{Var}(c) \subseteq \mathrm{Var}(\mathsf{R})$ if $\mathsf{anc} = \mathsf{d}(u)$; otherwise, u^{R} is u itself. Nodes $v^{\mathsf{L}}, v^{\mathsf{R}}$ are similarly defined. The resultant node's decomposition node is defined as the lowest common ancestor anc. To avoid repeating the same calculation, we store the results of previous calculations in the cache and use them as needed (Lines 3,22).

We show that st-DASC $\mathrm{CON}(u, v)$ represents Boolean function $f_u f_v$ by induction on the depth of u and v. The base cases are in Lines 4,5,6,10,13, and 15: for these cases, the returned left node or the node stored in res truly represents $f_u f_v$. In Line 17, by the induction hypothesis, $\mathrm{CON}(u_i, v)$ represents $f_{u_i} f_v$. Then, node res represents $\sum_i w_i (f_{u_i} f_v) = (\sum_i w_i f_{u_i}) f_v = f_u f_v$. In Line 20, by the induction hypothesis, $\mathrm{CON}(u^{\mathsf{L}}, v^{\mathsf{L}})$ and $\mathrm{CON}(u^{\mathsf{R}}, v^{\mathsf{R}})$ represent $f_{u^{\mathsf{L}}} f_{v^{\mathsf{L}}}$ and $f_{u^{\mathsf{R}}} f_{v^{\mathsf{R}}}$. Since we can write $f_u = f_{u^{\mathsf{L}}} f_{u^{\mathsf{R}}}$ and $f_v = f_{v^{\mathsf{L}}} f_{v^{\mathsf{R}}}$ by the definition of $u^{\mathsf{L}}, u^{\mathsf{R}}, v^{\mathsf{L}}, v^{\mathsf{R}}$, node res represents $(f_{u^{\mathsf{L}}} f_{v^{\mathsf{L}}})(f_{u^{\mathsf{R}}} f_{v^{\mathsf{R}}}) = (f_{u^{\mathsf{L}}} f_{u^{\mathsf{R}}})(f_{v^{\mathsf{L}}} f_{v^{\mathsf{R}}}) = f_u f_v$.

Thus, a logical conjunction can be calculated by recursively performing this method. The computational complexity is $O(m_1 m_2)$, where m_1, m_2 are the sizes of A, B. This is

Algorithm 1: CON(u, v)

Input : Two st-DASCs u, v respecting the same vtree
Output: An st-DASC corresponding to the conjunction of u and v

if *(i)* $\mathsf{d}(u) = \mathsf{d}(v)$*, u is a $\wedge$ node and v is a $+$ node, or (ii) $\mathsf{d}(v)$ is an ancestor of $\mathsf{d}(u)$* **then**
  swap(u, v)
if CACHE$(u, v) \neq$ *null* **then return** CACHE(u, v)
else if *u or v is 0 node* **then return** 0
else if *u is 1 node* **then return** v
else if *v is 1 node* **then return** u
else
  $\mathsf{s} \leftarrow \mathsf{d}(u)$, $\mathsf{t} \leftarrow \mathsf{d}(v)$, $\mathsf{anc} \leftarrow lca(\mathsf{s}, \mathsf{t})$
  if anc *is neither* s *nor* t **then**
    $res \leftarrow u \wedge v$
  else if *u, v are both literal nodes* **then**
    if *the labels of u and v are identical* **then**
      $res \leftarrow u$
    else
      $res \leftarrow 0$
  else if *u is a $+$ node* **then**
    `// Let the child nodes of` u `be` $\{(u_1, w_1), \ldots, (u_k, w_k)\}$
    $res \leftarrow$ a $+$ node with child nodes $\{(\mathrm{CON}(u_1, v), w_1), \ldots, (\mathrm{CON}(u_k, v), w_k)\}$
  else
    `// When` u, v `are both` $\wedge$ `nodes`
    $\mathsf{L} \leftarrow \mathsf{anc}^l$, $\mathsf{R} \leftarrow \mathsf{anc}^r$
    $res \leftarrow \mathrm{CON}(u^{\mathsf{L}}, v^{\mathsf{L}}) \wedge \mathrm{CON}(u^{\mathsf{R}}, v^{\mathsf{R}})$
    $\mathsf{d}(res) \leftarrow \mathsf{anc}$
  CACHE$(u, v) \leftarrow res$
return res

because, except for the cache, the recursion is called at most the number of different node pairs, and the internal processing takes at most the number of children on one side. Note that the output of CON satisfies the definition of st-DASC. Indeed, the output is structured decomposable because CON decomposes according to the input vtree.

CT We denote the number of models as CT(f) for Boolean function f. We consider calculating the number of models of function f_v corresponding to each node v of an st-DASC. Note that we always consider f_v a Boolean function of n input variables in when addressing the number of models. If v is a literal node, $\mathrm{CT}(f_v) = 2^{n-1}$. Similarly, $\mathrm{CT}(f_v) = 0$ if v is a 0 node, and $\mathrm{CT}(f_v) = 2^n$ if v is a 1 node. If v is a $+$ node with child nodes $\{(c_1, w_1), (c_2, w_2), \ldots, (c_k, w_k)\}$,

$$\mathrm{CT}(f_v) = \sum_{\mathbf{a}} f_v(\mathbf{a}) = \sum_{\mathbf{a}} \sum_i w_i f_{c_i}(\mathbf{a}) = \sum_i w_i \left(\sum_{\mathbf{a}} f_{c_i}(\mathbf{a}) \right) = \sum_i w_i \mathrm{CT}(f_{c_i}). \quad (1)$$

If v is a $\wedge$ node with child nodes $\{c_1, c_2\}$,

$$\begin{aligned} \mathrm{CT}(f_v) &= \sum_{\mathbf{a}} f_v(\mathbf{a}) \\ &= \sum_{\mathbf{y}} \sum_{\mathbf{z}} f_{c_1}(\mathbf{y}, \mathbf{z}) f_{c_2}(\mathbf{y}, \mathbf{z}) \\ &= \sum_{\mathbf{y}} f_{c_1}(\mathbf{y}, \mathbf{1}) \sum_{\mathbf{z}} f_{c_2}(\mathbf{1}, \mathbf{z}) \quad (2) \\ &= \frac{\sum_{\mathbf{a}} f_{c_1}(\mathbf{a})}{2^{|Z|}} \frac{\sum_{\mathbf{a}} f_{c_2}(\mathbf{a})}{2^{|Y|}} \\ &= \mathrm{CT}(f_{c_1}) \mathrm{CT}(f_{c_2}) / 2^n, \quad (3) \end{aligned}$$

where X is the underlying variable sets, $Y = \mathrm{Var}(c_1)$ and $Z = X \backslash Y$. Note that $\mathrm{Var}(c_1) \cap \mathrm{Var}(c_2) = \emptyset$ by the definition of decomposability and thus $\mathrm{Var}(c_2) \subseteq X \backslash \mathrm{Var}(c_1) = Z$. In Eq. (2), $(\mathbf{y}, \mathbf{1})$ is the assignment where all the elements corresponding to Z are 1 and those corresponding to Y are $\mathbf{y}$. $(\mathbf{1}, \mathbf{z})$ is defined in a similar way. Since f_{c_1} depends only on Y, $f_{c_1}(\mathbf{y}, \mathbf{z}) = f_{c_1}(\mathbf{y}, \mathbf{1})$ for any $\mathbf{y}, \mathbf{z}$. In a similar way, we can show that $f_{c_2}(\mathbf{y}, \mathbf{z}) = f_{c_2}(\mathbf{1}, \mathbf{z})$ for any $\mathbf{y}, \mathbf{z}$. Therefore, the equality (2) holds. CT(f_r) is the number of models of an st-DASC.

Next we show that CT(f) can be computed in $O(m_1)$ time. First, we compute the CT values for the literal and constant nodes as described above. Then every internal node's CT value can be computed by recursively applying (1) and (3) in a bottom-up manner. (1) and (3) denote that the CT value for any internal node can be computed in linear time in the number of outgoing edges from the node. Thus, the overall time is proportional to $O(m_1)$, where m_1 is the number of edges in the input st-DASC. This method can be easily extended to weighted model counting by multiplying the weight to a value corresponding to the leaf nodes.

¬C ¬C is a transformation that constructs a formula corresponding to the logical negation of given formula A. ¬C can be performed by preparing a $+$ node, attaching A to its child node with a -1-signed edge, and attaching a leaf node to the other with a label 1 and a 1-signed edge. The computational complexity is $O(1)$.

CD CD is a transformation that constructs a formula corresponding to $A|t$ for given formula A and consistent term t, and can be performed in the same way as in NNF. For any variable x, if x is included in t, then each leaf node labeled with x is replaced with constant node 1, and each leaf node labeled with $\neg x$ is replaced with constant node 0. Similarly, if $\neg x$ is included in t, then each leaf node labeled with $\neg x$ is replaced with constant node 1, and each leaf node labeled with x is replaced with constant node 0. Note that the result of this operation satisfies the restriction that each node corresponds to a Boolean function, since it corresponds to copying the outputs for certain assignments to other assignments. Since the entire tree is searched to re-label the leaf nodes, the computational complexity is $O(m_1)$.

Compilation from a term and a clause In preparation for proving that other operations can be performed in polytime, we show the following lemma.

Lemma 1. For any vtree and term t, an $O(|t|)$-sized st-DASC corresponding to t can be constructed. Similarly, for any vtree and clause γ, an $O(|\gamma|)$-sized st-DASC corresponding to γ can be constructed.

Proof. We construct an st-DASC corresponding to term t and a vtree as follows. First, we replace each leaf node of the vtree with a corresponding literal or value. For each leaf vtree node v with $\mathrm{Var}(\mathsf{v}) = x$, v is replaced by x if x is contained in t, replaced by $\neg x$ if $\neg x$ is contained in t, and replaced by 1 if neither is the case. Then each internal node of the vtree is replaced by a $\wedge$ node. Finally, the redundant nodes are contracted, i.e., the $\wedge$ node with 1 node as a child is eliminated and its parent is directly connected to the other child. The size is $O(|t|)$, since it has strictly $|t|$ leaf nodes and $2(|t| - 1)$ edges.

The st-DASC corresponding to a clause can be constructed as follows. Any clause with $|\gamma|$-literals can be represented as the negation of a term with $|\gamma|$-literals, and the st-DASC can perform negation by using a $+$ node with signed edges as shown above. Therefore, using negation and the above term construction, we can construct an st-DASC with $O(|\gamma|)$ size corresponding to any clause. □

CO, VA, CE, EQ, SE, IM These operations can be performed using CT, $\wedge$BC, CD, and $\neg$C. CO is a query to determine whether the model exists, which can be judged by whether the result of CT is non-zero. VA is a query to determine whether the function is always 1, that is, whether the result of CT equals 2^n. SE is a query to determine whether the model set of g includes the model set of f, which can be checked by whether $\mathrm{CT}(f \wedge g) = \mathrm{CT}(f)$. EQ is a query to determine whether two given functions are equal, which can be determined by performing SE on each of them. IM is identical as SE except that formula f is replaced by term t, which can be determined by whether $\mathrm{CT}(f|t) = 2^n$. CE, which determines whether the model set of clause γ includes the model set of formula f, also resembles SE, which can be reduced to IM on $\neg f$ and $\neg\gamma$. The computational complexity is as follows. CO and VA can be performed in $O(m_1)$, which is due to the complexity of CT. The computational complexity of SE and EQ are $O(m_1 m_2)$, since the most time-consuming part is taking the logical conjunction. The computational complexity of IM and CE is $O(m_1)$ since it can be performed by combining CD, CT, and $\neg$C; all of these operations can be performed in $O(m_1)$.

ME ME is a query that enumerates models of a given st-DASC of Boolean function f, which can be achieve by combining CD and CO. For details, see this work (Darwiche and Marquis 2002). The overall complexity is $O(n' m_1 \mathrm{CT}(f))$.

$\vee$BC $\vee$BC is a transformation that constructs a formula corresponding to the logical disjunction of given formulas A and B. Since $f \vee g = \neg(\neg f \wedge \neg g)$ by De Morgan's law, $\vee$BC can be achieved by combining the operations corresponding to $\wedge$BC and $\neg$C described above. The computational complexity and size are both $O(m_1 m_2)$, since the most time-consuming part is taking the logical conjunction.

SFO SFO is a transformation that constructs a formula corresponding to $(A|x) \vee (A|\neg x)$ for given formula A and variable x. Since CD and $\vee$BC can be calculated in polynomial time, SFO can also be calculated by combining them. The computational complexity is $O(m_1^2)$, since again the most time-consuming part is taking the logical disjunction.

Non-Polynomial Operations

The full proofs for the intractability of $\wedge$C, $\vee$C, and FO on the st-DASCs are found in the Appendix. In this subsection, we show proof sketches.

$\wedge$C, $\vee$C Assume that $\wedge$C can be performed in polynomial time on an st-DASC. Let $f = (x_1 \vee \neg y_1) \wedge (\neg x_1 \vee y_1) \wedge (x_2 \vee \neg y_2) \wedge (\neg x_2 \vee y_2) \wedge \cdots \wedge (x_n \vee \neg y_n) \wedge (\neg x_n \vee y_n)$. We represent f by an st-DASC with specific vtree T. Since each clause in f can be represented by linear size st-DASC, we can construct an st-DASC representing f whose size is polynomial in n by the assumption. However, we can also prove that the size of the st-DASC representing f must be exponential in n, which is a contradiction. Therefore, $\wedge$C cannot be performed in polynomial time on st-DASC.

Considering $\neg f$, we can similarly prove the intractability of $\vee$C.

FO We again consider function f appearing in the proof of the intractability of $\wedge$C. Let C_i be the i-th clause of f and $h = (z_1 \wedge \neg C_1) \vee (\neg z_1 \wedge z_2 \wedge \neg C_2) \vee \ldots (\neg z_1 \wedge \neg z_2 \wedge \cdots \wedge z_{2n} \wedge \neg C_{2n})$. We again consider the st-DASC representation of h with specific vtree T. We can prove that there is an st-DASC representing h whose size is polynomial in n. However, $\exists\{z_1, \ldots, z_{2n}\}.f = \neg C_1 \vee \cdots \vee \neg C_{2n} = \neg(C_1 \wedge \cdots \wedge C_{2n}) = \neg f$. If we assume that FO can be performed in polynomial time, since st-DASC can perform $\neg$C, we can construct an st-DASC representing f in polynomial time in the size of the st-DASC of h. This means that the size of the constructed st-DASC is polynomial in n. However, we already proved that the size of an st-DASC representing f becomes exponential in n, which is a contradiction. Thus, we cannot perform FO in polynomial time on an st-DASC.

Proof of Theorem 2

The proof of Theorem 2 consists of two elements. First we show the existence of a poly-size encoding from st-d-DNNFs to st-DASCs. Second, we show that some functions can be represented in polynomial-sized st-DASCs but cannot in polynomial-sized st-d-DNNFs. This can be shown using existing results for st-d-DNNFs.

Encoding from st-d-DNNFs to st-DASCs Encoding from an st-d-DNNF to an st-DASC can be accomplished by simply replacing the $\vee$ nodes with $+$ nodes having 1 signs for all edges heading to child nodes. The st-DASC constructed in this way represents the same Boolean function as the original st-d-DNNF since $\vee$ nodes can be replaced by $+$ nodes without changing the output value due to determinism. Therefore, st-DASCs are at least as succinct as st-d-DNNFs.

	SDD	Structured d-DNNF	st-DASC
CO	✓	✓	✓
VA	✓	✓	✓
CE	✓	✓	✓
IM	✓	✓	✓
EQ	✓	✓	✓
SE	✓	✓	✓
CT	✓	✓	✓
ME	✓	✓	✓

Table 2: Tractable queries for SDDs, st-d-DNNFs, and st-DASCs: Symbol ✓means that representation class can answer the query in polynomial time in representation size.

	SDD	Structured d-DNNF	st-DASC
CD	✓	✓	✓
FO	•	•	•
SFO	✓	•	✓
∧C	•	•	•
∧BC	✓	✓	✓
∨C	•	•	•
∨BC	✓	•	✓
¬C	✓	•	✓

Table 3: Tractable transformations for SDDs, st-d-DNNFs, and st-DASCs. Symbol ✓means that representation class can perform this transformation in polynomial time in representation sizes and • means it cannot.

Existence of the separation A previous result on the succinctness of st-d-DNNFs can be used to prove Theorem 2.

Theorem 3. (Vinall-Smeeth 2024) For every $n \in \mathbb{N}$, there exists Boolean function f with an equivalent st-d-DNNF of size n, such that any structured DNNF equivalent to $\neg f$ has size $n^{\tilde{\Omega}(\log n)}$.

Here, $\tilde{\Omega}(f)$ is identical as $\Omega(f)$ notation except that polylogarithmic factors of f are ignored.

Let us take function f, which can be represented by an st-d-DNNF of size n, although $\neg f$ cannot be represented in size polynomial of n by st-d-DNNFs. Since st-d-DNNFs are a subset of structured DNNFs, such a Boolean function f exists from Theorem 3. Since an st-d-DNNF can be encoded in an st-DASC in polynomial time, as shown in the previous section, f can be represented by an st-DASC of polynomial size. Since ¬C of an st-DASC can be performed in polynomial time, $\neg f$ can be represented by a polynomial-sized st-DASC. Therefore, $\neg f$ is a function that can be represented by an st-DASC of size polynomial in n but cannot by an st-d-DNNF of size polynomial in n.

Discussion

This section compares the polynomial operations of st-DASCs and other representation classes and gives an intuitive understanding of them.

Tables 2 and 3 show the possible operations on each representation class, where ✓denotes that the operation can be performed in polynomial time and • denotes it cannot be. One feature of st-DASCs is that they can perform operations equivalent to SDDs in polynomial time for the operations on the knowledge compilation map. Within st-DASCs, all the operations that can be performed in polynomial time in st-d-DNNFs can be performed, and additional operations, such as ¬C, ∨BC, and SFO, can also be performed in polynomial time. Furthermore, Theorem 2 shows that st-DASCs are more succinct than st-d-DNNFs. Such high succinctness and tractability of st-DASCs can be attributed to negation. St-DASCs can easily realize ¬C by a -1 sign and ∨BC and SFO can be reduced to ¬C. Furthermore, as shown in the proof of Theorem 2, the fact that negation can be taken also contributes to establishing the proof of succinctness.

Apparently, st-DASCs are more succinct and have more tractable operations than st-d-DNNFs, thus overcoming the trade-off discussed above. Nonetheless, there remains a possibility that certain operations, which are not listed in the knowledge compilation map, can be performed on st-d-DNNFs but not on st-DASCs. An example is the Boolean programming problem (Knuth 2011), which is a query for finding the model that maximizes the sum of weights associated with each literal. It resembles the maximum a posteriori query (MAP) of probabilistic circuits. The Boolean programming problem can be performed in polynomial time on st-d-DNNFs, although it remains unclear whether it can be performed on st-DASCs. Therefore, st-DASCs are not necessarily always superior to st-d-DNNFs.

Conclusion

In this paper, we proposed st-DASCs as representations of Boolean functions and analyzed their succinctness and possible operations in polytime. The results show that st-DASCs are more succinct than st-d-DNNFs, and that among the operations in the knowledge compilation map, those equivalent to SDDs can be performed in polynomial time on st-DASCs.

Exponential separation is an area of future exploration. Since this paper shows only a quasi-polynomial separation between st-d-DNNFs and st-DASCs, determining whether an exponential separation can be located between them remains future work. We can use a compilation algorithm for st-d-DNNFs as well as for st-DASCs since there is encoding from an st-d-DNNF to an st-DASC. However, whether a better compilation can be achieved is another issue to be addressed. This paper deals with the case that edge is associated with the sign, but we would like to investigate whether the succinctness can be improved when signs extended to integer weights.

Appendix: Proofs for Non-Polynomial Operations

In this section, we prove that ∧C, ∨C, and FO cannot be performed in polynomial time on st-DASCs.

Before proceeding to the proof, we define the notion of

linear vtrees. A vtree is linear if the left child of every internal node is always a leaf node. Within a linear vtree, all leaves have different depths, except for the lowest two. Thus, given a total order among input binary variables, we can construct a corresponding linear vtree by assigning variables to leaves such that the larger variable according to the total order is assigned to the deeper leaf. Here, the second largest variable is assigned to the left lowest leaf and the largest variable to the right lowest leaf.

$\wedge$C, $\vee$C Assume that $\wedge$C can be performed in polynomial time on st-DASC. Let $f = (x_1 \vee \neg y_1) \wedge (\neg x_1 \vee y_1) \wedge (x_2 \vee \neg y_2) \wedge (\neg x_2 \vee y_2) \wedge \cdots \wedge (x_n \vee \neg y_n) \wedge (\neg x_n \vee y_n)$. We represent f by an st-DASC respecting linear vtree T constructed from ordering $x_1 < \cdots < x_n < y_1 < \cdots < y_n$. Since each clause can be represented by a linear-sized st-DASC, f can be represented by a polynomial-sized st-DASC by assumption.

Function f is evaluated as 1 when $(x_1, x_2 \ldots, x_n) = (y_1, y_2, \ldots, y_n)$. Now we consider partial assignment $(x_1, \ldots, x_n) = (a_1, \ldots, a_n) =: \mathbf{a}$. We also consider corresponding term $t_{\mathbf{a}} \coloneqq l_1 \wedge \cdots \wedge l_n$, where l_i is x_i if $a_i = 1$, or $\neg x_i$ otherwise. Then conditioned function $f|t_{\mathbf{a}}$ is evaluated as 1 when $(y_1, \ldots, y_n) = \mathbf{a}$. Considering the CD operation, the st-DASC representing $f|t_{\mathbf{a}}$ can be built by replacing all the leaves labeled with x_i or $\neg x_i$ with either 0 or 1. After that, any $\wedge$ node v, where $\mathrm{Var}(v) = \mathrm{Var}(r)$ for root r and the left child of v is a constant node, can be eliminated by replacing it with either a 0 node (when the left child is a 0 node) or the right child (when the left child is a 1 node). After that, adjacent $+$ nodes can also be combined into one $+$ node. This means that we can construct an st-DASC representing $f|t_{\mathbf{a}}$ such that the root node is $+$ and every child of the root is also present in the original st-DASC of f.

Since the original st-DASC of f is polynomial-sized, by fixing a polynomial number of Boolean functions, we can represent $f|t_{\mathbf{a}}$ for any $\mathbf{a}$ by a linear combination of the fixed functions. In other words, if we consider a Boolean function of n variables as a vector of 2^n elements by lining up the outputs of the function for all possible assignments, the rank of the set of all conditioned functions, i.e., $\{f|t_{\mathbf{a}} \mid \mathbf{a}\}$, is polynomial. However, each $f|t_{\mathbf{a}}$ is a standard unit vector where only one element is 1 and the other elements are 0, and the vectors are different if the assignments are also different. This means that the rank of $\{f|t_{\mathbf{a}} \mid \mathbf{a}\}$ is 2^n. This contradicts the proposition that its rank is polynomial, and thus we prove that $\wedge$C cannot be performed in polynomial time.

Considering $\neg f$, we can similarly prove the intractability of $\vee$C.

FO We again consider function f appearing in the proof of intractability of $\wedge$C. Let C_i be the i-th clause of f and $h = (z_1 \wedge \neg C_1) \vee (\neg z_1 \wedge z_2 \wedge \neg C_2) \vee \ldots (\neg z_1 \wedge \neg z_2 \wedge \cdots \wedge z_{2n} \wedge \neg C_{2n})$. We consider the st-DASC representation of h respecting linear vtree T constructed from ordering $z_1 < \cdots < z_{2n} < x_1 < \cdots < x_n < y_1 < \cdots < y_n$. We later prove that there is an st-DASC representing h whose size is polynomial in n. However, $\exists\{z_1, \ldots, z_{2n}\}.f = \neg C_1 \vee \cdots \vee \neg C_{2n} = \neg(C_1 \wedge \cdots \wedge C_{2n}) = \neg f$. If we assume that FO can be performed in polynomial time, since st-DASC can perform $\neg$C, we can construct an st-DASC representing f in polynomial time with respect to the size of an st-DASC of h. This means that the size of the constructed st-DASC is polynomial in n. However, we already proved that the size of an st-DASC representing f becomes exponential in n, which is a contradiction. Thus, we cannot perform FO in polynomial time on st-DASCs.

Below, we prove that h can be represented as an st-DASC of polynomial size. Let $v_1, \ldots, v_{2n}$ be a $+$ node satisfying the following: (I) the left child of v_i is a $\wedge$ node whose left child is leaf z_i and right child represents $\neg C_i$. (II) For $i = 1, \ldots, 2n-1$, the right child of v_i is a $\wedge$ node whose left child is leaf $\neg z_i$ and whose right child is v_{i+1}. (III) The right child of v_{2n} is just a 0 node. We can easily observe that the st-DASC rooted at v_1 respects vtree T and it represents the function h. Since every C_i can be represented by a linear-sized st-DASC, the overall size is polynomial in n.

Acknowledgements

The authors thank the anonymous reviewers for their valuable feedback, corrections, and suggestions. This work was supported by JST CREST (Grant Number JPMJCR22D3, Japan) and JSPS KAKENHI (Grant Number JP20H05963, Japan).

References

Ahmed, K.; Teso, S.; Chang, K.-W.; Van den Broeck, G.; and Vergari, A. 2022. Semantic probabilistic layers for neuro-symbolic learning. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, 29944–29959. Curran Associates, Inc.

Álvaro Torralba; Alcázar, V.; Kissmann, P.; and Edelkamp, S. 2017. Efficient symbolic search for cost-optimal planning. *Artificial Intelligence*, 242: 52–79.

Bonet, B.; and Geffner, H. 2006. Heuristics for planning with penalties and rewards using compiled knowledge. In *Proceedings of the 10th International Conference on Principles of Knowledge Representation and Reasoning*, 452–462.

Bryant, R. E. 1986. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers*, C-35(8): 677–691.

Chaki, S.; and Gurfinkel, A. 2018. *BDD-Based Symbolic Model Checking*, 219–245. Springer International Publishing. ISBN 978-3-319-10575-8.

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

Choi, A.; Kisa, D.; and Darwiche, A. 2013. Compiling probabilistic graphical models using sentential decision diagrams. In *In Proceedings of the 12th European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty*, 121–132.

Choi, Y.; Vergari, A.; and Van den Broeck, G. 2020. Probabilistic circuits: a unifying framework for tractable probabilistic models. Available at: http://starai.cs.ucla.edu/papers/ProbCirc20.pdf.

Darwiche, A. 1999. Compiling knowledge into decomposable negation normal form. In *Proceedings of the 16th International Joint Conference on Artifical Intelligence*, 284–289.

Darwiche, A. 2003. A differential approach to inference in Bayesian networks. *Journal of the ACM*, 50(3): 280–305.

Darwiche, A. 2011. SDD: A new canonical representation of propositional knowledge bases. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence*, 819–826.

Darwiche, A. 2014. *Tractable Knowledge Representation Formalisms*, 141–172. Cambridge University Press.

Darwiche, A.; and Marquis, P. 2002. A knowledge compilation map. *Journal of Artificial Intelligence Research*, 17(1): 229–264.

Hardy, G.; Lucet, C.; and Limnios, N. 2007. K-terminal network reliability measures with binary decision diagrams. *IEEE Transactions on Reliability*, 56(3): 506–515.

Huang, J.; and Darwiche, A. 2005. On compiling system models for faster and more scalable diagnosis. In *Proceedings of the 20th National Conference on Artificial Intelligence*, volume 1, 300–306.

Inoue, T.; Takano, K.; Watanabe, T.; Kawahara, J.; Yoshinaka, R.; Kishimoto, A.; Tsuda, K.; Minato, S.; and Hayashi, Y. 2014. Distribution loss minimization with guaranteed error bound. *IEEE Transactions on Smart Grid*, 5(1): 102–111.

Kisa, D.; Van den Broeck, G.; Choi, A.; and Darwiche, A. 2014. Probabilistic sentential decision diagrams. In *Proceedings of the 14th International Conference on the Principles of Knowledge Representation and Reasoning*, 558–567.

Knuth, D. E. 2011. *The Art of Computer Programming, Volume 4A: Combinatorial Algorithms, Part 1*. Addison-Wesley Professional.

Maehara, T.; Suzuki, H.; and Ishihata, M. 2017. Exact computation of influence spread by binary decision diagrams. In *Proceedings of the 26th International Conference on World Wide Web*, 947–956.

Manhaeve, R.; Dumancic, S.; Kimmig, A.; Demeester, T.; and De Raedt, L. 2018. DeepProbLog: neural probabilistic logic programming. In *Proceedings of Advances in Neural Information Processing Systems 31 (NeurIPS 2018)*.

Pipatsrisawat, K.; and Darwiche, A. 2007. Clone: solving weighted Max-SAT in a reduced search space. In *Proceedings of the 20th Australian Joint Conference on Advances in Artificial Intelligence*, 223–233.

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, 517–522.

Poon, H.; and Domingos, P. 2011. Sum-product networks: a new deep architecture. In *Proceedings of the 27th Conference on Uncertainty in Artificial Intelligence*, 337–346.

Shen, Y.; Choi, A.; and Darwiche, A. 2016. Tractable operations for arithmetic circuits of probabilistic models. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, 3943–3951.

Vergari, A.; Choi, Y.; Liu, A.; Teso, S.; and Van den Broeck, G. 2021. A compositional atlas of tractable circuit operations for probabilistic inference. In *Proceedings of the 35th International Conference on Neural Information Processing Systems*, 13189–13201.

Vinall-Smeeth, H. 2024. Structured d-DNNF is not closed under negation. In *Proceedings of the 33rd International Joint Conference on Artificial Intelligence*, 3593–3601.

Xu, J.; Zhang, Z.; Friedman, T.; Liang, Y.; and Van den Broeck, G. 2018. A semantic loss function for deep learning with symbolic knowledge. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *PMLR*, 5502–5511.