An Efficient Low-Rank Tensors Representation for Algorithms in Complex Probabilistic Graphical Models

Gaspard Ducamp GASPARD.DUCAMP@LIP6.FR, GASPARD.DUCAMP@IBM.COM

Philippe Bonnard PHILIPPE.BONNARD@FR.IBM.COM

IBM France Lab, 9 rue de Verdun, 94253 Gentilly, France

Anthony Nouy ANTHONY.NOUY@EC-NANTES.FR

Centrale Nantes, LMJL, 1 rue de la Noe, 44321 Nantes, France

Pierre-Henri Wuillemin PIERRE-HENRI.WUILLEMIN@LIP6.FR

LIP6, 4 place Jussieu, 75005 Paris, France

Abstract

Probabilistic Graphical Models form a class of compact representations of high-dimensional probability distributions by decomposing these distributions in a set of multivariate factors (potentials). Every exact algorithm (for probabilistic inference, MAP, etc.) operates on a specific representation of these potentials. However complex probabilistic models often lead to very large potentials which dramatically impact both the space and time complexities of these algorithms and which can make inference in complex models intractable. In this paper we propose a new approach based on low-rank tensor representation to approximate and operate with these potentials. The low-rank tensor representation is used for the approximation of potentials with controlled precision and an important reduction in the number of parameters. Every operator used in such algorithms (multiplication, addition, projection, etc.) can be defined within this representation, leading to an approximation framework where every algorithm for PGMs can be easily implemented. As an instance of this framework, we present a classical message passing algorithm in Bayesian networks using the tensor train format. By reducing significantly the computational complexity and the memory usage, the proposed approach makes probabilistic inference much more scalable. These results are illustrated by experiments on dynamic Bayesian networks and classical Bayesian networks performed using a Python implementation with TensorFlow, T3F and pyAgrum.

Keywords: Probabilistic Graphical Models; Approximate Inference; Tensor Decomposition; Low-rank approximation; Tensor Train Format.

1. Introduction

The growing demand for explainable AI, as opposed to the concept of black box approaches used in some machine learning fields, puts the *probabilistic graphical models* (PGMs) community forward, PGM being, by nature, easier to understand by human experts. However, the growing number of data tends to make inferences in PGM intractable, the space and time efficiency of algorithms being directly linked to the complexity of the studied model. When exact algorithms suffer from space complexity in large-scale models, approximate ones can only offer a trade-off between time complexity and precision, sometimes without a guarantee of convergence toward a stationary distribution. The use of low-rank tensor methods is a possible way to mitigate the *curse of dimensionality* for discrete high-dimensional models.

In previous works, (Savicky and Vomlel, 2007) proposed to manipulate the structure of a Bayesian network (BN) and use tensor rank-one decomposition to approximate some special forms of conditional probability tables (CPTs) and (Vomlel and Tichavský, 2014) use CP decomposition of tensors corresponding to CPTs of threshold functions, exactly l-out-of-k functions, and their noisy counterparts. In a more general case, tensor methods combined with exact algorithms could provide a new approach to deal with complex PGM in a controlled and tractable way. This approach is not limited to Bayesian networks or to inference algorithms. Every representation of a high-dimensional multivariate function as a product of multivariate factors, every algorithm that operates on a commutative semiring of such multivariate factors are limited by the dimension of these very factors and then could benefit from this compact representation with controlled approximation. As an example of the use of low-rank tensor representation for PGM, we propose in this paper to focus on probabilistic inference in Bayesian networks using the tensor train format.

The outline of the paper is as follows. In Section 2, we briefly introduce Bayesian networks, describe a basic tree-based inference algorithm for computing posterior and marginal probabilities and show how it could benefit from low-rank tensor methods in terms of memory consumption and computational complexity. In Section 4, we present an algorithm using the tensor train format to approximate potentials in probabilistic graphical models such as Bayesian networks and Dynamic Bayesian networks. Finally, in Section 5, we present promising results of such an approach.

2. Bayesian Networks : Model and Algorithm

2.1 Bayesian Networks

A Bayesian network provides a compact representation of the joint probability distribution of a set of random variables. These appear in the form of nodes in a directed acyclic graph (DAG), denoted $\mathcal{G}$, where the absence of arcs represents some conditional independencies. When the variables are discrete, each node is associated with a CPT that contains the conditional probabilities of the random variable with respect to its parents. A Bayesian network $\mathcal{B}$ is associated with $(\mathcal{G}, \Theta)$ where Θ denotes the set of all parameters of conditional probability distribution of each variable v given their parents π_v in $\mathcal{G}$: $P(v|\pi_v)$. For any graph G, let $\mathcal{V}(G)$ express the set of its vertices and $\mathcal{E}(G)$ the set of its edges.

2.2 Message-Passing Protocol in Junction Trees

An inference based on a message-passing algorithm uses a secondary structure called a Junction Tree, denoted $\mathcal{T}$, where variables are grouped into cliques (orange nodes in Figure 1) connected by separators (in green) such that when two cliques $\mathbf{C}_i$ and $\mathbf{C}_j$ are connected by a path, $\mathbf{C}_i \cap \mathbf{C}_j$ is a subset of every clique and separator on the path. The set of nodes adjacent to a node i is denoted by $\mathrm{Adj}(i)$, i.e., $\mathrm{Adj}(i) := \{k \in \mathcal{V}(\mathcal{T}) : (i, k) \in \mathcal{E}(\mathcal{T})\}$.

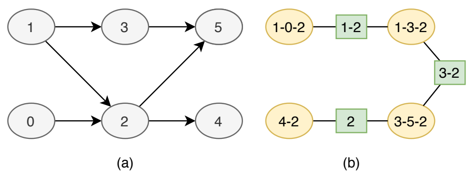

Figure 1: A BN (a) and a junction tree derived from it (b)

The complexity of an inference in a BN is NP-Hard (Cooper, 1990), growing exponentially in the tree-width of the network, the tree-width being related to the size of its largest clique determined by the products of the domains of its variables (Robertson and Seymour, 1986).

Algorithms, such as Shafer-Shenoy (Shenoy and Shafer, 1990), Hugin (Jensen et al., 1990) or Lazy Propagation (Madsen and Jensen, 2013), are based on a message passing protocol between adjacent nodes of the junction tree. For $(i, j) \in \mathcal{E}(\mathcal{T})$ we denote $\psi_{i\to j}$ the potential associated with the separator $\mathbf{S}_{ij}$ which represents the message from i to j over $\mathbf{S}_{ij}$. Since messages will be sent in both directions, we need to distinguish $\psi_{i\to j}$ from $\psi_{j\to i}$. In order to guarantee the correctness of computations, the message-passing protocol needs that the message $\psi_{i\to j}$ should be sent only when clique i has received messages from all of its neighbors except from j. Exchanging a message from i to j in the junction tree will propagate toward j information that has been gathered in i.

One way to organize messages computations is to perform two phases, namely *Collect* and *Distribute* from a predetermined root $r \in \mathcal{V}(\mathcal{T})$. During the *Collect* phase, messages are sent along edges from leaves toward r. During the *Distribute* phase, messages are sent from r towards the leaves. After propagating all the messages, in order to have the joint posterior distribution over the variables in $\mathbf{C}_i$ up to some normalization constant, all we need is to compute the product $\phi_i \times \prod_{(k,i)\in\mathcal{E}(\mathcal{T})} \psi_{k\to i}$, where ϕ_i denotes the potential associated with the clique $\mathbf{C}_i$.

To sum up, an exact inference with junction trees manipulates potentials with two operations, a marginalization and a product (and a division in the case of Hugin), the limiting factor in its feasibility being the size of the potentials. If a more compact representation of the information contained in the potentials can be found and aforementioned operations can be redefined using such an embedding, we could make a more scalable inference, at the cost of a controlled approximation.

3. Low-Rank Tensor Formats

Tensor methods have become a prominent tool for solving high-dimensional problems arising in physics, financial mathematics, statistics, uncertainty quantification, data science, and many other fields involving the approximation of high-dimensional functions or multidimensional arrays. For an introduction to tensor methods and their applications in numerical analysis and machine learning, the reader is referred to the monograph (Hackbusch, 2012) and the surveys (Kolda and Bader, 2009; Nouy, 2017; Bachmayr et al., 2016; Cichocki et al., 2016, 2017; Ji et al., 2019).

Here we consider tensors in the sense of multidimensional arrays. We denote by $\mathbb{R}^{n_1\times\cdots\times n_d}$ the space of tensors of order d and size $n_1 \times \ldots \times n_d$. The entries of a tensor $\mathbf{T} \in \mathbb{R}^{n_1\times\cdots\times n_d}$ are denoted by $\mathbf{T}(i_1, \ldots, i_d)$, $1 \le i_\nu \le n_\nu$, $1 \le \nu \le d$. The index i_ν is related to the ν-th mode of the tensor. A tensor $\mathbf{T}$ can be identified with a vector $\mathrm{vec}(\mathbf{T})$ whose entries are $\mathrm{vec}(\mathbf{T})(\overline{i_1, \ldots, i_d}) = \mathbf{T}(i_1, \ldots, i_d)$, with $\overline{i_1, \ldots, i_d} = i_d + (i_{d-1} - 1)n_d + \ldots + (i_1 - 1)n_2 \ldots n_d$.

The number of entries of a tensor grows exponentially with the order d, and so does the storage consumption and computational complexity of basic multilinear algebra operations between tensors. For handling high-order tensors, it is therefore essential to consider structured tensor formats, such as low-rank tensor formats.

3.1 Tensor Ranks and Related Tensor Formats

For any subset $\alpha \subset \{1,\dots,d\} := D$ and its complementary subset $\alpha^c = \{1,\dots,d\} \setminus \alpha$, a tensor $\mathbf{T}$ can be identified with a matrix $\mathcal{M}_\alpha(\mathbf{T})$ whose entries are (up to a permutation of indices)

$$\mathcal{M}_\alpha(\mathbf{T})(\overline{(i_\nu)_{\nu\in\alpha}}, \overline{(i_\nu)_{\nu\in\alpha^c}}) = \mathbf{T}(i_1,\dots,i_d).$$

The map $\mathcal{M}_\alpha$, called the α-matricisation operator, is a bijection from $\mathbb{R}^{n_1\times\cdots\times n_d}$ to $\mathbb{R}^{N_\alpha \times N_{\alpha^c}}$, where $N_\beta = \prod_{\nu\in\beta} n_\nu$. The rank of the matrix $\mathcal{M}_\alpha(\mathbf{T})$ is called the α-rank of $\mathbf{T}$, and denoted $\mathrm{rank}_\alpha(\mathbf{T})$. By convention, the D-rank and $\emptyset$-rank of a tensor are equal to 1.

Letting $S \subset 2^D$ be a set of subsets of D, we define the S-rank of a tensor $\mathbf{T}$ as the tuple $(\mathrm{rank}_\alpha(\mathbf{T}))_{\alpha\in S} \in \mathbb{N}^{|S|}$. For a given set S and a tuple $r = (r_\alpha)_{\alpha\in S}$, a tensor format $\mathcal{T}_r^S$ is defined as the set of tensors with S-rank less than r, $\mathcal{T}_r^S = \{\mathbf{T} \in \mathbb{R}^{n_1\times\cdots\times n_d} : \mathrm{rank}_\alpha(\mathbf{T}) \le r_\alpha, \alpha \in S\}$.

3.2 Tensor Train Format

The tensor train format has been introduced in (Oseledets, 2009; Oseledets and Tyrtyshnikov, 2009) in the context of numerical analysis. It was already known in quantum physics as Matrix Product State (see, e.g., (Schollwöck, 2011)). This format corresponds to the tensor format $\mathcal{T}_r^S$ with $S = \{\emptyset, \{1\}, \{1,2\}, \dots, \{1,\dots,d-1\}, D\}$. Given a tuple of integers $r = (r_0, r_1, \dots, r_d)$, with $r_0 = r_d = 1$, a tensor $\mathbf{T}$ in the tensor format $\mathcal{T}_r^S$ admits the representation

$$\mathbf{T}(i_1,\dots,i_d) = \sum_{k_1=1}^{r_1} \cdots \sum_{k_{d-1}=1}^{r_{d-1}} \mathbf{T}^{(1)}(1,i_1,k_1)\cdots\mathbf{T}^{(d)}(k_{d-1},i_d,1) \tag{1}$$

where the $\mathbf{T}^{(i)} \in \mathcal{R}^{r_{i-1}\times n_i \times r_i}$ are order-3 tensors called TT cores. The minimal integers $(r_0, r_1, \dots, r_d)$ such that $\mathbf{T}$ has a representation (of the form 1) is called the TT-rank of $\mathbf{T}$.

Storage Complexity The storage complexity of a tensor with TT ranks bounded by R and mode sizes bounded by N is in $O(dNR^2)$. This tensor format allows to circumvent the curse of dimensionality for classes of tensors with TT-rank uniformly bounded or growing polynomially with d.

3.3 Approximation in Tensor Train Format

Every tensor have an exact representation in the tensor train format although without any compression (possibly with high representation ranks). Many algorithms have been proposed for the approximation of tensors in tensor train format. Algorithm 1, introduced in (Oseledets, 2011), allows to obtain an approximation $\mathbf{T}'$ of a given tensor $\mathbf{T}$ in the tensor train format with a prescribed relative precision ϵ, i.e. $\|\mathbf{T} - \mathbf{T}'\|_F \le \epsilon \|\mathbf{T}\|_F$, where $\|\cdot\|_F$ denotes the Frobenius (or canonical) tensor norm. The algorithm relies on standard singular value decompositions of matrices. For more details on the tensor train format and its applications, see, e.g., (Gelß, 2017). The algorithm is here described for the case where the input $\mathbf{T}$ is a multidimensional array but it can be easily adapted to the case where the input is in the TT format.

4. Message Passing Algorithm using Tensor Train Format

Using the TT format seems to be a promising approach for an approximation of potentials in PGMs: (i) an approximation with controlled precision can be obtained using Algorithm 1; (ii) an approximation can be found using an upper limit for TT ranks, allowing to easily control memory usage;

Algorithm 1 *Compress($\mathbf{T}, \epsilon$) - Higher-order truncated SVD for the approximation in TT format*

```
    Input : Tensor $\mathbf{T} \in \mathbb{R}^{n_1 \times \dots \times n_d}$ and tolerance $\epsilon$
    Output Approximation $\mathbf{T}'$ of $\mathbf{T}$ in TT format with cores $\mathbf{U}^{(1)}, \dots, \mathbf{U}^{(d)}$ and ranks $r_0, \dots, r_d$
1: Set $r_0 = 1$ and $r_d = 1$. Set $\mathbf{A} \in \mathbf{R}^{r_0 \times n_1 \times \dots \times n_d}$ such that $\mathbf{A}(1, i_1, \dots, i_d) = \mathbf{T}(i_1, \dots, i_d)$
2: for $\nu = 1, \dots, d-1$ do
3:     $A = \mathcal{M}_{\{1,2\}}(\mathbf{A}) \in \mathbf{R}^{(r_{\nu-1} n_\nu) \times (n_{\nu+1} \cdots n_d)}$
4:     Compute SVD of $A$, i.e. $A = U \Sigma V^T$ with $\Sigma = \mathrm{diag}(\sigma_1, \dots, \sigma_d) \in \mathbb{R}^{s \times s}$, $U \in \mathbb{R}^{(r_{\nu-1} n_\nu) \times s}$ and $V \in \mathbb{R}^{(n_{\nu+1} \cdots n_d) \times s}$
5:     Set $r_\nu \leq s$ to the smallest index such that $\sigma^2_{r_{\nu+1}} + \dots + \sigma^2_s \leq \epsilon^2/(d-1)$
6:     Discard rows and columns of $U, \Sigma$, and $V$ corresponding to singular values $\sigma_{r_{\nu+1}}, \dots, \sigma_s$
7:     Define the $\nu$-th core $\mathbf{U}^{(\nu)} = \mathcal{M}^{-1}_{\{1,2\}}(U) \in \mathbb{R}^{r_{\nu-1} \times n_\nu \times r_\nu}$
8:     Define $\mathbf{A} = \mathcal{M}^{-1}_{\{1\}}(\Sigma V^T) \in \mathbb{R}^{r_\nu \times n_{\nu+1} \times \dots \times n_d}$
9: Define the $d$-th core $\mathbf{U}^{(d)} = \mathcal{M}^{-1}_{\{1\}}(\mathbf{A})$
```

(iii) in order to limit tensors manipulation when computing the product of potentials, we can use an explicit order on the cores. We propose here to use a topological order of the nodes in the DAG.

In order to compute the product between a clique and a separator as well as the marginalization of a potential and, therefore, define a new version of the Shafer-Shenoy algorithm using tensors in TT format we have to introduce some of the classical operations in tensor algebra.

4.1 Operations Between Tensors

We here recall some definitions of elementary operations between tensors (see, e.g., (Lee and Cichocki, 2018; Hackbusch, 2012) for more details).

Kronecker product. The Kronecker product (denoted $\otimes$) of two tensors $\mathbf{A} \in \mathbb{R}^{I_1 \times \dots \times I_N}$ and $\mathbf{B} \in \mathbb{R}^{J_1 \times \dots \times J_N}$ yields a tensor $\mathbf{C} = \mathbf{A} \otimes \mathbf{B}$ of size $I_1 J_1 \times \cdots \times I_N J_N$ with entries

$$\mathbf{C}(\overline{i_1 j_1}, \dots, \overline{i_N j_N}) = \mathbf{A}(i_1, \dots, i_N)\mathbf{B}(j_1, \dots, j_N).$$

Partial Kronecker product. The partial Kronecker product of two tensors along the modes α is denoted $\boxtimes_\alpha$. For two tensors $\mathbf{A} \in \mathbb{R}^{R_1 \times \cdots \times R_M \times I_1 \times \cdots \times I_N}$ and $\mathbf{B} \in \mathbb{R}^{S_1 \times \cdots \times S_M \times I_1 \times \cdots \times I_N}$, the partial Kronecker product $\boxtimes_{\{1,\dots,M\}}$ along modes $\{1, \dots, M\}$ yields a tensor $\mathbf{C} = \mathbf{A} \boxtimes_{\{1,\dots,M\}} \mathbf{B}$ of size $R_1 S_1 \times \cdots \times R_M S_M \times I_1 \times \cdots \times I_N$ with sub-tensors

$$\mathbf{C}(:, \dots, :, i_1, \dots, i_N) = \mathbf{A}(:, \dots, :, i_1, \dots, i_N) \otimes \mathbf{B}(:, \dots, :, i_1, \dots, i_N).$$

Mode-(M,1) contracted product. The mode-(M,1) contracted product (denoted $\times^1$) of tensors $\mathbf{A} \in \mathbb{R}^{I_1 \times \dots \times I_M}$ and $\mathbf{B} \in \mathbb{R}^{J_1 \times \dots \times J_N}$ with $I_M = J_1$ yields a tensor $\mathbf{C} = \mathbf{A} \times^1 \mathbf{B}$ of size $I_1 \times \cdots \times I_{M-1} \times J_2 \times \cdots \times J_N$ with entries

$$\mathbf{C}(i_1, \dots, i_{M-1}, j_2, \dots, j_N) = \sum_{i_M=1}^{I_M} \mathbf{A}(i_1, \dots, i_M)\mathbf{B}(i_M, j_2, \dots, j_N).$$

Hadamard product. The element-wise product or Hadamard product (denoted $\circledast$) of two tensors $\mathbf{A}$ and $\mathbf{B}$ of same size $n_1 \times \ldots \times n_d$ yields a tensor $\mathbf{C} = \mathbf{A} \circledast \mathbf{B}$ with entries

$$\mathbf{C}(i_1, \dots, i_d) = \mathbf{A}(i_1, \dots, i_d)\mathbf{B}(i_1, \dots, i_d).$$

4.2 Hadamard Product in Tensor Train Format

A tensor $\mathbf{T}$ with the representation in TT format (Equation (1)) can be written

$$\mathbf{T} = \mathbf{T}^{(1)} \times^1 \cdots \times^1 \mathbf{T}^{(d)}.$$

If $\mathbf{T}$ and $\mathbf{T}'$ are two tensors with the same size and with representations in TT formats $\mathbf{T} = \mathbf{T}^{(1)} \times^1 \cdots \times^1 \mathbf{T}^{(d)}$ and $\mathbf{T}' = \mathbf{T}'^{(1)} \times^1 \cdots \times^1 \mathbf{T}'^{(d)}$, then their Hadamard product $\mathbf{T} \circledast \mathbf{T}'$ has a representation in TT format

$$\mathbf{T} \circledast \mathbf{T}' = (\mathbf{T}^{(1)} \boxtimes_{\{1,3\}} \mathbf{T}'^{(1)}) \times^1 \cdots \times^1 (\mathbf{T}^{(d)} \boxtimes_{\{1,3\}} \mathbf{T}'^{(d)})$$

where $\boxtimes_{\{1,3\}}$ is the partial Kronecker product along modes 1 and 3.

4.3 Potentials Algebra with Tensor Train Format

Using algorithm 1 (denoted ***Compress***$(\cdot, \epsilon)$), we are now able to compress any multivariate function (Conditional Probability Distribution, potentials, etc.) in a tensor train format controlled by a given relative error. In order to build algorithms for PGM using this compressed format, we now need to define the operations used by such algorithms.

Clique-Separator Product. The product of the potential ϕ of a clique with one of its separators ψ can be obtained by using a Hadamard product between tensors. However, this requires ϕ and ψ to be tensors with the same order and size. Since separator's variables form a subset of the clique's variables, ψ has to be identified with a tensor ψ' with the order and size of ϕ. It is sufficient to consider the case where ϕ is a tensor of order d and size $n_1 \times \ldots \times n_d$ depending on variables $(i_1, \ldots, i_d)$ and ψ is a tensor of order $d-1$ depending on variables $(i_1, ..., i_{\nu-1}, i_{\nu+1}, ..., i_d)$. Then ψ' is the tensor of order d such that $\psi'(i_1, ..., i_d) = \psi(i_1, ..., i_{\nu-1}, i_{\nu+1}, ..., i_d)$, and the Hadamard product $\phi \circledast \psi$ has to be interpreted as $\phi \circledast \psi'$.

Remark 1 *If ψ has a representation in TT format with cores $\mathbf{T}^{(\mu)}$ and representation ranks r_ν, $\mu \in \{1, \ldots, \nu-1, \nu+1, \ldots, d\}$, then ψ' has a representation in TT format with cores $\mathbf{T}'^{(\mu)} = \mathbf{T}^{(\mu)}$ for $\mu \in \{1, \ldots, \nu-1, \nu+1, \ldots, d\}$ and $\mathbf{T}'^{(\nu)} \in \mathbb{R}^{r_{\nu-1} \times n_\nu \times r_\nu}$ such that $\mathbf{T}'^{(\nu)}(k_{\nu-1}, i_\nu, k_\nu) = \delta_{k_{\nu-1}, k_\nu}$, where δ represents the Kronecker delta.*

Marginalization. Variables in a separator $\psi_{i \to j}$ between two cliques $\mathbf{C}_i$ and $\mathbf{C}_j$ being a subset of $\mathbf{C}_i$ and $\mathbf{C}_j$'s variables, we can marginalize the tensor ϕ associated with $\mathbf{C}_i$ in order to form $\psi_{i \to j}$.

Remark 2 *If ϕ has a representation in TT format with cores $\mathbf{T}^{(\nu)}$, $\nu \in \{1, \ldots, d\}$ and $\psi_{i \to j}$ is a separator over $(i_1, ..., i_{\nu-1}, i_{\nu+1}, ..., i_d)$, then $\psi_{i \to j}$ has a representation in TT format with cores $\mathbf{T}'^{(\nu)} = \mathbf{T}^{(\nu)}$ for $\nu \in \{1, \ldots, \nu-1, \nu+2, \ldots, d\}$ and $\mathbf{T}'^{(\nu+1)} = \sum_{i_\nu} \mathbf{T}^{(\nu)}(:, i_\nu, :) \times^1 \mathbf{T}^{(\nu+1)}$ (for a right marginalization).*

4.4 Shafer-Shenoy with Tensor Train Format

We can now redefine the Shafer-Shenoy algorithm, a classical message passing algorithm described in 2.2, using TT and the previously defined operations. The ***Marginalize*** operation will marginalize ϕ over the variables that are not in $\psi_{i \to j}$ (see algorithms 2,3,4).

Algorithm 2 *CollectTT*$(\mathcal{T}, i, j, \epsilon)$

```
    Input : an initialized JT $\mathcal{T}$, $i, j \in \mathcal{V}(\mathcal{T})$ and a tolerance $\epsilon$
    Output Recursively computed $\psi_{i\to j}$
1: $\phi \leftarrow \phi_i$
2: for $k \in \mathrm{Adj}(i)\backslash\{j\}$ do
3:     CollectTT($\mathcal{T}, k, i, \epsilon$)
4:     $\phi \leftarrow$Compress($\phi \circledast \psi_{k\to i}, \epsilon$)
5: $\psi_{i\to j} \leftarrow$Marginalize($\phi, \psi_{i\to j}$)
```

Algorithm 3 *DistributeTT*$(\mathcal{T}, i, j, \epsilon)$

```
    Input : an initialized JT $\mathcal{T}$, $i, j \in \mathcal{V}(\mathcal{T})$ and a tolerance $\epsilon$
    Output Recursively computed $\psi_{i\to j}$
1: $\phi \leftarrow \phi_i$
2: for $k \in \mathrm{Adj}(i)\backslash\{j\}$ do
3:     $\phi \leftarrow$Compress($\phi \circledast \psi_{k\to i}, \epsilon$)
4: $\psi_{i\to j} \leftarrow$Marginalize($\phi, \psi_{i\to j}$)
5: for $l \in \mathrm{Adj}(j)\backslash\{i\}$ do
6:     DistributeTT($\mathcal{T}, j, l$)
```

Algorithm 4 *Shafer-Shenoy TT*$(\mathcal{T}, r, \epsilon)$

```
    Input : a JT $\mathcal{T}$ of $\mathcal{B} = (\mathcal{G}, \Theta)$, a root $r \in \mathcal{V}(\mathcal{T})$ and a tolerance $\epsilon$
    Output a JT $\mathcal{T}$ with messages in both directions on all the separators
1: for $v \in \mathcal{V}(\mathcal{B})$ do
2:     Assign Compress($P(v|\pi_v)$) to a clique C s.t $(v \cup \pi_v) \subseteq$ C
3: CollectTT($\mathcal{T}, r, r, \epsilon$)
4: DistributeTT($\mathcal{T}, r, r, \epsilon$)
```

5. Experimental Results

In order to obtain experimental results, we develop a first implementation using several packages : T3F (`https://t3f.readthedocs.io`) for the manipulation of tensors in TT format, TensorFlow (`www.tensorflow.org`) for tensors related operations and pyAgrum (`https://agrum.gitlab.io`) for the manipulation of Bayesian networks, junction trees and potentials. For the discussion, we compare 2 implementations : Shafer-Shenoy (called SS) and Shafer-Shenoy with tensor train format (called SSTT). The two implementations are identical as much as possible except that the first one manipulates potentials (model and operations implemented in C++) and the second one manipulates tensors in tensor train format (model and operations implemented in TensorFlow or T3F with mixed python and C++). We compare both inference time (denoted T_{SS} for SS, T_{SSTT} for SSTT) as well as the number of parameters (in cliques and separators) at the end of each inferences ($\#_{SS}$ for SS, $\#_{SSTT}$ for SSTT) and the compression factor between them ($\frac{\#_{SS}}{\#_{SSTT}}$, denoted τ). All the tests have been performed on a dual E5-2630v2@2.60GHz with 32Go of RAM.

5.1 Models from the Literature

Table 1 showcases how using our approach on classical BNs improves space complexity of inferences, especially on large BNs. The TT format is not helpful when the JT have small cliques (like in Asia or Alarm). In that case, the tensor train may contain more parameters than its original multidimensional counterpart. However, in case of complex network such as Munin1, it helps by greatly reducing the number of parameters and the inference time. In the case of Link (724 nodes, 1125 arcs), our Shafer-Shenoy implementation couldn't finish the inference, due to a lack of memory (SSTT took 207 seconds). TT operations being computationally expensive, time gains are not linear with the memory savings, as Barley shows.

BN	Alarm	Asia	Barley	Carpo	Child	Diabetes	Hailfinder	Insurrance	Link	Mildew	Munin1	Pigs	Water
Ratio	132.61	98.83	**0.69**	29.51	112.72	27.45	123.15	101.32	**207.10**	3.14	**0.03**	59.218	200.07
τ	0.78	0.60	**116.43**	0.71	0.74	**11.27**	**1.14**	**4.26**	-	**30.29**	**4289.57**	**40.63**	**63.93**

Table 1: Inference time ratio ($\frac{T_{SSTT}}{T_{SS}}$) and compression factor (τ) for classical BN ($\epsilon = 0.001$)

5.2 Dynamic Bayesian Networks

In order to generate BN with growing complexity, we relied on Dynamic Bayesian Network (dBN) (Dagum et al., 1992), dBNs are a generalisation of Markov Chain and can be seen as BNs that relate variables to each other over adjacent time steps (called time slices). Once the relation between two time slices is defined (in a graph called 2TBN), they can be easily deployed (unrolled) for a particular number of steps. When we create a junction tree from an unrolled dBN, the cliques tend to be very large, often making exact inference intractable (Murphy, 2002).

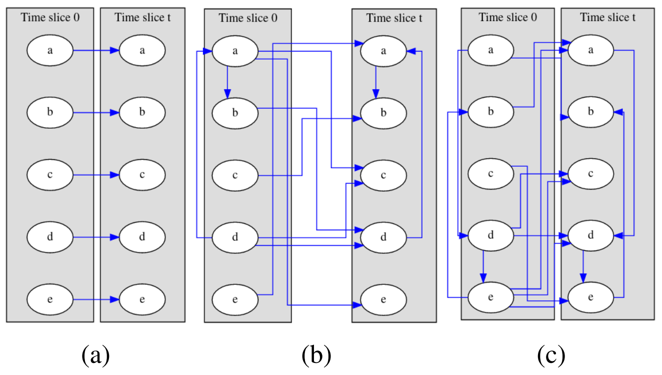

Figure 2: 2TBN of : (a) dBN 1, (b) dBN 2, (c) dBN 3

For our experiments, we have defined 3 dBNs, whose related 2TBN are presented in Figure 2. Each variable's domain size is 10 and CPT were randomly generated. The dBN 1 (Figure 2.a) is expected to be the worst case for our tensor train based algorithm : it is exactly five Markov chains and all the cliques have a size of 2 while dBN 2 and 3 (Figure 2.b and 2.c) are growing in complexity, with more and more arcs intra/inter slices.

Error of Approximation using SSTT. To evaluate the relative errors of our algorithm, we performed inferences ten times on dBN 2 with 50 time slices and randomized CPTs for each iteration. We then compare the exact value of each probability p in each posterior and its approximated version with SSTT $\hat{p}$. Finally we observe the relative error $\frac{|p-\hat{p}|}{p}$. If other evaluation criteria such as the absolute error or a Kullback–Leibler divergence between p and $\hat{p}$ have been considered and even considering its drawbacks, the relative error seemed to us to be the easiest one to interpret. Figure 3 shows that, as expected, a higher tolerance ϵ in the approximation operation $compress(T, \epsilon)$ increases the maximum relative error but lower the inference time. Interestingly, Figure 4 shows that the maximum error is almost constant with the number of time slices. It seems to indicate that there is few error propagation along the JT branches.

Inference Time and Memory Usage. Comparing inference time of Shafer-Shenoy and our algorithm shows that, the more complex and memory intensive an inference is, the more interesting is

	dBN 1	dBN 2	dBN 3	
Nb. Time slices	$\frac{T_{SSTT}}{T_{SS}}$	$\frac{T_{SSTT}}{T_{SS}}$	$\frac{T_{SSTT}}{T_{SS}}$	T_{SSTT}
4	179.89	61.07	7.79	2.95 s
5	69.14	38.83	1.46	4.31 s
7	69.47	1.47	-	**21.07 s**
15	64.24	**0.38**	-	**57.40 s**
50	55.40	**0.33**	-	**167.42 s**
75	45.97	**0.27**	-	**318.71 s**
100	34.59	**0.26**	-	**382.31 s**
150	32.68	**0.21**	-	**522.02 s**

Table 2: Ratio between inference times for SSTT and SS ($\epsilon = 0.05$)

Nb. Time slices	SS	SSTT	τ
4	3.33E+05	4.15E+04	8.0
5	1.04E+06	1.75E+05	5.9
7	1.71E+07	1.16E+05	147.8
15	1.09E+08	6.81E+05	160.5
50	5.05E+08	4.63E+06	109.1
75	8.01E+08	4.57E+06	175.4
100	1.07E+09	9.20E+06	115.9
150	1.66E+09	1.29E+07	128.8

Table 3: Number of parameters and compression factor (τ) in SS and SSTT (dBN 2, $\epsilon = 0.05$)

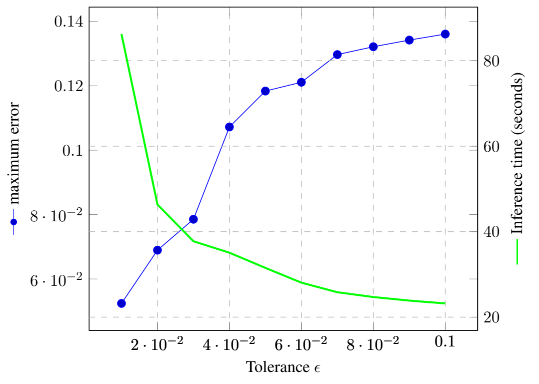

Figure 3: Evolution of the max error ($\frac{|p-\hat{p}|}{p}$) and inference time in dBN 2 with 50 timeslices

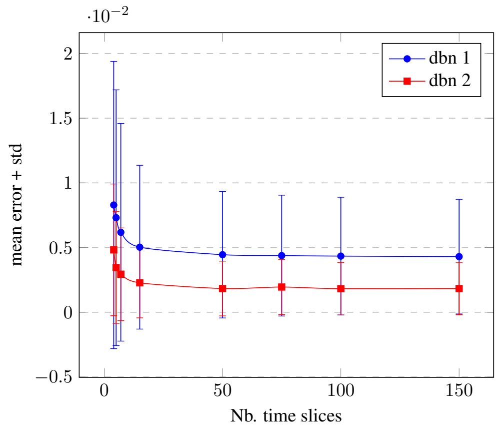

Figure 4: Evolution of the mean error ($\frac{|p-\hat{p}|}{p}$) in posteriors ($\epsilon = 0.05$)

the usage of the tensor train format, as shown in the Table 2. In the case of dBN 1, a decomposition is unnecessary since the size of potentials won't change, only their number will linearly grow with the number of time slices. With dBN 2, on the other hand, the TT version of Shafer-Shenoy shows how much compressing potentials can help to scale inference. Finally, in the case of dBN 3, memory lacks are observed over 7 time slices for the standard algorithm when the TT version can easily scale linearly to 150 time slices. The limiting factor being the treewidth of the junction tree, using a low-rank representation for potentials, such as the TT format, greatly improves memory usage, as shown in Table 3. For a tolerance factor ϵ of 0.05, the number of parameters in our model uses less than a hundredth of the space used by the exact one, with a maximum relative error of 0.12.

5.3 Comparison with Another Approximate Inference

To extend our comparison, we test our method against another approximate inference, Loopy Belief Propagation (LBP) (Murphy et al., 1999), rather than sampling methods that don't offer any guarantee in terms of inferences times. Performances measures shown in Table 4 are similar between SSTT and LBP. However, it is important to note that LBP does not provide any indication of errors for the exact marginal approximation. Even worse, LBP may not always converge for this approximation. On the other hand, SSTT proposes a scalable and controlled approximation.

BN	$\frac{\lvert p-\hat{p}\rvert}{p}$ SSTT ϵ = 1e-3	$\frac{\lvert p-\hat{p}\rvert}{p}$ SSTT ϵ = 1e-5	$\frac{\lvert p-\hat{p}\rvert}{p}$ LBP
Alarm	7,63E-03	2,24E-05	7,50E-01
Asia	4,12E-02	4,02E-09	7,60E-03
Barley	1,20E+00	1,13E-02	2,05E+00
Carpo	7,75E-03	5,52E-14	8,10E-16
Child	3,00E-04	1,47E-08	3,01E-01
Diabetes	2,12E+00	2,00E+00	5,67E+01
Hailfinder	8,10E-04	2,17E-06	4,51E-02
Insurrance	5,41E+00	3,38E-01	1,35E+00
Mildew	4,28E+12	1,35E+07	1,48E+02
Munin1	1,32E+15	1,92E+02	2,08E+00
Pigs	3,06E-14	3,06E-14	1,25E-01
Water	1,86E+02	4,20E+01	6,50E+00

Table 4: Maximum error between SSTT with multiple tolerances and LBP

6. Discussion and Future Works

The representation of potentials in tensor train format proposed in this paper allowed a controlled trade-off between time/space complexity and precision during inference. Even if we focused on only one algorithm (Shafer-Shenoy), it is straightforward to see how much this approach could be generalized in other contexts, such as Markov Random Fields (Koller and Friedman, 2009) or Probabilistic Relational Model (Torti et al., 2010; Medina Oliva et al., 2010), where systems can easily be deployed to represent large-scale worlds. Moreover, this compact representation could greatly improve as well the expression of conditional probability tables with a large set of parents such as (not decomposable) ICI models (Zagorecki et al., 2006) or aggregators (Wuillemin and Torti, 2012). Since the operations involved in our algorithms are heavily parallelizable, they could also benefit from a GPGPU pipeline using CUDA. In some cases, as Figure 1 shows, converting potential into tensors with TT format is unnecessary or even counterproductive compared to using the standard calculation. Hence, an hybrid inference determining for each potential the best algebra might be a solution. Finally, tensor train format is one particular case of more general tree tensor networks, which are tensor formats associated with dimension partition trees (Hackbusch and Kuhn, 2009; Nouy, 2019). The architecture of the network (given by the tree) can have a significant impact on the approximation power of a tree tensor network. Therefore, in the context of PGM, much higher performances could be expected by considering tree tensor networks with tree adaptation methods, as proposed in (Grelier et al., 2018; Grelier et al., 2019).

Acknowledgments

This work was supported by IBM France/ANRT CIFRE grant #2018/0251

References

M. Bachmayr, R. Schneider, and A. Uschmajew. Tensor networks and hierarchical tensors for the solution of high-dimensional partial differential equations. *Foundations of Computational Mathematics*, pages 1–50, 2016.

A. Cichocki, N. Lee, I. Oseledets, A.-H. Phan, Q. Zhao, and D. P. Mandic. Tensor networks for dimensionality reduction and large-scale optimization: Part 1 Low-rank tensor decompositions. *Foundations and Trends® in Machine Learning*, 9(4-5):249–429, 2016.

A. Cichocki, N. Lee, I. Oseledets, A.-H. Phan, Q. Zhao, M. Sugiyama, and D. P. Mandic. Tensor networks for dimensionality reduction and large-scale optimization: Part 2 Applications and future perspectives. *Foundations and Trends® in Machine Learning*, 9(6):249–429, 2017.

G. F. Cooper. The computational complexity of probabilistic inference using Bayesian belief networks. *Artificial Intelligence*, 1990.

P. Dagum, A. Galper, and E. Horvitz. Dynamic Network Models for Forecasting. In *Uncertainty in Artificial Intelligence*. 1992.

P. Gelß. *The Tensor-Train Format and Its Applications*. PhD thesis, 2017.

C. Gonzales, L. Torti, and P.-H. Wuillemin. aGrUM: a Graphical Universal Model framework. In *International Conference on Industrial Engineering, Other Applications of Applied Intelligent Systems*, Proceedings of the 30th International Conference on Industrial Engineering, Other Applications of Applied Intelligent Systems, Arras, France, June 2017.

E. Grelier, A. Nouy, and M. Chevreuil. Learning with tree-based tensor formats. *arXiv e-prints*, page arXiv:1811.04455, Nov. 2018.

E. Grelier, A. Nouy, and R. Lebrun. Learning high-dimensional probability distributions using tree tensor networks. *arXiv preprint arXiv:1912.07913*, 2019.

W. Hackbusch. *Tensor Spaces and Numerical Tensor Calculus*, volume 42. 01 2012.

W. Hackbusch and S. Kuhn. A New Scheme for the Tensor Representation. *Journal of Fourier analysis and applications*, 15(5):706–722, 2009.

F. V. Jensen, K. G. Olesen, and S. K. Andersen. An algebra of Bayesian belief universes for knowledge-based systems. *Networks*, 20(5):637–659, 1990.

Y. Ji, Q. Wang, X. Li, and J. Liu. A survey on tensor techniques and applications in machine learning. *IEEE Access*, PP:1–1, 10 2019.

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

D. Koller and N. Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

N. Lee and A. Cichocki. Fundamental tensor operations for large-scale data analysis using tensor network formats. *Multidimensional Systems and Signal Processing*, 2018.

A. L. Madsen and F. V. Jensen. Lazy propagation in junction trees, 2013.

G. Medina Oliva, P. Weber, E. Levrat, and B. Iung. Use of probabilistic relational model (PRM) for dependability analysis of complex systems. In *IFAC Proceedings Volumes (IFAC-PapersOnline)*, 2010.

K. P. Murphy. *Dynamic Bayesian networks: Representation, inference and learning*. PhD thesis, University of California, Berkeley, 2002.

K. P. Murphy, Y. Weiss, and M. I. Jordan. Loopy belief propagation for approximate inference: An empirical study. In *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*, UAI'99, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

A. Nouy. Low-rank methods for high-dimensional approximation and model order reduction. In P. Benner, A. Cohen, M. Ohlberger, and K. Willcox, editors, *Model Reduction and Approximation: Theory and Algorithms*. SIAM, Philadelphia, PA, 2017.

A. Nouy. Higher-order principal component analysis for the approximation of tensors in tree-based low-rank formats. *Numerische Mathematik*, 141(3):743–789, Mar 2019.

A. Novikov, P. Izmailov, V. Khrulkov, M. Figurnov, and I. Oseledets. Tensor train decomposition on tensorflow (t3f). *arXiv preprint arXiv:1801.01928*, 2018.

I. V. Oseledets. A new tensor decomposition. *Doklady Mathematics*, 80(1):495–496, Aug. 2009.

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

I. V. Oseledets and E. E. Tyrtyshnikov. Breaking the curse of dimensionality, or how to use SVD in many dimensions. *SIAM Journal on Scientific Computing*, 31(5):3744–3759, Jan. 2009.

N. Robertson and P. Seymour. Graph minors. ii. algorithmic aspects of tree-width. *Journal of Algorithms*, 7(3):309 – 322, 1986.

P. Savicky and J. Vomlel. Exploiting tensor rank-one decomposition in probabilistic inference. *Kybernetika*, 43:747, 01 2007.

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

P. P. Shenoy and G. Shafer. Axioms for probability and belief-function propagation. In *Uncertainty in Artificial Intelligence*. Elsevier, Amsterdam, 1990.

L. Torti, P. H. Wuillemin, and C. Gonzales. Reinforcing the object-oriented aspect of probabilistic relational models. In *Proceedings of the 5th European Workshop on Probabilistic Graphical Models, PGM 2010*, 2010. ISBN 9789526033143.

J. Vomlel and P. Tichavský. Probabilistic inference with noisy-threshold models based on a cp tensor decomposition. *International Journal of Approximate Reasoning*, 55(4):1072 – 1092, 2014. Special issue on the sixth European Workshop on Probabilistic Graphical Models.

P.-H. Wuillemin and L. Torti. Structured Probabilistic Inference. *International Journal of Approximate Reasoning*, 53(7):946–968, Oct. 2012.

A. Zagorecki, M. Voortman, and M. J. Druzdzel. Decomposing local probability distributions in Bayesian networks for improved inference and parameter learning. In G. Sutcliffe and R. Goebel, editors, *Proceedings of the Nineteenth International Florida Artificial Intelligence Research Society Conference, Melbourne Beach, Florida, USA, May 11-13, 2006*. AAAI Press, 2006.