Une Représentation Efficace en Tenseurs de Faible Rang pour les Algorithmes dans les Modèles Graphiques Probabilistes Complexes

Gaspard Ducamp, Philippe Bonnard [1], Anthony Nouy [2], Pierre-Henri Wuillemin [3],

[1] IBM France Lab, 9 rue de Verdun, 94253 Gentilly, France
[2] Centrale Nantes, LMJL, 1 rue de la Noe, 44321 Nantes, France
[3] LIP6, 4 place Jussieu, 75005 Paris, France
ducamp.gaspard@gmail.com, philippe.bonnard@fr.ibm.com, anthony.nouy@ec-nantes.fr,
pierre-henri.wuillemin@lip6.fr

Abstract

Les modèles graphiques probabilistes forment une classe de représentations compactes de distributions de probabilité à haute dimension en décomposant ces distributions en un ensemble de facteurs multivariés (potentiels). Chaque algorithme exact (pour l'inférence probabiliste, MAP, etc.) opère sur une représentation spécifique de ces potentiels. Cependant, les modèles probabilistes complexes conduisent souvent à des potentiels de grandes dimensions qui ont un impact important sur les complexités spatiales et temporelles de ces algorithmes et qui peuvent conséquemment rendre l'inférence dans les modèles complexes intraitable. Dans cet article, nous proposons une nouvelle méthode pour approcher et manipuler ces potentiels basée sur une représentation tensorielle de faible rang. Cette représentation est utilisée pour l'approximation des potentiels et permet d'avoir une précision contrôlée et une réduction importante du nombre de paramètres. Chaque opérateur utilisé dans de tels algorithmes (multiplication, addition, projection, etc.) peut être défini dans cette représentation tensorielle, ce qui conduit à un cadre d'approximation dans lequel chaque algorithme pour les PGM peut être facilement implémenté. Pour illustrer cette capacité, nous présentons un algorithme classique de passage de messages dans les réseaux Bayésiens en utilisant le format train de tenseurs. En réduisant considérablement la complexité de calcul et l'utilisation de la mémoire, l'approche proposée permet aux inférences de passer à l'échelle. Ces résultats sont illustrés par des expériences sur des réseaux Bayésiens dynamiques et des réseaux Bayésiens classiques, réalisées à l'aide d'une implémentation Python avec TensorFlow, TAP et pyAgrum.

Introduction

Ce document adapte et complète le travail présenté dans (Ducamp et al. 2020) tout en s'inscrivant dans le cadre général des travaux effectués dans (Ducamp 2021) où se pose notamment la question du passage à l'échelle des inférences (probabilistes) dans des systèmes complexes. En effet, lorsque les algorithmes exacts souffrent de la complexité spatiale dans les modèles à grande échelle, les algorithmes approchés ne peuvent offrir qu'un compromis entre complexité temporelle et précision, parfois sans garantie de convergence vers une distribution stationnaire. L'objectif de ce papier est de montrer que l'utilisation de méthodes tensorielles de faible rang peut être un moyen possible d'atténuer le *fléau de la dimensionnalité* pour les modèles discrets à haute dimension.

Dans des travaux antérieurs, (Savicky and Vomlel 2007) ont proposé de manipuler la structure d'un réseau Bayésien et d'utiliser la décomposition tensorielle dite de "rang un" pour approcher *certaines* formes particulières de CPTs, (Vomlel and Tichavský 2014) utilisent la décomposition CP des tenseurs correspondant aux CPTs des fonctions seuil, des fonctions exactement *l-out-of-k*, et de leurs contreparties bruitées. Nous proposons, dans un cas plus général, des méthodes tensorielles qui combinées avec des algorithmes exacts pourraient fournir une nouvelle approche pour traiter les CPT complexes d'une manière contrôlée et passant à l'échelle. Il est important de noter que cette approche n'est pas limitée aux réseaux Bayésiens ni aux algorithmes d'inférence. Toute représentation d'une fonction multivariée de haute dimension comme un produit de facteurs multivariés, tout algorithme qui opère sur un demi-anneau commutatif de tels facteurs multivariés sont limités par la dimension de ces mêmes facteurs et pourraient donc bénéficier de cette représentation compacte avec une approximation contrôlée, notre approche se veut plus générale que celles proposées dans les travaux cités ci-dessus. Comme exemple d'utilisation d'une représentation tensorielle de faible rang pour les PGM, nous proposons dans cet article de nous concentrer sur l'inférence probabiliste dans les réseaux Bayésiens en utilisant le format train de tenseurs.

Dans ce papier nous présenterons dans un premier temps certaines notions élémentaires de l'algèbre tensorielle, nous discuterons ensuite de méthodes avancées pour simplifier la représentation spatiale de tenseurs de grande dimension afin de, finalement, proposer une première approche pour l'inférence dans des systèmes complexes basée sur ces représentations.

Formats Tensoriels de Faible Rang

Les méthodes tensorielles sont devenues un outil de premier plan pour résoudre des problèmes à haute dimension en physique, en finance, en statistiques, en quantification de l'incertitude, en science des données et dans de nombreux autres domaines impliquant l'approximation de fonctions à haute dimension ou de tableaux multidimensionnels. Pour une introduction aux méthodes tensorielles et à leurs applications en analyse numérique et apprentissage automatique, le lecteur est invité à consulter l'ouvrage (Hackbusch 2012) et les articles (Kolda and Bader 2009; Nouy 2017; Bachmayr, Schneider, and Uschmajew 2016; Cichocki et al. 2016, 2017; Ji et al. 2019).

Dans cet article, nous définissons les tenseurs comme une généralisation des notions de scalaires, vecteurs et matrices à un plus grand nombre de dimensions, c'est-à-dire comme des tableaux multidimensionnels. Alors que les vecteurs ont des entrées v_i avec un indice et que les matrices ont des entrées M_{ij} avec deux indices, les tenseurs portent sur d indices. Ce nombre d'indices est appelé le *ordre* du tenseur.

Nous désignons par $\mathbb{R}^{n_1 \times \cdots \times n_d}$ l'espace des tenseurs d'ordre d et de taille $n_1 \times \ldots \times n_d$. Les composantes d'un tenseur $\mathbf{T} \in \mathbb{R}^{n_1 \times \cdots \times n_d}$ sont désignées par $\mathbf{T}(i_1, \ldots, i_d)$ (parfois $\mathbf{T}_{i_1,\ldots,i_d}$), $1 \leq i_\nu \leq n_\nu$, $1 \leq \nu \leq d$ où l'indice i_ν est lié au ν-ième mode (ou dimension) du tenseur.

Un tenseur $\mathbf{T}$ peut être identifié à un vecteur $\text{vec}(\mathbf{T})$ dont les entrées sont $\text{vec}(\mathbf{T})(\overline{i_1, \ldots, i_d} = \mathbf{T}(i_1, \ldots, i_d)$, avec $\overline{i_1, \ldots, i_d} = i_d + (i_{d-1} - 1)n_d + \ldots + (i_1 - 1)n_2 \ldots n_d$.

Opérations entre Tenseurs

Afin de manipuler les tenseurs, nous devons introduire un certain nombre d'opérations élémentaires. Pour une liste plus détaillée et exhaustive, le lecteur pourra se reporter à (Lee and Cichocki 2018; Hackbusch 2012). Les deux premières opérations sur les tenseurs que nous devons introduire sont le produit de Kronecker et sa contrepartie partielle.

Produit de Kronecker. Le produit de Kronecker (noté $\otimes$) de deux tenseurs $\mathbf{A} \in \mathbb{R}^{I_1 \times \ldots \times I_N}$ et $\mathbf{B} \in \mathbb{R}^{J_1 \times \ldots \times J_N}$ donne un tenseur $\mathbf{C} = \mathbf{A} \otimes \mathbf{B}$ de taille $I_1 J_1 \times \cdots \times I_N J_N$ avec des entrées

$$\mathbf{C}(\overline{i_1 j_1}, \ldots, \overline{i_N j_N}) = \mathbf{A}(i_1, \ldots, i_N)\mathbf{B}(j_1, \ldots, j_N) \quad (1)$$

Nous utiliserons le produit de Kronecker plutôt que le produit externe (plus général) car il permet une conservation des ordres (alors que le produit externe conserve les rangs). Cette caractéristique est importante pour nous puisque la représentation des tenseurs avec laquelle nous allons travailler par la suite n'utilise que des tenseurs d'ordre 3.

Produit de Kronecker partiel. Le produit de Kronecker partiel de deux tenseurs le long des modes $\mathcal{I}$ est noté $\boxtimes_{\mathcal{I}}$. Le produit $\boxtimes_{\{1,\ldots,M\}}$ selon les modes $\{1, \ldots, M\}$ pour deux tenseurs $\mathbf{A} \in \mathbb{R}^{R_1 \cdots \times R_M \times I_1 \times \cdots \times I_N}$ et $\mathbf{B} \in \mathbb{R}^{S_1 \times \cdots \times S_M \times I_1 \cdots \cdots \times I_N}$ donne un tenseur $\mathbf{C} = \mathbf{A} \boxtimes_{\{1,\cdots,M\}} \mathbf{B}$ de taille $R_1 S_1 \cdots \times R_M S_M \times I_1 \times \cdots \times I_N$ avec des sous-tenseurs

$$\begin{aligned}\mathbf{C}(:, \ldots, :, i_1, \ldots, i_N) =& \mathbf{A}(:, \ldots, :, i_1, \ldots, i_N) \\ & \otimes \mathbf{B}(:, \ldots, :, i_1, \ldots, i_N)\end{aligned} \quad (2)$$

La dernière opération dont nous avons besoin est le produit contracté qui correspond à une contraction entre deux indices tensoriels. Dans notre cas, ce sera toujours entre le dernier ($\text{M}^{\text{ième}}$) mode du premier tenseur et le premier mode du deuxième tenseur.

Produit contracté (M,1) Le produit contracté (M,1) (noté $\times^1$) des tenseurs $\mathbf{A} \in \mathbb{R}^{I_1 \times \ldots \times I_M}$ et $\mathbf{B} \in \mathbb{R}^{J_1 \times \ldots \times J_N}$ avec $I_M = J_1$ donne un tenseur $\mathbf{C} = \mathbf{A} \times^1 \mathbf{B}$ de taille $I_1 \times \cdots \times I_{M-1} \times J_2 \times \cdots \times J_N$ avec des entrées

$$\begin{aligned}\mathbf{C}(i_1, \ldots, i_{M-1}, j_2, \ldots, j_N) = \sum_{i_M=1}^{I_M} & \mathbf{A}(i_1, \ldots, i_M) \\ & \mathbf{B}(i_M, j_2, \ldots, j_N)\end{aligned} \quad (3)$$

Maintenant que nous avons défini les opérations nécessaires à nos calculs, nous devons trouver une représentation pour nos tenseurs qui nous permette de réduire leur dimensionnalité. Effectivement, étant donné que le nombre d'entrées d'un tenseur $\mathbf{T}^{(d)}$ croît de manière exponentielle avec l'ordre d — et qu'il en va de même pour la consommation de mémoire et la complexité de calcul des opérations d'algèbre multilinéaire de base entre tenseurs —, il est essentiel de considérer des formats de tenseurs structurés, tels que les formats de tenseurs à faible rang, pour manipuler les tenseurs d'ordre élevé.

Rangs Tensoriels et Formats Tensoriels Associés

Dans cette section, nous allons présenter quelques outils permettant de contourner la principale limitation liée à l'utilisation des tenseurs *pleins*.

Un tenseur élémentaire $\mathbf{T} = \mathbf{T}^{(1)} \otimes \ldots \otimes \mathbf{T}^{(d)}$ est le produit tensoriel de d tenseurs d'ordre 1 (vecteurs) $\mathbf{T}^{(\nu)} \in \mathbb{R}^{n_\nu}$, dont les entrées sont $\mathbf{T}(i_1, \ldots, i_d) = \mathbf{T}^{(1)}(i_1) \ldots \mathbf{T}^{(d)}(i_d)$. Il est à noter que si l'on considère un ensemble de marginales de variables aléatoires toutes indépendantes les unes des autres, le potentiel associé au produit de ces marginales est un tenseur élémentaire. Le rang canonique d'un tenseur $\mathbf{T}$, noté $\text{rank}(\mathbf{T})$, est l'entier minimal r tel que le tenseur puisse être écrit comme une somme de r tenseurs élémentaires.

Chaque tenseur $\mathbf{T}$ peut être représenté comme une combinaison linéaire de r tenseurs d'ordre 1, une telle représentation est appelée *décomposition polyadique canonique* (Hitchcock 1927) (et parfois appelée CANDECOMP (Carroll and Chang 1970) ou PARAFAC

(Harshman 1970)). Ce format a notamment été utilisé dans (Savicky and Vomlel 2007; Vomlel and Tichavský 2014) afin de factoriser certains CPTs lors d'inférences probabilistes. Cependant, le calcul du rang canonique d'un tel tenseur est NP difficile (Hillar and Lim 2013, Theorem 1.13). D'autres représentations, plus complexes mais aussi plus structurées, peuvent être définies.

Pour tout sous-ensemble $\alpha \subset \{1,\dots,d\} := D$ et son sous-ensemble complémentaire $\alpha^c = \{1,\dots,d\} \setminus \alpha$, un tenseur $\mathbf{T}$ peut être identifié grâce à une matrice $\mathcal{M}_\alpha(\mathbf{T})$ dont les entrées sont, à une permutation d'indices près :

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

L'application $\mathcal{M}_\alpha$, appelée opérateur de matricisation α, est une bijection de $\mathbb{R}^{n_1\times\cdots\times n_d}$ vers $\mathbb{R}^{N_\alpha\times N_{\alpha^c}}$, où $N_\beta = \prod_{\nu\in\beta} n_\nu$. Le rang de la matrice $\mathcal{M}_\alpha(\mathbf{T})$ est appelé rang α de $\mathbf{T}$, et est noté $\mathrm{rank}_\alpha(\mathrm{T})$. Par convention, les rangs D et $\emptyset$ d'un tenseur sont égaux à 1.

En considérant que $S \subset 2^D$ est un ensemble de sous-ensembles de D, nous définissons le S-rank d'un tenseur $\mathbf{T}$ comme le tuple $(\mathrm{rank}_\alpha(\mathrm{T}))_{\alpha\in S} \in \mathbb{N}^{|S|}$. Pour un ensemble donné S et un tuple $r = (r_\alpha)_{\alpha\in S}$, un format de tenseur $\mathcal{T}_r^S$ est défini comme l'ensemble des tenseurs dont le rang S est inférieur à r,

$$\mathcal{T}_r^S = \{\mathbf{T} \in \mathbb{R}^{n_1\times\cdots\times n_d} : \mathrm{rank}_\alpha(\mathrm{T}) \le r_\alpha, \alpha \in S\}$$

Lorsque S est un arbre de partition de dimension sur D (avec la racine D et les feuilles $\{\nu\}, 1 \le \nu \le d$) ou un sous-ensemble d'un tel arbre, $\mathcal{T}_r^S$ est appelé un format tensoriel arborescent (Falcó, Hackbusch, and Nouy 2018). Ce format comprend le format de Tucker pour un arbre trivial (où $S = \{\{1,\dots,d\},\{1\},\dots,\{d\}\}$), le format de Tucker hiérarchique (HT) (Hackbusch and Kuhn 2009) pour un arbre binaire, et le format train de tenseurs décrit ci-dessous.

Le Format Train de Tenseurs

Le format train de tenseurs (TT) a été introduit dans (Oseledets 2009; Oseledets and Tyrtyshnikov 2009) dans le contexte de l'analyse numérique. Il était déjà connu en physique quantique sous le nom de Matrix Product State (voir, par exemple, (Schollwöck 2011)). Ce format correspond au format tensoriel $\mathcal{T}_r^S$ avec $S = \{\emptyset,\{1\},\{1,2\},\dots,\{1,\dots,d-1\},D\}$. Étant donné un tuple d'entiers $r = (r_0, r_1, \dots, r_d)$, avec $r_0 = r_d = 1$, un tenseur $\mathbf{T}$ dans le format tensoriel $\mathcal{T}_r^S$ admet la représentation

$$\mathbf{T}(i_1,\dots,i_d) = \sum_{k_1=1}^{r_1} \cdots \sum_{k_{d-1}=1}^{r_{d-1}} \mathbf{T}^{(1)}(1,i_1,k_1)\cdots \mathbf{T}^{(d)}(k_{d-1},i_d,1) \quad (4)$$

où les $\mathbf{T}^{(i)} \in \mathbb{R}^{r_{i-1}\times n_i\times r_i}$ sont des tenseurs d'ordre 3 appelés noyaux. Les entiers minimaux $(r_0, r_1, \dots, r_d)$ tels que $\mathbf{T}$ a une représentation (selon l'équation 4) est appelé le rang TT de $\mathbf{T}$.

Complexité de stockage La complexité de stockage d'un tenseur avec des rangs TT bornés par R et des tailles de mode bornées par N est en $O(dNR^2)$. Ce format de tenseur permet de contourner le fléau de la dimensionnalité pour les classes de tenseurs avec des rangs TT uniformément bornés ou croissant polynomialement avec d.

Approximation dans le Format Train de Tenseurs

Chaque tenseur a une représentation exacte dans le format TT (Oseledets 2011), éventuellement avec des rangs de représentation élevés.

Afin de contrôler la taille de ses rangs, de nombreux algorithmes ont été proposés pour non seulement transformer mais aussi compresser les tenseurs complets en tenseurs au format TT. L'algorithme décrit ci-dessous, introduit dans (Oseledets 2011)[p. 2301], nous permet d'obtenir une approximation $\widetilde{\mathbf{T}}_\epsilon$ au format TT d'un tenseur donné $\mathbf{T}$ avec une précision relative fixée ϵ, *i.e.* , $\left\|\mathbf{T} - \widetilde{\mathbf{T}}_\epsilon\right\|_F \le \epsilon \|\mathbf{T}\|_F$, où $\|\cdot\|_F$ désigne la norme tensorielle de Frobenius (ou canonique) (si $\mathbf{T} \in \mathbb{R}^{\{n_1,\cdots,n_d\}}$, $\|\mathbf{T}\|_F^2 = \sum_{i_1,\cdots,i_d} \mathbf{T}^2_{i_1,\cdots,i_d}$). L'algorithme s'appuie sur des décompositions en valeurs singulières de matrices. Pour plus de détails sur le format TT et ses applications, voir (Gelß 2017). L'algorithme est ici décrit pour le cas où l'entrée $\mathbf{T}$ est un tenseur complet (un tableau multidimensionnel) mais une version où l'entrée est directement au format TT existe également (Oseledets 2011)[p. 2305].

Algorithm 1 *Compress($\mathbf{T}, \epsilon$) - Approximation au format TT à l'aide de SVD tronquée d'ordre supérieur*

Input : Tensor $\mathbf{T} \in \mathbb{R}^{n_1\times\cdots\times n_d}$ and tolerance ϵ

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 $r_\nu \le s$ to the smallest index such that $\sigma^2_{r_{\nu+1}} + \dots + \sigma_s^2 \le \epsilon^2/(d-1)$
6: Discard rows and columns of U, Σ, and V corresponding to singular values $\sigma_{r_{\nu+1}},\dots,\sigma_s$
7: Define the ν-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 $\mathbf{A} = \mathcal{M}^{-1}_{\{1\}}(\Sigma V^T) \in \mathbb{R}^{r_\nu\times n_{\nu+1}\times\cdots\times n_d}$
9: Define the d-th core $\mathbf{U}^{(d)} = \mathcal{M}^{-1}_{\{1\}}(\mathbf{A})$

Réseaux Bayésiens et Trains de Tenseurs

L'utilisation du format TT semble être une approche prometteuse pour réduire la consommation mémoire

des potentiels dans les PGM. Un tel format permet de passer d'une représentation des potentiels qui évolue non plus exponentiellement avec le nombre de parents mais linéairement, avec un contrôle sur l'approximation induite.

Compte tenu du contexte de cet article, nous avons décidé de nous concentrer sur une expérimentation autour d'une inférence qui manipulerait les potentiels comme des tenseurs au format TT au lieu de tableaux multidimensionnels.

Proposer une réinterprétation de l'inférence de Shafer-Shenoy (Shenoy and Shafer 1990) comme première expérience était simple puisqu'elle ne nécessite de redéfinir que quelques opérations, ce qui la rend facile à comparer à l'algorithme standard. Résumons les avantages d'un tel format :

1. Ce changement dans la représentation des données pourrait être intéressant non seulement pour les inférences mais pour tous les algorithmes qui manipulent de grands potentiels.
2. Une approximation avec une précision contrôlée peut être obtenue en utilisant la la procédure **Compress** définie précédemment ;
3. Une approximation peut être trouvée en utilisant une limite supérieure pour les rangs TT, permettant de contrôler facilement l'utilisation de la mémoire ;

Avant de redéfinir les opérations élémentaires de Shafer-Shenoy pour les tenseurs au format TT et, finalement, l'algorithme complet, nous devons introduire une autre opération sur les tenseurs, le produit de Hadamard. À titre de comparaison, nous donnons la définition de cette opération pour les tenseurs pleins.

Produit de Hadamard. Le produit de Hadamard (noté $\circledast$) de deux tenseurs $\mathbf{A}$ et $\mathbf{B}$ de même taille $n_1 \times \ldots \times n_d$ donne un tenseur $\mathbf{C} = \mathbf{A} \circledast \mathbf{B}$ avec des entrées

$$\mathbf{C}(i_1, \ldots, i_d) = \mathbf{A}(i_1, \ldots, i_d)\mathbf{B}(i_1, \ldots, i_d) \qquad (5)$$

Produit de Hadamard au Format TT

Rappelons qu'un tenseur $\mathbf{T}$ avec la représentation au format TT (équation 4) peut s'écrire

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

Si $\mathbf{A}$ et $\mathbf{B}$ sont deux tenseurs de même taille et dont les représentations aux formats TT $\mathbf{A} = \mathbf{A}^{(1)} \times^1 \cdots \times^1 \mathbf{A}^{(d)}$ et $\mathbf{B} = \mathbf{B}^{(1)} \times^1 \cdots \times^1 \mathbf{B}^{(d)}$, alors leur produit de Hadamard $\mathbf{A} \circledast \mathbf{B}$ a une représentation au format TT :

$$\mathbf{A} \circledast \mathbf{B} = (\mathbf{A}^{(1)} \boxtimes_{\{1,3\}} \mathbf{B}^{(1)}) \times^1 \cdots \times^1 (\mathbf{A}^{(d)} \boxtimes_{\{1,3\}} \mathbf{B}^{(d)}) \qquad (6)$$

où $\boxtimes_{\{1,3\}}$ est le produit partiel de Kronecker selon les modes 1 et 3, défini dans l'équation 2.

$\mathbf{A}^{(i)} \in \mathbb{R}^{r^A_{i-1} \times n^A_i \times r^A_i}$ et $\mathbf{B}^{(i)} \in \mathbb{R}^{r^A_{i-1} \times n^B_i \times r^B_i}$ désignent les noyaux de $\mathbf{A}$ et $\mathbf{B}$ pour tout $i \in (1, \ldots, d)$. Pour pouvoir calculer le produit de Kronecker partiel $\mathbf{A}^{(i)} \boxtimes_{\{1,3\}} \mathbf{B}^{(i)}$ nous devons nous assurer que $n^A_i = n^B_i$. Dans ce but et pour éviter d'avoir à manipuler la position des noyaux lors de l'inférence, nous allons forcer un ordre arbitraire sur les variables au sein d'un potentiel (et pour des raisons de simplicité, nous utiliserons un ordre topologique).

Algèbre des Potentiels Avec le Format TT

En utilisant la procédure **Compress**, nous sommes maintenant capables de compresser tout tenseur associé à une fonction multivariée (CPT, potentiels, etc.) dans un format TT avec une précision donnée. Afin de construire des algorithmes utilisant ce format compressé, nous devons maintenant définir les opérations utilisées par de tels algorithmes. Dans notre cas, le produit clique-séparateur et une marginalisation.

Produit clique-séparateur. Le produit du potentiel ϕ d'une clique avec l'un de ses séparateurs ψ peut être obtenu en utilisant un produit de Hadamard entre tenseurs. Cependant, cela nécessite que ϕ et ψ soient des tenseurs de même ordre et de même taille. Puisque les variables du séparateur forment un sous-ensemble des variables de la clique, ψ doit être identifié à un tenseur ψ' ayant l'ordre et la taille de ϕ. Il suffit de considérer le cas où ϕ est un tenseur d'ordre d et de taille $n_1 \times \ldots \times n_d$ dépendant des variables $(i_1, \ldots, i_d)$ et ψ est un tenseur d'ordre $d-1$ dépendant des variables $(i_1, ..., i_{\nu-1}, i_{\nu+1}, ..., i_d)$. Alors ψ' est le tenseur d'ordre d tel que $\psi'(i_1, ..., i_d) = \psi(i_1, ..., i_{\nu-1}, i_{\nu+1}, ..., i_d)$, et le produit de Hadamard $\phi \circledast \psi$ doit être interprété comme $\phi \circledast \psi'$.

Propriété Si ψ a une représentation au format TT avec des noyaux $\mathbf{T}^{(\mu)}$ et des rangs de représentation r_ν, $\mu \in \{1, \ldots, \nu-1, \nu+1, \ldots, d\}$, alors ψ' a une représentation au format TT avec des noyaux $\mathbf{T'}^{(\mu)} = \mathbf{T}^{(\mu)}$ pour $\mu \in \{1, \ldots, \nu-1, \nu+1, \ldots, d\}$ et $\mathbf{T'}^{(\nu)} \in \mathbb{R}^{r_{\nu-1} \times n_\nu \times r_\nu}$ tel que $\mathbf{T'}^{(\nu)}(k_{\nu-1}, i_\nu, k_\nu) = \delta_{k_{\nu-1}, k_\nu}$, où δ représente le delta de Kronecker.

Soit ϕ et $\psi \in \mathbb{R}^{i_1 \times \ldots \times \ldots, i_d}$ deux tenseurs au format TT avec $\phi^{(i)} \in \mathbb{R}^{r^\phi_{i-1} \times n_i \times r^\phi_i}$ et $\psi'^{(i)} \in \mathbb{R}^{r^{\psi'}_{i-1} \times n_i \times r^{\psi'}_i}$ pour tous les $i \in \{1, \ldots, d\}$ alors si nous désignons le tenseur ϕ' dans le format TT qui est égal à $\phi \circledast \psi$ et en raison des produits de Kronecker partiels internes utilisés (cf. équation 6), nous avons $\phi'^{(i)} \in \mathbb{R}^{(r^\phi_{i-1} \times r^{\psi'}_{i-1}) \times n_i \times (r^\phi_i \times r^{\psi'}_i)}$. Étant donné le nombre (généralement) important de produits clique-séparateur et afin de limiter la croissance des rangs internes de ϕ', il est nécessaire de recompresser chacun de ces résultats en utilisant la procédure **Compress**.

Marginalisation. L'adaptation de l'opération de marginalisation au cas des potentiels sous forme de trains de tenseurs est plus simple. Les variables d'un séparateur $\psi_{i \to j}$ entre deux cliques $\mathbf{C}_i$ et $\mathbf{C}_j$ étant

un sous-ensemble des variables de $\mathbf{C}_i$ et $\mathbf{C}_j$, on peut marginaliser le tenseur ϕ associé à $\mathbf{C}_i$ afin de former $\psi_{i\to j}$. Si ϕ a une représentation au format TT avec des noyaux $\mathbf{T}^{(\nu)}$, $\nu \in \{1,\dots,d\}$ et $\psi_{i\to j}$ est un séparateur sur $(i_1,...,i_{\nu-1},i_{\nu+1},....,i_d)$, alors $\psi_{i\to j}$ a une représentation au format TT avec des noyaux $\mathbf{T'}^{(\nu)} = \mathbf{T}^{(\nu)}$ pour $\nu \in \{1,\dots,\nu-1,\nu+2,\dots,d\}$ et, pour une marginalisation à droite :

$$\mathbf{T'}^{\nu+1)} = \sum_{i_\nu} \mathbf{T}^{\nu)}(:,i_\nu,:) \times^1 \mathbf{T}^{\nu+1)}$$

Cette opération est (généralement) moins coûteuse que son équivalent sur les tenseurs complets. Dans les algorithmes suivants, nous désignerons par **Marginalize**$(\phi,\psi_{i\to j})$ l'opération qui marginalise ϕ sur les variables qui ne sont pas dans $\psi_{i\to j}$.

Shafer-Shenoy et Trains de Tenseurs

Nous pouvons maintenant redéfinir l'algorithme de Shafer-Shenoy, un algorithme classique à base de propagation de messages, en utilisant le format TT et nos opérations nouvellement définies. Soit $\mathcal{B} = (\vec{\mathcal{G}},\mathbb{P})$ un réseau Bayésien avec $\mathbb{P}$ caractérisé par $\Theta = \{\mathbb{P}(\mathrm{X}|\,\mathrm{Pa}_{\mathrm{X}}^{\vec{\mathcal{G}}},\forall \mathrm{X} \in \mathcal{V}(\vec{\mathcal{G}})\}$. Avant d'initialiser un arbre de jonction $\mathcal{T}$ selon $\mathcal{B}$, nous pouvons construire un nouvel ensemble de potentiels Θ', qui sera utilisé pour générer les potentiels associés à chaque clique tels que :

$$\Theta' = \{\theta'_{\mathrm{X}} | \theta'_{\mathrm{X}} = \mathbf{Compress}(\mathbb{P}(\mathrm{X}|\,\mathrm{Pa}_{\mathrm{X}}^{\vec{\mathcal{G}}}),\epsilon), \forall \mathrm{X} \in \mathcal{V}(\vec{\mathcal{G}})\}$$

Cette phase n'est pas particulièrement coûteuse et pourrait être effectuée avant l'inférence (afin de stocker et réutiliser le modèle tensorisé, par exemple). Les algorithmes suivants décrivent le fonctionnement de nos inférences pour un arbre de jonction $\mathcal{T}$, une racine r dans cet arbre et une tolérance ϵ donnés.

$$\mathbf{T'}^{(\nu+1)} = \sum_{i_\nu} \mathbf{T}^{(\nu)}(:,i_\nu,:) \times^1 \mathbf{T}^{(\nu+1)}$$

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

Input : an initialized JT $\mathcal{T}$, $i,j \in \mathcal{V}(\mathcal{T})$ and a tolerance ϵ
Output Recursively computed $\psi_{i\to j}$

$\phi \leftarrow \phi_i$
for $k \in \mathrm{Adj}(i)\backslash\{j\}$ **do**
  CollectTT$(\mathcal{T},k,i,\epsilon)$
  $\phi \leftarrow$ ***Compress***$(\phi \circledast \psi_{k\to i},\epsilon)$
$\psi_{i\to j} \leftarrow$ ***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 ϵ
Output Recursively computed $\psi_{i\to j}$

$\phi \leftarrow \phi_i$
for $k \in \mathrm{Adj}(i)\backslash\{j\}$ **do**
  $\phi \leftarrow$ ***Compress***$(\phi \circledast \psi_{k\to i},\epsilon)$
$\psi_{i\to j} \leftarrow$ ***Marginalize***$(\phi,\psi_{i\to j})$
for $l \in \mathrm{Adj}(j)\backslash\{i\}$ **do**
  DistributeTT$(\mathcal{T},j,l)$

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

Input : a JT $\mathcal{T}$ of $\mathcal{B} = (\vec{\mathcal{G}},\Theta')$, a root $r \in \mathcal{V}(\mathcal{T})$, a tolerance ϵ
Output a JT $\mathcal{T}$ with messages in both directions on all the separators

for $\mathrm{X} \in \mathcal{V}(\vec{\mathcal{G}})$ **do**
  Assign θ'_{X} to a clique $\mathbf{C}$ s.t $(\mathrm{X} \cup \mathrm{Pa}_{\mathrm{X}}^{\vec{\mathcal{G}}}) \subseteq \mathbf{C}$
CollectTT$(\mathcal{T},r,r,\epsilon)$
DistributeTT$(\mathcal{T},r,r,\epsilon)$

Un avantage de notre approche, qui ne modifie que les opérations élémentaires et les structures de données utilisées, est qu'elle permet de facilement adapter des algorithmes classiques à son usage. Conséquemment, notre nouvelle version de l'algorithme de Shafer Shenoy est très proche de la version classique basée sur l'utilisation de tableaux multidimensionnels. Si les différentes opérations élémentaires ne sont pas aussi gourmandes en mémoire que leur version à base de tenseurs pleins, nous verrons que les appels systématiques à l'algorithme de compression après chaque produit est une limitation. Cependant, nous pensons que ce problème pourrait être surmonté, comme suggéré par (Kressner and Periša 2017).

Résultats Expérimentaux

Pour obtenir nos résultats expérimentaux, nous avons développé une première implémentation de notre algorithme en utilisant plusieurs bibliothèques : T3F (Novikov et al. 2018) pour la manipulation des tenseurs au format TT, TensorFlow pour les opérations liées aux tenseurs et pyAgrum (Ducamp, Gonzales, and Wuillemin 2020) pour la manipulation des réseaux Bayésiens, des arbres de jonction et des potentiels. Pour des raisons de stabilité et de praticité, nous sommes ensuite passés à une autre bibliothèque pour manipuler les tenseurs au format TT, Tensap (Nouy and Grelier 2020). Notre code est disponible sous forme la d'une bibliothèque Python appelé TenGeRine [1].

Nous comparons deux implémentations : un Shafer-Shenoy classique (appelée SS) et Shafer-Shenoy utilisant le format TT (appelée SSTT). Les deux implémentations sont identiques autant que possible sauf que la première manipule des potentiels (modèle et opérations implémentés en C++) et la seconde manipule des tenseurs au format TT (modèle et opérations implémentés avec TensorFlow ou Tensap avec un mélange de Python et de C++). Nous comparons à la fois le temps d'inférence (noté T_{SS} pour SS, T_{SSTT} pour SSTT) ainsi que le nombre de paramètres (des cliques et séparateurs) à la fin de chaque inférence ($\#_{SS}$ pour SS, $\#_{SSTT}$ pour SSTT) et le facteur de compression entre eux ($\frac{\#_{SS}}{\#_{SSTT}}$, noté τ). Tous les tests ont été effectués sur un processeur Intel dual E5-2630v2@2.60GHz et 32Go de RAM. Nous ne profitons pas *encore* des fa-

[1] `https://gitlab.com/agrumery/tengerine`

cilités de calcul offertes par les infrastructures de calculs tensoriels avancés (GPGPU et parallélisation).

Modèles Issus de la Littérature

Nous comparons d'abord nos algorithmes sur des modèles classiques de la littérature [2]. Notre objectif est ici de vérifier que l'utilisation de trains de tenseurs permet bien de réduire l'espace mémoire sans introduire trop d'erreurs (dues à la compression) dans les potentiels calculés.

Temps d'Inférence et Compression Le tableau 1 montre comment l'utilisation de notre approche sur les BN classiques peut améliorer l'utilisation de la mémoire pour les inférences, en particulier dans les réseaux complexes. Au lieu de regarder le nombre de nœuds/arcs dans les modèles, nous avons considéré qu'il était plus pertinent de regarder la taille des cliques, en particulier des plus grandes, car c'est le critère discriminant pour les inférences basées sur les arbres de jonction (Robertson and Seymour 1986).

BN	Param. clique max	Ratio temps	τ
Asia	64	91.9	0.6
Carpo	1.00E+03	365.51	0.78
Child	1.30E+03	152.05	0.72
Alarm	5.80E+03	114.6	0.86
Insurrance	**1.10E+07**	89.5	**3.21**
Pigs	**1.20E+09**	65	**37.79**
Hailfinder	2.10E+09	170.15	0.94
Water	**6.50E+10**	3.5	**187.48**
Munin1	**1.20E+13**	**0.05**	**2148.21**
Diabetes	**1.60E+13**	41.31	**2.03**
Barley	**9.60E+14**	1.71	**35.8**
Mildew	**2.70E+15**	2.4	**19.9**
Link	**4.90E+27**	**207.1**	**-**

Table 1: Nombre maximum de paramètres dans une clique avec SS, ratio de temps d'inférence $\frac{T_{SSTT}}{T_{SS}}$ et taux de compression (τ) pour divers BNs classiques ($\epsilon = 0.001$)

Le format TT ne semble pas être utile lorsque les JT ont de petites cliques, comme avec Asia ou en Alarm. En effet, lorsque les cliques et les séparateurs sont petits, le format TT peut augmenter le nombre de paramètres nécessaires pour les décrire. Dans le cas d'Asia, par exemple, la plus grande clique est un tenseur d'ordre 6 avec seulement 64 valeurs. Il n'est donc pas vraiment surprenant de voir le nombre de paramètres augmenter lors de la conversion d'un tel tenseur en 6 tenseurs d'ordre 3. Ces résultats indiquent qu'il devrait y avoir une limite inférieure dans la dimensionnalité des potentiels à partir de laquelle l'utilisation du format TT serait contre-productive.

Cependant, dans le cas d'un réseau complexe tel que Munin1, le format TT permet de réduire considérablement le nombre de paramètres (par un facteur de 2148 !). Dans le cas de Link (592 cliques, dont une à 20 dimensions), notre implémentation classique de Shafer-Shenoy n'a pas pu terminer l'inférence, par manque de mémoire (SSTT a pris 207 secondes). Les économies de mémoire ne semblent pas impliquer une réduction du temps d'inférence, comme le montre par exemple le réseau Barley, mais cela n'est pas surprenant puisque, comme mentionné précédemment, notre prototype n'utilise pas toutes les optimisations possibles liées à l'utilisation de méthodes tensorielles. En outre, les opérations sur TT sont, dans leur forme actuelle, assez complexes et coûteuses en termes de calcul. Nous pensons qu'il y a beaucoup de place pour l'amélioration d'un point de vue algorithmique, comme suggéré dans (Kressner and Periša 2017), où les auteurs proposent des stratégies efficaces pour limiter les coûts de calcul associés à l'utilisation de produits Hadamard entre tenseurs au format Tucker.

Erreur d'Approximation Rappelons que lors de l'utilisation de l'algorithme **Compress**, le paramètre de tolérance ϵ est utilisé par des SVDs tronquées successives pour réduire la taille du tenseur initial en supprimant autant que possible les informations *les moins pertinentes*. Cette suppression introduit, par conséquent, une erreur dans les calculs, qui doit être discutée.

A cet égard, nous comparons dans le tableau 2 la valeur exacte de chaque probabilité p dans chaque postérieur et sa version approchée avec la SSTT $\hat{p}$ associée aux inférences du tableau 1. On observe l'erreur absolue $|p - \hat{p}|$ ainsi que l'erreur relative $\frac{|p-\hat{p}|}{p}$. Si d'autres critères d'évaluation tels que la divergence de Kullback-Leibler entre p et $\hat{p}$ ont été envisagés, ils nous ont semblé être les plus faciles à interpréter.

Si les erreurs absolues semblent contenues pour une tolérance de $\epsilon = 10^{-3}$, dans une fourchette de un pour cent, les erreurs relatives peuvent être élevées, suggérant que les petites probabilités sont mal approchées. Dans le cas du Mildew, par exemple, l'erreur relative maximale se produit lorsqu'une probabilité de $1,35e^{-11}$ est approchée avec $4,60e^{-08}$. Heureusement, comme le montre la figure 1, plus la tolérance est faible, plus les erreurs sont réduites.

	Absolute error		Relative error	
BN name	**Maximum**	**Moyenne**	**Maximum**	**Moyenne**
Alarm	3,78e-04	4,10e-05	9,73e-03	3,53e-04
Asia	3,96e-04	1,98e-04	3,96e-02	4,56e-03
Barley	4,09e-04	2,18e-05	9,99e-01	9,04e-03
Carpo	7,76e-04	7,68e-05	9,80e-01	8,20e-03
Child	7,69e-05	3,69e-06	1,53e-03	3,43e-05
Diabetes	8,25e-04	5,46e-05	1,00e+00	7,27e-03
Hailfinder	2,76e-04	5,68e-06	8,09e-04	2,11e-05
Insurance	4,97e-04	4,13e-05	8,40e-01	1,10e-02
Mildew	2,43e-04	1,32e-05	3,40e+03	4,60e+00
Munin1	3,19e-03	2,30e-04	6,84e+02	1,16e+00
Pigs	3,33e-15	6,29e-16	1,33e-14	2,25e-15
Water	8,85e-04	6,88e-05	1,00e+00	3,27e-03

Table 2: Erreurs absolues ($|p - \hat{p}|$) et relatives ($\frac{|p-\hat{p}|}{p}$) pour nos BN classiques ($\epsilon = 0.001$)

Le cas de Pigs, présenté dans la figure 2, est très intéressant. Ce modèle, complexe par sa taille (et par

[2] `https://gitlab.com/agrumery/pgmrepository`

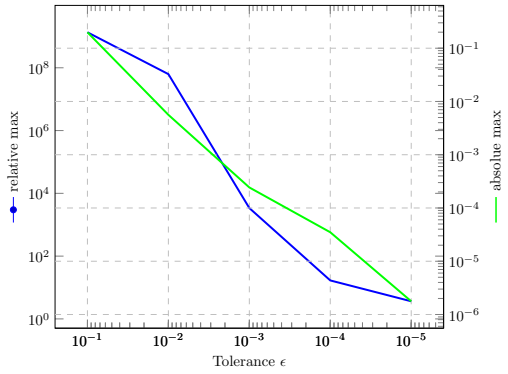


Figure 1: Évolution des erreurs maximales pour Mildew en fonction de multiples seuils de tolérance

la présence de quelques grandes cliques) se compresse très bien (par un facteur de 38) sans introduire d'erreurs notables (de l'ordre de la précision machine).

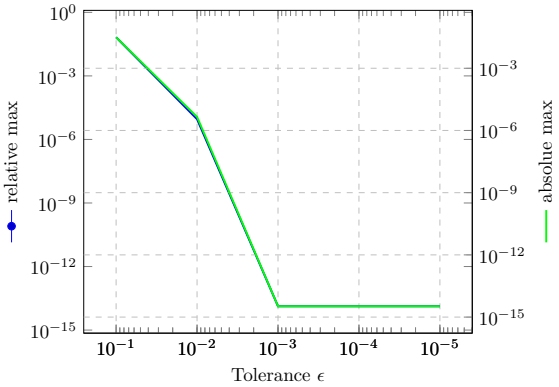


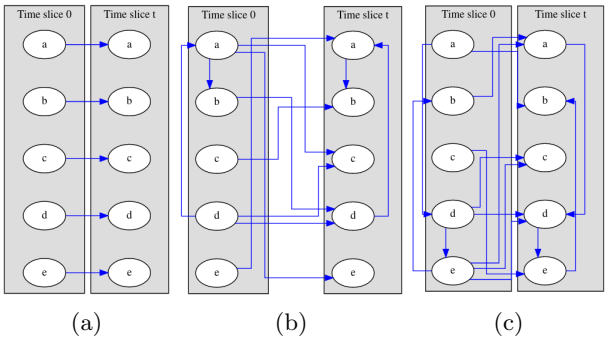
Figure 2: Évolution des erreurs maximales pour Pigs en fonction de multiples seuils de tolérance

Une première analyse suggère que cela pourrait être dû au fait que les CPTs de Pigs soient très déterministes : dans ce cas, nous nous attendons à ce que les tenseurs pleins associés à chaque potentiel soient peu denses et donc facilement compressés avec des SVD tronqués.

Si ces premiers résultats sont encourageants, nous avons voulu nous placer dans des cas arbitrairement plus complexes pour nous assurer du passage à l'échelle de notre approche.

SSTT et Réseaux Complexes

Afin de générer des BNs de plus en plus complexes, nous avons utilisé des réseaux Bayésiens dynamiques (dBN) (Dagum, Galper, and Horvitz 1992), les dBN sont une généralisation des chaînes de Markov et peuvent être considérés comme des BN qui relient les variables entre elles sur des pas de temps adjacents. Une fois que la relation entre deux pas de temps est définie (dans un graphe appelé 2TBN), ils peuvent être facilement déployés (déroulés) pour un nombre arbitraire de pas de temps. Lorsque nous créons un arbre de jonction à partir d'un dBN déroulé, les cliques ont tendance à être très grandes, ce qui rend souvent l'inférence exacte intraitable (Murphy 2002).



(a) (b) (c)

Figure 3: 2TBN of : (a) dBN_1, (b) dBN_2, (c) dBN_3

Pour nos expériences, nous avons défini 3 dBN, dont les 2TBN associés sont présentés dans la figure 3. La taille du domaine de chaque variable est de 10 et les CPT ont été générés aléatoirement. Le dBN_1 (figure 3.a) devrait être le pire cas pour notre algorithme basé sur le train tensoriel : il comporte exactement cinq chaînes de Markov et toutes les cliques ont une taille de 2 tandis que les dBN_2 et dBN_3 (figures 3. b et 3.c) sont de plus en plus complexes, avec de plus en plus d'arcs intra/inter tranches.

Temps d'Inférence et Coût Mémoire La comparaison du temps d'inférence de Shafer-Shenoy et de notre algorithme confirme que plus une inférence est complexe et gourmande en mémoire, plus l'utilisation du format Train Tensor est intéressante, comme le montre le tableau 3. Dans le cas de dBN_1, une décomposition est inutile puisque la taille des potentiels ne changera pas, seul leur nombre augmentera linéairement avec le nombre de pas de temps. Avec dBN_2, en revanche, la version TT de Shafer-Shenoy montre à quel point la compression des potentiels peut aider à dimensionner l'inférence. Enfin, dans le cas de dBN_3, des manques de mémoire sont observés sur 7 pas de temps pour l'algorithme standard alors que la version TT évolue linéairement jusqu'à 150 pas de temps. Le facteur limitant étant la largeur de l'arbre de jonction, l'utilisation d'une représentation à faible rang pour les potentiels, comme le format TT, améliore considérablement l'utilisation de la mémoire, comme le montre le tableau 4. Pour un facteur de tolérance ϵ de $0,05$, le nombre de paramètres de notre modèle utilise moins d'un centième de l'espace utilisé par le modèle exact, avec une erreur relative maximale de $0,12$.

Erreur d'Approximation avec SSTT Pour évaluer les erreurs introduites par la compression, nous avons effectué les inférences dix fois sur dBN_2 avec 50 pas de temps et des CPT aléatoires pour chaque itération. Les figures 5 et 4 montrent que, comme

	$\mathbf{dBN_1}$	$\mathbf{dBN_2}$	$\mathbf{dBN_3}$	
Nb. pas	$\frac{T_{SSTT}}{T_{SS}}$	$\frac{T_{SSTT}}{T_{SS}}$	$\frac{T_{SSTT}}{T_{SS}}$	T_{SSTT}
4	179.89	61.07	7.79	2.95 s
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 3: Ratio entre les temps d'inférence pour SSTT et SS ($\epsilon = 0.05$)

Nb. pas	**SS**	**SSTT**	τ
4	3.33e+05	4.15e+04	8.0
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 4: Nombre de paramètres et facteurs de compression (τ) avec SS et SSTT (dBN_2, $\epsilon = 0.05$)

prévu, une tolérance inférieure ϵ dans l'algorithme **Compress** diminue les erreurs relatives et absolues maximales mais augmente le temps d'inférence.

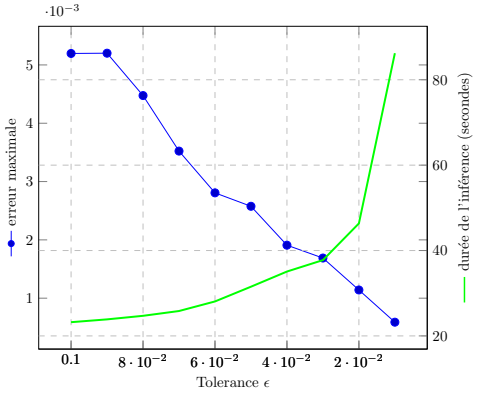


Figure 4: Évolution de l'erreur absolue maximale et temps d'inférence pour dBN_2 avec 50 pas de temps

Il est intéressant de noter que la figure 6 montre que l'erreur maximale est presque constante avec le nombre de pas de temps (alors qu'on aurait pu s'attendre à ce qu'elle soit amplifiée lorsque le nombre de pas de temps augmente). Cela semble indiquer qu'il y a peu de propagation d'erreurs durant les longues séquences de calculs approchés.

Ces premiers résultats expérimentaux sur des modèles complexes sont très encourageants ; ils tendent à confirmer que l'utilisation de formats tensoriels de faibles rangs permet de réaliser passer à l'échelle des

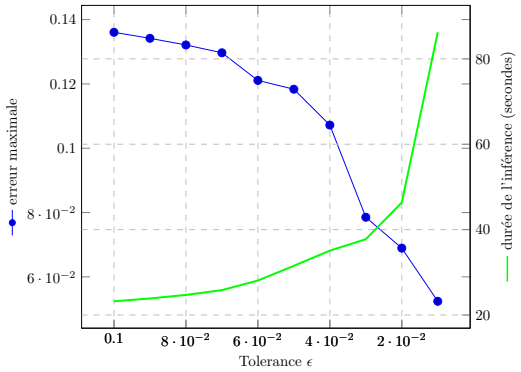


Figure 5: Évolution de l'erreur relative maximale et temps d'inférence pour dBN_2 avec 50 pas de temps

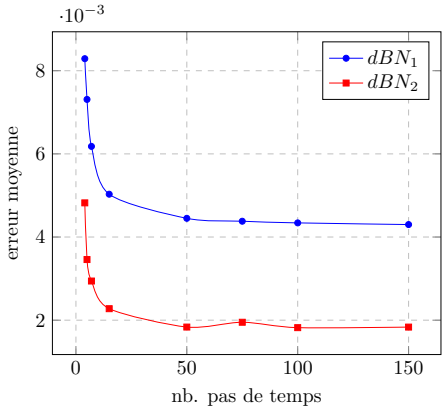


Figure 6: Évolution de l'erreur relative moyenne en fonction du nombre de pas de temps ($\epsilon = 0.05$)

inférences avec une erreur contrôlable dans des cas où une inférence exacte serait impossible en raison d'un manque de mémoire.

Comparaison avec une Autre Inférence Approchée

Afin d'étendre notre comparaison, nous avons comparé notre approche à une autre inférence approchée, Loopy Belief Propagation (LBP) (Murphy, Weiss, and Jordan 1999) (plutôt qu'aux méthodes d'échantillonnage qui n'offrent aucune garantie en termes de temps d'inférence). Les résultats préliminaires présentés dans le tableau 5 nous montrent deux choses :

- Notre approche offre de meilleurs résultats en ce qui concerne l'erreur absolue,
- Une réduction de la tolérance utilisée lors de la troncature nous permet d'avoir globalement de bien meilleures erreurs relatives que LBP sur cet ensemble de modèles. Lorsque les résultats de LBP sont meilleurs, c'est généralement de façon marginale, à l'exception de Mildew dont les très petites marginales sont bruitées par notre approximation.

	Maximum absolute error			Maximum relative error		
BN	**1e-3**	**1e-5**	**LBP**	**1e-3**	**1e-5**	**LBP**
Alarm	3.78e-04	1.38e-06	2.85e-02	9.73e-03	4.29e-05	3.48e-01
Asia	3.96e-04	6.40e-09	1.10e-03	3.96e-02	1.42e-08	2.27e-03
Barley	4.09e-04	1.30e-06	1.58e-02	9.99e-01	1.11e-02	1.08e+01
Carpo	7.76e-04	7.28e-06	8.63e-03	9.80e-01	8.57e-03	1.24e-01
Child	7.69e-05	5.50e-09	5.69e-03	1.53e-03	1.55e-08	3.46e-02
Diabetes	8.25e-04	1.77e-05	2.28e-02	1.00e+00	9.99e-01	2.65e-01
Hailfinder	2.76e-04	5.45e-07	1.87e-03	8.09e-04	2.17e-06	6.91e-03
Insurrance	4.97e-04	3.31e-06	1.57e-02	8.40e-01	2.52e-01	1.03e-01
Mildew	2.43e-04	1.77e-06	4.41e-03	3.40e+03	3.64e+00	4.66e-01
Munin1	3.19e-03	1.28e-05	1.20e-02	6.84e+02	9.99e-01	4.44e-01
Pigs	3.33e-15	3.33e-15	4.45e-03	1.33e-14	1.33e-14	1.24e-02
Water	8.85e-04	5.20e-06	6.36e-04	1.00e+00	9.77e-01	1.25e-01

Table 5: Erreurs maximales entre SSTT avec tolérances multiples et LBP

Par ailleurs, et bien que LBP soit plus rapide que notre algorithme, il ne garantit pas la convergence et, de fait, ne permet pas d'estimer la qualité de l'approximation. Ces premiers résultats sont encourageants, si l'utilisation de notre algorithme est, en l'état actuel, plus chronophage que cet algorithme d'inférence approché classique, il semble néanmoins proposer une approximation contrôlée qui passe bien à l'échelle.

Discussion et Travaux Futurs

La représentation des potentiels sous forme de trains de tenseurs proposée dans cet article permet d'avoir un compromis contrôlé entre la complexité temps/espace et la précision lors de l'inférence. Même si nous nous sommes concentrés sur un seul algorithme (Shafer-Shenoy), il est facile de voir à quel point cette approche pourrait être généralisée dans d'autres contextes, tels que les champs aléatoires de Markov (Koller and Friedman 2009) ou les modèles relationnels probabilistes (Torti, Wuillemin, and Gonzales 2010; Medina Oliva et al. 2010), où les systèmes peuvent facilement être déployés pour représenter des mondes à grande échelle. De plus, cette représentation compacte pourrait également améliorer grandement l'expression des tableaux de probabilités conditionnelles avec un grand ensemble de parents tels que les modèles ICI (non décomposables) (Zagorecki, Voortman, and Druzdzel 2006) ou les agrégateurs (Wuillemin and Torti 2012), des premiers travaux dans cette direction ont d'ailleurs été présentés dans (Ducamp 2021). Comme les opérations impliquées dans nos algorithmes sont fortement parallélisables, elles pourraient également bénéficier d'un pipeline GPGPU utilisant, par exemple, CUDA. Dans certains cas, comme le montre la figure 1, la conversion du potentiel en tenseurs au format TT est inutile ou même contre-productive par rapport à l'utilisation d'une algèbre standard. Par conséquent, une inférence hybride déterminant pour chaque potentiel la meilleure algèbre à utiliser pourrait être une solution. Enfin, le format TT est un cas particulier de réseaux tensoriels arborescents plus généraux, qui sont des formats tensoriels associés à des arbres de partition de dimension : (Hackbusch and Kuhn 2009; Nouy 2019). L'architecture du réseau (donnée par l'arbre) peut avoir un impact significatif sur les capacités d'approximation d'un réseau tensoriel arborescent. Dans le contexte de la PGM, des performances beaucoup plus intéressantes pourraient être attendues en considérant des réseaux tensoriels arborescents avec des méthodes d'adaptation de l'arbre, comme proposé dans (Grelier, Nouy, and Chevreuil 2018; Grelier, Nouy, and Lebrun 2019).

References

Bachmayr, M.; Schneider, R.; and Uschmajew, A. 2016. Tensor Networks and Hierarchical Tensors for the Solution of High-Dimensional Partial Differential Equations. *Foundations of Computational Mathematics* 1–50.

Carroll, J. D.; and Chang, J.-J. 1970. Analysis of individual differences in multidimensional scaling via an n-way generalization of "Eckart-Young" decomposition. *Psychometrika* 35(3): 283–319.

Cichocki, A.; Lee, N.; Oseledets, I.; Phan, A.-H.; Zhao, Q.; and Mandic, D. P. 2016. 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.

Cichocki, A.; Lee, N.; Oseledets, I.; Phan, A.-H.; Zhao, Q.; Sugiyama, M.; and Mandic, D. P. 2017. 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.

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

Ducamp, G. 2021. *Probabilistic rules optimization and compilation*. Ph.D. thesis, Sorbonne Université, Paris.

Ducamp, G.; Bonnard, P.; Nouy, A.; and Wuillemin, P.-H. 2020. An Efficient Low-Rank Tensors Representation for Algorithms in Complex Probabilistic Graphical Models. In Jaeger, M.; and Nielsen, T. D., eds., *Proceedings of the 10th International Conference on Probabilistic Graphical Models*, volume 138 of *Proceedings of Machine Learning Research*, 173–184. PMLR.

Ducamp, G.; Gonzales, C.; and Wuillemin, P.-H. 2020. aGrUM/pyAgrum : a toolbox to build models and algorithms for Probabilistic Graphical Models in Python. In *PGM 2020 (The 10th International Conference on Probabilistic Graphical Models)*. Aalborg, Danemark. To be published.

Falcó, A.; Hackbusch, W.; and Nouy, A. 2018. Tree-based tensor formats. *SeMA Journal* .

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

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

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

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

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

Harshman, R. 1970. Foundations of the PARAFAC procedure: Models and conditions for an "explanatory" multi-model factor analysis.

Hillar, C.; and Lim, L.-H. 2013. Most tensor problems are NP-hard.

Hitchcock, F. L. 1927. The Expression of a Tensor or a Polyadic as a Sum of Products. *Journal of Mathematics and Physics* 6(1-4): 164–189.

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

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

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

Kressner, D.; and Periša, L. 2017. Recompression of Hadamard Products of Tensors in Tucker Format. *SIAM Journal on Scientific Computing* 39: A1879–A1902.

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

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

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

Murphy, K. P.; Weiss, Y.; and Jordan, M. I. 1999. 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: Morgan Kaufmann Publishers Inc.

Nouy, A. 2017. Low-Rank Methods for High-Dimensional Approximation and Model Order Reduction. In Benner, P.; Cohen, A.; Ohlberger, M.; and Willcox, K., eds., *Model Reduction and Approximation: Theory and Algorithms*. SIAM, Philadelphia, PA.

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

Nouy, A.; and Grelier, E. 2020. anthony-nouy/tensap v1.1. doi:10.5281/zenodo.3903331. URL `https://doi.org/10.5281/zenodo.3903331`.

Novikov, A.; Izmailov, P.; Khrulkov, V.; Figurnov, M.; and Oseledets, I. 2018. Tensor Train decomposition on TensorFlow (T3F). *arXiv preprint arXiv:1801.01928* .

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

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

Oseledets, I. V.; and Tyrtyshnikov, E. E. 2009. Breaking the Curse of Dimensionality, Or How to Use SVD in Many Dimensions. *SIAM Journal on Scientific Computing* 31(5): 3744–3759.

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

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

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

Shenoy, P. P.; and Shafer, G. 1990. Axioms for Probability and Belief-Function Propagation. In *Uncertainty in Artificial Intelligence*. Amsterdam: Elsevier.

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

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

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

Zagorecki, A.; Voortman, M.; and Druzdzel, M. J. 2006. Decomposing Local Probability Distributions in Bayesian Networks for Improved Inference and Parameter Learning. In Sutcliffe, G.; and Goebel, R., eds., *Proceedings of the Nineteenth International Florida Artificial Intelligence Research Society Conference, Melbourne Beach, Florida, USA, May 11-13, 2006*. AAAI Press.