From Tensor Networks to Tractable Circuits, and Back

Arend-Jan Quist[1] , **Marc Farreras**[1,2] , **Alexis de Colnet**[1] , **John van de Wetering**[2] , **Alfons Laarman**[1]

[1]Leiden University, Leiden, The Netherlands
[2]University of Amsterdam, Amsterdam, The Netherlands
{a.quist,a.e.h.de.colnet,a.w.laarman}@liacs.leidenuniv.nl, j.m.m.vandewetering@uva.nl

Abstract

Tensor networks and circuits are widely used data structures to represent pseudo-Boolean functions. These two formalisms have been studied primarily in separate communities, and this paper aims to establish equivalences between them. We show that some classes of tensor networks that are appealing in practice correspond to classes of circuits with specific properties that have been studied in knowledge compilation as *tractable circuits*. In particular, we prove that matrix product states (tensor trains) coincide with nondeterministic edge-valued decision diagrams and that tree tensor networks exactly correspond to structured-decomposable circuits. These correspondences enable direct transfer of structural and algorithmic results; for example, canonicity and tractability guarantees known for circuits yield analogous guarantees for the associated tensor networks, and vice versa.

1 Introduction

Tensor networks and tractable circuits have emerged as two powerful systems for representing high-dimensional pseudo-Boolean functions. Tensor networks, and in particular tensor trains (TTs or matrix product states) (Schollwöck 2011; Oseledets 2011) and tree tensor networks (TTNs) (Shi, Duan, and Vidal 2006), are mainly used in quantum physics, linear algebra, and increasingly in machine learning (Zuidberg Dos Martires 2025; Loconte et al. 2025; Loconte, Mengel, and Vergari 2025). Tractable circuits, in various normal forms, such as decision diagrams (DDs) (Bryant 1986) and DNNF (Darwiche 2001), found extensive applications in formal verification, probabilistic analysis, and automated reasoning. Despite their conceptual similarities, these formalisms have largely been developed separately by different communities.

In this paper, we establish key correspondences between the two formalisms. We show that the DD-equivalent of a TT is a non-deterministic (nd) extension of the well-known edge-valued DD (EVDD) (Lai, Pedram, and Vrudhula 1994; Miller, Thornton, and Goodman 2006). The TTN, on the other hand, has a circuit equivalent in structured DNNF (Pipatsrisawat and Darwiche 2008), extended with edge weights. Based on these two correspondences, we then study weak and strong (decision) determinism, yielding syntactic or semantic properties that define the corresponding circuits and tensors. Table 1 summarizes our results. These equivalences are constructive and allow bi-directional transformations with polynomial (often linear) overhead.

These two main correspondences (TT $\equiv$ nd-EVDD and TTN $\equiv$ EV-SDNNF) enable cross-fertilization between two disciplines that have hitherto been largely separated. For instance, the optimal variable order in DDs has been extensively studied and is known to be NP-complete. Similarly, on the tensor network side, the size of a TT has been linked to the so-called Schmidt rank, and optimizing the order is also known to be hard. Canonicity is inherent in the definition of (deterministic) DDs, but much harder to obtain for TTs, where one has to resort to singular value decomposition. The tractability of operations has been extensively studied for circuits (Darwiche and Marquis 2002), while different operations were considered for tensor networks (Vinkhuijzen, Coopmans, and Laarman 2026) and little is known about their composability (Vergari et al. 2021). Finally, on the tensor network side, approximation by truncation is common practice, but it is not known yet what this means on the DD side, nor how it relates to approximate algorithms with rigorous theoretical guarantees for the same purpose (Arenas et al. 2021; Meel and de Colnet 2025). Our equivalences establish how these results relate to each other.

In sum, this work contributes to a unified view of symbolic and algebraic representations for pseudo-Boolean functions, including settings with real, negative, or complex weights. The provided equivalences can be used to transfer techniques, lower bounds, and algorithms between two well-developed but mostly separate fields. Additional opportunities for cross-fertilization between the tensor network community and the decision diagram community are discussed in more detail in Section 8.

Tensor network type		**Decision diagram type**
Tensor Train (TT)	$\Leftrightarrow$	Non-Deterministic EVDD
Deterministic TT	$\Leftrightarrow$	EVDD
Tree Tensor Network (TTN)	$\Leftrightarrow$	EV-SDNNF
Decision TTN	$\Leftrightarrow$	Decision EV-SDNNF
Deterministic TTN	$\Leftrightarrow$	Deterministic EV-SDNNF

Table 1: Tensor network and decision diagram equivalences.

2 Preliminaries

Semi-rings are written $\mathbb{K} = (S, +, \times, 0, 1)$, with S the underlying set of elements. A Boolean variable is a variable that can take only two values: 0 or 1 (*false* or *true*). A (truth) assignment to a set of Boolean variables X is a mapping $\alpha\colon X \to \{0,1\}$. The set of assignments to X is denoted by $\{0,1\}^X$. We use the notation $\{0,1\}^n$, $n \in \mathbb{N}$, to refer to the set of assignments to $\{x_1, \ldots, x_n\}$. A pseudo-Boolean function from X to $\mathbb{K}$ is a mapping $f\colon \{0,1\}^X \to \mathbb{K}$.

2.1 Tractable Circuits

We introduce classes of circuits to represent pseudo-Boolean functions. These circuits are sometimes referred to as *tractable circuits* because they allow efficient poly-time procedures for several problems that are otherwise hard (Darwiche and Marquis 2002; Vergari et al. 2021)

Pseudo-Boolean circuits. A $(+,\times)$-circuit over $\mathbb{K} = (S, +, \times, 0, 1)$ is a directed acyclic graph (DAG) without multi-edges whose internal nodes, or *gates*, are each labeled with the symbol $+$ and with the symbol $\times$ and whose sources, or inputs, are each labeled with an indicator variable $\mathbf{1}[x = 0]$ or $\mathbf{1}[x = 1]$ for x a Boolean variable. Nodes with an outgoing edge to a gate g are called the inputs or children of g. Gates have a finite but unbounded number of inputs. The set of variables of a gate g, denoted by $var(g)$ is the set of all x's for which there is a circuit input $\mathbf{1}[x = b]$, $b \in \{0,1\}$, among the descendants of g. Each incoming edge to a $+$ gate is labeled with a value in S, called its weight. We write $g = w_1 \cdot g_1 + \cdots + w_k \cdot g_k$ to denote that g is a $+$ gate with children $g_1, \ldots, g_k$ and that the incoming edge from g_i carries weight w_i. We write $g = g_1 \times \cdots \times g_k$ to denote that g is a $\times$ gate with children $g_1, \ldots, g_k$. Each gate corresponds to a pseudo-Boolean function over $var(g)$ with a value in S. The value computed by g on a (Boolean) assignment α to $var(g)$ is denoted by $g(\alpha)$. For $\times$ gates we have that $g(\alpha) = g_1(\alpha_1) \times \cdots \times g_k(\alpha_k)$ and for $+$ gates we have that $g(\alpha) = w_1 \times g_1(\alpha_1) + \cdots + w_k \times g_k(\alpha_k)$, where α_i is the restriction of α to $var(g_i)$. When $\mathbb{K}$ is the Boolean semi-ring $(\{0,1\}, \vee, \wedge, 0, 1)$, the $(+,\times)$ circuits coincide with the *circuits in negation normal form* (NNF). When $\mathbb{K}$ is the real field $(\mathbb{R}, +, \times, 0, 1)$ they are the *arithmetic circuits*. When $\mathbb{K}$ is the semi-ring of the non-negative reals $(\mathbb{R}_+, +, \times, 0, 1)$ they are the *monotone arithmetic circuits*.

Determinism. A $(+,\times)$ circuit is *deterministic* when, for every $+$ node $w_1 \cdot g_1 + \cdots + w_k \cdot g_k$ we have that $w_i \times g_i \times w_j \times g_j$ is the 0 function for every $i \neq j$. So either w_i or w_j is the 0 weight, or $g_i \times g_j$ is the 0 function.

Decomposability. A $(+,\times)$ circuit is *decomposable* when, for every $\times$ gate $g = g_1 \times \cdots \times g_k$, it holds that $var(g_i) \cap var(g_j) = \emptyset$ for every $1 \leq i < j \leq k$. In a sense, in a decomposable circuit, the $\times$ splits the variables.

Structured-decomposability. In this paper, we consider decomposable circuits that can only split variables in a specific way. The legal splits are encoded in a *vtree* (variable tree). Let X be a set of Boolean variables. A *vtree* over X is a pair (T, λ) with T a rooted binary tree and $\lambda : leaves(T) \to X$ a bijection. For every $v \in V(T)$, we denote by $var(v)$ the set of variables on the leaves of the subtree rooted at v. Formally, $var(v) = \lambda(v)$ when $v \in leaves(T)$, and $var(v) = var(v_\ell) \cup var(v_r)$ when v is an internal node with children v_ℓ and v_r. A $(+,\times)$ circuit C is *structured by the vtree* (T, λ) over $var(C)$ when there is a mapping ϕ from C's gates to $V(T)$ such that

- gates g map to nodes $\phi(g)$ such that $var(g) \subseteq var(\phi(g))$
- each $\times$ gate g has exactly two children g_ℓ and g_r, $\phi(g)$ is an internal node v of T and there exist $v'_\ell \in \{v_\ell\} \cup descendants(v_\ell)$ and $v'_r \in \{v_r\} \cup descendants(v_r)$ such that $\phi(g_\ell) = v'_\ell$ and $\phi(g_r) = v'_r$.
- each $+$ gate $g = w_1 \cdot g_1 + \cdots + w_k \cdot g_k$ is such that $\phi(g_i)$ is either $\phi(g)$ or a descendant of $\phi(g)$.

A circuit structured by a vtree is *structured-decomposable*. Structured-decomposability implies decomposability.

Edge Valued Decision Diagrams (EVDD)

Edge-valued decision diagrams are popular representations for pseudo-Boolean functions to a semi-ring. We review some classes of edge-valued decision diagrams and recall that, in a sense, they are subsumed by classes of circuits.

Read-once EVDD. A read-once edge-valued decision diagram over $\mathbb{K}$ is a rooted DAG G with a single source and a single sink. Each internal node is labeled with a Boolean variable and has either one outgoing edge labeled 0 or 1, or two outgoing edges, one labeled 0 and the other labeled 1. The endpoint of a 0-edge (resp. 1-edge) is called the 0-child (resp. 1-child). The 0-child and the 1-child may be identical. Each edge e is labeled with a value $w(e) \in \mathbb{K}$. G is *read-once* in the sense that no variable x labels more than one node along the same source-to-sink path. A read-once EVDD is used to represent a function $f\colon \{0,1\}^X \to \mathbb{K}$ with X the finite set of variables labeling its internal nodes. For every assignment α to X, we follow the unique path p_α in G starting at the source of G and following the $\alpha(x)$-edge whenever encountering a node labeled with x. Let $E(p_\alpha)$ be the set of edges in p_α. If p_α reaches the sink then $f(\alpha) = \prod_{e \in E(p_\alpha)} w(e)$. Otherwise $f(\alpha) = 0$. The set of variables labeling internal nodes of G is denoted by $var(G)$.

An *ordered* EVDD is a read-once EVDD where there is a total ordering $\prec$ on $var(G)$ respecting the variable order along all paths, i.e., if v is an ancestor of u then $x_v \prec x_u$.

Non-deterministic EVDD. Non-deterministic read-once EVDD (nd-EVDD) is defined like read-once EVDD except that each node can have several 0-edges and several 1-edges. The definitions of read-once and ordered properties translate to non-deterministic EVDD in a natural way. However, we need to explain how to interpret the diagram as a pseudo-function f. For an assignment α to X (with $X \supseteq var(G)$) we now have a set P_α of paths associated with α, namely, all paths starting from the source, ending at the sink, and

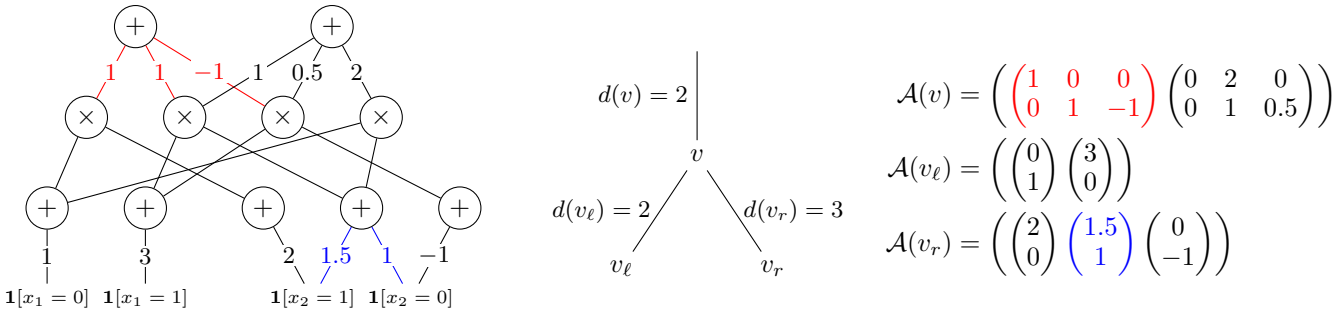


Figure 1: Example of a structured-decomposable circuit (left) and its corresponding tree tensor network (right). The matrices of the tensors correspond to adjacency matrices between the layers of the circuit.

that follow an $\alpha(x)$-edge whenever encountering a node labeled with x, then we have $f(\alpha) = \sum_{p \in P_\alpha} \prod_{e \in E(p)} w(e)$. If $P_\alpha = \emptyset$ and $f(\alpha) = 0$. In the paper, we mainly focus on non-deterministic ordered EVDD, so we will refer to them as EVDD when the context is clear.

2.2 Tensor Networks

In the following, $\mathbb{K} = (S, +, \times, 0, 1)$ is a semi-ring. We write $\times$ as $\cdot$ if the context is clear.

Tensors. A *tensor* A of order r with dimensions $d_1 \times \cdots \times d_r$ over $\mathbb{K}$ is an element in $\mathbb{K}^{d_1 \times \cdots \times d_r}$. The components, or *entries*, of A are denoted by $A[i_1, \ldots, i_r]$ where $i_k \in [d_k]$ for every $k \in [r]$. When a dimension is 2 we sometimes take its entry index in $\{0, 1\}$ rather than in $\{1, 2\}$. Given A and B two tensors over $\mathbb{K}$ of the same dimension $d_1 \times \cdots \times d_r$, we denote by $A \odot B$ their element-wise product, that is, $A \odot B$ is a tensor over $\mathbb{K}$ of dimension $d_1 \times \cdots \times d_r$ such that $(A \odot B)[i_1, \ldots, i_r] = A[i_1, \ldots, i_r] \times B[i_1, \ldots, i_r]$.

Tensor contraction generalizes matrix multiplication. Let A and B be two tensors over $\mathbb{K}$ with dimensions $d_1 \times \cdots \times d_r$ and $n_1 \times \cdots \times n_s$ respectively. Suppose $d_k = n_l$. The contraction of k^{th} dimension of A with the l^{th} dimension of B is the tensor over $\mathbb{K}$ denoted by $A *_{k,l} B$ of dimensions $d_1 \times \cdots \times d_{k-1} \times d_{k+1} \times \cdots \times d_r \times n_1 \times \cdots \times n_{l-1} \times n_{l+1} \times \cdots \times n_s$ whose entries are as follows, for $\vec{e}_1 \in [d_1] \times \cdots \times [d_{k-1}]$, $\vec{e}_2 \in [d_{k+1}] \times \cdots \times [d_r]$, $\vec{e}_3 \in [n_1] \times \cdots \times [n_{l-1}]$, $\vec{e}_4 \in [n_{l+1}] \times \cdots \times [n_s]$,

$$(A *_{k,l} B)[\vec{e}_1, \vec{e}_2, \vec{e}_3, \vec{e}_4] = \sum_{a \in [d_k]} A[\vec{e}_1, a, \vec{e}_2] \cdot B[\vec{e}_3, a, \vec{e}_4]$$

A function $f \colon \{0,1\}^n \to \mathbb{K}$ can be represented by the order-n tensor $A \in \mathbb{K}^{2 \times \cdots \times 2}$ with entries $A[\alpha] = f(\alpha)$ for $\alpha \in \{0,1\}^n$. We may use other tensor representations of f equivalent under tensor isomorphisms. For instance, f may be represented as a vector in $\mathbb{K}^{2^n}$. Furthermore, a collection of d functions $(f^{(1)}, \ldots, f^{(d)}) \in (\{0,1\}^n \to \mathbb{K})^d$ can be jointly represented as a tensor over $A \in \mathbb{K}^{2 \times \cdots \times 2 \times d}$ with entries $A[\alpha, i] = f^{(i)}(\alpha)$ where $\alpha \in \{0,1\}^n$ and $i \in [d]$.

Tensor trains (TTs). A *tensor train* (TT), also called a matrix product state (MPS), is an ordered array $\mathcal{A}$ of n tensors, denoted as $\mathcal{A}(r)$ with $r \in [n]$. Each tensor $\mathcal{A}(r)$ with $r \in [n]$ has components $\mathcal{A}(r)[j_r, j_{r+1}, i]$, with $i \in \{0, 1\}$. The indices j_r and j_{r+1} are called virtual indices, with $j_r \in [\chi_r]$, $j_{r+1} \in [\chi_{r+1}]$, where χ_r is an integer for $r \in [n]$. We constrain $\chi_1 = \chi_{n+1} = 1$, and remove this dimension if the context is clear. So, tensors at the ends have order two, while others have order three. The maximum over χ_r is called the *bond dimension* of the tensor train. The tensor train $\mathcal{A}$ represents the tensor $\hat{\mathcal{A}}$ over $\mathbb{K}^{2^n}$, seen as a function $f : \{0,1\}^n \mapsto \mathbb{K}$ with $f(\alpha)$ defined as

$$\hat{\mathcal{A}}[\alpha] = (\mathcal{A}(1) *_{2,1} \mathcal{A}(2) *_{2,1} \cdots *_{2,1} \mathcal{A}(n-1) *_{2,1} \mathcal{A}(n))[\alpha],$$

where the variable x_r belongs to tensor $\mathcal{A}(r)$.

Tree tensor networks. A *binary tree tensor network (TTN)* is a tuple $(\mathcal{A}, T, d)$ where T is a rooted binary tree directed from the root to the leaves, $\mathcal{A}$ is a mapping from $V(T)$ to tensors over $\mathbb{K}$ and d maps $V(T)$ to $\mathbb{N}$. If v is a leaf of T then $\mathcal{A}(v)$ is a $2 \times d(v)$ tensor over $\mathbb{K}$. If v is an internal node of T with children v_ℓ and v_r then $\mathcal{A}(v)$ is a $d(v_\ell) \times d(v_r) \times d(v)$ tensor over $\mathbb{K}$. We define $\hat{\mathcal{A}}(v)$ has $\hat{\mathcal{A}}(v) = \mathcal{A}(v)$ for leaves of T and $\hat{\mathcal{A}}(v) = \mathcal{A}(v) *_{1,2} \hat{\mathcal{A}}(v_\ell) *_{2,2} \hat{\mathcal{A}}(v_r)$ for internal nodes. One can check $\hat{\mathcal{A}}(v)$ is a $2 \times \cdots \times 2 \times d(v)$ tensor over $\mathbb{K}$ where the number of 2 is the number of leaves below v (including v) in T. It will be convenient to see $\hat{\mathcal{A}}(v)$ as a $2^k \times d(v)$ tensor and thus $\hat{\mathcal{A}}(v)[\cdot, i]$, for $i \in [d(v)]$ as a pseudo-Boolean function over k variables. Therefore, a n-leaves binary TTN over $\mathbb{K}$ computes a sequence of $d(\text{root}(T))$ functions in $\{0,1\}^n \to \mathbb{K}$. We will refer to binary TTNs simply as TTNs, since we only consider this form.

Example 1. *Figure 1 provides, on the right, an example of TTN with T a binary tree with 3 nodes: v, v_ℓ, and v_r. One can check that $\hat{\mathcal{A}}(v)$ is*

$$\begin{pmatrix} \begin{pmatrix} 0 & 2 \\ 6 & 4.5 \end{pmatrix} \begin{pmatrix} 2 & 3 \\ 1.5 & 4.5 \end{pmatrix} \end{pmatrix}$$

So the TTN represents two pseudo-Boolean functions: the first maps $(x_1 = 0, x_2 = 0)$ to 0, $(x_1 = 0, x_2 = 1)$ to 2, etc. The second function maps $(x_1 = 0, x_2 = 0)$ to 2, $(x_1 = 0, x_2 = 1)$ to 3, etc.

3 From Tensor Train to (Non-deterministic) EVDD, and Back

We explain the equivalence between TT and non-deterministic EVDD, and how to efficiently transform between them. Moreover, we show that a TT where all matrices in the tensor train have at most one non-zero entry per row corresponds to a deterministic EVDD. Figure 2 shows an example of how TT and nd-EVDD relate to each other.

3.1 From Tensor Train to nd-EVDD

Assume that we have a tensor train $\mathcal{A}(1), \ldots, \mathcal{A}(n)$ and its corresponding dimensions $\chi_1, \ldots, \chi_{n+1}$. The nd-EVDD that is equivalent to the given tensor train is defined as follows. For every $r \in [n+1]$, we define χ_r nodes $v_s^{(r)}$, with $s \in [\chi_r]$ labeled with variable x_r. The node $v_1^{(1)}$ will be the source of the EVDD, and $v_1^{(r+1)}$ is the sink of the EVDD, which is not labeled by a variable by definition. There are b-edges between nodes $v_s^{(r)}$ and $v_t^{(r+1)}$ with weight $\mathcal{A}(r)[s,t,b]$, where $b \in \{0,1\}, r \in [n], s \in [\chi_r], t \in [\chi_{r+1}]$.

We recursively define $\hat{\mathcal{A}}(n) = \mathcal{A}(n)$ and $\hat{\mathcal{A}}(r) = \mathcal{A}(r) *_{2,1} \hat{\mathcal{A}}(r+1)$ for $r \in [n-1]$. Note that $\hat{\mathcal{A}}(1)$ is the $2 \times \cdots \times 2$ tensor that represents the function f.

We will show by induction on r that the EVDD defined above represents the same function as the tensor train. We show that every node $v_s^{(r)}$ represents the function $f^{(s)}(\alpha) = \hat{\mathcal{A}}(r)[s,\alpha]$ with $\alpha \in \{0,1\}^{n-r}$.

Base case The node $v_s^{(n)}$ has b-edge with weight $\mathcal{A}(n)[s,b]$ to the sink, and hence represents the function $f^{(s)}(x) = \mathcal{A}(n)[s,x]$.

Induction case Let α be an assignment and let $w_s = v_s^{(r)}$ be a node. We define $P_\alpha^{w_s}$ as the set of paths from node w_s to the sink, following assignment α. Then node w_s represents the function $f^{(s)}(\alpha) = \sum_{p \in P_\alpha^{w_s}} \prod_{e \in E(p)} w(e)$. Let $P_\alpha^{w,u}$ be the subset of paths from P_α^w that go via node u. Then we can rewrite with $u_t = v_t^{(r+1)}$

$$\begin{aligned} f^{(s)}(\alpha) &= \sum_{t \in [\chi_{r+1}]} \sum_{p \in P_\alpha^{w_s,u_t}} \prod_{e \in E(p)} w(e) \\ &= \sum_{t \in [\chi_{r+1}]} w(w_s \to^{\alpha(x_r)} u_t) \sum_{p \in P_\alpha^{u_t}} \prod_{e \in E(p)} w(e) \\ &= \sum_{t \in [\chi_{r+1}]} w(w_s \to^{\alpha(x_r)} u_t) \cdot f^{(t)}(\alpha), \end{aligned}$$

where $w_s \to^b u_t$ is the b-edge from w_s to u_t, and $f^{(t)}$ is the function represented by node u_t. By construction, we have $w(w_s \to^{\alpha(x_r)} u_t) = \mathcal{A}(r)[s,t,\alpha(x_r)]$. By induction, we know that $f^{(t)}$ equals $\hat{\mathcal{A}}(r+1)[t,\cdot]$. Hence, we have by definition of tensor contraction and of $\hat{\mathcal{A}}(r)$

$$\begin{aligned} f^{(s)}(\alpha) &= \sum_{t \in [\chi_{r+1}]} \mathcal{A}(r)[s,t,\alpha(x_r)] \cdot \hat{\mathcal{A}}(r+1)[t,\alpha] \\ &= (\mathcal{A}(r) *_{2,1} \hat{\mathcal{A}}(r+1))[s,\alpha] = \hat{\mathcal{A}}(r)[s,\alpha], \end{aligned}$$

which shows the induction.

Hence, the source v_1^1 represents the function $f = \hat{\mathcal{A}}(1)$.

Theorem 1. *There is a procedure that, given a TT $\mathcal{A}(1), \ldots, \mathcal{A}(n)$ representing the function $f\colon \{0,1\}^n \to \mathbb{K}$, constructs in linear time in the size of $\mathcal{A}$ an nd-EVDD that represents the function f.*

3.2 From nd-EVDD to Tensor Train

First, we assume that EVDDs are *complete*, i.e., that on every path all variables from X are read. This can always be enforced in time linear in the size of the EVDD and $|X|$ as follows. On every edge connecting two nodes that skips k intermediate variables, we place k new nodes labeled with the intermediate variables, and pairs of 0-edges and 1-edges carrying weight 1 that connect the newly introduced nodes to each other and to the two nodes of the original edge.

Now, we enumerate the nodes that belong to the same variable. Every node will be labeled as $v_s^{(r)}$, where x_r is the variable of the node, and $s \in [\chi_r]$ is the enumerator. The sink will be labeled as $v_1^{(r+1)}$ We define the tensors $\mathcal{A}(r)[s,t,b]$ as the weight of the b-edge connecting $v_s^{(r)}$ and $v_t^{(r+1)}$, for $r \in [n]$. This can be done in quadratic time, as there might be edges with weight 0.

We will show by induction that the tensor $\hat{\mathcal{A}}(r)[s,\cdot]$ represents the same functions $f^{(s)}\colon \{0,1\}^{n-r} \to \mathbb{K}$ with $s \in [\chi_r]$ as nodes $\{v_s^{(r)}\}_{s \in [\chi_r]}$.

Base case. For every assignment α there is only a single path from $v_s^{(n)}$ to sink. So $f^{(s)}$ takes values $f^{(s)}(\alpha) = \sum_{p \in P_\alpha} \prod_{e \in p} w(e) = w(v_s^{(r)} \to^{\alpha(x)} v_1^{(r+1)})$. Tensor $\hat{\mathcal{A}}(n)[s,\cdot]$ represents the functions $\hat{\mathcal{A}}(n)[s,\alpha] = w(v_s^{(r)} \to^{\alpha(x_n)} v_1^{(r+1)}) = f^{(s)}(\alpha)$.

Induction case. Assume that tensor $\hat{\mathcal{A}}(r+1)$ represents the functions $g^{(t)}$, corresponding to the functions represented by nodes $v_t^{(r+1)}$. For every assignment α, we see that the tensor $\hat{\mathcal{A}}(r)$ can be decomposed as follows.

$$\begin{aligned} \hat{\mathcal{A}}(r)[s,\alpha] &= (\mathcal{A}(r) *_{2,1} \hat{\mathcal{A}}(r+1))[s,\alpha] \\ &= \sum_{t \in [\chi_{r+1}]} \mathcal{A}(r)[s,t,\alpha(x_r)] \cdot \hat{\mathcal{A}}(r+1)[t,\alpha] \\ &= \sum_{t \in [\chi_{r+1}]} w(v_s^{(r)} \to^{\alpha(x_r)} v_t^{(r+1)}) \cdot g^{(t)}(\alpha) \end{aligned}$$

We can decompose $g^{(t)}(\alpha)$ by their definitions as functions representing nodes $v_t^{(r+1)}$. Thus we can rewrite $\hat{\mathcal{A}}(r)[s,\alpha]$

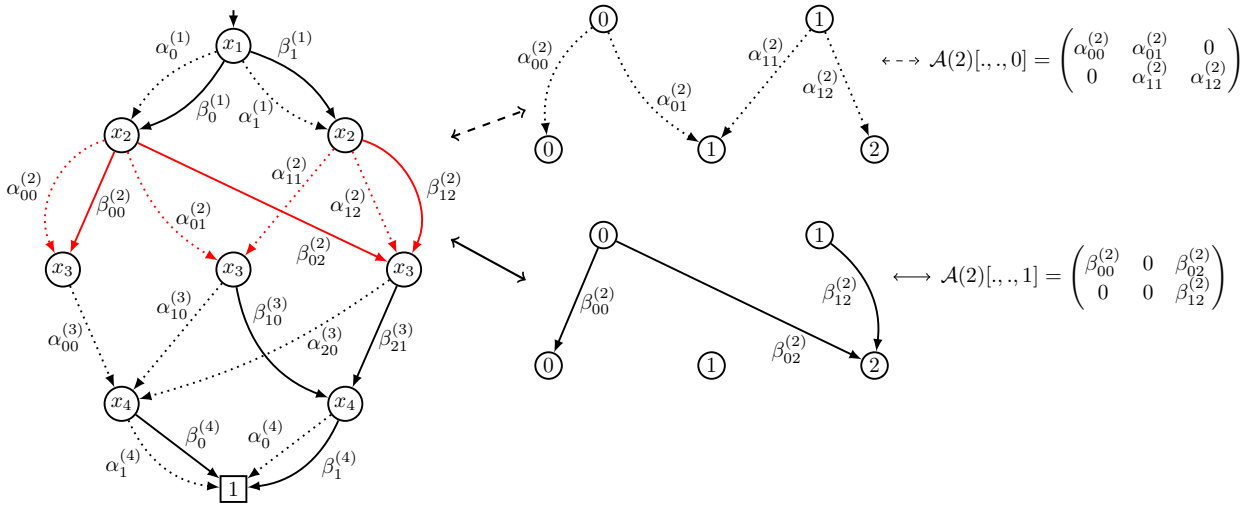


Figure 2: A non-deterministic edge-valued decision diagram (nd-EVDD) and its equivalence to a tensor train (TT). A pseudo-Boolean function $f(x_1, \ldots, x_4)$ is encoded as an nd-EVDD, which is structurally equivalent to a TT representation. The red edges highlight the bipartite subgraph between levels 2 and 3 of the diagram, whose adjacency matrix directly corresponds to one of the tensors in the TT. Dashed and black double arrows indicate how this bipartite graph decomposes into two subgraphs: one for low edges and one for high edges, encoding the matrices $\mathcal{A}_0^{(2)}$ and $\mathcal{A}_1^{(2)}$, respectively, each of dimension 2×3. In this construction, nodes at the upper level represent matrix rows, nodes at the lower level represent columns, and an edge between them corresponds to a non-zero matrix element, with the edge weight specifying its value; zero-valued elements are omitted for clarity.

as

$$\begin{aligned}
&\sum_{t \in [\chi_{r+1}]} w(w_s \to^{\alpha(x_r)} u_t) \sum_{p \in P_\alpha^{u_t}} \prod_{e \in E(p)} w(e) \\
&= \sum_{t \in [\chi_{r+1}]} \sum_{p \in P_\alpha^{w_s, u_t}} \prod_{e \in E(p)} w(e) \\
&= \sum_{p \in P_\alpha^{v_s^{(r)}}} \prod_{e \in E(p)} w(e),
\end{aligned}$$

i.e., exactly the functions represented by nodes $\{v_s^{(r)}\}_{s \in [\chi_r]}$. Hence, $\hat{\mathcal{A}}(1)$ represents the same function as the source node $v_1^{(1)}$, so the constructed tensor train represents the same function as the given EVDD.

Theorem 2. *There is a procedure that, given a nd-EVDD G representing the function $f\colon \{0,1\}^n \to \mathbb{K}$, constructs in quadratic time in the size of G and linear time in n a tensor train $\mathcal{A}(1), \ldots, \mathcal{A}(n)$ that represents the function f.*

3.3 Deterministic EVDD

From the construction of the tensor train in Section 3.2, we directly observe the relation between deterministic EVDDs and tensor trains. For deterministic EVDDs, there is at most one outgoing 0-edge and at most one outgoing 1-edge. Therefore, in the corresponding tensor train, the tensors $\mathcal{A}(r)[s,t,i]$ have at most one non-zero value in the vector $\mathcal{A}(r)[s,\cdot,i]$ for $s \in \chi_r, i \in \{0,1\}$. In other words, all matrices $\mathcal{A}(r)[\cdot,\cdot,i]$ have at most one non-zero entry per row. By the construction of an EVDD from a TT in Section 3.1, we see that tensor trains with this property always correspond to a deterministic EVDD.

(Vinkhuijzen, Coopmans, and Laarman 2026) showed that a TT is always at least as succinct as a deterministic EVDD up to a polynomial factor, but there exist families of states for which TT is exponentially more succinct than deterministic EVDD. This corresponds to our observation that deterministic EVDDs correspond to a strict subset of tensor trains.

4 From Tree Tensor Networks to Structured-Decomposability, and Back

We explain the equivalence of tree tensor networks and structured-decomposable $(+,\times)$-circuits by showing two transformations. Figure 1 shows an example of a tree tensor network and an equivalent structured-decomposable circuit.

4.1 From TTN to Circuits

Let $(\mathcal{A}, T, d)$ be a TTN. The vtree of the corresponding structured-decomposable circuit is $\mathcal{T} = (T, \lambda)$, where T is the underlying tree of the TTN and λ maps its n leaves, read from left to right, to the variables $x_1, \ldots, x_n$. For $v \in V(T)$, we denote by T_v the tree below v in T. We explain the translation inductively.

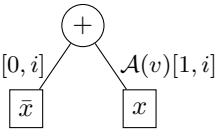
Let v be a node of T. Recall that if $\hat{\mathcal{A}}(v)$ is a $2 \times \cdots \times 2 \times d(v)$ tensor over $\mathbb{K}$, with m 2s corresponding to the m leaves below v, then we interpret each $\hat{\mathcal{A}}(v)[\cdot, k]$ for $k \in [d(v)]$ as

a pseudo-Boolean function in $\{0,1\}^m \to \mathbb{K}$. So $\hat{\mathcal{A}}(v)$ corresponds to a sequence of $d(v)$ pseudo-Boolean functions.

Terminal node. If v is a leaf of T then $\hat{\mathcal{A}}(v) = \mathcal{A}(v)$ is a $2 \times d(v)$ tensor over $\mathbb{K}$. Let x be the variable $\lambda(v)$ corresponding to v. We have

$$\hat{\mathcal{A}}(v)[x,i] = \mathcal{A}(v)[0,i] \cdot \mathbf{1}[x=0] + \mathcal{A}(v)[1,i] \cdot \mathbf{1}[x=1]$$

where, $\mathbf{1}[x=b]$ is the indicator function that returns 1 when x is set to b and 0 otherwise. We then define the gates $g_v^1, \ldots, g_v^{d(v)}$ as follows

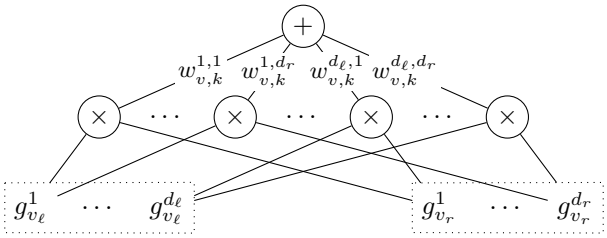
$g_v^i \quad = \quad$ 

It is readily verified that g_v^i computes $\hat{\mathcal{A}}(v)[\cdot,i]$ and that it is structured by the vtree made of a single leaf labeled with x.

Internal node. Let v be an internal node of T with left and right children v_ℓ and v_r. Then we have $\hat{\mathcal{A}}(v) = \mathcal{A}(v) * \hat{\mathcal{A}}(v_\ell) * \hat{\mathcal{A}}(v_r)$. To improve readability we write $d_\ell = d(v_\ell)$ and $d_r = d(v_r)$. By inductive assumption, we have already constructed the circuits below the gates $g_{v_\ell}^i$ and $g_{v_r}^j$ for every $i \in [d_\ell]$ and $j \in [d_r]$. Let $n = n_\ell + n_r$ be the number of leaves below v, with n_ℓ below v_ℓ and n_r below n_r. For $k \in [d(v)]$, the k^{th} function $\hat{\mathcal{A}}(v)[\cdot,k]$ is

$$\hat{\mathcal{A}}(v)[x_1,...,x_n,k] = \sum_{\substack{k_\ell,k_r \\ \in [d_\ell]\times[d_r]}} \begin{array}{r} \hat{\mathcal{A}}(v_r)[x_{n_\ell+1},...,x_n,k_r] \\ \cdot \hat{\mathcal{A}}(v_\ell)[x_1,...,x_{n_\ell},k_\ell] \\ \cdot \mathcal{A}(v)[k_\ell,k_r,k] \end{array} \quad (1)$$

We define the gates $g_v^1, \ldots, g_v^{d(v)}$ as follows. $g_v^k =$



where $w_{v,k}^{i,j} = \mathcal{A}(v)[i,j,k]$. It follows from the above expression of $\hat{\mathcal{A}}(v)[\cdot,k]$ that, if for every $i,j \in [d_\ell] \times [d_r]$, $g_{v_\ell}^i$ computes $\hat{\mathcal{A}}(v_\ell)[\cdot,i]$ and $g_{v_r}^j$ computes $\hat{\mathcal{A}}(v_r)[\cdot,j]$, then g_v^k computes $\hat{\mathcal{A}}(v)[\cdot,k]$. By induction, every circuit in the left box is structured by the vtree $\mathcal{T}_{v_\ell}$ and every circuit in the right box is structured by the vtree $\mathcal{T}_{v_r}$. Thus, the circuit under g_v^k is structured by T_v. One counts that the construction of g_v^k introduces $1 + d_\ell d_r$ new gates. The same $d_\ell d_r$ $\times$-gates are reused by all $g_v^1, \ldots, g_v^{d(v)}$ and thus constructing all these gates requires $d(v) + d_r d_l = d(v) + d(v_\ell)d(v_r)$ new gates and $(2 + d(v))d(v_\ell)d(v_r)$ new edges in total. Therefore, assuming the circuits in the boxes are accessed in $O(1)$ time, constructing $g_v^1, \ldots, g_v^{d(v)}$ takes time linear in the size of $\mathcal{A}(v)$.

Thus, this TTN-to-circuit translation is a linear-time construction. If $(\mathcal{A}, T, d)$ represents a single function, i.e., if $\hat{\mathcal{A}}$ is a $2^n \times 1$ tensor, then the final circuit has a single output gate representing this exact function. Otherwise, if the TTN represents a sequence of s functions, then the final circuit has exactly s output gates, one per function.

Theorem 3. *There is a procedure that, given a TTN $(\mathcal{A}, T, d)$ representing the functions $f^1, \ldots, f^s \colon \{0,1\}^n \to \mathbb{K}$, constructs in linear time in the size of $\mathcal{A}$ an s-output structured-decomposable circuit such that, for every $k \in [s]$, the k^{th} output gate represents f^k.*

For every vtree node v, the circuit has exactly $d(v)$ + gates $g_v^1, \ldots, g_v^{d(v)}$ mapped to v and g_v^k computes exactly the function $\hat{\mathcal{A}}(v)[\cdot,k]$.

4.2 From Circuits to TTN

The backward translation from structured-decomposable circuits to TTN is basically the reverse of the forward translation explained above but the circuit has to go through some (simple) preprocessing. Let C be an $(S, +, \times, 0, 1)$ circuit structured by $\mathcal{T} = (T, \lambda)$ over the variables $x_1, \ldots, x_n$ and let ϕ be the mapping between C's gate and T's node. We modify C (while preserving the function computed) so that it respects the following properties:

1. The output gates of C are all $+$ gates.
2. For each $+$ gate g, the children of g are either circuit inputs or $\times$ gate.
3. There is at most one input gate $\mathbf{1}[x_i = b]$ per $i \in [n]$ and $b \in \{0,1\}$.

The first two properties are easily enforced in linear time by interleaving a dummy $+$ gate with a single incoming edge carrying the unit weight at every edge of C that connects a $\times$ gate to another $\times$ gate or to a circuit input. The third property is also implemented in linear time by merging identical inputs. Next, for every $v \in V(T)$, we fix an arbitrary total order upon the $+$ gates mapped to v by ϕ. We denote the resulting gate sequence by $g_v^1, \ldots, g_v^{d_v}$. We modify C so that it becomes *fully-connected* in the following sense:

4. For every g_v^k, if v is the leaf mapped to x by λ, then g_v^k is of the form $w_{v,k}^0 \cdot \mathbf{1}(x=0) + w_{v,k}^1 \cdot \mathbf{1}(x=1)$ for some weights $w_{v,k}^0, w_{v,k}^1 \in S$; if however v is an internal node with left and right children v_ℓ and v_r, then $g_v^k = \sum_{i,j \in [d_{v_\ell}] \times [d_{v_r}]} w_{v,k}^{i,j} \cdot (g_{v_\ell}^i \cdot g_{v_r}^j)$ with $w_{v,k}^{i,j} \in S$.

The fully-connected property is implemented in quadratic time by adding every missing incoming edge to $+$ gates with 0 weight.

Now we can define the tensor $\mathcal{A}(v)$. If v is a leaf node then $\mathcal{A}(v)$ is the $2 \times d_v$ tensor defined by $\mathcal{A}(v)[0,k] = w_{v,k}^0$ and $\mathcal{A}(v)[1,k] = w_{v,k}^1$. If v is an internal node then $\mathcal{A}(v)$ is the $d_{v_\ell} \times d_{v_r} \times d_v$ tensor defined by $\mathcal{A}(v)[i,j,k] = w_{v,k}^{i,j}$. The final tree tensor network is $(\mathcal{A}, T, d)$, with d mapping each $v \in V(T)$ to d_v. If C respects the four properties 1., 2., 3., and 4., then the TTN is built in linear time.

Transforming the TTN back to a structured-decomposable circuit as in Section 4.1 gives back C, hence the following reverse statement to Theorem 3.

Theorem 4. *There is a procedure that, given an s-outputs structured-decomposable circuit C structured by $((T, \lambda), \phi)$ and representing $f^1, \ldots, f^s \colon \{0,1\}^n \to \mathbb{K}$, processes C in quadratic time into an equivalent circuit C' also structured by (T, λ) and then turns C' in linear time into a TTN $(\mathcal{A}, T, d)$ representing $f^1, \ldots, f^s$.*

In addition, for every $v \in V(T)$, if C' has d_v + gates $g_v^1, \ldots, g_v^{d_v}$ mapped to v by ϕ, then $d(v) = d_v$ and $\hat{\mathcal{A}}(v)[\cdot, k]$ exactly compute g_v^k for every $k \in [d_v]$.

The whole procedure described in Theorem 4 takes quadratic time, while that of turning TTN into circuits takes linear time. This difference is purely due to the fact that, when constructing the TTN, we populate the tensors with many 0s corresponding to missing edges in the circuits. With a more compact representation of the tensors in the TTN, for instance, storing only a list of non-zero entries, the transformation can be done more quickly.

5 Determinism

Determinism is a crucial property of circuits in many applications. Here we investigate what this property becomes in the world of TTN.

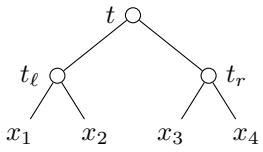
5.1 Decision Determinism

To enforce determinism in practice, one option is to have all $+$ gates of the form

$$g_0 \cdot \mathbf{1}[x = 0] + g_1 \cdot \mathbf{1}[x = 1]$$

for some variable x and circuits g_0 and g_1. Circuits that are structured-decomposable and respect the above property are called *decision structured-decomposable*: at gate g, we *decide* whether to evaluate g_0 or g_1 based on the value of x. We also say that the gate *branches* on x.

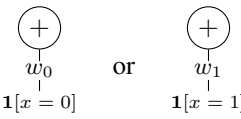
From a theoretical viewpoint, one disadvantage of these circuits is that, while every function can be represented as a decision structured-decomposable circuit for *some* vtree, not all functions can be represented as decision structured-decomposable circuits for *all* vtrees. For example, consider the function $x_1 + x_2 + x_3 + x_4$ in the semi ring of integers and the vtree



In a decision structured-decomposable circuit respecting this vtree, $+$ gates can only be mapped to t_ℓ and t_r, whereas gates mapped to t must be decomposable products. Thus, to get such circuit for $x_1 + x_2 + x_3 + x_4$ we would have to write $x_1 + x_2 + x_3 + x_4$ as a product of functions $f_\ell(x_1, x_2) \cdot f_r(x_3, x_4)$. But this is not possible since we would need $f_\ell(0,0) \cdot f_r(0,0) = 0$, which would imply that $f_\ell(0,0) = 0$ or $f_r(0,0) = 0$, which in turn would imply that $f_\ell(1,1) \cdot f_r(0,0) = 0$ or $f_\ell(0,0) \cdot f_r(1,1) = 0$, while the correct answer is 2.

Decision structured-decomposable circuits are equivalent to TTNs that respect a local property, i.e., a property that is respected at every node v of the tree by the tensor $\mathcal{A}(v)$. The functions $f_v^1, \ldots, f_v^{d(v)}$ can only be of one type, which only depends on the position of v in the tree.

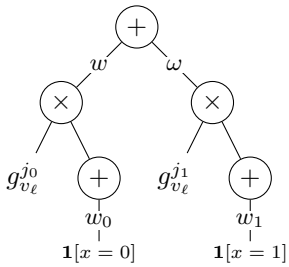
Terminal node. If v is a leaf of T for the variable x then each f_v^i is a function computed by a circuit of the form



So $\mathcal{A}(v)$ is a $2 \times d(v)$ tensor such that, for every $i \in [d(v)]$,

$$\mathcal{A}(v)[0, i] = 0 \quad \text{or} \quad \mathcal{A}(v)[1, i] = 0$$

Almost terminal node. If v is an internal node with one of its children that is a leaf, say v_r, then each f_v^i is a function computed by a circuit of the form

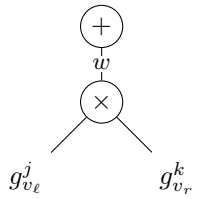


where x is the variable for v_r. So the matrix $\mathcal{A}(v)[\cdot, \cdot, i]$ has only two entries that are not 0, that respectively correspond to the $\times$ gates: there are at most two indexes $j_0, j_1 \in [d(v_\ell)]$ and at most two indexes $k_0, k_1 \in [d(v_r)]$ such that

$$\mathcal{A}(v)[j_0, k_0, i] \neq 0 \quad \text{and} \quad \mathcal{A}(v)[j_1, k_1, i] \neq 0,$$

furthermore, $f_{v_r}^{k_0} = w_0 \cdot \mathbf{1}[x = 0]$ and $f_{v_r}^{k_1} = w_1 \cdot \mathbf{1}[x = 1]$. The case where v_ℓ is a leaf of T is analogous.

Internal node. If v is an internal node of T whose children are not leaves of T then each f_v^i is a function computed by a circuit of the form



So the matrix $\mathcal{A}(v)[\cdot, \cdot, i]$ has only one entry that is not 0, which corresponds to the $\times$ gate: there is only one pair of index $j, k \in [d(v_\ell)] \times [d(v_r)]$ such that $\mathcal{A}(v)[j, k, i] \neq 0$.

Thus, decision structured-decomposable circuits correspond to TTNs with very sparse tensors. We call *decision TTNs* the TTN whose tensors $\mathcal{A}(v)$ are of the type described above. We then have the following equivalence.

Theorem 5. *There is a procedure that, given an s-outputs decision structured-decomposable circuit representing $f^1, \ldots, f^s \colon \{0,1\}^n \to \mathbb{K}$, constructs in polynomial time a decision TTN representing the same s functions.*

Theorem 6. *There is a procedure that, given a decision TTN representing $f^1, \ldots, f^s \colon \{0,1\}^n \to \mathbb{K}$, constructs in polynomial time an s-outputs decision structured-decomposable circuit representing the same s functions.*

5.2 General Determinism

Restricting to decision gates is not the only way to enforce determinism. In this section, we discuss what general determinism corresponds to in TTN.

On structured-decomposable circuits, checking that a $+$ gate $w_1 \cdot g_1 + w_2 \cdot g_2$ is deterministic is doable in quadratic time: compute a structured-decomposable circuit representing $g_1 \times g_2$ and check whether its support is empty. The product operation runs in quadratic time because g_1 and g_2 are structured similarly. Checking that a gate is a decision gate is done locally by looking at its children, but checking general determinism seems to involve operations on all descendants of the $+$ gate. The same is to be expected for the deterministic version of TTN.

Definition 1. A TTN $(\mathcal{A}, T, d)$ is *deterministic* when, for every $v \in V(T)$, every $k \in [d(v)]$ and every two distinct entries of $\mathcal{A}(v)[i_1, j_1, k]$ and $\mathcal{A}(v)[i_2, j_2, k]$ of $\mathcal{A}(v)[\cdot, \cdot, k]$, if both entries are non-zero then $\hat{\mathcal{A}}(v_\ell)[\cdot, i_1] \odot \hat{\mathcal{A}}(v_\ell)[\cdot, i_2] = 0$ or $\hat{\mathcal{A}}(v_r)[\cdot, j_1] \odot \hat{\mathcal{A}}(v_r)[\cdot, j_2] = 0$.

Recall that $\odot$ is the element-wise product of tensors. We claim that the procedures described in Theorems 3 and 4 turn deterministic circuits into deterministic TTN and vice versa.

From Circuits to TTN. In Theorem 4, the processing from C to C' adds single-input $+$ gates and adds incoming edges to $+$ gates carrying weight 0. So if C is deterministic, then C' is too. The procedure then constructs for every $+$ gate $g_v^k = \sum_{i,j \in [d_{v_\ell}] \times [d_{v_r}]} w_{v,k}^{i,j} \cdot g_{v_\ell}^i \cdot g_{v_r}^j$ of C' a matrix $\mathcal{A}(v)[\cdot, \cdot, k]$ such that $\mathcal{A}(v)[i, j, k] = w_{v,k}^{i,j}$.

When C' is deterministic, for each $(i_1, j_1) \neq (i_2, j_2)$ with $w_{v,k}^{i_1,j_1} \neq 0$ and $w_{v,k}^{i_2,j_2} \neq 0$ we have

$$g_{v_\ell}^{i_1} \cdot g_{v_r}^{j_1} \cdot g_{v_\ell}^{i_2} \cdot g_{v_r}^{j_2} = 0$$

Since $var(g_{v_\ell}^{i_1} \cdot g_{v_\ell}^{i_2}) \cap var(g_{v_r}^{j_1} \cdot g_{v_r}^{j_2}) \subseteq var(v_\ell) \cap var(v_r) = \emptyset$ we deduce that

$$g_{v_\ell}^{i_1} \cdot g_{v_\ell}^{i_2} = 0 \quad \text{or} \quad g_{v_r}^{j_1} \cdot g_{v_r}^{j_2} = 0$$

By Theorem 4, for every assignment α to $var(v_\ell)$ we have $\hat{\mathcal{A}}(v_\ell)[\alpha, i_1] = g_{v_\ell}^{i_1}(\alpha)$, $\hat{\mathcal{A}}(v_\ell)[\alpha, i_2] = g_{v_\ell}^{i_2}(\alpha)$ and for every assignment β to $var(v_r)$ we have $\hat{\mathcal{A}}(v_r)[\beta, j_1] = g_{v_r}^{j_1}(\beta)$ and $\hat{\mathcal{A}}(v_r)[\beta, j_2] = g_{v_r}^{j_2}(\beta)$. Thus we indeed have that

$$\hat{\mathcal{A}}(v_\ell)[\cdot, i_1] \odot \hat{\mathcal{A}}(v_\ell)[\cdot, i_2] = 0$$
$$\text{or } \hat{\mathcal{A}}(v_r)[\cdot, j_1] \odot \hat{\mathcal{A}}(v_r)[\cdot, j_2] = 0$$

From TTN to circuit. The procedure of Theorem 3 turns the TTN $(\mathcal{A}, T, d)$ into a structured-decomposable circuit respecting vtree $((T, \lambda), \phi)$ where each gate g_v^k mapped to v by ϕ computes exactly $\hat{\mathcal{A}}(v)[\cdot, k]$. We then have $g_v^k = \hat{\mathcal{A}}(v)$

$$= \sum_{k_\ell, k_r \in [d_\ell] \times [d_r]} \mathcal{A}(v)[k_\ell, k_r, k] \cdot \hat{\mathcal{A}}(v_\ell)[\cdot, k_\ell] \cdot \hat{\mathcal{A}}(v_r)[\cdot, k_r]$$
$$= \sum_{k_\ell, k_r \in [d_\ell] \times [d_r]} \mathcal{A}(v)[k_\ell, k_r, k] \cdot g_{v_\ell}^{k_\ell} \cdot g_{v_r}^{k_r}$$

If the TTN is deterministic, then $\mathcal{A}(v)[k_\ell, k_r, k] \neq 0$ and $\mathcal{A}(v)[h_\ell, h_r, k] \neq 0$ and $(k_\ell, k_r) \neq (h_\ell, h_r)$ forces $\hat{\mathcal{A}}(v_\ell)[\cdot, k_\ell] \cdot \hat{\mathcal{A}}(v_\ell)[\cdot, h_\ell] = 0$ or $\hat{\mathcal{A}}(v_r)[\cdot, k_r] \cdot \hat{\mathcal{A}}(v_r)[\cdot, h_r] = 0$ and therefore $g_{v_\ell}^{k_\ell} \cdot g_{v_\ell}^{h_\ell} = 0$ or $g_{v_r}^{k_r} \cdot g_{v_r}^{h_r} = 0$. It follows that g_v^k is indeed a deterministic $+$ gates.

Lemma 1. *The procedure of Theorem 3 turns deterministic TTN into deterministic structured-decomposable circuits.*

Lemma 2. *The procedure of Theorem 4 turns deterministic structured-decomposable circuits into deterministic TTN.*

6 Application to Representations for Quantum States

Tensor networks and structured circuits are popular ways to describe quantum states. In this section, we explain how these representations are used in quantum computing.

6.1 Quantum States

Quantum states are described by a complex vector, called the quantum state vector. A quantum state consisting of n qubits is represented by a vector of size 2^n. In quantum computing and quantum mechanics, quantum state vectors are usually written in Dirac notation. In Dirac notation, a vector $\vec{v}$ is written as $|\phi\rangle$ (or with another Greek letter around the brackets $|.\rangle$). The basis vector $\vec{e_x} = (0, 0, \ldots, 0, 1, 0, \ldots, 0)^T$, with a 1 at the x'th position and all 0's elsewhere, is written as $|x\rangle$, with $x \in \{0, 1, \ldots, 2^n - 1\}$, or its equivalent Boolean representation $x \in \{0,1\}^n$.

This notation directly relates pseudo-Boolean functions to vectors: every quantum state vector can be written as

$$|\phi\rangle = \sum_{\vec{x} \in \{0,1\}^n} f(\vec{x}) \, |\vec{x}\rangle \, ,$$

where f is a pseudo-Boolean function of the form $f \colon \{0,1\}^n \to \mathbb{C}$. Therefore, every quantum state, which is represented by a quantum state vector, can also be represented by a pseudo-Boolean function. Because every pseudo-Boolean function can be represented by a tensor network or structured circuit, a quantum state can, in fact, also be represented by those representations.

6.2 Tensor Trains in Quantum Computing

Recall that a TT is defined as an ordered array of n tensors of order three, denoted by $\mathcal{A}(r)$, where r indicates the position of each tensor within the array. The function interpretation

of a TT can be directly translated to a quantum state representation, yielding the state

$$|\phi\rangle = \sum_{x_1,\ldots,x_n \in \{0,1\}^n} \mathcal{A}(1)[x_1] \cdots \mathcal{A}(n)[x_n] \, |x_1, \ldots, x_n\rangle \,,$$

where we omitted the $*_{2,1}$ contractions between the tensors, as they are natural multiplications of the corresponding matrices.

6.3 EVDD in Quantum Computing

Recall that an EVDD is a graph with weighted edges. Its function interpretation for an assignment is the sum of products over paths from root to sink corresponding to the assignment. This may appear abstract or difficult to interpret in the context of quantum states, but there seems to be a fairly clean reformulation in Dirac notation.

First, we observe that every node in an EVDD represents a function $f\colon \{0,1\}^k \to \mathbb{K}$ itself. Therefore, as every function is related to a vector, every node in the EVDD represents a vector over $\mathbb{K}$ of length 2^k. We will write the vector represented by node v as $|\phi_v\rangle$. Now, we can recursively write for every internal node v of the EVDD

$$\begin{aligned} |\phi_v\rangle = \quad & |0\rangle \otimes \sum_{u \text{ a 0-child of } v} w(v \to^0 u) \, |\phi_u\rangle \\ & + |1\rangle \otimes \sum_{u \text{ a 1-child of } v} w(v \to^1 u) \, |\phi_u\rangle \,, \end{aligned}$$

and the sink node is defined as $|\phi_{sink}\rangle = 1$. Here, we used the Kronecker product $\otimes$, which acts on basis vectors as $|x\rangle \otimes |y\rangle = |x, y\rangle$ for any $x \in \{0,1\}^k$ and $y \in \{0,1\}^\ell$.

7 Related Work

Tensor networks originate in quantum many-body physics, and have since been applied to combinatorial optimization, numerical partial differential equations, and machine learning (Cirac et al. 2021; Chamon and Mucciolo 2012; Garcia-Saez and Latorre 2011; Khoromskij 2014; Li, Precup, and Rabusseau 2022). These representations compactly encode high-dimensional tensors while supporting efficient contraction algorithms. Beyond tree-like topologies, cyclic variants like Projected Entangled Pair States (PEPS) (Cirac et al. 2021) have been considered.

On the circuit side, other classes of circuits have been studied. For instance, decomposable circuits that do not implement *structured-decomposability* are generally much more compact than the circuits considered in the present work (Pipatsrisawat and Darwiche 2008), but we fail to see their counterpart in the tensor network world. Among structured-decomposable circuits, sentential decision diagrams (SDDs) (Darwiche 2011; Kisa et al. 2014) refine structured decomposability via canonicity relative to a vtree. SDDs have a TTN equivalent, and canonical SDDs should yield some kind of canonical TTN. But canonical SDDs themselves are rather unsatisfactory since, in order to attain canonicity, one has to give up the tractability of several operations (like conditioning or computing conjunction) (van den Broeck and Darwiche 2015). Similar drawbacks are expected for their TTN equivalent.

Tensor-based methods have been investigated for model counting (Kourtis et al. 2019; Dudek, Dueñas-Osorio, and Vardi 2019; Dudek, Phan, and Vardi 2020), an area closely related to proposition decomposable and deterministic circuits (in particular, decision-DNNF and d-DNNF circuits). Tensor-based proof systems for model counting have recently been investigated (Beyersdorff et al. 2026) and compared to existing proof systems, which, for most of them, construct tractable circuits. Compared to our work, tensor-based methods for model counting do not aim at computing a representation of the formula, but only its model count.

Among recent research relating circuits and tensor networks we want to mention (Loconte et al. 2025; Loconte, Javaloy, and Vergari 2025) where the connection between TTN and tractable probabilistic circuits is investigated in one direction (namely, from TTN to circuits). We should also mention (Onaka et al. 2025), which uses tensor trains (TT) to represent Boolean functions but only considers deterministic TT. One small difference is that their TT use entries 1, 0 and -1 while ours would only use 1 and 0 in the Boolean semi-ring.

Connections between decision diagrams and quantum state representations have been explored in quantum circuit simulation (Miller, Thornton, and Goodman 2006). Our equivalences between tensor trains and non-deterministic EVDDs, and between TTNs and structured-decomposable circuits, formalize and generalize this relationship. More broadly, our work contributes to recent efforts to understand tractable representations from an algebraic perspective, including settings with negative or complex weights (Valiant 1979; Loconte, Mengel, and Vergari 2025).

The constructions related to (decision) determinism in structured-decomposable circuits are closely connected to the recently published notion of partition circuits and partition trees (Zuidberg Dos Martires 2024; Zuidberg Dos Martires 2025). This line of work further motivates bridging quantum information theory with knowledge representation and compilation.

8 Discussion

The established equivalences between tensor networks and tractable circuits provide a formal bridge between two communities that have developed largely in isolation: the tensor-network community (primarily in quantum physics, linear algebra, and machine learning) and the knowledge compilation community (primarily in formal verification, probabilistic reasoning, and automated reasoning). The transformations between these two data structures with only linear (or quadratic) overhead enable direct transfer of e.g. lower bounds, heuristics, and analytical results in both directions.

From a theoretical perspective, there is a large arsenal of tools for proving *lower bounds* on circuits (Bova et al. 2016; de Colnet and Mengel 2020) based on rectangle cover techniques, rank- or width-based methods, etc. The underlying concepts of these tools are sometimes studied for tensor networks under different names, for instance the relationship between linear treewidth and Schmidt rank is well known (Legeza and Sólyom 2003). Our contributions suggest that these tools directly translate to the tensor network

domain, as often used in other fields. For example, the DNNF lower bounds from (de Colnet 2020) on permutation functions should also transfer to deterministic-TTN, which is useful since these functions correspond to Clifford operators, which are ubiquitous in, e.g., quantum error correction (Cross and Vandeth 2025).

There have been many heuristics for *variable ordering optimization* for both circuits and MPS (Legeza and Sólyom 2003), with strong evidence of NP-hardness (Bollig and Wegener 1996; Hillar and Lim 2013). For decision diagram circuits (BDD and SDD (Choi and Darwiche 2013)), dynamic variable reordering has been extensively explored but remains largely unknown in the tensor network domain. Our mappings open possibilities for applying the techniques developed for decision diagrams to tensor networks.

The circuit community has spent efforts creating *canonical tractable representations* of functions, in particular SDD and PSDD (van den Broeck and Darwiche 2015). Canonicity is a highly desirable property from a practical perspective. Canonical forms are also important for theoretical and numerical studies using tensor networks (Acuaviva and others 2023), but it is unknown how to compute them cheaply (SVD/Gaussian elimination is required (Schollwöck 2011)). Our contribution changes this picture for deterministic-TTN. However, in practice there exist methods for structured problems to find optimal TTN (Li et al. 2024), which is a weaker property than canonicity, which might be used for structured decomposable circuits.

There exist *approximation procedures* for circuits and tensor networks that could also be transferred from one domain to the other. On the tensor network side, a liberal form of approximation is often done through *truncation* (Schollwöck 2011). Truncation has only recently been studied for decision diagrams (Hillmich et al. 2022), but never for circuits before. On the circuit side, in Boolean settings, polytime methods for *approximating #P-hard problems* have been shown to apply to decomposable circuits (Meel and de Colnet 2026; Arenas et al. 2021). There is hope that these methods, known as FPRAS, are applicable to #P-hard problems on real-valued decomposable circuits and, therefore, to TTN and MPS (tensor contraction, a key operation for tensor networks, is #P-hard (Biamonte, Morton, and Turner 2015)).

The equivalences we found have strong implications for the *succinctness and tractability* of these data structures and, therefore, when combined with the available knowledge compilation maps (Darwiche and Marquis 2002; Fargier et al. 2014; Vinkhuijzen, Coopmans, and Laarman 2026), can help choose an appropriate representation method for concrete cases, not only for circuits but also for tensor networks.

In summary, recent advances in knowledge compilation—efficiently computable canonical forms, lower bounds, parameterized complexity results, heuristics, and approximation schemes—can now benefit the tensor-network community, and vice versa. We expect that further exploration of these connections, including extensions to cyclic tensor networks such as PEPS or to unstructured circuits, will yield both theoretical insights and practical gains across machine learning, quantum computing, and automated reasoning.

AI Declaration

In the preparation of this paper AI was only used to spot typos and improve readability.

Acknowledgments

This publication is part of the project Divide & Quantum (with project number 1389.20.241) of the research programme NWA-ORC which is (partly) financed by the Dutch Research Council (NWO). This work was supported by the Netherlands Organization for Scientific Research (NWO/OCW), as part of QuantumLimits (project number SUMMIT.1.1016).

References

Acuaviva, A., et al. 2023. The minimal canonical form of a tensor network. *arXiv preprint arXiv:2209.14358*.

Arenas, M.; Croquevielle, L. A.; Jayaram, R.; and Riveros, C. 2021. #NFA admits an FPRAS: efficient enumeration, counting, and uniform generation for logspace classes. *J. ACM* 68(6):48:1–48:40.

Beyersdorff, O.; Giesen, J.; Goral, A.; Hoffman, T.; Kasche, K.; and Staudt, C. 2026. Proof systems for tensor-based model counting. In *To appear in AAAI 2026*.

Biamonte, J. D.; Morton, J.; and Turner, J. 2015. Tensor network contractions for #SAT. *Journal of Statistical Physics*.

Bollig, B., and Wegener, I. 1996. Improving the variable ordering of OBDDs is NP-complete. *IEEE Transactions on computers* 45(9):993–1002.

Bova, S.; Capelli, F.; Mengel, S.; and Slivovsky, F. 2016. Knowledge compilation meets communication complexity. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence (IJCAI)*, 1008–1014.

van den Broeck, G., and Darwiche, A. 2015. On the role of canonicity in knowledge compilation. In Bonet, B., and Koenig, S., eds., *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA*, 1641–1648. AAAI Press.

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

Chamon, C., and Mucciolo, E. R. 2012. Virtual parallel computing and a search algorithm using matrix product states. *Phys. Rev. Lett.* 109:030503.

Choi, A., and Darwiche, A. 2013. Dynamic minimization of sentential decision diagrams. In *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence (AAAI)*.

Cirac, J. I.; Pérez-García, D.; Schuch, N.; and Verstraete, F. 2021. Matrix product states and projected entangled pair states: Concepts, symmetries, theorems. *Reviews of Modern Physics* 93(4).

Cross, A., and Vandeth, D. 2025. Small binary stabilizer subsystem codes. *arXiv preprint arXiv:2501.17447*.

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

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

Darwiche, A. 2011. SDD: A new canonical representation of propositional knowledge bases. In *IJCAI Proceedings-International Joint Conference on Artificial Intelligence*.

de Colnet, A., and Mengel, S. 2020. Lower bounds for approximate knowledge compilation. *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence (IJCAI)*.

de Colnet, A. 2020. A lower bound on DNNF encodings of pseudo-boolean constraints. In Pulina, L., and Seidl, M., eds., *Theory and Applications of Satisfiability Testing - SAT 2020 - 23rd International Conference, Alghero, Italy, July 3-10, 2020, Proceedings*, Lecture Notes in Computer Science, 312–321. Springer.

Dudek, J. M.; Dueñas-Osorio, L.; and Vardi, M. Y. 2019. Efficient contraction of large tensor networks for weighted model counting through graph decompositions. *CoRR* abs/1908.04381.

Dudek, J. M.; Phan, V. H. N.; and Vardi, M. Y. 2020. DPMC: weighted model counting by dynamic programming on project-join trees. In Simonis, H., ed., *Principles and Practice of Constraint Programming - 26th International Conference, CP 2020, Louvain-la-Neuve, Belgium, September 7-11, 2020, Proceedings*, volume 12333 of *Lecture Notes in Computer Science*, 211–230. Springer.

Fargier, H.; Marquis, P.; Niveau, A.; and Schmidt, N. 2014. A knowledge compilation map for ordered real-valued decision diagrams. In *Proceedings of the AAAI Conference on Artificial Intelligence*.

Garcia-Saez, A., and Latorre, J. I. 2011. An exact tensor network for the 3SAT problem.

Hillar, C. J., and Lim, L.-H. 2013. Most tensor problems are np-hard. *Journal of the ACM (JACM)*.

Hillmich, S.; Zulehner, A.; Kueng, R.; Markov, I. L.; and Wille, R. 2022. Approximating decision diagrams for quantum circuit simulation. *ACM Transactions on Quantum Computing* 3(4):1–21.

Khoromskij, B. N. 2014. Tensor numerical methods for high-dimensional PDEs: Basic theory and initial applications.

Kisa, D.; van den Broeck, G.; Choi, A.; and Darwiche, A. 2014. Probabilistic sentential decision diagrams. In *KR*.

Kourtis, S.; Chamon, C.; Mucciolo, E.; and Ruckenstein, A. E. 2019. Fast counting with tensor networks. *SciPost Physics* 7(5):060.

Lai, Y.-T.; Pedram, M.; and Vrudhula, S. 1994. EVBDD-based algorithms for integer linear programming, spectral transformation, and function decomposition. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 13(8):959–975.

Legeza, Ö., and Sólyom, J. 2003. Optimizing the density-matrix renormalization group method using quantum information entropy. *Physical Review B*.

Li, W.; Ren, J.; Yang, H.; Wang, H.; and Shuai, Z. 2024. Optimal tree tensor network operators for tensor network simulations: Applications to open quantum systems. *The Journal of Chemical Physics*.

Li, T.; Precup, D.; and Rabusseau, G. 2022. Connecting weighted automata, tensor networks and recurrent neural networks through spectral learning.

Loconte, L.; Mari, A.; Gala, G.; Peharz, R.; de Campos, C.; Quaeghebeur, E.; Vessio, G.; and Vergari, A. 2025. What is the relationship between tensor factorizations and circuits (and how can we exploit it)? *Trans. Mach. Learn. Res.* 2025.

Loconte, L.; Javaloy, A.; and Vergari, A. 2025. How to square tensor networks and circuits without squaring them. *CoRR* abs/2512.17090.

Loconte, L.; Mengel, S.; and Vergari, A. 2025. Sum of squares circuits. In *Proceedings of the AAAI Conference on Artificial Intelligence*.

Meel, K. S., and de Colnet, A. 2025. An FPRAS for model counting for non-deterministic read-once branching programs. In *28th International Conference on Database Theory (ICDT 2025)*, 30–1. Schloss Dagstuhl–Leibniz-Zentrum für Informatik.

Meel, K. S., and de Colnet, A. 2026. #CFG and #DNNF admit FPRAS. *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA)*.

Miller, D.; Thornton, M.; and Goodman, D. 2006. A decision diagram package for reversible and quantum circuit simulation. In *2006 IEEE International Conference on Evolutionary Computation*, 2428–2435.

Onaka, R.; Nakamura, K.; Nishino, M.; and Yasuda, N. 2025. Tensor decomposition meets knowledge compilation: A study comparing tensor trains with OBDDs. In Walsh, T.; Shah, J.; and Kolter, Z., eds., *AAAI-25, Sponsored by the Association for the Advancement of Artificial Intelligence, February 25 - March 4, 2025, Philadelphia, PA, USA*, 15109–15117. AAAI Press.

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

Pipatsrisawat, K., and Darwiche, A. 2008. New compilation languages based on structured decomposability. In *AAAI*, volume 8, 517–522.

Schollwöck, U. 2011. The density-matrix renormalization group in the age of matrix product states. *Annals of Physics* 326(1):96–192.

Shi, Y.-Y.; Duan, L.-M.; and Vidal, G. 2006. Classical simulation of quantum many-body systems with a tree tensor network. *Phys. Rev. A* 74:022320.

Valiant, L. G. 1979. Negation can be exponentially powerful. In *Proceedings of the eleventh annual ACM symposium on theory of computing*, 189–196.

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 Ranzato, M.; Beygelzimer, A.; Dauphin, Y. N.; Liang, P.; and Vaughan, J. W., eds., *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, 13189–13201.

Vinkhuijzen, L.; Coopmans, T.; and Laarman, A. 2026. A knowledge compilation map for quantum information. In *Proceedings of the AAAI Conference on Artificial Intelligence*.

Zuidberg Dos Martires, P. 2024. Probabilistic neural circuits. *Proceedings of the AAAI Conference on Artificial Intelligence* 38(15):17280–17289.

Zuidberg Dos Martires, P. 2025. A quantum information theoretic approach to tractable probabilistic models. *arXiv preprint arXiv:2506.01824*.