Tensor Decomposition Meets Knowledge Compilation: A Study Comparing Tensor Trains with OBDDs

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

A knowledge compilation map analyzes tractable operations in Boolean function representations and compares their succinctness. This enables the selection of appropriate representations for different applications. In the knowledge compilation map, all representation classes are subsets of the negation normal form (NNF). However, Boolean functions may be better expressed by a representation that is different from that of the NNF subsets. In this study, we treat tensor trains as Boolean function representations and analyze their succinctness and tractability. Our study is the first to evaluate the expressiveness of a tensor decomposition method using criteria from knowledge compilation literature. Our main results demonstrate that tensor trains are more succinct than ordered binary decision diagrams (OBDDs) and support the same polytime operations as OBDDs. Our study broadens their application by providing a theoretical link between tensor decomposition and existing NNF subsets.

Introduction

Knowledge compilation, a technique for handling propositional reasoning at high-speed, has been studied for decades. The logical formula in propositional logic can be converted into a representation by pre-computation, allowing various queries to be quickly answered on such converted representations. Typical target representations for knowledge compilation include *Ordered Binary Decision Diagrams (OBDDs)* (Bryant 1986), *deterministic Decomposable Negation Normal Form (d-DNNF)* (Darwiche 2001b), and *Sentential Decision Diagrams (SDDs)* (Darwiche 2011). *A Negation Normal Form (NNF)* (Darwiche 2001a) is a generalization of all these approaches and Boolean function representation classes can be considered as adding constraints, such as determinism and structured decomposability, to the NNF. However, the possibility remains that Boolean functions can be represented more succinctly by focusing on representations other than NNF.

Tensor decomposition represents a tensor as the sum of the products of smaller tensors. Tensor decomposition resembles knowledge compilation because it succinctly represents a large tensor with a set of smaller tensors. A tensor train decomposes a high-dimensional tensor into a linear product of lower-dimensional tensors (Oseledets 2011). Many operations can be performed on a tensor train, such as sum, inner product, and Hadamard product.

As the truth table of an n-ary Boolean function can naturally be regarded as an n-dimensional binary tensor, an exact tensor decomposition representing such a binary tensor can be regarded as a target representation for knowledge compilation. Our study measures the performance of a tensor train as a representation of a Boolean function. For comparison with existing Boolean function representations, we follow the approach of the knowledge compilation map (Darwiche and Marquis 2002), which clarifies the succinctness between Boolean function representations and the operations that can be performed on each representation in polytime. In this study, we analyze the succinctness between OBDDs and tensor trains. In addition to the popularity of OBDDs, considering that a linear ordering of variables can be introduced into tensor trains such as OBDDs, we argue that a comparison between tensor trains and OBDDs would be beneficial to the community.

The main result of this study is twofold. First, we show that tensor trains (deemed to be Boolean function representations) are more succinct than OBDDs; that is, they can represent any function with at most a polynomial times larger size than OBDDs and a function that cannot be represented by OBDDs with a polynomial times larger size than tensor trains exists. Second, we prove that tensor trains can perform all operations on the knowledge compilation map (Darwiche and Marquis 2002) that OBDDs can perform in polynomial time (polytime) with respect to the size of the representation. These results are interesting because there is usually a trade-off between succinctness and the abundance of operations that can be performed in polytime, as suggested by knowledge compilation maps. Moreover, because OBDDs are among the most powerful representations of many NNF subsets in terms of the number of operations supported in polytime, it is surprising that the tractability of operations of tensor trains and OBDDs are identical. It is also remarkable that tensor trains emerged from a different field than NNFs; to the best of our knowledge, this is the first study that argues that tensor decomposition provides more succinctness and tractability in representing a Boolean function than NNF subsets. The full version of this paper will be available at arXiv.

Related Works

Since the publication of an innovative paper (Darwiche and Marquis 2002), the knowledge compilation map has been extended when new knowledge representation languages appear (Fargier and Marquis 2008; Darwiche 2014).

The knowledge compilation map has contributed to many varied applications and has also been extended beyond Boolean functions (Fargier, Marquis, and Niveau 2013; Choi, Vergari, and Van den Broeck 2020). However, tensor decomposition-based methods have not been evaluated with these maps. Knowledge compilation maps, which play an important role in understanding knowledge representation languages, have fueled the emergence of several important applications of knowledge compilation techniques (Chavira and Darwiche 2008; Manhaeve et al. 2018; Xu et al. 2018). An approach similar to tensor decomposition is expected to contribute to numerous applications.

Tensor decomposition (Kolda and Bader 2009) is a method to represent a large-scale high-dimensional tensor as the sum of the products of smaller or low-dimensional tensors. High-dimensional tensors are frequently used in research areas such as machine learning, numerical calculations, and quantum physics. Because an m-dimensional tensor has an exponential number of elements with respect to m, tensor decomposition methods are indispensable for addressing high-dimensional tensors. Canonical Polyadic (CP) decomposition (Carroll and Chang 1970) and Tucker decomposition (Tucker 1966) are the initial examples; other variants have been proposed, including hierarchical Tucker decomposition (Hackbusch and Kühn 2009), which is more succinct than CP decomposition (Khrulkov, Novikov, and Oseledets 2018). However, these decompositions do not place much emphasis on operations. As a tensor decomposition that supports several operations, a tensor train (Oseledets 2011) was proposed, which decomposes an m-dimensional tensor into the product of m 3-dimensional tensors. Tensor trains were originally invented in the quantum physics community as matrix product states (Fannes, Nachtergaele, and Werner 1992). Several tensor operations for tensor trains were developed in a previous study whose runtimes are polynomial with the decomposition size(Oseledets 2011).

Two other studies bridge Boolean function representations and tensor decompositions: Tensor Decision Diagrams (Hong et al. 2022) and Quantum Multiple-valued Decision Diagrams (QMDD) (Niemann et al. 2016). Both represent tensors as decision graphs, similar to OBDDs. However, these studies addressed the representation of real-valued tensors instead of Boolean functions. Although a method for model counting using tensor decomposition (Dudek and Vardi 2020) exists, this method does not introduce tensor decomposition into the knowledge compilation or analyze its succinctness or tractability.

Our study focuses on the use of tensor decomposition methods as representations of Boolean functions and analyzes their succinctness and tractable Boolean function operations. To the best of our knowledge, no similar studies have been previously conducted.

Preliminaries

We briefly introduce the OBDD, tensor train, and knowledge compilation map. For convenience, we describe the values that a binary variable can assume as 0 and 1, which corresponds to *false* and *true* in the ordinary Boolean expression. Within such an expression, for binary values a and b, the conjunction $a \wedge b$, disjunction $a \vee b$, and negation $\neg a$ can be computed by $\min\{a, b\}$, $\max\{a, b\}$, and $1 - a$, respectively.

An *n-ary Boolean function* considers n binary variables as inputs and returns a binary output. The set of assignments of the input variables such that the output of the Boolean function is 1 is known as a *model*. A Boolean function is *consistent* when it includes a model. We introduce operations that transform one Boolean function into another.

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

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

OBDD: An OBDD is a data structure that represents a Boolean function with a rooted directed acyclic graph, $B = (N, A)$, where N denotes a node set and A denotes a set of oriented edges. Its structure is as follows. We denote the root of the graph by $R \in N$. Node set N comprises two terminal $\{\bot, \top\}$ and non-terminal nodes. From each non-terminal node $v \in N \setminus \{\bot, \top\}$, exactly two oriented edges exist: LO and HI. We denote the nodes to which these edges indicate as $\mathsf{lo}(v)$ and $\mathsf{hi}(v)$. $v \in N \setminus \{\bot, \top\}$ is given a corresponding label for just one of the input variables, denoted by $\mathsf{var}(v)$. Each directed edge must have a direction that is consistent with the determined order of the input variables. This structure is known as a *variable order*. For each node $v \in N$ of the OBDD, the corresponding Boolean function f_v is defined as follows:

(i) If $v = \bot$, then $f_v = 0$ for any assignment of input.

(ii) If $v = \top$, then $f_v = 1$ for any assignment of input.

(iii) If $v \in N \setminus \{\bot, \top\}$, then $f_v = (\neg \mathsf{var}(v) \wedge f_{\mathsf{lo}(v)}) \vee (\mathsf{var}(v) \wedge f_{\mathsf{hi}(v)})$.

We denote the function corresponding to the OBDD with R as the root using f_R. The *size* of an OBDD comprises the number of its nodes. When considering the operations on two OBDDs, those that can be performed depend on whether their variable orders are identical. $\text{OBDD}_<$ is defined to distinguish between the two.

Definition 3. For the total order $<$ among the input variables, $\text{OBDD}_<$ is a subset of the OBDD satisfying the following condition: if node v_1 is an ancestor of v_2, then $\mathsf{var}(v_1) < \mathsf{var}(v_2)$.

Figure 1 shows an example of an OBDD with variable order $x_1 < x_2 < x_3$ representing Boolean function $f = (\neg x_1 \wedge \neg x_2) \vee (\neg x_1 \wedge \neg x_3) \vee (\neg x_2 \wedge \neg x_3)$. Terminal nodes are depicted as squares, whereas internal nodes are represented by circles. The LO and HI edges are indicated by the dashed and solid arrows, respectively.

Tensor train: A tensor is an m-dimensional array with $d_1 \times \cdots \times d_m$ elements. Each dimension of a tensor is known as a *mode*. We denote the element of tensor $\mathcal{X}$ that corresponds to $(a_1, \ldots, a_m)$ by $\mathcal{X}(a_1, \ldots, a_m)$, where $a_j \in \{0, \ldots, d_j - 1\}$ for $1 \leq j \leq m$. For an m-dimensional tensor $\mathcal{X}$ of size $d_1 \times \cdots \times d_m$ and an ℓ-dimensional tensor $\mathcal{Y}$ of size $d'_1 \times \cdots \times d'_\ell$, where $d_m = d'_1$, *mode product* $\mathcal{XY}$ satisfies

$$\mathcal{XY}(a_1, \ldots, a_{m-1}, a'_2, \ldots, a'_\ell) = \sum_{i=0}^{d_m - 1} \mathcal{X}(a_1, \ldots, a_{m-1}, i)\mathcal{Y}(i, a'_2, \ldots, a'_\ell)$$

for any $a_1, \ldots, a_{m-1}, a'_2, \ldots, a'_\ell$. This mode product $\mathcal{XY}$ result in an $(m + \ell - 2)$-dimensional tensor of size $d_1 \times \cdots \times d_{m-1} \times d'_2 \times \cdots \times d'_\ell$. Note that this is a generalization of the matrix product to the tensors.

Tensor train (Oseledets 2011) represents $d_1 \times \cdots \times d_m$ tensor $\mathcal{T}$ using $r_i \times d_i \times r_{i+1}$ tensors $\mathcal{A}_i$ as $\mathcal{T} = \mathcal{A}_1 \mathcal{A}_2 \cdots \mathcal{A}_m$, where $r_1 = r_{m+1} = 1$. $r = \max r_i$ is known as the *rank* of the tensor train. The size of the tensor train representation is the sum of the sizes of $\mathcal{A}_i$. The size of $\mathcal{A}_i$ is $r_i d_i r_{i+1}$, the number of the elements. Because $r_i d_i r_{i+1} \leq r^2 d_i$, the size of the tensor train representation is bounded by $r^2 \sum_i d_i$. In the following, we focus on $\overbrace{2 \times \cdots \times 2}^{m}$ tensors and denote them as $2^{\times m}$ tensors. For such tensors, $\sum_i d_i = \mathcal{O}(m)$; thus, the number of elements in the tensor train representation is at most $\mathcal{O}(mr^2)$.

In the following, we explain the use of tensor trains to represent Boolean functions. The following definition implies that a Boolean function can be represented by a tensor while skipping certain variables.

Definition 4. Let $\mathcal{T}$ be a $2^{\times m}$ binary tensor with $1 \leq m \leq n$ and $\pi : [m] \to [n]$ be a mapping, where $[n] = \{1, \ldots, n\}$. A pair $(\mathcal{T}, \pi)$ is a tensor representation of an n-ary Boolean function representing an n-ary Boolean function f whose output equals $\mathcal{T}(a_1, \ldots, a_m)$ for the input assignments of variables $x_1, \ldots, x_n$ satisfying $x_j = a_{\pi^{-1}(j)}$ if $j \in Q$, where $(a_1, \ldots, a_m) \in \{0, 1\}^m$ and $Q = \{\pi(j) \mid j \in [m]\}$.

For instance, $f(x_1, x_2) = x_1 \vee \neg x_2$ can be represented by pair $(\mathcal{T}, \pi)$, where $\mathcal{T}$ is 2×2 tensor (= matrix)

$$\mathcal{T} = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}, \tag{1}$$

and $\pi(1) = 1$ and $\pi(2) = 2$. Similarly, $f(x_1, x_2) = \neg x_2$ can be represented as a pair of $\mathcal{T} = (1, 0)^\top$ and $\pi(1) = 2$.

Now, we can assume that a tensor train is a succinct representation of a Boolean function:

Definition 5. Let $(\mathcal{T}, \pi)$ be a tensor representation of an n-ary Boolean function f, where $\mathcal{T}$ is a $2^{\times m}$ binary tensor. A pair of tensor sequences, $\mathcal{A}_1, \ldots, \mathcal{A}_m$ and π, is a tensor train representation of f, where $\mathcal{A}_i$ for $i \in \{1, \ldots, m\}$ are 3-dimensional ternary tensors; that is, every element of $\mathcal{A}_i$ is in $\{-1, 0, 1\}$ and satisfies $\mathcal{T} = \mathcal{A}_1 \cdots \mathcal{A}_m$, i.e., $\mathcal{A}_1 \cdots \mathcal{A}_m$ is a tensor train decomposition of $\mathcal{T}$.

For instance, binary tensor $\mathcal{T}$ in Eq. (1) can be represented as a tensor train $\mathcal{T} = \mathcal{A}_1 \mathcal{A}_2$, where $\mathcal{A}_1$ and $\mathcal{A}_2$ are $1 \times 2 \times 2$ and $2 \times 2 \times 1$ tensors defined as:

$$\mathcal{A}_1 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \quad \mathcal{A}_2 = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}.$$

The requirement that $\mathcal{A}_1, \ldots, \mathcal{A}_m$ are ternary tensors ensures that a constant number of bits are required to represent each element of $\mathcal{A}_i$. We use this fact to compare the succinctness of the OBDD$_<$ and tensor trains in later sections. As shown below, this requirement does not change the set of polytime operations supported by tensor trains. We use notation $\mathcal{A}_i(\cdot, b, \cdot)$ for representing the slice of the $r \times d \times q$ tensor with $b \in \{0, \ldots, d - 1\}$, which corresponds to the $r \times q$ matrix. Given tensor train $\mathcal{T} = \mathcal{A}_1 \cdots \mathcal{A}_m$, we can represent $\mathcal{T}(a_1, \ldots, a_m)$ using these matrices as follows:

$$\mathcal{T}(a_1, \ldots, a_m) = \mathcal{A}_1(\cdot, a_1, \cdot) \cdots \mathcal{A}_m(\cdot, a_m, \cdot). \tag{2}$$

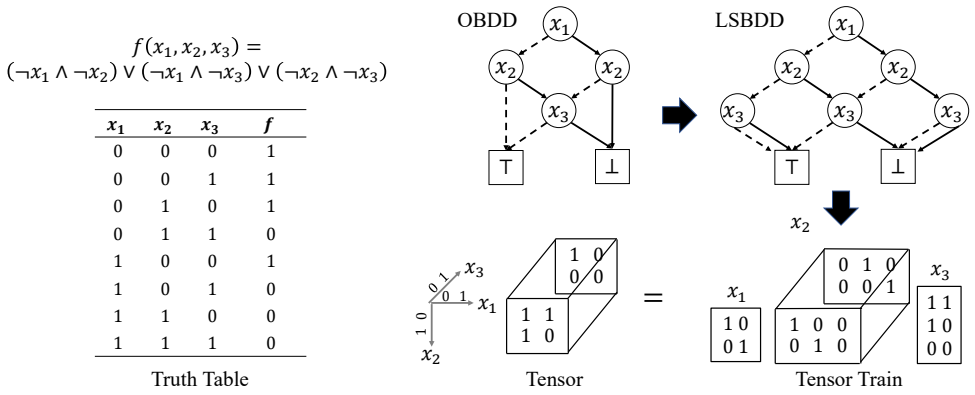
We say a Boolean function is represented in TT language if it is represented as a pair of tensor trains $\mathcal{A}_1, \ldots, \mathcal{A}_m$ and mapping π defined above. Moreover, given a total order $<$ over the input variables $x_1, \ldots, x_n$ of an n-ary Boolean function f, we define a subclass of TT following $<$.

Definition 6. For a total order $<$ among input variables $x_1, \ldots, x_n$, TT$_<$ is a subclass of TT where mapping π satisfies $x_{\pi(i)} < x_{\pi(j)}$ for all $1 \leq i < j \leq m$. We say such mapping π *follows* variable order $<$.

Knowledge compilation map: The knowledge compilation map (Darwiche and Marquis 2002) is an attempt to compare different target representations of knowledge compilation. Since the publication of a seminal paper (Darwiche and Marquis 2002), researchers have been evaluating representations of Boolean functions based on two key dimensions introduced in the paper: the succinctness of the target compilation language and the class of queries and transformations supported by representations in polytime.

Succinctness evaluates the space efficiency of the representations. Because the existing representations for Boolean functions in the literature are all NNF subsets, succinctness has been evaluated by the number of edges in an NNF. This assumption is justified by considering the space required for both representations. The space required for tensor train representation is $\mathcal{O}(\sum_{i=1}^m |\mathcal{A}_i| + m \log n)$ bits if we limit the possible values of elements of $\mathcal{A}_i$ to $\{-1, 0, 1\}$. Here, $m \leq \sum_{i=1}^m |\mathcal{A}_i|$ holds because $|\mathcal{A}_i| > 0$ for every i. In contrast, OBDD B needs at least $\Omega(|B| \log n)$ bits to represent a non-terminal node because every node is associated with a variable. Therefore, to retain the tensor train representation of size $S(\coloneqq \sum_i |\mathcal{A}_i|)$, we only need the most constant factor of space required for the OBDD representation of the same size $S(\coloneqq |B|)$.

Queries and transformations are operations related to Boolean functions. Although queries do not change the



x_1	x_2	x_3	f
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

Figure 1: Example of transforming an OBDD into an equivalent tensor train. We first transform an OBDD into an equivalent level-wise smooth OBDD (LSBDD) and then construct a tensor train from the LSBDD. The OBDD and the LSBDD follow the variable order $x_1 < x_2 < x_3$ and the π of the tensor train is defined as $\pi(i) = i$, which follows the order.

Boolean functions, a transformation returns a modified Boolean function as its output. In knowledge compilation literature, representations are evaluated based on whether they support polytime operations over a set of queries and transformations. We use the following lemma to determine whether a tensor train supports an operation in polytime:

Lemma 1. The size of the tensor train representation, $\sum_{i=1}^{m} |\mathcal{A}_i|$, of a Boolean function can be lower bounded by $r + m$, where r denotes the rank and m denotes the number of modes.

Proof. Because $r = \max r_i$, at least one tensor $\mathcal{A}_i$ whose size $2r_i r_{i+1}$ is larger than r exists and the size of every tensor is $|\mathcal{A}_i| > 0$. Therefore, $\sum_{i=1}^{m} |\mathcal{A}_i| > r + (m-1)$ and, thus, $\sum_{i=1}^{m} |\mathcal{A}_i| \geq r + m$. □

According to Lemma 1, any polynomial comprising r and m is also a polynomial of the size of the tensor train representation. Therefore, we prove the polytime complexity of operations on tensor train representations by showing the complexities of the polynomials of r and m.

Main Results

We present the results of evaluating the succinctness and tractability of the operations of tensor trains. First, we introduce the notion of succinctness to compare the expressive efficiencies of the OBDDs and tensor trains.

Definition 7. (Darwiche and Marquis 2002) For two representation classes (A and B) of a Boolean function, A is described as at least as *succinct* as B, denoted as $A \leq B$, if polynomial p exists such that, for every representation $\beta \in B$, equivalent representation $\alpha \in A$ where $|\alpha| \leq p(|\beta|)$ exists. Here, $|\alpha|$ and $|\beta|$ are the sizes of α and β, respectively. Furthermore, representation class A is more *succinct* than B if it is at least as succinct as B, even though B is not as succinct as A.

The following are the main results of this study.

Theorem 1. $\text{TT}_<$ is more succinct than $\text{OBDD}_<$.

This theorem implies that any OBDD β of size $|\beta|$ can be translated into tensor train α whose size $|\alpha|$ satisfies $|\alpha| \leq p(|\beta|)$ with polynomial p. In contrast, a sequence of Boolean functions $f_1, f_2, \ldots$ exists such that the size of the OBDD representing f_i is exponentially larger than that of the corresponding $\text{TT}_<$.

Moreover, we show that all queries and transformations introduced by (Darwiche and Marquis 2002) that can be performed in polytime with $\text{OBDD}_<$ can also be performed in polytime with $\text{TT}_<$. Table 1 summarizes an explanation of the computational complexity of each operation.

Theorem 2. Suppose that Boolean functions f, g are represented as $\text{TT}_<$ with an identical variable order $<$. Then, CO, VA, EQ, SE, CT, $\wedge$BC, $\vee$BC, $\neg$C, and SFO can be performed in polytime with respect to the size of the tensor train representations of f, g. CE, IM, and CD can be performed in polytime with respect to the tensor representation size of f and the number of literals in the clause (for CE) or the term (for IM and CD). ME can be performed in polytime with respect to the tensor train representation size of f and the number of models.

We also proved that the tensor train does not support transformations that $\text{OBDD}_<$ does not support.

Theorem 3. $\wedge$C, $\vee$C, and FO cannot be performed in polytime in the size of tensor trains.

The proofs of Theorems 2 and 3 are provided in the full version. This high-level concept is presented in the next section.

These theorems show that $\text{TT}_<$ achieves a good trade-off between its succinctness and the richness of the supporting operations, as $\text{TT}_<$ is more succinct than $\text{OBDD}_<$ while it supports the same set of polytime operations. The only existing representation that achieves the same trade-off as OBDD is SDD, which supports the same set of queries and operations as OBDD and is more succinct than OBDD (Bova 2016). However, TT is clearly different from SDD, because

	Name	Explanation	Complexity
Query	CO	Answering whether f has a model.	$\mathcal{O}(mr^2)$
	VA	Answering whether f is valid; that is, f evaluates to 1 for any assignment of variables.	$\mathcal{O}(mr^2)$
	CE	Answering whether $f \models \gamma$ holds for a clause γ.	$\mathcal{O}((m+\ell)r^2)$
	EQ	Answering whether f and g are equivalent.	$\mathcal{O}((m+\ell)rq(r+q))$
	SE	Answering whether $f \models g$ holds.	$\mathcal{O}((m+\ell)rq(r+q))$
	IM	Answering whether $\gamma \models f$ holds for a term γ.	$\mathcal{O}((m+\ell)r^2)$
	CT	Counting the number of models of f.	$\mathcal{O}(mr^2)$
	ME	Enumerating the models of f.	$\mathcal{O}(Mmr^2)$
Transformation	$\wedge$BC	Constructing $\mathrm{TT}_<$ representing the logical conjunction of f, g.	$\mathcal{O}((m+\ell)r^2q^2)$
	$\vee$BC	Constructing $\mathrm{TT}_<$ representing the logical disjunction of f, g.	$\mathcal{O}((m+\ell)r^2q^2)$
	$\neg$C	Constructing $\mathrm{TT}_<$ representing the logical negation of f.	$\mathcal{O}(mr^2)$
	CD	Constructing $\mathrm{TT}_<$ representing $f\|\gamma$, f conditioned by a ℓ-literals term γ.	$\mathcal{O}(\ell r^2)$
	SFO	Constructing $\mathrm{TT}_<$ representing $\exists x.f$. for a variable x.	$\mathcal{O}(mr^4)$
	$\wedge$C	Constructing $\mathrm{TT}_<$ representing the logical conjunction of $f_1, f_2, \ldots, f_k$.	•
	$\vee$C	Constructing $\mathrm{TT}_<$ representing the logical disjunction of $f_1, f_2, \ldots, f_k$.	•
	FO	Constructing $\mathrm{TT}_<$ representing $\exists \mathbf{X}.f$ for a variable set $\mathbf{X}$.	•

Table 1: Explanations of operations and computational complexity on tensor trains. We use m and r (resp. ℓ and q) to represent the number of modes and the tensor train rank corresponding to f (resp. g). We also use ℓ as the number of modes of a tensor train representing a term or a clause. M denotes the number of models of f. Symbol • implies that the operation cannot be performed in polytime on the tensor train regardless of whether P $\neq$ NP.

the latter is a subset of NNF, whereas TT is not. Therefore, $\mathrm{TT}_<$ is a unique representation that is not equivalent to any existing representation in the knowledge compilation map.

Proofs

High-Level Ideas

Before proceeding to detailed proofs of the theorems, we describe the high-level ideas. First, we refer to the succinctness of the tensor train representation (Theorem 1). We show that $\mathrm{TT}_<$ is as succinct as $\mathrm{OBDD}_<$ by providing a transformation scheme from an OBDD to a tensor train in a layer-wise manner; that is, we design each tensor $\mathcal{A}_i$ such that it represents a set of OBDD nodes corresponding to the i-th variable. The transformation is illustrated in Figure 1. This transformation only increases the size by a polynomial factor and we can prove that $\mathrm{TT}_<$ is as succinct as $\mathrm{OBDD}_<$. Second, we show that $\mathrm{OBDD}_<$ is not as succinct as $\mathrm{TT}_<$ using hidden-weighted bit (HWB) functions, which are known to result in an exponential-size OBDD. We demonstrate that a rank $2n$ tensor train is sufficient to represent an n-ary HWB.

In the proof of Theorem 2, we show that most of the operations can be written as a combination of the sum, inner product, and Hadamard product (i.e., element-wise product) of tensors. For instance, considering that output of the tensor train is limited to $\{0, 1\}$, the model counting and logical conjunction correspond to the inner and Hadamard products. The inner and Hadamard products can be performed in polytime on the tensor trains and most other operations can be performed by combining them.

Theorem 3 is proven through contradictions. Assuming that the operation is possible in polynomial time, it follows that certain functions can be represented by tensor trains of a polynomial size. However, a contradiction can be observed because a known lower bound exists on the rank of the tensor train. Owing to space limitations, the proofs of Theorems 2 and 3 are provided in the full version.

Proof of Theorem 1

Proof that $\mathrm{TT}_<$ is at least as succinct as $\mathrm{OBDD}_<$: First, we show that $\mathrm{TT}_<$ is at least as succinct as $\mathrm{OBDD}_<$; that is, any Boolean functions that can be represented in a size $|\beta|$ $\mathrm{OBDD}_<$ can also be represented in a size $|\alpha|$ $\mathrm{TT}_<$ satisfying $|\alpha| \leq p(|\beta|)$, where p is a polynomial. We can prove this by showing a transformation from an OBDD into a tensor train representing the same Boolean function, which does not change the variable order $<$. The transformation comprises two steps: (i) transforming the OBDD into a level-wise smooth OBDD (LSBDD) and (ii) transforming the LSBDD into an equivalent tensor train. Because both transformations will cause at most polynomial increases in representations, we can show that the tensor train is as succinct as the OBDD.

We convert an OBDD into the corresponding LSBDD, which we define as follows:

Definition 8. Let $V \subseteq \{x_1, \ldots, x_n\}$ be a subset of variables and we assume a total order $<$ over variables $x_1, \ldots, x_n$ exists. An LSBDD with respect to V is an OBDD where every non-terminal node v satisfies (i) $\mathsf{var}(v) \in V$ and (ii) no $x \in V$ exists such that $\mathsf{var}(v) < x < \mathsf{var}(c)$ for every child c unless $\mathsf{var}(v)$ is the largest variable among V.

Note that LSBDD differs from a complete OBDD (Bollig and Pröger 2012), in which every path from the root to a sink node has length n. They differ in that the LSBDD allows variable skipping whereas the complete OBDD does not. LSBDD is introduced because converting OBDD to complete OBDD increases the representation size by a factor of n at worst, which is not a polynomial in the original representation size. Figure 1 shows an example of OBDD and

LSBDD with respect to $V = \{x_1, x_2, x_3\}$, representing the same Boolean function $f = (\neg x_1 \wedge \neg x_2) \vee (\neg x_1 \wedge \neg x_3) \vee (\neg x_2 \wedge \neg x_3)$.

We demonstrate a procedure for transforming an OBDD into an equivalent LSBDD. Without loss of generality, we assume that the OBDD follows variable order $x_1 < x_2 < \cdots < x_n$. Let $V(B)$ be the set of labels appearing in OBDD B defined as $V(B) = \{\mathsf{var}(v) \mid v \in N \setminus \{\top, \bot\}\}$. We show that we can construct LSBDD respecting $V(B)$ whose size is bounded by a polynomial of the size of OBDD B.

Lemma 2. For any OBDD B, we can construct an LSBDD equivalent to B with respect to $V(B)$ and equivalent to B, whose size is bounded by a polynomial of $|B|$ in polytime.

Proof. We can convert any OBDD B into an LSBDD with respect to $V(B)$ by repeating the following procedure: Let e be an attributed edge; that is, either the LO or HI edge in B whose source is v and whose target is c. If c is the terminal and $x \in V(B)$ such that $\mathsf{var}(v) < x$ exists, we remove e and add new node u with $\mathsf{var}(u) = \max V(B)$ and $\mathsf{lo}(u) = \mathsf{hi}(u) = c$. Then, we add edge e' whose source is v and target is u. Edge e' has the same attribute with e; that is, if e is a LO (resp. HI) edge, then e' is also an LO (resp. HI) edge. If c is a non-terminal node and $x \in V(B)$ satisfying $\mathsf{var}(v) < x < \mathsf{var}(c)$ exists, we remove e and add new node u with $\mathsf{var}(u) = \hat{x}$ and $\mathsf{lo}(u) = \mathsf{hi}(u) = c$, where $\hat{x}$ is the largest $x \in V(B)$ satisfying $\mathsf{var}(v) < x < \mathsf{var}(c)$. Then, we add an edge e' whose source is v and the target is u. The edge e' has the same attribute with e. We repeat this substitution until no further substitution is possible to obtain an LSBDD.

We can show that the OBDD obtained by the above procedure is an LSBDD with respect to $V(B)$ because every non-terminal node satisfies the conditions in Definition 8. In addition, the obtained LSBDD represents the same Boolean function because the Boolean function corresponding to c equals u after substitution. Furthermore, we can show that the size of the LSBDD is not larger than $|B|(|V(B)| - 1)$ because we add at most $|V(B)| - 1$ nodes for every non-terminal node in B. Therefore, the size of an LSBDD is bounded by $p(|B|)$, where p is a polynomial because $|V(B)| \leq |B|$. □

Next, we describe the procedure for transforming an LSBDD into $\mathrm{TT}_<$. Let $L = (N, E)$ be an LSBDD, where N denotes a set of nodes and E denotes a set of attributed edges. Let $V(L)$ be a set of variables appearing in L and $m = |V(L)|$. We first define mapping π by setting $x_{\pi(i)}$, which is the i-th smallest variable in $V(B)$ for $1 \leq i \leq m$. Next, we define $\mathcal{A}_1, \ldots, \mathcal{A}_m$. Let S_i $(1 \leq i \leq m)$ be a sequence of all non-terminal nodes $v \in N$ satisfying $\mathsf{var}(v) = x_{\pi(i)}$. We order and number the nodes in S_i arbitrarily and denote as $v_{i,0}, v_{i,1}, \ldots, v_{i,r_i-1}$, where $r_i = |S_i|$. For $1 \leq i < m$, we set $\mathcal{A}_i$ as an $r_i \times 2 \times r_{i+1}$ tensor representing the nodes in S_i:

$$\mathcal{A}_i(j,0,k) = \begin{cases} 1 & (\mathsf{lo}(v_{i,j}) = v_{i+1,k}) \\ 0 & (\text{otherwise}) \end{cases},$$

$$\mathcal{A}_i(j,1,k) = \begin{cases} 1 & (\mathsf{hi}(v_{i,j}) = v_{i+1,k}) \\ 0 & (\text{otherwise}) \end{cases}.$$

Similarly, we define $\mathcal{A}_m$ as an $r_m \times 2 \times 1$ tensor representing nodes in S_m:

$$\mathcal{A}_m(j,0,0) = \begin{cases} 1 & (\mathsf{lo}(v_{m,j}) = \top) \\ 0 & (\mathsf{lo}(v_{m,j}) = \bot) \end{cases},$$

$$\mathcal{A}_m(j,1,0) = \begin{cases} 1 & (\mathsf{hi}(v_{m,j}) = \top) \\ 0 & (\mathsf{hi}(v_{m,j}) = \bot) \end{cases}.$$

Each tensor $\mathcal{A}_i$ encodes edges in the LSBDD. In other words, if an edge connecting $v_{i,j}$ to $c_{i+1,k}$ exists, the corresponding element of $\mathcal{A}_i$ is set to 1. Tensor $\mathcal{A}_m$ represents edges connecting nodes in S_m to the $\top$ terminal node.

Lemma 3. The pair of tensor sequence $\mathcal{A}_1, \ldots, \mathcal{A}_m$ and mapping π obtained by the above procedure is a tensor train representation of the Boolean function that the input LSBDD corresponds to.

The full proof of Lemma 3 is given in the full version. Lemma 3 shows that $\mathrm{TT}_<$ can be constructed to represent an equivalent Boolean function with $\mathrm{OBDD}_<$. The number of elements in $\mathcal{A}_i$ is $2r_ir_{i+1}$, where r_i denotes the number of non-terminal LSBDD nodes in S_i. Because $|L| = \sum_{i=1}^{m} r_i$, $\sum_i |\mathcal{A}_i|$ is bounded by $|L|^2$, $\mathrm{TT}_<$ is at least as succinct as $\mathrm{OBDD}_<$. Because efficient compilation methods for the OBDD (Huang and Darwiche 2004; Knuth 2011) exist, they can also be used for the compilation of a tensor train by encoding an OBDD to a tensor train.

Proof that $\mathrm{OBDD}_<$ is not as succinct as $\mathrm{TT}_<$: In the following, we demonstrate the existence of a function that can be represented in a polynomial-sized tensor train but not in a polynomial-sized OBDD. We consider the HWB function (Bryant 1991) defined as follows.

Definition 9. Let $\mathbf{x} = \{x_1, x_2, \ldots, x_n\}$ be the input variable. The Boolean function HWB_n is defined as follows:

$$\mathrm{HWB_n}(\mathbf{x}) = \begin{cases} 0 & (\mathbf{x} = \mathbf{0}) \\ x_{\sum x_i} & (\text{otherwise}) \end{cases},$$

where $\mathbf{0}$ is an all-0 vector.

HWB_n is a function that outputs x_k when the number of 1s in the input is k. HWB has long been used to measure the succinctness of Boolean function representations. In particular, the following characteristic of OBDDs is known.

Lemma 4. (Bollig et al. 1999) The OBDD size of HWB_n is $\Omega(2^{0.2n})$.

To prove that an OBDD is not as succinct as a tensor train, proving the following proposition is sufficient.

Proposition 1. HWB_n can be represented by a tensor train of size $\mathcal{O}(n^2)$.

To prove the above proposition, we introduce a tensor train representing a Boolean function whose rank is $2n$. We then show that the representation equals HWB_n.

For simplicity, we assume that the variable order is $x_1 < \cdots < x_n$. An extension is easy for more general ordering.

Let $\mathcal{A}_0$ be a $2n$-dimensional vector such that only the first element is 1 and the others are 0. Similarly, $\mathcal{A}_{n+1}$ is defined as a $2n$-dimensional vector such that only the $(n+1)$-st element is 1 and the others are 0. We also define a sequence of $2n \times 2 \times 2n$ tensors $\mathcal{A}_i$ $(i \in \{1, 2, \ldots, n\})$ as follows:

$$\mathcal{A}_i(\cdot, 0, \cdot) = \begin{pmatrix} I & O \\ O & I \end{pmatrix}, \quad \mathcal{A}_i(\cdot, 1, \cdot) = \begin{pmatrix} I_1 & I'_i \\ O & I_{n-1} \end{pmatrix},$$

where O is an $n \times n$ matrix whose elements are all 0, I is an $n \times n$ identity matrix, I_k is an $n \times n$ matrix obtained by cyclically right-shifting each row vector of I by k, and I'_k is the left-right inversion of I_k, i.e., $I'_{ki,j} = I_{ki,n-1-j}$. Under this definition, $\mathbf{v}^\top I_k$ is a vector obtained by cyclically right-shifting the elements of n-dimensional vector $\mathbf{v}$ by k. In addition, we define $\tilde{\mathcal{A}}_i$ $(1 \leq i \leq n)$ as follows:

$$\tilde{\mathcal{A}}_1 = \mathcal{A}_0\mathcal{A}_1, \quad \tilde{\mathcal{A}}_i = \mathcal{A}_i, \quad \tilde{\mathcal{A}}_n = \mathcal{A}_n\mathcal{A}_{n+1}.$$

Next, we show that $((\tilde{\mathcal{A}}_1, \ldots, \tilde{\mathcal{A}}_n), \pi)$ is a tensor train representing HWB_n, where $\pi : [n] \to [n]$ is an identity function for the assumed variable order. Let $\mathcal{T} = \mathcal{A}_0\mathcal{A}_1 \cdots \mathcal{A}_{n+1}$. The output of the tensor train corresponding to input values $(a_1, \ldots, a_n) \in \{0,1\}^n$ for variables $x_1, \ldots, x_n$ is $\mathcal{A}_0^\top \mathcal{A}_1(\cdot, a_1, \cdot) \cdots \mathcal{A}_n(\cdot, a_n, \cdot)\mathcal{A}_{n+1}$ by following Eq. (2).

Let $\mathcal{V}_i(a_1, \ldots, a_i)$ be a $2n$-dimensional vector defined as $\mathcal{V}_i(a_1, \ldots, a_i) = \mathcal{A}_0^\top \mathcal{A}_1(\cdot, a_1, \cdot) \cdots \mathcal{A}_i(\cdot, a_i, \cdot)$. We show that $\mathcal{V}_i$ retains sufficient information regarding the values assigned to variables $x_1, \ldots, x_i$ for evaluating HWB_n. We design the first n elements of $\mathcal{V}_i$ to encode $\sum_{j=1}^{i} a_j$ and the remaining n elements to store values $a_1, \ldots, a_i$ while maintaining the first element as $a_{\sum a_j}$. Therefore, the $(n+1)$-st element of $\mathcal{V}_m$ is equal to the HWB_n output. The following lemma shows that $\mathcal{V}_i$ stores such information.

Lemma 5. For any $k \in \{0, 1, \ldots, n\}$ and $(a_1, \ldots, a_k) \in \{0,1\}^k$, $\mathcal{V}_k = \mathcal{A}_0\mathcal{A}_1(\cdot, a_1, \cdot) \cdots \mathcal{A}_k(\cdot, a_k, \cdot)$ is a $2n$-dimensional vector that satisfies the following properties: (i) For the first n elements, only one element is 1 and the others are 0, and the j-th element is 1 iff $\sum_{i=1}^{k} a_i = j - 1 \pmod n$. (ii) The remaining n elements are made by cyclically left-shifting $(a_1, \ldots, a_k, 0, \ldots, 0)$ by j. Therefore, the $(n+1)$-th element of $\mathcal{V}_k$ equals $a_{\sum_{i=1}^{k} a_i}$.

Proof. This is shown by mathematical induction and holds for $k = 0$. Assuming this holds for $k = i$, we divide the case according to a_{i+1}. When $a_{i+1} = 0$, $\mathcal{V}_{i+1} = \mathcal{V}_i\mathcal{A}_{i+1}(\cdot, 0, \cdot) = \mathcal{V}_i$. This satisfies (i) and (ii) because the number of 1's is the same as $\mathcal{V}_i$ and $(a_1, \ldots, a_i, 0, \ldots, 0) = (a_1, \ldots, a_{i+1}, 0, \ldots, 0)$. When $a_{i+1} = 1$, the first n elements of $\mathcal{V}_{i+1}$ are obtained by circularly right-shifting the first n elements of $\mathcal{V}_i$ by 1 by $\mathcal{V}_i\mathcal{A}_{i+1}(\cdot, 1, \cdot)$ from the definition of $\mathcal{A}_{i+1}$. The remaining n elements of $\mathcal{V}_{i+1}$ are made by cyclically left-shifting $(a_1, \ldots, a_{i+1}, 0, \ldots, 0)$ such that the first element corresponds to $a_{\sum a_i}$. This vector is obtained from the remaining n elements of $\mathcal{V}_i$ by setting the element corresponding to x_{i+1} to 1 and circularly left-shifting the vector by 1. This is also performed by $\mathcal{V}_i\mathcal{A}_{i+1}(\cdot, 1, \cdot)$. These results demonstrate the validity of the subject through mathematical induction. □

Proof of Proposition 1. By Lemma 5, the $(n+1)$-st element of $\mathcal{V}_n = \mathcal{A}_0\mathcal{A}_1(\cdot, a_1, \cdot) \cdots \mathcal{A}_n(\cdot, a_n, \cdot)$ corresponds to the $(\sum_{i=1}^{n} a_i)$-th value of the input; that is, it corresponds to $\mathrm{HWB}_i(\mathbf{x})$. Multiplying $\mathcal{A}_{n+1}$ from the right corresponds to selecting the $(n+1)$-st element of $\mathcal{V}_n$, the tensor train representing HWB_n. Therefore, $((\tilde{\mathcal{A}}_1, \ldots, \tilde{\mathcal{A}}_n), \pi)$ is a tensor train representing HWB_n with $\mathcal{O}(n^2)$ size. □

Note that the above proofs hold even if the elements of $\mathcal{A}_i$ are limited to $\{-1, 0, 1\}$.

Proof Sketch of Theorem 2

The proof of Theorem 2 is rather long, so we will only explain the proof sketch. For details, please refer to the full version.

We can verify that, within the tensor representation in Definition 4, every operation in Table 1 can be reduced to basic tensor operations, such as the sum, Hadamard product, and inner product defined below.

Definition 10. For two $d_1 \times \cdots \times d_m$ tensors $\mathcal{A}, \mathcal{B}$, their sum $+$, Hadamard product $\circ$, and inner product $\cdot$ are defined as follows:

$$(\mathcal{A}+\mathcal{B})(a_1, \ldots, a_m) = \mathcal{A}(a_1, \ldots, a_m) + \mathcal{B}(a_1, \ldots, a_m),$$
$$(\mathcal{A}\circ\mathcal{B})(a_1, \ldots, a_m) = \mathcal{A}(a_1, \ldots, a_m)\mathcal{B}(a_1, \ldots, a_m),$$
$$\mathcal{A} \cdot \mathcal{B} = \sum_{a_1, \ldots, a_m} \mathcal{A}(a_1, \ldots, a_m)\mathcal{B}(a_1, \ldots, a_m),$$

where $\mathcal{A} + \mathcal{B}$ and $\mathcal{A} \circ \mathcal{B}$ are $d_1 \times \cdots \times d_m$ tensors and $\mathcal{A} \cdot \mathcal{B}$ is a scalar value.

For example, we show that CT, the query of counting the number of models, is reduced to taking inner product with an all-one tensor and $\wedge$BC, the transformation of taking the logical conjunction of two Boolean function, is reduced to taking Hadamard product of two tensors. It is also known that these operations can be performed efficiently even if the tensors are represented as tensor trains of the same mode order (Oseledets 2011).

However, to perform tensor operations, the modes of the tensors must be aligned. Within tensor train representations, this implies that we should align their mappings π. Therefore, we prepare the method used to align the mappings. In addition, we show that clauses and terms can be represented by a constant-rank tensor train. Finally, by aligning the mappings of two tensor train representations, we show that the operations listed in Table 1, except for $\wedge$C, $\vee$C, and FO, can be reduced to the combinations of sum, Hadamard product, and inner product.

Proof of Theorem 3

In this section, we prove that $\wedge$C, $\vee$C, and FO cannot be performed in polynomial time on tensor trains.

$\wedge$C, $\vee$C: Assume that $\wedge$C can be performed in polynomial time on a tensor train. Let $f = (x_1 \vee \neg y_1) \wedge (\neg x_1 \vee y_1) \wedge (x_2 \vee \neg y_2) \wedge (\neg x_2 \vee y_2) \wedge \cdots \wedge (x_n \vee \neg y_n) \wedge (\neg x_n \vee y_n)$ and consider representing f by a tensor train with order $x_1 < \cdots < x_n < y_1 < \cdots < y_n$. Because each clause can be represented by a linear size tensor train, f can also

be represented by a polynomial size (polysize) tensor train based on this assumption.

Let $A(x_1, \ldots, x_n; y_1, \ldots, y_n)$ be the unfolding matrix of the tensor representation of f; that is, $(x_1, \ldots, x_n)$ corresponds to the row, and $(y_1, \ldots, y_n)$ corresponds to the column. The rank of this matrix is the lower bound of the tensor train rank (Oseledets 2011). Because f outputs 1 if and only if $(x_1, \ldots, x_n) = (y_1, \ldots, y_n)$, $A(x_1, \ldots, x_n; y_1, \ldots, y_n)$ is an identity matrix; thus, its rank is 2^n. Therefore, the rank of the tensor train is also $\Omega(2^n)$, contradicting that f can be represented by a polynomial size tensor train.

By considering $\neg f$, we can similarly prove $\vee$C with the power of De Morgan's law.

FO: Assume that FO can be performed in polynomial time on tensor trains. Let $f = (x_1 \vee \neg y_1) \wedge (\neg x_1 \vee y_1) \wedge (x_2 \vee \neg y_2) \wedge (\neg x_2 \vee y_2) \wedge \cdots \wedge (x_n \vee \neg y_n) \wedge (\neg x_n \vee y_n)$. Let C_i be the i-th clause of f and $h = (z_1 \wedge \neg C_1) \vee (\neg z_1 \wedge z_2 \wedge \neg C_2) \vee \ldots (\neg z_1 \wedge \ldots \neg z_{2n-1} \wedge z_{2n} \wedge \neg C_{2n})$. h can be represented by a polysize OBDD with order $z_1 < \cdots < z_{2n} < x_1 < \cdots < x_n < y_1 < \cdots < y_n$, which has nodes $v_1, \ldots, v_{2n}$ whose labels are $\mathsf{var}(v_i) = z_i$ for $1 \leq i \leq 2n$, $\mathsf{lo}(v_i) = v_{i+1}$ for $1 \leq i \leq 2n-1$, $\mathsf{lo}(v_{2n}) = \bot$. The substructure below $\mathsf{hi}(v_i)$ represents C_i for $1 \leq i \leq 2n$. As every C_i can be represented by a polysize OBDD, the overall OBDD representing h is also polysized. According to Theorem 1, h can be represented by a polysize tensor train. In addition, by the definition of forgetting, we have $\exists\{z_1, \ldots, z_{2n}\}.h = \neg C_1 \vee \cdots \vee \neg C_{2n} = \neg f$. By assumption, we obtain a polysize tensor train representing $\neg f = \neg C_1 \vee \cdots \vee \neg C_{2n}$. Because $\neg$C is also tractable on tensor trains, we obtain a polysize tensor train representing f. However, this contradicts the lower bound of the rank of f mentioned in the above proof regarding $\wedge$C. Therefore, we cannot perform FO in polynomial time on tensor trains.

Discussion

Our theoretical results demonstrate that the power of the tensor trains is equivalent to that of OBDDs. However, their limitations persist as representations of Boolean functions. For instance, a bounded conjunction is performed using the Hadamard product of tensor trains and the rank of the output tensor always becomes the product of the ranks of the input tensor trains, whereas the output OBDDs tend to be small in practice. Future work on TT-based representations will include finding a way to perform practically efficient operations.

In this study, we compared the succinctness of TT using OBDD. It is difficult to find succinctness between the TT and the other representations if we apply standard strategies. It is evident that no encoding from the NNF subsets exists that does not support the same set of queries (e.g., DNNF) to TT. SDD (Darwiche 2011) is the only representation supporting the same set of polytime operations and it is expected that TT will not be more succinct than SDD because it has a more flexible structure for variables than OBDD and TT. Conversely, the NNF subsets are not expected to be more succinct than TT because they cannot represent +1,-1 weights.

Raw tensor representation supports parallelism. Similarly, given that a tensor train comprises multiple tensors, we can perform calculations in parallel, as demonstrated in (Yang, Krompass, and Tresp 2017), thereby benefiting from GPU acceleration. One advantage of tensor-based parallel processing is its independence from variable linear ordering, which eliminates the need to consider order.

Conclusion

This study examined the succinctness and richness of the operations of the tensor train as a representation of Boolean functions by comparing them with the OBDD. We proved that the tensor train is more succinct than the OBDD and can efficiently perform every operation that the OBDD can perform in polytime. These results suggest that the tensor decomposition is a powerful representation of Boolean functions. Because encoding from OBDD to TT exists, compilation for OBDD can also be used for TT. However, it is yet to be determined whether a better compilation method exists. The relationship between tensor trains and SDD or other knowledge representations is a topic for future research.

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

Bollig, B.; Löbbing, M.; Sauerhoff, M.; and Wegener, I. 1999. On the complexity of the hidden weighted bit function for various BDD models. *RAIRO - Theoretical Informatics and Applications - Informatique Théorique et Applications*, 33(2): 103–115.

Bollig, B.; and Pröger, T. 2012. An efficient implicit OBDD-based algorithm for maximal matchings. In *Proceedings of the 6th International Conference on Language and Automata Theory and Applications*, 143–154. Springer.

Bova, S. 2016. SDDs are exponentially more succinct than OBDDs. In *Proceedings of the 30th AAAI Conference on Artificial Intelligence*, 929–935.

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

Bryant, R. E. 1991. On the complexity of VLSI implementations and graph representations of Boolean functions with application to integer multiplication. *IEEE Transactions on Computers*, 40(2): 205–213.

Carroll, J. D.; and Chang, J.-J. 1970. Analysis of individual differences in multidimensional scaling via an n-way generalization of "Eckart-Young" decomposition. *Psychometrika*, 35: 283–319.

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

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. 2001a. Decomposable negation normal form. *Journal of The ACM*, 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. 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.

Dudek, J. M.; and Vardi, M. Y. 2020. Parallel weighted model counting with tensor networks. arXiv:2006.15512.

Fannes, M.; Nachtergaele, B.; and Werner, R. F. 1992. Finitely correlated states on quantum spin chains. *Communications in Mathematical Physics*, 144: 443–490.

Fargier, H.; and Marquis, P. 2008. Extending the knowledge compilation map: Krom, Horn, affine and beyond. In *Proceedings of the 23rd National Conference on Artificial Intelligence*, volume 1, 442–447.

Fargier, H.; Marquis, P.; and Niveau, A. 2013. Towards a knowledge compilation map for heterogeneous representation languages. In *In Proceedings of the 23rd International Joint Conference on Artificial Intelligence*, 877–883.

Hackbusch, W.; and Kühn, S. 2009. A new scheme for the tensor representation. *Journal of Fourier Analysis and Applications*, 15: 706–722.

Hong, X.; Zhou, X.; Li, S.; Feng, Y.; and Ying, M. 2022. A tensor network based decision diagram for representation of quantum circuits. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 27(6): 1–30.

Huang, J.; and Darwiche, A. 2004. Using DPLL for efficient OBDD construction. In *Proceedings of the 7th International Conference on Theory and Applications of Satisfiability Testing*, 157–172.

Khrulkov, V.; Novikov, A.; and Oseledets, I. 2018. Expressive power of recurrent neural networks. In *In Proceedings of the 6th International Conference on Learning Representations*.

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

Kolda, T. G.; and Bader, B. W. 2009. Tensor decompositions and applications. *SIAM Review*, 51(3): 455–500.

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)*.

Niemann, P.; Wille, R.; Miller, D. M.; Thornton, M. A.; and Drechsler, R. 2016. QMDDs: efficient quantum function representation and manipulation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 35(1): 86–99.

Oseledets, I. V. 2011. Tensor-train decomposition. *SIAM Journal on Scientific Computing*, 33(5): 2295–2317.

Tucker, L. R. 1966. Some mathematical notes on three-mode factor analysis. *Psychometrika*, 31: 279–311.

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.

Yang, Y.; Krompass, D.; and Tresp, V. 2017. Tensor-train recurrent neural networks for video classification. In *Proceedings of the 34th International Conference on Machine Learning*, 3891–3900.