Recommandation par inférence bayésienne. Application à la configuration de produit

Hélène Fargier, Pierre-François Gimenez, Jérôme Mengin

Recommandation par inférence bayésienne

Application à la configuration de produit

Hélène Fargier , Pierre-François Gimenez , Jérôme Mengin

IRIT, CNRS, Université de Toulouse, France
{helene.fargier,pierre-francois.gimenez,jerome.mengin}@irit.fr

RÉSUMÉ. *Cet article traite du problème de la recommandation de valeurs dans le cadre de la configuration interactive de produit. L'idée est d'apprendre, hors ligne et à partir d'un historique de vente, des indépendances entre variables sous la forme d'un réseau bayésien ; on peut ensuite utiliser les tables du réseau bayésien appris pour recommander, à chaque étape de la configuration, les options les plus probablement choisies ; ou on peut estimer directement à partir de l'historique les probabilités nécessaires : nous proposons pour cela une variation de l'algorithme* Recursive Conditioning. *Nos expérimentations sur des données réelles montrent que ces approches sont compatibles avec une exploitation en ligne en termes de temps CPU et possèdent une très bonne précision : leur taux de succès est proche du meilleur possible.*

ABSTRACT. *This paper deals with the problem of value recommendation in interactive product configuration. It proposes to learn independencies off-line with a Bayesian network from a sales history. Afterwards, the conditional probability tables can be used to recommend for each step of the configuration the most probable values; we can also estimate those probabilities directly from the sales history: we propose for that a variation of the Recursive Conditioning algorithm. Our experiments on real world dataset show that these approaches are compatible with an online exploitation in terms of CPU time, and have a very good accuracy – their success rate is close to the best possible one.*

MOTS-CLÉS : *réseau bayésien, systèmes de recommandation, configuration de produit, inférence bayésienne.*

KEYWORDS: *Bayesian network, recommender systems, product configuration, Bayesian inference.*

DOI:10.3166/RIA.32.39-74

1. Introduction

Dans des contextes de vente en ligne, un des facteurs de limitation principaux est la difficulté pour l'utilisateur de trouver un article qui corresponde à ses préférences

et, orthogonalement, la difficulté que peut avoir le fournisseur à guider ses clients potentiels. Cette difficulté augmente avec la taille de l'e-catalogue, qui est typiquement exponentielle quand les produits considérés sont configurables. De tels produits sont en effet définis par un ensemble fini de composants, d'options, ou plus généralement par un ensemble de variables, dont les valeurs doivent être choisies par l'utilisateur. L'espace de recherche est ainsi fortement combinatoire (par exemple, dans le problème de configuration de voiture décrit dans (Astesana *et al*., 2010), la définition d'un véhicule utilitaire *Trafic* fait intervenir environ 150 variables pour un catalogue de quelques 10^{27} variantes). Il est généralement exploré pas à pas durant la session de configuration : à chaque pas, l'utilisateur choisit librement une variable qui n'a pas encore été assignée et choisit une valeur. Le problème est que l'utilisateur peut être déstabilisé par la profusion des possibilités : trop de choix peuvent le dérouter et l'amener à abandonner la configuration ou à configurer un produit qui ne le satisfait pas. Ce phénomène est appelé « mass confusion » (Huffman, Kahn, 1998).

Notre objectif est d'équiper ce type de configurateur avec une fonctionnalité de recommandation, qui permette au système de mettre en avant, à chaque pas de la configuration (chaque fois que l'utilisateur sélectionne une variable d'intérêt), les valeurs qui sont les plus susceptibles de lui convenir. Dans le contexte que nous avons étudié, l'ordre de choix des variables n'est pas fixé, et varie avec les utilisateurs : certains préfèrent commencer la configuration selon des critères techniques, d'autres esthétiques, d'autres financiers, etc. ; aucune connaissance sur l'utilisateur n'est disponible, ni sur son ordre de choix, ni sur lui-même ou ses éventuels achats précédents – un utilisateur n'achète pas souvent des produits complexes, comme des voitures ou des cuisines. On possède en revanche une large liste de produits précédemment configurés et achetés par d'autres utilisateurs (des "historiques de vente").

La problématique de la recommandation d'article a été abondamment étudiée (Adomavicius, Tuzhilin, 2005 ; Ricci *et al*., 2011 ; Jannach *et al*., 2010). Les méthodes proposées s'appuient principalement soit sur des méthodes de filtrage collaboratif ; soit sur des méthodes basées sur le contenu. Malheureusement, ces méthodes ne peuvent pas être utilisées telles quelles ici :

– La première raison est que le nombre de produits possibles est colossal – exponentiel dans le nombre de variables à configurer. Cela rend beaucoup de méthodes numériques, comme la populaire factorisation de matrices, impossibles à utiliser en pratique.

– La seconde raison est que le contexte de la configuration interactive diffère du contexte classique de la recommandation de produit : on ne cherche pas à recommander un produit mais seulement *la valeur* d'une de ses variables, sélectionnée par l'utilisateur.

La littérature sur la configuration classique (voir (Hotz *et al*., 2014) pour un survey du domaine) ne peut pas beaucoup aider dans notre contexte, puisqu'elle a considéré la problématique de la recommandation essentiellement sous l'angle de la recommandation de solution (de configuration complète) étant donné la spécification des desiderata de l'utilisateur par des contraintes (Tiihonen, Felfernig, 2010 ; Falkner *et al*., 2011 ;

Tiihonen, Felfernig, 2017) ; une problématique qui reste donc proche de la recommandation d'item. Peu de solutions ont été proposées au problème de la recommandation interactive de valeur, bien que son intérêt ait été identifié il y a plus d'une dizaine d'années (Coster *et al.*, 2002 ; Tiihonen, Felfernig, 2010). Des approches basées sur les k plus proches voisins sont possibles (Coster *et al.*, 2002) mais sont limitées à de petits historiques.

Toute la difficulté, si nous voulons nous rapprocher de méthodes du type filtrage collaboratif, est que nous ne disposons d'aucune connaissance *a priori* sur l'utilisateur, comme son âge ou son sexe, ni sur ses habitudes d'achats, car la plupart des configurateurs en ligne s'utilisent anonymement. Nous ne disposons pas non plus d'un rating des produits ni de la part des utilisateurs précédents, ni de la part de l'utilisateur courant. Ce manque d'informations initiale n'est pas rédhibitoire. En effet, à mesure que l'utilisateur fait des choix sur les valeurs à affecter aux variables, on obtient des informations sur ses préférences, ce qui nous permet de personnaliser progressivement la recommandation. Notre idée générale est de recommander, pour une certaine variable à une certaine étape de la configuration, la valeur qui a été la plus souvent choisie par les précédents utilisateurs dans un contexte similaire. Par contexte, nous entendons l'ensemble des variables précédemment sélectionnées et les valeurs qu'on leur a affectées (c'est la configuration partielle que l'utilisateur actuel est en train de compléter). Notre approche se base sur l'idée qu'il existe une distribution de probabilité p, définie sur tout l'espace des produits configurés, qui quantifie les chances qu'a un produit d'être le produit préféré de l'utilisateur actuel. Cette distribution de probabilité dépend de l'utilisateur, de son environnement, etc., et n'est pas connue, mais en l'absence de ces informations, on peut supposer que l'historique de ventes nous donne une approximation acceptable de cette probabilité : les produits précédemment configurés et achetés par des utilisateurs étaient ceux qu'ils préféraient.

Notre contribution s'articule autour de deux points. Le premier est l'utilisation de réseaux bayésiens pour apprendre cette distribution p et inférer (calculer) la valeur la plus probable, une idée qui n'a jamais été abordée dans la littérature sur la recommandation en configuration interactive. Le second est une nouvelle méthode d'inférence bayésienne qui permet d'utiliser des historiques évolutifs.

Le présent article est structuré comme suit : dans la première section, nous aborderons formellement la problématique de l'apprentissage de réseaux bayésiens pour la configuration et introduirons nos notations. Une courte présentation des réseaux bayésiens, de leurs propriétés, des méthodes d'apprentissage et d'inférence sera faite en section 3, ainsi qu'une description de l'algorithme *Recursive Conditioning*. La section 4 présentera le second algorithme, *Direct Recursive Conditioning*. Nous testerons en section 5 ces deux approches sur des jeux de données réels.

2. Problématique et notations

Un problème de configuration est défini par un ensemble $\mathcal{X}$ de n variables discrètes, chaque variable X prenant ses valeurs dans un domaine fini $\underline{X}$. Une confi-

guration complète est un vecteur $\mathbf{o} \in \prod_{X \in \mathcal{X}} \underline{X}$; nous notons $\underline{\mathcal{X}}$ l'ensemble qui les contient toutes.

Si $\mathbf{U}$ est un ensemble de variables, alors $\underline{\mathbf{U}}$ désigne l'ensemble des configurations partielles $\prod_{X \in \mathbf{U}} \underline{X}$. Dans la suite de cet article, nous utiliserons la notation suivante : les lettres majuscules en caractères gras représenteront des ensembles de variables et nous noterons une instanciation d'un de ces ensembles par la lettre minuscule correspondante, par exemple $\mathbf{u} \in \underline{\mathbf{U}}$.

Si $\mathbf{U}$ et $\mathbf{V}$ sont deux ensembles de variables, et si $\mathbf{v} \in \underline{\mathbf{V}}$, alors $\mathbf{v}[\mathbf{U}]$ est la restriction de $\mathbf{v}$ à $\mathbf{U} \cap \mathbf{V}$. $\mathbf{u}$ est dit compatible avec $\mathbf{v}$ si $\mathbf{u}[\mathbf{U} \cap \mathbf{V}] = \mathbf{v}[\mathbf{U} \cap \mathbf{V}]$; on écrira dans ce cas $\mathbf{u} \sim \mathbf{v}$. Enfin, si $\mathbf{u}$ et $\mathbf{v}$ sont compatibles, nous noterons $\mathbf{uv}$ le tuple qui étend $\mathbf{u}$ avec les valeurs de $\mathbf{v}$ sur les variables de $\mathbf{V} \setminus \mathbf{U}$ (ou, de manière équivalente, $\mathbf{uv}$ étend $\mathbf{v}$ avec les valeurs de $\mathbf{u}$ sur les variables de $\mathbf{U} \setminus \mathbf{V}$).

Dans un problème de configuration interactive, l'utilisateur construit variable par variable le produit qui l'intéresse. À chaque étape, on note $\mathbf{Assigned}$ l'ensemble des variables assignées et $\mathbf{assigned}$ le tuple de leurs valeurs. L'utilisateur choisira alors librement la prochaine variable à assigner : nous la notons Next. Puis le système proposera une valeur parmi celles de $\underline{\mathrm{Next}}$. Notre approche est de présenter à l'utilisateur la valeur $x \in \underline{\mathrm{Next}}$ qui soit la plus susceptible de lui plaire, connaissant ses choix antérieurs $\mathbf{assigned}$.

Dans notre contexte, nous disposons d'historiques de ventes sur lesquels le système base ses recommandations. Formellement, un historique de vente est un (multi-) ensemble $\mathcal{H} \subseteq \underline{\mathcal{X}}$ de configurations complètes qui correspondent à des produits configurés et achetés par de précédents utilisateurs. En pratique, cet historique est souvent évolutif, à savoir qu'il s'enrichit continuellement de nouvelles ventes.

À partir de cette problématique, plusieurs modélisations des préférences utilisateurs seraient envisageables. Par exemple des modèles ordinaux comme les CP-nets (Conditional Preference nets (Boutilier *et al*., 2004)), les arbres lexicographiques (Booth *et al*., 2010) et les PCP-nets (Probabilistic CP-nets (Bigot *et al*., 2013)). La difficulté avec les deux premiers modèles est que les procédures d'acquisition existantes sont des procédures d'élicitation, qui construisent à partir de questions le CP-net de l'utilisateur courant – de nature fondamentalement comparative, ils ne fonctionnent pas de manière passive à partir d'une liste d'exemplaires ayant plu à d'autres utilisateurs. On rencontre la même limitation avec les procédures d'acquisition de modèles numériques comme les GAI nets (Gonzales, Perny, 2004 ; Braziunas, Boutilier, 2005). Les PCP-nets ont l'avantage de pouvoir être appris à partir d'exemplaires vendus (Bigot *et al*., 2014) ; ces exemplaires sont alors interprétés comme les produits préférés de différents utilisateurs. Néanmoins, les PCP-nets, comme les CP nets, ont une expressivité limitée et ne peuvent pas représenter certains ordres de préférence, ce qui pourrait limiter leur précision en recommandation.

Il est également possible d'apprendre les préférences utilisateurs sous la forme d'une loi de probabilité ; plus un produit aura une probabilité élevée, plus il sera considéré comme préférable. C'est cette idée que nous allons développer.

Chaque utilisateur ayant ses propres préférences, on retrouve dans l'historique une grande variété de produits. Cependant, nous n'avons aucune information sur les préférences des utilisateurs ou leur environnement au moment de la configuration. À la place, nous supposons l'existence d'une loi de probabilité p sur l'ensemble des configurations complètes, telle que $p(\mathbf{o})$ soit la probabilité que $\mathbf{o} \in \underline{\mathcal{X}}$ soit le produit préféré d'un utilisateur inconnu dans un environnement inconnu. Si on considère que l'historique de ventes est un échantillon de $\underline{\mathcal{X}}$ qui suit la loi de p, on pourra l'utiliser pour estimer cette loi.

Pour une valeur $x \in \underline{\mathrm{Next}}$, $p(\mathrm{Next} = x \mid \mathbf{assigned})$ est la probabilité marginale conditionnelle que la valeur x soit la valeur préférée de l'utilisateur actuel pour Next sachant ses choix précédents. Nous pouvons alors recommander la valeur la plus probable :

$$\underset{x \in \underline{X}}{\operatorname{argmax}}\ p(\mathrm{Next} = x \mid \mathbf{assigned})$$

Une première idée, naïve, pour calculer $p(x \mid \mathbf{assigned})$ pourrait être de calculer la fréquence de x dans l'ensemble des produits vendus qui vérifient $\mathbf{assigned}$. Dans la suite de cet article, on notera $\#(\mathbf{u})$ le nombre de configurations complètes dans $\mathcal{H}$ compatibles avec la configuration partielle $\mathbf{u}$. Si $\#(\mathbf{u})$ est assez grand, on peut estimer $p(\mathbf{u})$ par :

$$p(\mathbf{u}) = \frac{\#(\mathbf{u})}{|\mathcal{H}|}$$

Cette estimation sera fiable si $\#(\mathbf{u})$ est assez grand, donc si $\mathbf{U}$ est assez petit. Si $\mathbf{U}$ est trop grand, nous chercherons à décomposer le calcul en plusieurs calculs sur des ensembles plus petits, jusqu'à pouvoir effectuer une estimation dans $\mathcal{H}$. Un moyen de réaliser cette estimation par décomposition est d'apprendre un réseau bayésien qui sera utilisé en ligne, pour la recommandation, via des opérations de conditionnement et de calcul de marginales.

Dans les sections qui suivent, nous travaillons sur le problème central tel qu'il vient d'être décrit – considérer l'historique de vente comme étant représentatif d'une distribution mal connue et calculer, à partir de méthodes d'inférence bayésienne, la probabilité des différentes valeurs d'une variable Next, connaissant $\mathbf{assigned}$.

En pratique, le problème est plus complexe, puisque d'une part l'historique peut être évolutif et que d'autre part, comme c'est généralement le cas en configuration, toutes les combinaisons de valeurs ne peuvent pas conduire à un objet réalisable – les combinaisons de valeurs sont soumises à des contraintes métiers, qu'elle soient techniques, commerciales, liées aux stocks, etc. On ne peut pas par exemple laisser l'utilisateur construire une voiture décapotable avec un toit ouvrant. De plus, ces contraintes peuvent varier avec le temps – les éléments de l'historique de vente ne satisfont pas toutes les contraintes à respecter le jour de la configuration, et peuvent avoir été soumises à des contraintes qui ne sont plus à respecter. C'est typiquement le cas pour des contraintes de « packs » (qui sont souvent saisonnières) ou lorsque de nouvelles options, couleurs, etc. sont proposées. Nous ne traiterons pas directement du problème

de la prise en compte des contraintes dans cet article, supposant que l'on a un moyen de savoir quelles sont les valeurs cohérentes avec les choix de configurations courants de l'utilisateur – différentes techniques de propagation de contraintes, de cohérence globale (Bessiere *et al.*, 2013) ou de compilation (Pargamin, 2002 ; Amilhastre *et al.*, 2002 ; Hadzic *et al.*, 2007) peuvent être utilisées. Notre approche est de résoudre le problème central, et de proposer à l'utilisateur la plus probable des valeurs cohérentes avec la configuration courante.

3. Réseaux bayésiens

Les réseaux bayésiens permettent de représenter de manière compacte une distribution de probabilité sur un espace combinatoire, en utilisant les indépendances existant entre certaines variables d'un problème donné. Nous rappelons dans cette section la définition d'un réseau bayésiens et certaines propriétés qui seront utiles par la suite.

3.1. Définition et propriétés

DÉFINITION 1. — *Un réseau bayésien (Pearl, 1989) sur l'ensemble de variables $\mathcal{X}$ est une paire $(\mathcal{G}, \Theta)$ où*

– *$\mathcal{G}$ est un graphe orienté acyclique (ou DAG pour «* directed acyclic graph *») dont les sommets sont les variables de $\mathcal{X}$; dans la suite, pour une variable X, $\mathbf{Pa}(X)$ dénote l'ensemble des* parents *de X dans $\mathcal{G}$ (c'est-à-dire l'ensemble des variables Y telles qu'il y a un arc de Y vers X) ;*

– *Θ est un ensemble de lois de probabilités conditionnelles : pour chaque variable $X \in \mathcal{X}$, $\Theta(X \mid \mathbf{Pa}(X))$ dénote la loi de probabilité de X connaissant les valeurs des parents de X. Si toutes les variables sont discrètes, alors pour tout X, $\Theta(X \mid \mathbf{Pa}(X))$ est une table de probabilités conditionnelles. De plus, si $x \in \underline{X}$, $\mathbf{u} \in \underline{\mathbf{Pa}(X)}$, alors $\Theta(x \mid \mathbf{u})$ dénote la probabilité que X prenne la valeur x quand $\mathbf{u}$ est le vecteur de valeurs affectées aux parents de X.*

Un réseau bayésien $(\mathcal{G}, \Theta)$ définit de manière unique une loi de probabilité $p_{(\mathcal{G},\Theta)}$ sur $\underline{\mathcal{X}}$. Dans notre cas, il représente la distribution de probabilité sur les configurations complètes. La probabilité d'une configuration complète $\mathbf{o} \in \underline{\mathcal{X}}$ est :

$$p_{(\mathcal{G},\Theta)}(\mathbf{o}) = \prod_{X \in \mathcal{X}} \Theta(\mathbf{o}[X] \mid \mathbf{o}[\mathbf{Pa}(X)]) \tag{1}$$

Dans la suite de cet article, lorsque ça ne prêtera pas à confusion, nous omettrons l'indice indiquant le réseau bayésien et noterons simplement p la loi de probabilité définie par $(\mathcal{G}, \Theta)$.

EXEMPLE 2. — Un exemple de réseau bayésien est représenté sur la figure 1. La probabilité de la configuration $abcdef$ est donnée par :

$$p(abcdef) = p(a) \times p(c \mid a) \times p(e \mid c) \times p(f) \times p(d \mid cf) \times p(b \mid ad)$$

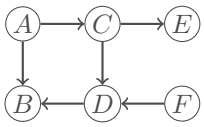


Figure 1. Un exemple de réseau bayésien

□

Les réseaux bayésiens permettent cette représentation compacte d'une loi de probabilité sur un espace combinatoire car ils représentent implicitement de nombreuses indépendances conditionnelles entre certaines variables ou groupes de variables.

DÉFINITION 3. — *Étant donnée une distribution de probabilité p sur $\mathcal{X}$, et trois sous-ensembles disjoints* $\mathbf{U}$, $\mathbf{V}$ *et* $\mathbf{Z}$ *de* $\mathcal{X}$, *on dit qu'il y a indépendance conditionnelle entre* $\mathbf{U}$ *et* $\mathbf{V}$ *sachant* $\mathbf{Z}$, *ce qu'on note* $\mathbf{U} \perp\!\!\!\perp \mathbf{V} \mid \mathbf{Z}$, *si :*

$$p(\mathbf{UV} \mid \mathbf{Z}) = p(\mathbf{U} \mid \mathbf{Z})p(\mathbf{V} \mid \mathbf{Z}),$$

c'est-à-dire si $p(\mathbf{uv} \mid \mathbf{z}) = p(\mathbf{u} \mid \mathbf{z}) \times p(\mathbf{v} \mid \mathbf{z})$ *pour toutes instanciations* $\mathbf{u}$, $\mathbf{v}$ *et* $\mathbf{z}$ *de* $\mathbf{U}$, $\mathbf{V}$ *et* $\mathbf{Z}$.

De telles indépendances probabilistes sont représentées de manière implicite dans un réseau bayésien :

PROPOSITION 4. — *Étant donné un réseau bayésien* $(\mathcal{G}, \Theta)$ *sur* $\mathcal{X}$, *toute variable* X *est indépendante de ses non-descendants sachant ses parents. Formellement, si* $\mathbf{N}$ *désigne l'ensemble des variables* Y *telles qu'il n'y a pas de chemin dans* $\mathcal{G}$ *de* X *à* Y, *alors* $X \perp\!\!\!\perp \mathbf{N} \mid \mathbf{Pa}(X)$.

C'est l'exploitation de ces indépendances conditionnelles qui a fait le succès des réseaux bayésiens, puisqu'elles permettent de simplifier des calculs de probabilités marginales, et peuvent être détectées par des tests statistiques lorsqu'on veut apprendre un réseau bayésien à partir d'un jeu de données.

3.2. Apprentissage de réseaux bayésiens

L'apprentissage d'un réseau bayésien à partir de données se fait en deux étapes : d'abord trouver la structure du réseau, c'est-à-dire le DAG qui sous-tend le réseau bayésien, puis trouver ses paramètres, c'est-à-dire les distributions de probabilité conditionnelles. Lors des deux phases, on cherche à maximiser la vraisemblance du jeu de données d'apprentissage.

Étant donné qu'apprendre le réseau bayésien qui maximise la vraisemblance d'un jeu de données est un problème NP-difficile (Chickering, 1995), des stratégies heuristiques ont dû être trouvées. Il y a principalement deux familles d'approches dans l'apprentissage de la structure du réseau : celles basées sur un score et celles dites de recherche sous contraintes.

Les premières recherchent la structure d'un réseau qui maximise un score qui mesure à quel point le réseau explique bien les données (Cooper, Herskovits, 1991). Le score prend également en compte la complexité du réseau afin de limiter le surapprentissage. Ce score peut être une fonction bayésienne, comme le score bayésien de Dirichlet (Cooper, Herskovits, 1992) ou peut provenir de la théorie de l'information, comme le critère AIC (*Akaike Information Criterion* (Akaike, 1998)).

La seconde famille d'approches se base sur des calculs d'indépendances conditionnelles. En effet, un réseau bayésien mémorise des indépendances conditionnelles de la distribution de probabilité qu'il représente ; réciproquement, on peut estimer des indépendances conditionnelles sur un jeu de données d'apprentissage et en déduire des informations sur la structure du réseau bayésien. Un exemple précurseur de cette approche est *Inductive-Causation* de Pearl (Verma, Pearl, 1990) ; une méthode plus récente est *Sparse Candidate* (Spirtes *et al*., 2000).

Une fois la structure apprise, l'apprentissage des tables de probabilités conditionnelles est simple ; la aussi, l'apprentissage vise à maximiser la vraisemblance des données.

En général, le réseau bayésien appris représente une approximation de la loi du jeu de données initial, et ce pour plusieurs raisons :

– afin de limiter le surapprentissage, le graphe a une complexité limitée ; on veille, par exemple, à limiter le nombre de parents qu'a un nœud. Cette limitation restreint le pouvoir de représentation du réseau et peut entraîner une perte d'information ;

– les tables de probabilités conditionnelles sont estimées à partir de données et sont donc de mauvaises estimations s'il n'y a pas assez de données.

3.3. Inférence dans un réseau bayésien

Un réseau bayésien $(\mathcal{G}, \Theta)$ étant donné, lorsqu'on souhaite calculer la probabilité d'une configuration partielle $\mathbf{q}$, une solution naïve est d'additionner les probabilités de toutes les configurations complètes compatibles avec $\mathbf{q}$:

$$p(\mathbf{q}) = \sum_{\mathbf{o} \sim \mathbf{q}} p(\mathbf{o}) = \sum_{\mathbf{o} \sim \mathbf{q}} \prod_{X \in \mathcal{X}} \Theta(\mathbf{o}[X] \mid \mathbf{o}[\mathbf{Pa}(X)]) \qquad (2)$$

Dans cette équation, pour une taille de $\mathbf{q}$ fixée, le nombre de configurations $\mathbf{o}$ dont il faut additionner les probabilités est une fonction exponentielle du nombre de variables. Calculer une loi marginale à partir d'un réseau bayésien est connu pour être NP-difficile (Dagum, Luby, 1993). Heureusement, des méthodes efficaces et rapides existent.

Des méthodes d'inférence exacte, comme *Variable Elimination* (N. L. Zhang, Poole, 1994), *Value Elimination* (Bacchus *et al*., 2002), *Jointree algorithms* (Lauritzen, Spiegelhalter, 1988), *Cutset Conditioning* (Pearl, 1985), *Recursive Conditioning* (Darwiche, 2001) exploitent les indépendances représentées dans le réseau pour découper cette

somme-produit en sous-sommes et en sous-produits. *Variable Elimination* et les méthodes *Jointree* sont coûteuses en espace tandis que *Recursive Conditioning* permet une inférence qui peut être polynomiale en espace. Même si ces méthodes visent une tâche NP-difficile, les algorithmes sont assez efficaces sur des jeux de données réels pour permettre une utilisation en ligne.

Il existe également des méthodes d'inférence approchées qui se répartissent en deux classes, basées l'une sur la propagation de croyance (voir par exemple (Kim, Pearl, 1983)) et l'autre sur l'échantillonnage stochastique (cf. (Lemmer, Kanal, 2014)). Cette dernière utilise des méthodes de Monte-Carlo, qui tirent des affectations complètes de $\mathcal{X}$ selon la loi du réseau bayésien (ce qui est facile) et utilisent ces affectations pour estimer des probabilités.

3.4. Application à la recommandation en configuration

Étant donné que nous disposons d'un historique de vente, qu'on suppose représentatif d'une distribution de probabilité sur $\underline{\mathcal{X}}$ correspondant à la probabilité qu'une configuration donnée soit choisie par un utilisateur, nous avons choisi d'explorer la possibilité d'utiliser le formalisme des réseaux bayésiens pour calculer, à chaque étape de la configuration, la valeur la plus probable de la prochaine variable à instancier.

La première étape consiste donc à apprendre, hors ligne, en utilisant des méthodes classiques décrites ci-dessus, un réseau bayésien qui approxime la loi de probabilité dont l'historique de vente est un échantillon – nous ne reviendrons pas sur ce point.

La seconde étape consiste à calculer la loi marginale *a posteriori* de Next connaissant $\mathbf{assigned}$, qu'on note $p(\mathrm{Next} \mid \mathbf{assigned})$ (il s'agit d'une fonction définie sur $\underline{\mathrm{Next}}$).

Avec un réseau bayésien, il est plus aisé de calculer une marginale jointe comme $p(\mathrm{Next}, \mathbf{assigned})$. On va donc exprimer la marginale *a posteriori* en fonction de la marginale jointe :

$$p(\mathrm{Next} \mid \mathbf{assigned}) = \frac{p(\mathrm{Next}, \mathbf{assigned})}{\sum_{\mathbf{x} \in \underline{\mathrm{Next}}} p(x, \mathbf{assigned})}$$

En fait, la marginale *a posteriori* $p(\mathrm{Next} \mid \mathbf{assigned})$ n'est qu'une normalisation de la marginale jointe $p(\mathrm{Next}, \mathbf{assigned})$. Comme on ne s'intéresse qu'à l'ordre des probabilités sur les valeurs de Next, et non à la valeur précise de ces probabilités, on se contentera par la suite du calcul de la probabilité marginale $p(\mathrm{Next}, \mathbf{assigned})$.

Les méthodes classiques d'inférence mentionnées ci-dessus peuvent être utilisées pour ces calculs de probabilités marginales. Nous proposons en section 4 une autre approche, qui n'utilise que les indépendances impliquées par la structure du réseau bayésien appris : les probabilités elles-mêmes seront directement estimées à partir de l'historique de vente. Notre approche est basée sur un algorithme particulier d'inférence, que nous rappelons dans la prochaine section.

3.5. Recursive conditioning

L'équation 2 montre que calculer une probabilité marginale revient à additionner des produits de probabilités conditionnelles provenant des tables d'un réseau bayésien. En utilisant des indépendances, il est possible de factoriser et décomposer en partie ce calcul en calculs de *facteurs* plus petits.

DÉFINITION 5. — *Étant donné un réseau bayésien* $(\mathcal{G}, \Theta)$ *sur* $\mathcal{X}$*, et deux ensembles de variables* $\mathbf{U}, \mathbf{V} \subseteq \mathcal{X}$ *tels que* $\mathbf{Pa}(\mathbf{V}) \subseteq \mathbf{U} \cup \mathbf{V}$*,* $\mathbf{u} \in \underline{\mathbf{U}}$*, on définit le facteur* $F_{\mathbf{V}}(\mathbf{u})$ *de la manière suivante :*

$$F_{\mathbf{V}}(\mathbf{u}) = \sum_{\substack{\mathbf{v} \in \underline{\mathbf{V}} \\ \mathbf{v} \sim \mathbf{u}}} \prod_{X \in \mathbf{V}} \Theta(\mathbf{v}[X] \mid \mathbf{uv}[\mathbf{Pa}(X)])$$

Autrement dit, $F_{\mathbf{V}}(\mathbf{u})$ est le produit des probabilités conditionnelles provenant des tables de probabilités conditionnelles du réseau bayésien quand il est réduit aux sommets $\mathbf{V}$. Notons que $F_{\mathbf{V}}(\mathbf{U})$ n'est pas nécessairement une distribution de probabilité sur $\mathbf{U}$, car la somme des facteurs pour toutes les affectations $\mathbf{u}$ de $\mathbf{U}$ ne vaut pas nécessairement 1. Bien sûr, si p est la distribution de probabilité induite par $(\mathcal{G}, \Theta)$, on a $p(\mathbf{u}) = F_{\mathcal{X}}(\mathbf{u})$.

Pour calculer une probabilité marginale $p(\mathbf{u})$, l'algorithme *Recursive Conditioning* (Darwiche, 2001),dénoté RC dans la suite, utilise le conditionnement de certaines variables du réseau bayésien pour « couper » celui-ci en deux réseaux indépendants et décomposer ainsi le facteur $F_{\mathcal{X}}(\mathbf{u})$ en deux facteurs portant sur des ensembles plus petits et pouvant être calculés indépendamment. Ces deux facteurs pourront eux-même être décomposés récursivement, et ainsi de suite. C'est le fait que l'algorithme conditionne une partie du réseau bayésien pour pouvoir le couper et recommence récursivement qui lui a donné son nom : *Recursive Conditioning*. Il est basé sur la propriété suivante, implicite dans (Darwiche, 2001). [1]

PROPOSITION 6. — *Soit un réseau bayésien* $(\mathcal{G}, \Theta)$ *sur* $\mathcal{X}$*, et deux parties* $\mathbf{U}, \mathbf{V}$ *de* $\mathcal{X}$ *telles que* $\mathbf{Pa}(\mathbf{V}) \subseteq \mathbf{U} \cup \mathbf{V}$*. Soit* $\{\mathbf{V}_1, \mathbf{V}_2\}$ *une partition de* $\mathbf{V}$*, et soit* $\mathbf{C} = (\mathbf{Pa}(\mathbf{V}_1) \cap \mathbf{V}_2) \cup (\mathbf{Pa}(\mathbf{V}_2) \cap \mathbf{V}_1)$*, alors, si* $\mathbf{u} \in \underline{\mathbf{U}}$ *on a :*

$$F_{\mathbf{V}}(\mathbf{u}) = \sum_{\substack{\mathbf{c} \in \underline{\mathbf{C}} \\ \mathbf{c} \sim \mathbf{u}}} F_{\mathbf{V}_1}(\mathbf{u}[\mathbf{V}_1 \cup \mathbf{Pa}(\mathbf{V}_1)]\mathbf{c}) \times F_{\mathbf{V}_2}(\mathbf{u}[\mathbf{V}_2 \cup \mathbf{Pa}(\mathbf{V}_2)]\mathbf{c}) \qquad (3)$$

L'intérêt de cette décomposition de $F_{\mathbf{V}}(\mathbf{u})$ en sous-facteurs repose sur le fait que les variables de $\mathbf{V}_1 \setminus \mathbf{C}$ (resp. $\mathbf{V}_2 \setminus \mathbf{C}$) n'apparaissent pas dans le calcul de $F_{\mathbf{V}_2}$ (resp. $F_{\mathbf{V}_1}$), ce qui permet un allègement des calculs.

---

1. (Darwiche, 2001) utilise cette décomposition, mais utilise la notation Pr à la place de F ; toutefois, cette notation semble ambiguë, car les facteurs ne sont pas normalisés, et ne sont donc pas réellement des probabilités.

EXEMPLE 7. — Reprenons le réseau bayésien de l'exemple 2 et cherchons à calculer $p(bf)$. En utilisant l'équation 2 on aurait une somme de 16 produits de 6 probabilités élémentaires chacun :

$$p(bf) = F_{\mathcal{X}}(bf) = \sum_{acde} \Theta(a)\Theta(b \mid ad)\Theta(c \mid a)\Theta(d \mid cf)\Theta(e \mid c)\Theta(f)$$

En séparant $\mathcal{X}$ en deux parties $\{A, B, C\}$ et $\{D, E, F\}$, comme D est parent de B, et C est parent de E et D, on a :

$$p(bf) = F_{\mathcal{X}}(bf) = \sum_{cd} F_{\{A,B,C\}}(bcd) \times F_{\{D,E,F\}}(cdf)$$

De même, on peut décomposer $\{A, B, C\}$ en deux parties $\{A, B\}$ et $\{C\}$, où A est parent de C; et $\{D, E, F\}$ en $\{D\}$ et $\{E, F\}$, où F, déjà instancié, est parent de D. On a donc $F_{\{A,B,C\}}(bcd) = \sum_a F_{\{AB\}}(abd)F_{\{C\}}(ac)$ et $F_{\{D,E,F\}}(cdf) = F_{\{D\}}(cdf)F_{\{E,F\}}(cef)$. Finalement, la décomposition entièrement développée est la suivante :

$$p(bf) = \sum_{cd} \big(\sum_{a} \Theta(a)\Theta(b \mid ad)\Theta(c \mid a) \big) \times \Theta(d \mid cf)\Theta(f) \sum_{e} \Theta(e \mid c)$$

Il y a donc $2^{|\{C,D\}|}(2^{|\{A\}|} \times 3 + 2) = 32$ consultations des tables de probabilité conditionnelles Θ (sans consultation de la table pour E, puisque $\sum_e \Theta(e \mid c) = 1$). □

On peut remarquer qu'il existe plusieurs partitions possibles pour décomposer un facteur en sous-facteurs, et que toutes ne se valent pas : en effet, RC itère sur les instanciations d'un ensemble de variables $\mathbf{C}$ permettant de rendre les deux parties indépendantes. (Darwiche, 2001) propose d'éviter de calculer la décomposition à chaque inférence, mais plutôt de la planifier une fois pour toute – indépendamment des requêtes – en construisant un arbre de décomposition (appelé *dtree*). Un tel arbre de décomposition pour le réseau bayésien de l'exemple 2 est dessiné sur la figure 2 : à chaque nœud est associé un ensemble de variables ; si cet ensemble n'est pas un singleton, cet ensemble est partitionné en 2, et le nœud a deux enfants ; si cet ensemble est un singleton, c'est qu'on peut faire un appel à la table de probabilités conditionnelles correspondante. La figure montre aussi, sous chaque nœud binaire, l'ensemble de variables $\mathbf{C}$ de la proposition 6, appelé *cutset*. (Darwiche, 2001) donne plusieurs algorithmes permettant de calculer des *dtree* équilibrés tout en essayant de minimiser la taille des ensembles $\mathbf{C}$ à chaque étape, et étudie les gains en complexité en temps qu'on peut obtenir par l'utilisation d'un cache, pour éviter de refaire plusieurs fois les mêmes calculs lors du calcul d'une probabilité. L'algorithme 1 implémente l'approche de (Darwiche, 2001).

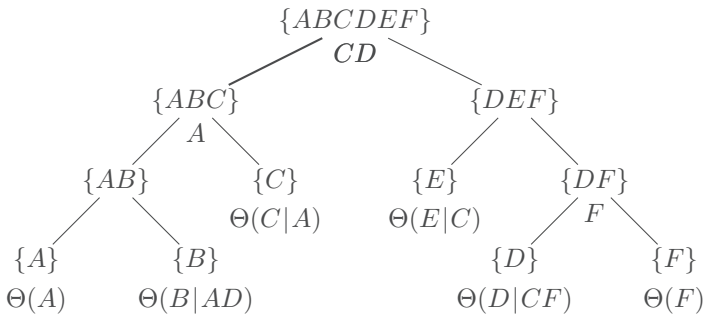


Figure 2. Un dtree *possible pour le réseau bayésien de l'exemple 2. Pour chaque nœud T, les variables au-dessus du nœud correspondent à $\mathbf{V}_T$. Si T est un nœud interne, les variables en-dessous correspondent au cutset de T. Si T est une feuille, il y a en-dessous la table de probabilités associée à la variable de T*

Algorithme 1 : RC

Input : T : un nœud du *dtree*, $\mathbf{u}$ une configuration partielle
Output : $F_{\mathbf{V}_T}(\mathbf{u})$
Algorithm RC (T, $\mathbf{u}$)

```
1  if facteur_cache(u) n'est pas défini then
2      if T est une feuille then facteur_cache(u) ← LookUp (T, u)
3      else
4          p ← 0
5          C ← le cutset de T
6          T1, T2 ← les deux enfants de T
7          for each c ∈ C, c ∼ u do
8              p ← p + RC(T1, uc) × RC(T2, uc)
9          facteur_cache(u) ← p
10 return facteur_cache(u)
```

Function LookUp (T, $\mathbf{u}$)

```
1  Θ_{X|Pa(X)} ← la table de probabilités conditionelles associée à T
2  if X ∈ U then return Θ_{X|Pa(X)}(u[X] | u[Pa(X)])
3  else return 1
```

4. Estimations directes de probabilités

4.1. Motivation

Lors d'un calcul de marginale avec peu de variables instanciées, l'algorithme *Recursive Conditioning* parcourt tous les nœuds du *dtree* en conditionnant à chaque fois un cutset.

Or, dans notre contexte, nous disposons d'un historique de ventes sur lequel le réseau bayésien a été appris. Cela signifie que pour calculer $p(bf)$ (cf. l'exemple 7), le plus simple n'est pas de conditionner récursivement comme le fait RC mais plutôt d'utiliser directement l'historique de ventes pour estimer la fréquence de bf dans les produits vendus. On aura une bonne précision si $\#(bf)$ est suffisamment grand.

Nous pouvons amener cette idée plus loin. L'algorithme RC découpe en deux l'ensemble des variables et travaille sur chacun d'eux, donc la cardinalité de l'ensemble des variables conditionnées va généralement baisser au fur et à mesure que l'algorithme ira profondément dans les branches du *dtree* (même en comptant les variables conditionnées du cutset qui s'y ajoutent). La proportion du jeu de données qui correspondra aux instanciations de ces variables conditionnées va augmenter par la même occasion. Cela signifie que plus un calcul est bas dans le *dtree*, plus il y a de chances pour que le jeu de données soit suffisant pour fournir une estimation du résultat. Ainsi, on pourra estimer grâce à l'historique de ventes des probabilités intermédiaires, évitant ainsi de descendre dans toutes les branches du *dtree*. Nous appelons cet algorithme, qui est une variation de RC qui exploite directement l'historique, *Direct Recursive Conditioning* (DRC).

On peut donc être tenté de transposer RC en un nouvel algorithme qui effectuerait les mêmes calculs que RC mais qui, dès que ce serait possible, utiliserait l'historique pour estimer directement des calculs intermédiaires. Malheureusement, cette démarche rencontre un obstacle : les calculs intermédiaires de RC ne peuvent pas être estimés directement à partir de l'historique. En effet, les calculs intermédiaires de RC sont des facteurs qui, le plus souvent, ne sont pas des distributions de probabilité. Or, à partir du jeu de données on ne peut estimer facilement que des probabilités et pas n'importe quel facteur.

DRC est donc basé sur deux idées maîtresses : quand c'est possible utiliser directement le jeu de données pour estimer les probabilités et, comme on le fait dans RC, conditionner pour simplifier les calculs, mais en adaptant cette idée de manière à ce que toutes les grandeurs intermédiaires soient des probabilités. Si on veut que tous les calculs intermédiaires de DRC soient des probabilités, cela nous empêche d'utiliser la même décomposition du calcul que RC qui n'est valide que pour des facteurs. Nous utilisons donc les indépendances conditionnelles que nous avons évoquées plus tôt.

4.2. Calcul d'indépendances conditionnelles

La détermination des indépendances conditionnelles peut se faire grâce à un outil : le graphe moralisé (Dawid *et al.*, 2007).

DÉFINITION 8. — *Le graphe moralisé (ou simplement graphe moral) $\mathcal{G}_m$ de $\mathcal{G}$ est un graphe non-orienté ayant les mêmes sommets que $\mathcal{G}$ et dans lequel il y a une arête entre A et B ssi l'une de ces conditions (au moins) est vérifiée :*

- *A est le parent de B dans $\mathcal{G}$;*
- *B est le parent de A dans $\mathcal{G}$;*

– *A et B ont un fils commun dans $\mathcal{G}$;*

Par exemple, le graphe moralisé du réseau bayésien de l'exemple 2 est présenté en figure 3.

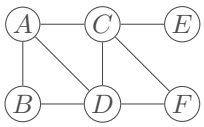


Figure 3. Le graphe moral correspondant au réseau bayésien de l'exemple 2

DÉFINITION 9. — *Dans un graphe non-orienté $\mathcal{G}_m$, on dit que $\mathbf{A}$ est séparé de $\mathbf{B}$ par $\mathbf{C}$ (ou que $\mathbf{C}$ est un séparateur pour $\mathbf{A}$ et $\mathbf{B}$) si, dans le graphe induit par $\mathcal{G}_m \setminus \mathbf{C}$, il n'existe aucun chemin entre $\mathbf{A}$ et $\mathbf{B}$.*

On note par $\mathrm{Anc}(\mathbf{U})$ l'ensemble des ancêtres de $\mathbf{U}$, y compris $\mathbf{U}$. On note $\mathcal{G}_\mathbf{U}$ la restriction du graphe $\mathcal{G}$ sur l'ensemble de ses nœuds $\mathbf{U}$.

PROPOSITION 10. — *(Cowell, 2006)*

Supposons que p soit strictement positive. Alors $\mathbf{U} \perp\!\!\!\perp \mathbf{V} \mid \mathbf{Z}$ si et seulement si, dans le graphe moralisé de $\mathcal{G}_{\mathrm{Anc}(\mathbf{UVZ})}$, $\mathbf{U}$ est séparé de $\mathbf{V}$ par $\mathbf{Z}$.

C'est pour cela que nous travaillerons dans DRC avec des séparateurs : ils nous permettront d'exploiter les indépendances conditionnelles qu'ils impliquent.

4.3. Direct Recursive Conditioning

Comme RC, l'algorithme DRC va récursivement décomposer un calcul jusqu'à atteindre des termes à estimer sur $\mathcal{H}$. Chacun des termes de la décomposition sera une probabilité jointe, qui pourra, s'il y a assez d'exemples compatibles dans le jeu de données, être estimée à partir de l'historique. La décomposition de DRC s'appuie sur la proposition suivante :

PROPOSITION 11. — *Soit un réseau bayésien $\mathcal{G}$ et $\mathbf{U}$ un ensemble de variables. Soit $\{\mathbf{G}_1, \mathbf{G}_2, \mathbf{C}\}$ une partition du graphe moralisé de $\mathcal{G}_{\mathrm{Anc}(\mathbf{U})}$. Si $\mathbf{C}$ sépare $\mathbf{G}_1$ de $\mathbf{G}_2$, alors pour toute configuration partielle $\mathbf{u}$:*

$$p(\mathbf{u}) = \sum_{\substack{\mathbf{c} \in \underline{\mathbf{C}} \\ \mathbf{c} \sim \mathbf{u}}} \frac{p(\mathbf{u}[\mathbf{G}_1]\mathbf{c})\, p(\mathbf{u}[\mathbf{G}_2]\mathbf{c})}{p(\mathbf{c})} \tag{4}$$

Le lecteur pourra trouver les preuves de cette proposition et de celles qui suivent en annexe, à la fin de l'article.

Nous souhaitons souligner deux différences importantes entre la décomposition de RC (équation (3)) et celle de DRC (équation (4)).

La première différence est que DRC décompose récursivement un terme en trois sous-termes, alors que RC décompose un terme en deux sous-termes. Intuitivement, on peut donc attendre de DRC qu'il conduise à une décomposition nécessitant plus de calculs.

Néanmoins, ce point est contrebalancé par la seconde différence entre ces deux décompositions : RC doit nécessairement itérer sur les valeurs du cutset alors que ce n'est pas toujours le cas pour DRC. En effet, si on suppose dans l'équation (4) que $\mathbf{U} \subseteq (\mathbf{G}_1 \cup \mathbf{C})$ (c'est-à-dire si $\mathbf{u}[\mathbf{G}_2]$ est vide), alors le calcul se réduit à :

$$p(\mathbf{u}) = \sum_{\substack{\mathbf{c} \in \mathbf{C} \\ \mathbf{c} \sim \mathbf{u}}} \frac{p(\mathbf{u}[\mathbf{G}_1]\mathbf{c})\, p(\mathbf{c})}{p(\mathbf{c})} = \sum_{\substack{\mathbf{c} \in \mathbf{C} \\ \mathbf{c} \sim \mathbf{u}}} p(\mathbf{u}[\mathbf{G}_1]\mathbf{c}) = p(\mathbf{u}[\mathbf{G}_1])$$

Ainsi, à chaque décomposition, DRC peut soit décomposer un terme en trois sous-termes, soit passer directement à la décomposition suivante.

Enfin, comme voulu, DRC peut arrêter son inférence tôt. Chaque appel de DRC vise à calculer une probabilité jointe $p(\mathbf{u})$. Or, nous rappelons que nous disposons d'un historique de vente $\mathcal{H}$. Si $\#(\mathbf{u})$ est assez grand, on peut facilement estimer $p(\mathbf{u})$ par :

$$p(\mathbf{u}) \simeq \frac{\#(\mathbf{u})}{|\mathcal{H}|}$$

Cette estimation sera fiable si $\mathbf{U}$ est assez petit. Afin de savoir quand il est possible d'estimer les probabilités à partir des données et quand ce n'est pas possible, on fixe un seuil s. Si $\#(\mathbf{u}) > s$, alors nous estimons $p(\mathbf{u})$ par comptage direct. Si $\#(\mathbf{u})$ est trop petit, DRC décompose le calcul comme indiqué par l'équation (4).

Il peut arriver que DRC ne puisse décomposer $\mathbf{U}$ et soit obligé d'estimer $p(\mathbf{u})$ avec l'historique même si $\#(\mathbf{u})$ est faible. En particulier, si $\#(\mathbf{u}) = 0$, alors $p(\mathbf{u})$ sera estimé à 0 ce qui peut être problématique. Une solution classique à ce problème est d'utiliser une distribution bêta comme loi a priori lors de l'estimation. En notant k l'hyperparamètre de la loi bêta appelé « taille d'échantillon équivalente », l'estimation de $p(\mathbf{u})$ devient :

$$p(\mathbf{u}) \simeq \frac{\#(\mathbf{u}) + \frac{k}{|\mathbf{U}|}}{|\mathcal{H}| + k}$$

Plus $\#(\mathbf{u})$ est petit, plus le terme de correction $\frac{k}{|\mathbf{U}|}$ devient prépondérant.

De la même manière que Darwiche a introduit les *dtree* afin de séparer l'apprentissage de la décomposition et l'inférence, nous allons introduire un principe graphique de décomposition. Étant donné que DRC décompose un calcul en trois sous-calculs, il s'agit d'un arbre ternaire, et pas binaire comme le *dtree* de RC.

DÉFINITION 12. — *Un arbre de décomposition ternaire d'un réseau bayésien $\mathcal{G}$ est un arbre ternaire fini dont chaque feuille correspond à un nœud et à ses parents.*

À chaque nœud T d'un arbre de décomposition ternaire est associé un ensemble de variables $\mathbf{V}_T$. De plus, si T est un nœud interne, on associe à T une partition $\{\mathbf{G}_{1,T}, \mathbf{G}_{2,T}, \mathbf{C}_T\}$ du graphe moralisé de $\mathcal{G}_{\mathrm{Anc}(\mathbf{V}_T)}$ telle que $\mathbf{C}_T$ sépare $\mathbf{G}_{1,T}$ et $\mathbf{G}_{2,T}$.

Le nœud T vérifie les propriétés suivantes :

- *Si T est la racine de l'arbre alors, $\mathbf{V}_T = \mathcal{X}$.*
- *Si T est un nœud interne avec pour enfants T_1, T_2 et T_C, alors :*
  - *l'ensemble associé à T_1 est $\mathbf{C}_T \cup \mathbf{G}_{1,T}$*
  - *l'ensemble associé à T_2 est $\mathbf{C}_T \cup \mathbf{G}_{2,T}$*
  - *l'ensemble associé à T_C est $\mathbf{C}_T$*
- *Si T est une feuille, alors $\mathbf{V}_T$ est composé d'un nœud et de ses parents.*

Algorithme 2 : DRC

Input : T un nœud d'un arbre de décomposition ternaire
$\mathbf{u}$ une affectation complète ou partielle
Output : $p(\mathbf{u})$ la probabilité de $\mathbf{u}$
Algorithme DRC (T, $\mathbf{u}$)

```
1  if U = ∅ then return 1
2  else
3      if proba_cache(u) n'est pas défini then
4          T_1, T_2, T_C ← les trois enfants de T
5          if #(u) > s ou T est une feuille then
6              proba_cache(u) ← (#(u) + k/|U|)/(|H| + k)
7          else if U ⊆ V_{T_C} then proba_cache(u) ← DRC(T_C, u)
8          else if U ⊆ V_{T_1} then proba_cache(u) ← DRC(T_1, u)
9          else if U ⊆ V_{T_2} then proba_cache(u) ← DRC(T_2, u)
10         else
11             p ← 0
12             for each c ∈ C_T, c ∼ u do
13                 p ← p + DRC(T_1, u[G_{1,T}]c) × DRC(T_2, u[G_{2,T}]c)
                       ÷ DRC(T_C, c)
14             proba_cache(u) ← p
15     return proba_cache(u)
```

DRC est présenté dans l'algorithme 2. La ligne 6 est le cas terminal : on estime la probabilité si T est une feuille ou si $\#(\mathbf{u})$ est assez grand.

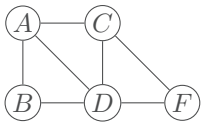
Les lignes 7, 8 et 9 correspondent au cas où la décomposition n'est pas nécessaire car un des deux ensembles $\mathbf{u}[\mathbf{G}_1]$ ou $\mathbf{u}[\mathbf{G}_2]$ est vide.

Étant donné que DRC itère sur un arbre fini, l'algorithme termine. Le calcul de DRC est basé sur la proposition 11 et renvoie une estimation de $p(\mathbf{u})$.

Nous nous sommes inspirés du mécanisme de cache de RC : dès que DRC souhaite calculer une probabilité, il vérifie qu'il ne l'a pas déjà calculée plus tôt (ligne 3). Si c'est le cas, la probabilité est présente dans $proba\_cache$.

Afin de compter rapidement dans l'historique de ventes (lignes 5 et 6), nous avons compilé celui-ci sous la forme d'un *Algebraic Decision Diagram* (Bahar *et al.*, 1997). Cette compilation qui peut compresser l'historique d'un ordre de grandeur, accélère les calculs par rapport à une approche plus naïve.

EXEMPLE 13. — Reprenons le réseau bayésien de l'exemple 2 et cherchons à calculer $p(bf)$. Le graphe moral de $\mathcal{G}_{\mathrm{Anc}(\{B,F\})}$ est :



E est ici la seule variable qui n'a pas de descendant dans $\mathbf{U} = \{B, F\}$. Nous utilisons la partition suivante : $\mathbf{G}_1 = \{A, B\}$, $\mathbf{C} = \{C, D\}$, $\mathbf{G}_2 = \{F\}$. Ceci nous amène à la décomposition suivante :

$$p(bf) = \sum_{cd} \frac{p(bcd)p(cdf)}{p(cd)}$$

Si nous appliquons récursivement cette décomposition, nous aboutissons à :

$$p(bf) = \sum_{cd} \frac{\sum_a \frac{p(abd)p(acd)}{p(ad)} p(cdf)}{p(cd)}$$

Nous pouvons remarquer que la décomposition est finie car toutes les probabilités restantes font intervenir des nœuds et certains de leurs parents.

Si on descend jusqu'aux feuilles de l'arbre, il y a $2^{|\{C,D\}|} \times (2 + 2^{|\{A\}|} \times 3) = 32$ estimations de probabilités, c'est-à-dire autant que de consultations de probabilités conditionnelles pour RC. De plus, notre approche nous permet un raccourci puissant : l'utilisation, à n'importe quelle étape, du comptage dans le jeu de données.

Si par exemple $p(bcd)$ peut être estimé directement depuis l'historique, alors la première décomposition suffit, ce qui nous conduirait à $2^{|\{C,D\}|} \times 3 = 12$ estimations. De plus, si $p(bf)$ est estimable directement, alors aucun décomposition n'est nécessaire.

Néanmoins, la consultation de la table de probabilité comme le fait RC et l'estimation d'une probabilité à partir de l'historique n'a pas le même coût : il est bien

plus rapide de consulter la table car, dans le cas des réseaux bayésiens, les tables de probabilités conditionnelles sont calculées une fois pour toute à l'apprentissage, donc avant l'inférence ; alors que dans notre cas, il faut ré-estimer ces probabilités pendant l'inférence. □

Précisons que le fait d'estimer, pendant l'inférence, les probabilités à partir du jeu de données initial ne constitue pas, à nos yeux, un *apprentissage*. L'objectif d'un apprentissage est d'obtenir un modèle pouvant expliquer les données observées. Or, l'inférence de DRC n'apprend aucun modèle. Nous rapprochons plutôt nos estimations de probabilités à de simples calculs intermédiaires qui existent également dans les autres algorithmes d'inférence, et qui ne sont pas conservés une fois l'inférence effectuée.

Nous tenons également à éclairer la différence entre notre approche et celle des algorithmes d'inférence approchée par échantillonnage stochastique (Lemmer, Kanal, 2014). Comme ces algorithmes DRC utilise un jeu d'affectations complètes pour estimer des probabilités. La différence majeure est que DRC utilise le jeu de donnée *initial*, celui qui a servi à apprendre le réseau bayésien, alors que les algorithmes d'inférence approchée génère (échantillonne) des données à partir du réseau bayésien. Ces méthodes approchées reposent donc sur des données qui ont subi deux détériorations : la phase d'apprentissage du réseau bayésien et l'échantillonnage, ce qui explique que ce soit seulement des méthodes *approchées*. Étant donné que DRC utilise les données initiales du problème, ce n'est pas une méthode approchée. Ceci se visualise très bien avec les résultats en termes de taux d'erreur, où DRC est aussi précis que Jointree et RC qui sont des méthodes d'inférence exacte.

Maintenant que nous savons comment utiliser un arbre de décomposition ternaire, nous devons voir comment en construire un.

4.4. Construction d'un arbre de décomposition ternaire

Afin de construire un arbre de décomposition ternaire, nous calculons récursivement les enfants d'un nœud T en trouvant une partition $\{\mathbf{G}_{1,T}, \mathbf{G}_{2,T}, \mathbf{C}_T\}$ du graphe moralisé de $\mathcal{G}_{\mathrm{Anc}(\mathbf{V}_T)}$ telle que $\mathbf{C}_T$ sépare $\mathbf{G}_{1,T}$ et $\mathbf{G}_{2,T}$.

Étant donné qu'on souhaite arrêter la décomposition lorsque $\mathbf{V}_T$ est constitué d'un nœud et de (certains de) ses parents, nous introduisons la notion d'ensemble de variables décomposable.

DÉFINITION 14. — *Un ensemble de variables* $\mathbf{V}$ *est dit* décomposable *dans* $\mathcal{G}$ *si le graphe moralisé du sous-graphe induit de* $\mathbf{V}$ *n'est pas complet.*

La construction d'un arbre de décomposition ternaire est détaillée dans l'algorithme 3. Nous supposons disposer d'une fonction *FindPartition*($\mathbf{V}$, $\mathrm{Anc}(\mathbf{V})$) qui renvoie une partition $\{\mathbf{G}_1, \mathbf{G}_2, \mathbf{C}\}$ de $\mathrm{Anc}(\mathbf{V})$ telle que $\mathbf{C}$ sépare $\mathbf{G}_1$ et $\mathbf{G}_2$. Cette fonction sera détaillée dans la prochaine sous-section.

Algorithme 3 : Construction d'un arbre de décomposition ternaire

Input : $\mathbf{V}$ un ensemble de variables
Output : un arbre de décomposition ternaire pour $\mathbf{V}$
Algorithme `ConstruitArbre(`$\mathbf{V}$`)`

```
1  V_T ← V
2  if V_T est décomposable then
3      {G_{1,T}, G_{2,T}, C_T} ← FindPartition(V_T, Anc(V_T))
4      T_1 ← ConstruitArbre(C_T ∪ G_{1,T})
5      T_2 ← ConstruitArbre(C_T ∪ G_{2,T})
6      T_C ← ConstruitArbre(C_T)
7      return nœud(T_1, T_2, T_C)
8  else return feuille(V_T)
```

Afin de permettre à l'algorithme 3 de terminer, il faut que chacun des sous-calculs « diminue » en taille en un certain sens ; si une partition permet une telle diminution, elle sera dite admissible. Avant d'introduire formellement cette notion, nous avons besoin d'introduire quelques notations.

Nous noterons par la suite $\mathbf{F}_{\mathcal{G}}(\mathbf{V})$ l'ensemble des nœuds de $\mathbf{V}$ qui n'ont pas de descendants dans $\mathcal{G}$ qui appartiennent à $\mathbf{V}$.

On rappelle que la distance $d_{\mathcal{G}_m}(X, Y)$ entre deux nœuds X et Y dans un graphe non-orienté $\mathcal{G}_m$ est la longueur du plus petit chemin qui relie X à Y dans $\mathcal{G}_m$. S'il n'existe aucun chemin entre X et Y, alors $d_{\mathcal{G}_m}(X, Y) = \infty$.

DÉFINITION 15. — *La hauteur de* $\mathbf{V}$ *dans le graphe moral* $\mathcal{G}_m$ *de* $\mathcal{G}_{\mathrm{Anc}(\mathbf{V})}$ *est la plus grande distance dans* $\mathcal{G}_m$ *entre un nœud de* $\mathbf{F}_{\mathcal{G}}(\mathbf{V})$ *et un nœud de* $\mathbf{V}$*, ce que nous notons :*

$$H_{\mathcal{G}_m}(\mathbf{V}) = \max_{\substack{X \in \mathbf{F}_{\mathcal{G}}(\mathbf{V}) \\ Y \in \mathbf{V}}} d_{\mathcal{G}_m}(X, Y)$$

DÉFINITION 16. — *Soit* $\mathcal{G}$ *un réseau bayésien,* $\mathbf{V}$ *un ensemble de variables et* $\mathcal{G}_m$ *le graphe moralisé de* $\mathcal{G}_{\mathrm{Anc}(\mathbf{V})}$*. Une partition* $\{\mathbf{G}_1, \mathbf{G}_2, \mathbf{C}\}$ *de* $\mathcal{G}_m$ *est dite* admissible *pour* $\mathbf{V}$ *si* $\mathbf{C}$ *sépare* $\mathbf{G}_1$ *et* $\mathbf{G}_2$ *et si, pour tout ensemble* $\mathbf{E} \in \{\mathbf{C} \cup (\mathbf{G}_1 \cap \mathbf{V}), \mathbf{C} \cup (\mathbf{G}_2 \cap \mathbf{V}), \mathbf{C}\}$ *:*

$$|\mathrm{Anc}(\mathbf{E})| < |\mathrm{Anc}(\mathbf{V})|)$$

ou

$$\begin{cases} |\mathrm{Anc}(\mathbf{E})| = |\mathrm{Anc}(\mathbf{V})|) \\ H_{\mathcal{G}_m}(\mathbf{E}) < H_{\mathcal{G}_m}(\mathbf{V}) \end{cases}$$

Nous pouvons donc construire un arbre de décomposition ternaire à partir du moment où nous disposons d'un moyen de construire des partitions admissibles. Et c'est justement l'objet de la sous-section qui suit.

4.5. Construction de partitions admissibles

Nous avons précédemment introduit la notion d'ensemble décomposable (définition 14). Nous allons montrer dans cette section qu'un ensemble de variables $\mathbf{V}$ est décomposable dans $\mathcal{G}$ si et seulement si $\mathbf{V}$ admet une partition admissible de $\mathcal{G}_m$ pour $\mathbf{V}$ (dans toute cette section, nous noterons $\mathcal{G}_m$ le graphe moralisé de $\mathcal{G}_{\mathrm{Anc}(\mathbf{V})}$).

Nous allons procéder en deux temps. Tout d'abord, nous allons montrer que si $\mathbf{V}$ n'est pas décomposable alors il n'existe aucune partition admissible. La seconde étape sera, à partir d'un ensemble $\mathbf{V}$, de construire une partition admissible.

En préambule, caractérisons les ensembles de variables décomposables avec les notions qui interviennent dans la définition de l'admissibilité :

LEMME 17. — *Soit un ensemble de variables $\mathbf{V}$ et un réseau bayésien $\mathcal{G}$. $\mathbf{V}$ est décomposable dans $\mathcal{G}$ si et seulement si $\mathbf{V} \neq \varnothing$ et $H_{\mathcal{G}_m}(\mathbf{V}) > 1$.*

Nous ne détaillerons pas outre mesure la première partie de notre problème, c'est-à-dire montrer qu'une ensemble de variables non décomposables n'admet pas de partition admissible de $\mathcal{G}_m$; nous rappelons que les preuves sont disponibles en annexe.

PROPOSITION 18. — *Soit $\mathbf{V}$ un ensemble de variables et $\mathcal{G}$ un réseau bayésien. Si $\mathbf{V}$ n'est pas décomposable dans $\mathcal{G}$, alors il n'existe pas de partition admissible de $\mathcal{G}_m$ pour $\mathbf{V}$.*

Nous supposons donc que par la suite $\mathbf{V}$ est décomposable dans $\mathcal{G}$, i.e. $\mathbf{V} \neq \varnothing$ et $H_{\mathcal{G}_m}(\mathbf{V}) \geq 2$ (d'après le lemme 17). Notre objectif est de construire une partition admissible de $\mathcal{G}_m$ pour $\mathbf{V}$. Nous utiliserons le lemme suivant :

LEMME 19. — *Soit $\mathcal{G}$ un réseau bayésien, $\mathbf{V}$ un ensemble de variables. Alors, pour tout ensemble $\mathbf{E} \subseteq \mathrm{Anc}(\mathbf{V})$, s'il existe une variable $X \in \mathbf{F}_{\mathcal{G}}(\mathbf{V})$ telle que $X \notin \mathbf{E}$, alors :*

$$|\mathrm{Anc}(\mathbf{E})| < |\mathrm{Anc}(\mathbf{V})|$$

Nous traitons séparément les deux cas où $|\mathbf{F}_{\mathcal{G}}(\mathbf{V})|$ vaut soit 1, soit plus.

Cas 1 : $|\mathbf{F}_{\mathcal{G}}(\mathbf{V})| = 1$

Notons alors X l'unique variable de $\mathbf{V}$ sans descendant dans $\mathbf{V}$. On définit l'ensemble $\mathbf{Z}$ des nœuds de $\mathbf{V}$ qui, dans $\mathcal{G}_m$, sont les plus éloignés de X :

$$\mathbf{Z} = \{Y \in \mathbf{V} \mid d_{\mathcal{G}_m}(X, Y) = H_{\mathcal{G}_m}(\mathbf{V})\}$$

Nous recherchons une partition telle que $X \in \mathbf{G}_1$ et $\mathbf{Z} \subseteq \mathbf{G}_2$. On peut tout de suite remarquer que, pour $\mathbf{E} \in \{\mathbf{C} \cup (\mathbf{G}_2 \cap \mathbf{V}), \mathbf{C}\}$, étant donné que $X \notin \mathbf{E}$, le lemme 19 implique que $|\mathrm{Anc}(\mathbf{E})| < |\mathrm{Anc}(\mathbf{V})|$.

Nous nous intéressons donc au troisième ensemble $\mathbf{C} \cup (\mathbf{G}_1 \cap \mathbf{V})$; nous allons faire en sorte que $H_{\mathcal{G}_m}(\mathbf{C} \cup (\mathbf{G_1} \cap \mathbf{V})) < H_{\mathcal{G}_m}(\mathbf{V})$. Pour cela, nous cherchons un séparateur $\mathbf{C}$ entre X et $\mathbf{Z}$ tel que $d_X(\mathbf{C}) < d_X(\mathbf{Z})$. Nous savons qu'un tel séparateur existe ; en effet, $\mathbf{C} = \mathbf{Pa}(X)$ est une solution possible :

– les seuls voisins de X dans $\mathcal{G}_m$ sont ses parents, donc $\mathbf{Pa}(X)$ sépare X du reste du graphe ;

– $d_X(\mathbf{Pa}(X)) < d_X(\mathbf{Z})$ car

$$d_X(\mathbf{Pa}(X)) = 1 < 2 \leq H_{\mathcal{G}_m}(\mathbf{V}) = d_X(\mathbf{Z})$$

On obtient alors $H_{\mathcal{G}_m}(\mathbf{C} \cup (\mathbf{G_1} \cap \mathbf{V})) < H_{\mathcal{G}_m}(\mathbf{V})$. Étant donné que $\mathbf{C} \cup (\mathbf{G}_1 \cap \mathbf{V}) \subseteq \mathrm{Anc}(\mathbf{V})$, $|\mathrm{Anc}(\mathbf{C} \cup (\mathbf{G}_1 \cap \mathbf{V}))| \leq |\mathrm{Anc}(\mathbf{V})|$. De plus, $H_{\mathcal{G}_m}(\mathbf{C} \cup (\mathbf{G}_1 \cap \mathbf{V})) < H_{\mathcal{G}_m}(\mathbf{V})$. Donc au final, $\{\mathbf{G}_1, \mathbf{G}_2, \mathbf{C}\}$ est admissible.

Cas 2 : $|\mathbf{F}_{\mathcal{G}}(\mathbf{V})| > 1$

Soit X et Y deux variables de $\mathbf{F}_{\mathcal{G}}(\mathbf{V})$. Soit une partition $\{\mathbf{G}_1, \mathbf{G}_2, \mathbf{C}\}$ tel que $\mathbf{C}$ sépare $\mathbf{G}_1$ de $\mathbf{G}_2$ et telle que $X \in \mathbf{G}_1$ et $Y \in \mathbf{G}_2$ (ce qui est possible car X et Y ne sont pas voisins).

On remarque alors que :

– $Y \notin \mathbf{C} \cup (\mathbf{G}_1 \cap \mathbf{V})$

– $X \notin \mathbf{C} \cup (\mathbf{G}_2 \cap \mathbf{V})$

– $X, Y \notin \mathbf{C}$.

On peut alors appliquer le lemme 19 pour chacun de ces ensembles et on peut en conclure que $\{\mathbf{G}_1, \mathbf{G}_2, \mathbf{C}\}$ est admissible.

Nous avons donc construit, pour tout ensemble de variables décomposable $\mathbf{V}$, une partition admissible. Nous pouvons donc en conclure l'équivalence entre existence d'une partition admissible et décomposabilité :

PROPOSITION 20. — *Soit* $\mathbf{V}$ *un ensemble de variables et* $\mathcal{G}$ *un réseau bayésien. Alors* $\mathbf{V}$ *est décomposable dans* $\mathcal{G}$ *si et seulement si* $\mathbf{V}$ *admet une partition admissible de* $\mathcal{G}_m$ *pour* $\mathbf{V}$.

Nous pouvons nous servir de cette preuve comme d'un algorithme de construction de partition admissible : c'est l'algorithme 4.

Nous faisons appel dans l'algorithme 4 à une fonction $VertexSeparator(\mathbf{A}, \mathbf{V}_1, \mathbf{V}_2)$ qui renvoie une partition $\{\mathbf{G}_1, \mathbf{G}_2, \mathbf{C}\}$ telle que :

– $\mathbf{C} \subseteq \mathbf{A}$;

– $\mathbf{V}_1 \subseteq \mathbf{G}_1$;

– $\mathbf{V}_2 \subseteq \mathbf{G}_2$;

– $\mathbf{G}_1$ est séparé de $\mathbf{G}_2$ par $\mathbf{C}$.

Il s'agit du problème VS (*vertex separator*), qui est NP-difficile (Bui, Jones, 1992). Dans notre implémentation, nous utilisons une réduction de VS vers HP (*Hypergraph Partitioning*) ; en effet, les deux problèmes sont duaux (E. Zhang, Gao, 2014). Cette réduction a pour but d'utiliser les heuristiques très efficaces développées pour HP.

Algorithme 4 : FindPartition

Input : $\mathcal{G}$ un réseau bayésien, $\mathbf{V}$ un ensemble de variables
Output : Si $\mathbf{V}$ est décomposable : $\{\mathbf{G}_1, \mathbf{G}_2, \mathbf{C}\}$ une partition admissible pour $\mathbf{V}$. Sinon : $\varnothing$.
Algorithme `FindPartition(`$\mathcal{G}$, $\mathbf{V}$`)`

$\mathcal{G}_m \leftarrow$ graphe moralisé de $\mathcal{G}_{\mathrm{Anc}(\mathbf{V})}$
$\mathbf{F} \leftarrow \{X$ sans descendant appartenant à $\mathbf{V}$ dans $\mathcal{G}\}$
if $|\mathbf{F}| = 0$ **then return** $\varnothing$
else if $|\mathbf{F}| = 1$ **then**
  $X \leftarrow$ l'unique variable de $\mathbf{F}$
  $d_X \leftarrow$ fonction de distance à X dans $\mathcal{G}_m$
  $l \leftarrow \max_{Y \in \mathbf{V}} d_X(Y)$
  if $l \leq 1$ **then return** $\varnothing$
  else
    $\mathbf{Z} \leftarrow \mathrm{argmax}_{Y \in \mathbf{V}} d_X(Y)$
    $\mathbf{P} \leftarrow \{Y \in \mathcal{G}_m \mid d_X(Y) < l\}$
    $\{\mathbf{G}_1, \mathbf{G}_2, \mathbf{C}\} \leftarrow$ VertexSeparator$(\mathbf{P}, \{X\}, \mathbf{Z})$
    return $\{\mathbf{G}_1, \mathbf{G}_2, \mathbf{C}\}$
else
  $X, Y \leftarrow$ deux variables de $\mathbf{F}$
  $\{\mathbf{G}_1, \mathbf{G}_2, \mathbf{C}\} \leftarrow$ VertexSeparator$(\mathrm{Anc}(\mathbf{V}), \{X\}, \{Y\})$
  return $\{\mathbf{G}_1, \mathbf{G}_2, \mathbf{C}\}$

Notre présentation de DRC est terminée. Nous avons montré comment construire un arbre de décomposition ternaire et comment réaliser l'inférence à partir de l'utilisation jointe de cet arbre et de l'historique.

5. Expérimentations

L'approche proposée dans cet article a été testée sur un case study composé de deux historiques de ventes fournis par Renault, le constructeur automobile français. Toutes les données sont disponibles à l'adresse http://www.irit.fr/~Helene.Fargier/BR4CP/benches.html. Ces jeux de données, nommés "*small*" et "*medium*", sont d'authentiques historiques de vente : chaque configuration correspond à une voiture vendue. Le jeu de données *small* a 48 variables et 27 088 configurations (27 088 véhicules configurés) ; le jeu de données *medium* a 44 variables et 14 786 configurations. La plupart des variables sont binaires, mais pas toutes.

Nos expérimentations ont été menées sur la méthode décrite en Section 3.5, qui apprend un réseau bayésien et recommande en calculant la marginale sur les variables d'intérêt, et sur l'algorithme DRC que nous venons de décrire. Nous avons utilisé la méthode *hc* du *package* R *bnlearn* (Scutari, 2010) pour apprendre le réseau bayésien.

Nous avons également essayé d'autres méthodes d'apprentissage (*tabu*, *mmhc*), sans que les résultats ne diffèrent significativement.

Nous avons testé deux algorithmes d'inférence dans les réseaux bayésiens, l'algorithme *jointree* fourni par la bibliothèque Jayes (Kutschke, 2013) et notre propre implémentation de RC. Dans les expériences suivantes, RC utilise un *dtree* qui a été généré par la méthode décrite par (Darwiche, Hopkins, 2001) basée sur le partitionnement d'un hypergraphe. Nous avons aussi évalué notre approche DRC. Les arbres de décomposition ternaires ont été construits en utilisant le logiciel hMETIS (Karypis, Kumar, 1998). Le seuil utilisé dans DRC est 50. La taille de l'échantillon équivalent utilisé est 10. Le code de DRC est accessible à l'adresse https://github.com/PFGimenez/PhD.

Les expériences ont été faites sur un ordinateur avec un processeur quad-cœur i5-3570 cadencé à 3,4Ghz, en utilisant un seul de ses cœurs. Les implémentations des algorithmes sont toutes écrites en Java et ont été exécutées avec la machine virtuelle Java *OpenJDK*.

Dans un premier temps, nous avons mené nos expériences sur le problème central, sans prendre en compte ni contraintes ni évolutivité de l'historique – les résultats sont décrits en section 5.1. Puis nous nous rapprochons du problème réel en considérant un contexte plus complexe – c'est le sujet des sections 5.2 (pour la prise en compte des contraintes) et 5.3 (pour la prise en compte d'historiques évolutifs).

5.1. Précision et efficacité du recommendeur : problème central

Protocole expérimental

Notre protocole imite, à partir de scénarios réels, le jeu de sessions de configuration. Chaque test est une simulation d'une session de configuration, c'est-à-dire une séquence de choix de variable et d'affectation. Chaque simulation correspond à une voiture de l'échantillon de départ. Pour chacune de ces voitures vendues, l'acheteur original a utilisé son propre ordonnancement de variables pour avancer dans la configuration. Malheureusement, les historiques de vente fournis par Renault détaillent les produits vendus mais pas l'ordre dans lequel ils ont été configurés. C'est pourquoi nous générons pour chaque produit P 10 sessions $session$, chacune issue d'un tirage aléatoire sur l'ordre des variables. Pour chaque variable X de la session, le recommandeur doit fournir une recommandation r pour X. Si r est égal à la valeur x stipulée par la session, alors la recommandation est un succès ; sinon, c'est un échec. Dans les deux cas, la variable X sera affectée à x.

Nous avons utilisé la procédure standard de validation croisée avec dix plis : le jeu de données a été séparé en dix échantillons de taille égale (donc dix plis de 2 708 véhicules chacun pour le cas *small* et de 1 478 véhicules pour le cas *medium*). Neuf échantillons servent à apprendre le réseau bayésien et à estimer les probabilités. Le dixième échantillon sert à effectuer les tests pour évaluer les algorithmes. Ce processus est répété de façon à utiliser chacun des dix échantillons comme échantillon de test.

Algorithme 5 : Protocole d'évaluation de la recommandation en configuration interactive

Input : Le jeu d'apprentissage $\mathcal{H}_{app}$ et le jeu de test $\mathcal{H}_{test}$
Output : Le taux de succès
main :
1 apprentissage de $\mathcal{H}_{app}$
2 $succes \leftarrow 0$
3 $erreur \leftarrow 0$
4 **for each** P **in** $\mathcal{H}_{test}$ **do**
5 $session \leftarrow$ affectation variables-valeurs P dans un ordre aléatoire
6 $Assigned \leftarrow \varnothing$
7 **for each** (Next, x) **in** $session$ **do**
8 $possibles \leftarrow$ ensemble des valeurs possibles
`// appel à l'algorithme qu'on évalue`
9 $r \leftarrow$ calcule_recommandation($\mathrm{Next} \mid Assigned, possibles$)
10 **if** $r = x$ **then** incrémenter $succes$
11 **else** incrémenter $erreur$
12 $Assigned \leftarrow Assigned \cup \{(\mathrm{Next}, x)\}$
13 **return** $succes \;/\; (succes + erreur)$

Au total, 13 002 240 (resp. 6 505 840) recommandations élémentaires sont effectuées lorsque le case study *small* (resp. *medium*) est joué sur les 27 088 (resp. 14 786) sessions. L'algorithme de recommandation est évalué par (i) le temps nécessaire au calcul d'une recommandation (ligne 9) et (ii) son taux le succès, c'est-à-dire le ratio entre le nombre de recommandations correctes et le nombre total de recommandations faites. Les résultats obtenus sont les valeurs moyennes sur les dix plis.

5.1.1. Oracle

Afin d'interpréter facilement les résultats de la validation croisée, nous souhaitons calculer le plus haut taux de succès atteignable pour le jeu de test. Si nous utilisons un algorithme qui connaît déjà le jeu de test, il pourrait l'utiliser en estimant sa distribution de probabilité. Il recommanderait alors pour la variable Next, sachant les valeurs déjà assignées $\mathbf{assigned}$, la valeur la plus probable dans le sous-ensemble des produits du jeu de test qui vérifient $\mathbf{assigned}$. Plus précisément, pour tout $x \in \underline{\mathrm{Next}}$, cet algorithme approcherait $p(x \mid \mathbf{assigned})$ par $\#(x, \mathbf{assigned})/\#(\mathbf{assigned})$. On remarque que $\#(\mathbf{assigned})$ n'est jamais nul car il y a au moins un produit dans le jeu de test qui le vérifie : celui dont la configuration est en train d'être simulée.

Nous appelons cet algorithme théorique "Oracle" et il est présenté dans l'algorithme 6. Son taux de succès est plus grand que celui de toute autre stratégie mais n'est pas de 100 % étant donné qu'il y a une certaine variabilité entre les utilisateurs. Le taux de succès de l'Oracle n'est généralement pas atteignable par les autres algo-

rithmes car l'Oracle a surappris le jeu de test, ce qui n'est pas le cas des algorithmes que nous évaluons.

Algorithme 6 : Oracle

Input : Le jeu de test $\mathcal{H}_{test}$
Output : Un majorant du taux de succès maximal
main :

$succes \leftarrow 0$
$erreur \leftarrow 0$
for each P **in** $\mathcal{H}_{test}$ **do**
    $session \leftarrow$ affectation variables-valeurs P dans un ordre aléatoire
    $Assigned \leftarrow \varnothing$
    for each (Next, x) **in** $session$ **do**
        $r \leftarrow \mathrm{argmax}_{x \in \underline{\mathrm{Next}}} \#(\{r, assigned\})$
        if $r = x$ **then** incrémenter $succes$
        else incrémenter $erreur$
        $Assigned \leftarrow Assigned \cup \{(\mathrm{Next}, x)\}$
return $succes \ / \ (succes + erreur)$

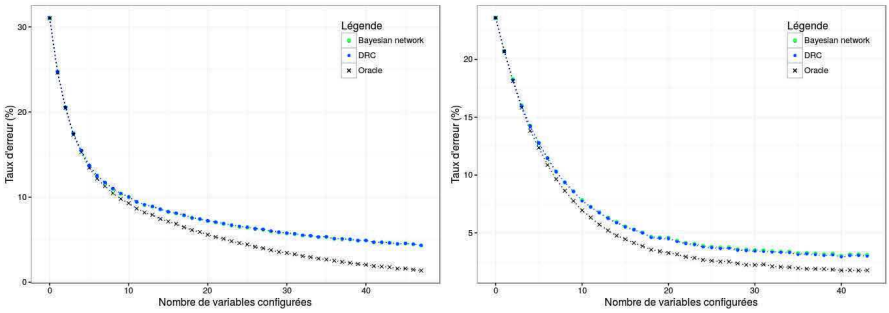
PROPOSITION 21. — *La probabilité de succès de l'algorithme Oracle est supérieure ou égale à tout algorithme évalué par validation croisée.*

Afin de pouvoir évaluer de manière absolue le comportement des recommandeur, nous avons calculé pour chaque jeu de test (une borne sup) du taux de réussite maximal. Bien sûr, il ne s'agit que d'une borne qui n'est pas forcément atteignable, car il y a un biais entre les lois du jeu d'apprentissage et du jeu de test.

5.1.2. Résultats

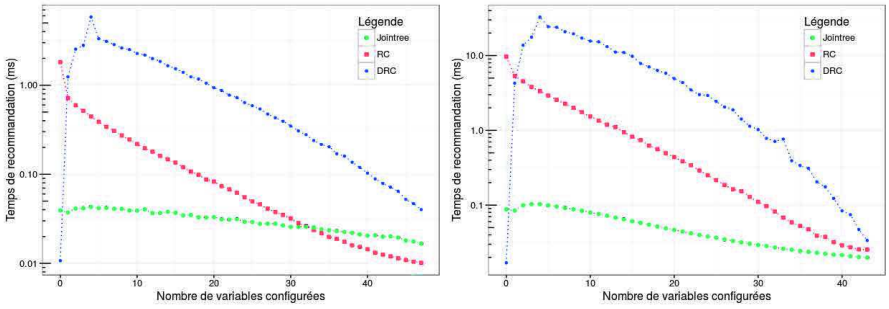
La figure 4 indique le taux de succès de l'inférence bayésienne classique (Jointree et RC qui ont la même précision) et DRC en fonction du nombre de variables configurées. L'Oracle est indiqué comme une ligne idéale. La figure 5 indique le temps de recommandation moyen de Jointree, RC et DRC en fonction du nombre de variables configurées.

Il n'y a pas de différence significative entre les taux des succès de l'apprentissage de réseau bayésien classique, que nous noterons RB, et DRC (les courbes sont quasiment confondues). Ces deux ensembles méthodes ont des taux de succès proches de la courbe idéale de l'Oracle. Nous pouvons également vérifier que plus le nombre de variables déjà configurées est grand, plus la recommandation est précise – le système a au bout de quelques choix mieux identifié l'utilisateur. Ce comportement est en accord avec avec les besoins en situation de configuration : le plus souvent, l'utilisateur ne prend le temps de réfléchir à ses choix que sur les dix ou quinze premières variables, qui sont celles qui l'intéressent vraiment. C'est pour les variables suivantes qu'il se reposera sur les recommandations du système.



(a) (b)

*Figure 4. Taux d'erreur moyen sur les jeux de données «*small*» (a) et «*medium*» (b)*



(a) (b)

*Figure 5. Temps moyen de recommandation sur les jeux de données «*small*» (a) et «*medium*» (b)*

Sur ces deux jeux de données, DRC a un temps de recommandation en forme de cloche, ce que nous expliquons de la manière suivante. Lorsqu'il y a très peu de variables configurées, il peut y avoir assez de configurations dans l'historique de ventes pour faire une estimation directe. Lorsqu'il y a plus de variables, le temps de calcul augmente rapidement car DRC descend plus bas dans l'arbre de décomposition ternaire tout en conditionnant à chaque fois des variables. À la fin de la recommandation, étant donné que la majorité des variables sont affectées, DRC a besoin de conditionner moins de variables, ce qui explique le gain de temps.

On peut voir que pour ces jeux de données, qui correspondent à une application réelle, le temps CPU de chacun des trois algorithmes est largement assez bon pour une utilisation en ligne : moins de 10 ms pour *small* et moins de 30 ms pour *medium* (voir figure 5).

5.2. Prise en compte des contraintes métier

En pratique, dans la plupart des applications en configuration les combinaisons de valeurs sont soumises à des contraintes métier, techniques ou commerciales. Notons que les éléments de l'historique de vente ne satisfont pas toutes les contraintes qui pèsent au jour de la configuration, et peuvent avoir été soumises à des contraintes qui ne sont plus à respecter. A titre d'exemple, seuls 55 % des véhicules de l'échantillon *medium* satisfont le jeu de contraintes qui nous a été fourni (2,5 % pour l'échantillon *small*). Ne garder que les éléments de l'échantillon qui satisfont les contraintes serait se priver d'une part importante de l'information disponible sur les préférences des utilisateurs précédents.

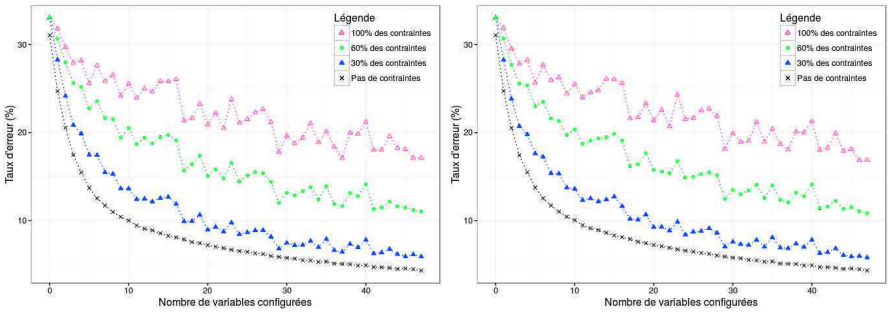
Notre approche générale est d'utiliser un moteur qui, après chaque choix de l'utilisateur, filtre les domaines des variables libres pour assurer la cohérence inverse globale – sont éliminées toutes les valeurs qui ne peuvent pas conduire à une extension complète de la configuration partielle courante qui satisfasse toutes les contraintes. S'il reste plus d'un choix, l'algorithme de recommandation calcule la restriction à la variable Next de la distribution de probabilité courante (conditionnée par les choix courants). Est ensuite proposée à l'utilisateur *la plus probable des valeurs cohérentes*.

Dans les expérimentations qui suivent, nous avons utilisé une approche par compilation de contraintes (le compilateur SALADD (Fargier *et al*., 2014) qui compile le CSP contenant les contraintes métier en un MDD équivalent).

Pour mesurer l'influence des contraintes sur le taux d'erreur, nous avons introduit les contraintes par paquets. Dans les figures qui suivent, on mesure la précision sur un CSP contenant 30 % (resp. 60 %, resp. 100 %) du jeu de contraintes total. Nous avons utilisé le même protocole en 10 plis que dans la section précédente, ne gardant du pli de test que les véhicules qui satisfont les contraintes. Moins de sessions sont donc joués que dans l'expérimentation précédente (à 100 % des contraintes, un total de 82 520 sessions pour *medium* et seulement 7 100 pour *small*), et une même session fera moins d'appels au recommandeur : la présence de contraintes fait qu'il est possible qu'à certaines étapes de la configuration une seule valeur soit possible pour Next – la recommandation est dite *triviale*. Nous n'avons pris en compte que les cas non triviaux dans la mesure du taux d'erreur. Le fait que moins de tests peuvent être effectués sur les jeux très contraints que sur les jeux moins contraints explique que sur small, où l'on possède peu de scenarii satisfaisant les contraintes, les courbes montrent une plus grande variabilité.

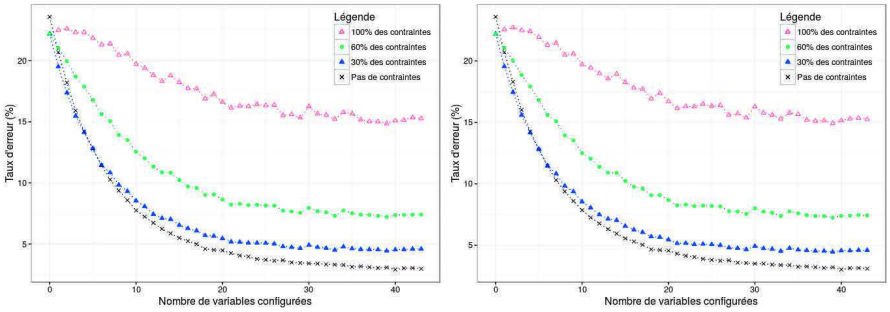
Sans surprise, la précision des deux approches baisse avec le taux de contrainte – ce qui s'explique par le fait qu'une grande partie de l'échantillon à partir duquel on infère les marginales ne satisfait pas les contraintes : le choix qui serait le plus satisfaisant pour l'utilisateur n'est pas toujours disponible, en dehors des premières variables.

On pourrait décider de ne lancer l'apprentissage que sur les produits satisfaisant les contraintes. Sur le problème de configuration de véhicule qui nous intéresse, il ne



(a) (b)

Figure 6. Taux d'erreur de DRC (a) et de RB (b) en fonction du taux de contrainte sur le problème small



(a) (b)

Figure 7. Taux d'erreur de DRC (a) et de RB (b) en fonction du taux de contrainte sur le problème medium

serait pas possible ni judicieux de n'apprendre que sur le sous-échantillon satisfaisant les contraintes – d'une part car, on l'a vu, de 55 % à 2,5 % des véhicules de l'échantillon satisfait les contraintes, d'autre part, car dans le problème réel les contraintes changent avec le temps. Le décalage entre contraintes et échantillon fait intrinsèquement partie de la problématique industrielle étudiée.

En résumé, les deux algorithmes d'inférence bayésienne restent comparables en termes de précision quand les contraintes sont prises en compte – elle décroît de manière similaire quand le taux de contraintes augmente, tout en restant bonne – moins de 25 % d'erreur dans tous les cas : dans trois cas sur quatre, l'utilisateur va "acheter" la valeur proposée par le système.

5.3. Précision avec mise à jour de l'historique

Les sections précédentes concluent sur l'intérêt des méthodes d'inférence bayésienne pour le problème central comme sur le problème avec contraintes, lorsque l'on considère un historique statique. En pratique, le problème est évolutif dans le temps : l'historique s'enrichit de nouvelles ventes au cours du temps. Nous nous sommes donc intéressés à une possibilité qu'offre DRC : inférer avec un historique évolutif.

Afin de détailler ce point, précisons que les données de Renault « *small* » sont datées, ce qui nous permet de les classer en quatre périodes, notés P1, P2, P3 et P4. Nous avons appris les dépendances entre variables avec les ventes de la première période et nous avons mesuré la précision des recommandations de RB et de DRC sur les périodes suivantes.

RB apprend le réseau bayésien (structure et tables) du période P1 pour recommander sur les période P1, P2, P3 et P4. DRC se base sur P1 pour apprendre l'arbre de décomposition ternaire, mais utilise, pour chaque période, tous les historiques à sa disposition : pour recommander sur le 10e pli de P2, il utilise P1 et les neufs autres plis de P2, pour recommander sur le 10e pli de P3, il utilise P1, P2 et les neufs autres plis de P3, etc. Le cas de P1 seul sert de "baseline" : on apprend sur 9 plis de P1 et on recommande sur le 10e.

Les résultats de cette expérience sont présentés dans le tableau 1 pour la variante sans contraintes et dans le tableau 2 pour la variante avec contraintes. Nous donnons ici le taux d'erreur moyen, indépendamment du nombre de variables déjà assignées.

Tableau 1. Taux d'erreur de DRC et de RB avec historique incrémental

Taux erreur	P1	P2	P3	P4
DRC	7,17 %	7,80 %	11,04 %	12,60 %
RB	7,19 %	7,97 %	11,68 %	14,25 %

Tableau 2. Taux d'erreur de DRC et de RB avec historique incrémental avec contraintes

Taux erreur	P1	P2	P3	P4
DRC	NA [a]	NA	25,23 %	21,00 %
RB	NA	NA	35,58 %	37,29 %

a. Sur le problème *small* aucun produit vendu dans les périodes P1 et P2 ne correspondent aux contraintes, les plis ne fournissent aucun de scenario de test

Nous pouvons constater une baisse de précision globale de DRC et de RB au fur et à mesure des périodes. Ce qui était prévisible et s'explique par l'évolution de l'offre : les contraintes qui définissent l'offre de vente à chaque période changent au cours du temps. Les acheteurs des premières périodes avaient accès à une gamme différente que les acheteur des dernières périodes, et la trace que l'on possède de leurs préférence – leurs achats – dirige le recommandeur vers des véhicules différents de ceux qui ont

été choisis dans les dernières gammes. Cependant, le taux d'erreur de DRC subit une dégradation moins prononcée que celle de RB, puisque DRC peut utiliser la base d'exemples mise à jour. L'écart est encore plus grand entre le taux d'erreur de DRC et de RB sur les dernières périodes dans l'expérience qui prend en compte les contraintes – puisque DRC peut utiliser des scenarii d'apprentissage qui satisfont les contraintes courantes, alors que l'approche statique n'a accès qu'à un historique "ancien", qui ne les satisfait souvent pas.

6. Conclusion

Cet article a proposé une approche du problème de la recommandation de valeur dans le cadre de la configuration interactive de produit par inférence bayésienne. Elle a été déclinée sous deux variantes : toutes deux sont basées sur l'indépendance bayésienne mais diffèrent sur leur façon d'utiliser le jeu de données. La première approche apprend un réseau bayésien dans une phase hors ligne. Ce réseau bayésien est ensuite utilisé pour les calculs de recommandation qui sont faits en ligne. L'autre approche, appelée *Direct Recursive Conditioning*, utilise la même structure de dépendance que le réseau bayésien appris mais les probabilités sont estimées en ligne, en utilisant directement le jeu de données. L'avantage de DRC est qu'il peut utiliser en direct un historique qui évolue, tant que les relations d'indépendance conditionnelle ne changent pas.

Nos expériences sur des cas concrets montrent que ces deux méthodes sont compatibles avec les exigences d'un configurateur en ligne, et les deux ont une très bonne précision ; proche du maximum possible, en fait.

Dans notre approche, les contraintes ne sont pas prises en compte lors de l'apprentissage du réseau bayésien. Cela a deux conséquences : d'une part, les dépendances entre variables correspondant à ces contraintes ne sont pas prises en compte lors de l'apprentissage de la structure du réseau bayésien, ce qui peut entraîner des erreurs lors de l'optimisation de la structure ; d'autre part, le réseau bayésien peut affecter une probabilité non nulle à une configuration impossible, ce qui nous oblige à filtrer à chaque étape les valeurs possibles pour la recommandation. Nous ne connaissons pas à ce jour de méthode permettant de prendre en compte de telles contraintes lors de l'apprentissage d'un réseau bayésien, mais c'est une direction de recherche importante.

Au-delà de la recommandation pour configuration interactive, l'approche de DRC peut être utilisée pour d'autres applications qui se basent sur l'inférence bayésienne à partir d'un jeu de données d'objets combinatoires. De manière orthogonale, l'idée d'utiliser des probabilités à la volée pourrait être étendue à d'autres méthodes d'inférence que RC.

Bibliographie

Adomavicius G., Tuzhilin A. (2005). Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, n° 6, p. 734–749.

Akaike H. (1998). Information theory and an extension of the maximum likelihood principle. In *Selected papers of Hirotugu Akaike*, p. 199–213. Springer New York.

Amilhastre J., Fargier H., Marquis P. (2002). Consistency restoration and explanations in dynamic CSPs application to configuration. *Artificial Intelligence*, vol. 135, n° 1-2, p. 199–234.

Astesana J., Cosserat L., Fargier H. (2010). Constraint-based vehicle configuration: A case study. In *Proceedings of ICTAI'10*, p. 68–75.

Bacchus F., Dalmao S., Pitassi T. (2002). Value elimination: Bayesian inference via backtracking search. In *Proceedings of UAI'02*, p. 20–28.

Bahar R. I., Frohm E. A., Gaona C. M., Hachtel G. D., Macii E., Pardo A. *et al*. (1997). Algebraic decision diagrams and their applications. *Formal Methods in System Design*, vol. 10, n° 2/3, p. 171–206.

Bessiere C., Fargier H., Lecoutre C. (2013). Global inverse consistency for interactive constraint satisfaction. In *Proceedings of CP'13*, p. 159–174.

Bigot D., Fargier H., Mengin J., Zanuttini B. (2013). Probabilistic conditional preference networks. In *Proceedings of UAI'13*, p. 72–81.

Bigot D., Mengin J., Zanuttini B. (2014). Learning probabilistic CP-nets from observations of optimal items. In *Proceedings of STAIRS'14*, vol. 264, p. 81–90.

Booth R., Chevaleyre Y., Lang J., Mengin J., Sombattheera C. (2010). Learning conditionally lexicographic preference relations. In *Proceedings of ECAI'10*, p. 269–274.

Boutilier C., Brafman R. I., Domshlak C., Hoos H. H., Poole D. (2004). CP-nets: A tool for representing and reasoning with conditional ceteris paribus preference statements. *Journal of Artificial Intelligence Research*, vol. 21, p. 135–191.

Braziunas D., Boutilier C. (2005). Local utility elicitation in GAI models. In *Proceedings of UAI'05*, p. 42–49.

Bui T. N., Jones C. (1992). Finding good approximate vertex and edge partitions is NP-hard. *Information Processing Letters*, vol. 42, n° 3, p. 153–159.

Chickering D. M. (1995). Learning Bayesian networks is NP-complete. In *Proceedings of AISTATS'95*, p. 121–130.

Cooper G. F., Herskovits E. (1991). A Bayesian method for constructing bayesian belief networks from databases. In *Proceedings of UAI'91*, p. 86–94.

Cooper G. F., Herskovits E. (1992). A Bayesian method for the induction of probabilistic networks from data. *Machine learning*, vol. 9, n° 4, p. 309–347.

Coster R., Gustavsson A., Olsson T., Rudstrom A., Rudstrom A. (2002). Enhancing web-based configuration with recommendations and cluster-based help. In *Proceedings of AH'02*.

Cowell R. G. (2006). *Probabilistic networks and expert systems: Exact computational methods for Bayesian networks*. Springer Science & Business Media.

Dagum P., Luby M. (1993). Approximating probabilistic inference in Bayesian belief networks is NP-hard. *Artificial Intelligence*, vol. 60, n° 1, p. 141–153.

Darwiche A. (2001). Recursive conditioning. *Artificial Intelligence*, vol. 126, n° 1-2, p. 5–41.

Darwiche A., Hopkins M. (2001). Using recursive decomposition to construct elimination orders, jointrees, and dtrees. In *Proceedings of ECSQARU'01*, p. 180–191.

Dawid P., Lauritzen S. L., Spiegelhalter D. J. (2007). *Probabilistic networks and expert systems: Exact computational methods for Bayesian networks*. Springer.

Falkner A. A., Felfernig A., Haag A. (2011). Recommendation technologies for configurable products. *Artificial Intelligence*, vol. 32, n° 3, p. 99–108.

Fargier H., Marquis P., Schmidt N. (2014). Compacité pratique des diagrammes de décision valués. normalisation, heuristiques et expérimentations. *Revue d'Intelligence Artificielle*, vol. 28, n° 5, p. 571–592.

Gonzales C., Perny P. (2004). GAI networks for utility elicitation. In *Proceedings of KR'04*, p. 224–234.

Hadzic T., Wasowski A., Andersen H. R. (2007). Techniques for efficient interactive configuration of distribution networks. In *Proceedings of IJCAI'07*, p. 100–105.

Hotz L., Felfernig A., Günter A., Tiihonen J. (2014). A short history of configuration technologies. In *Knowledge-based configuration*, p. 9–19. Morgan Kaufmann.

Huffman C., Kahn B. E. (1998). Variety for sale: Mass customization or mass confusion? *Journal of retailing*, vol. 74, n° 4, p. 491–513.

Jannach D., Zanker M., Felfernig A., Friedrich G. (2010). *Recommender systems - an introduction*. Cambridge University Press.

Karypis G., Kumar V. (1998). Multilevelk-way partitioning scheme for irregular graphs. *Journal of Parallel and Distributed computing*, vol. 48, n° 1, p. 96–129.

Kim J. H., Pearl J. (1983). A computational model for causal and diagnostic reasoning in inference systems. In *Proceedings of IJCAI'83*, vol. 83, p. 190–193.

Kutschke M. (2013). Jayes - Bayesian network library under eclipse public license.

Lauritzen S. L., Spiegelhalter D. J. (1988). Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society. Series B (Methodological)*, p. 157–224.

Lemmer J., Kanal L. (2014). Propagating uncertainty in Bayesian networks by probabilistic logic sampling. *Proceedings of UAI'14*, vol. 5, p. 149.

Pargamin B. (2002). Vehicle sales configuration : the cluster tree approach. In *Proceedings of ECAI'02 configuration workshop.*

Pearl J. (1985). A constraint-propagation approach to probabilistic reasoning. In *Proceedings of UAI'85*, p. 357–370.

Pearl J. (1989). *Probabilistic reasoning in intelligent systems - networks of plausible inference*. Morgan Kaufmann.

Ricci F., Rokach L., Shapira B., Kantor P. B. (Eds.). (2011). *Recommender systems handbook*. Springer.

Scutari M. (2010). Learning bayesian networks with the bnlearn R package. *Journal of Statistical Software*, vol. 35, n° 3, p. 1–22.

Spirtes P., Glymour C. N., Scheines R. (2000). *Causation, prediction, and search*. MIT press.

Tiihonen J., Felfernig A. (2010). Towards recommending configurable offerings. *International Journal of Mass Customisation*, vol. 3, n° 4, p. 389–406.

Tiihonen J., Felfernig A. (2017). An introduction to personalization and mass customization. *Journal of Intelligent Information Systems*, vol. 49, n° 1, p. 1–7.

Verma T., Pearl J. (1990). Equivalence and synthesis of causal models. In *Proceedings of UAI'90*, p. 255–270.

Zhang E., Gao L. (2014). A vertex separator-based algorithm for hypergraph bipartitioning. *Journal of Computers*, vol. 9, n° 8, p. 1886–1896.

Zhang N. L., Poole D. (1994). A simple approach to Bayesian network computations. In *Proceedings of AI'94*.

Preuves

PREUVE (preuve de la proposition 6). — Notons que $\mathbf{C} \subseteq \mathbf{V}$. Soient $\mathbf{V}'_1 = \mathbf{V}_1 \setminus \mathbf{C}$ et $\mathbf{V}'_2 = \mathbf{V}_2 \setminus \mathbf{C}$, alors :

$$F_{\mathbf{V}}(\mathbf{u}) = \sum_{\substack{\mathbf{c}\in\underline{\mathbf{C}} \\ \mathbf{c}\sim\underline{\mathbf{u}}}} \sum_{\substack{\mathbf{v}'_1\in\underline{\mathbf{V}'_1} \\ \mathbf{v}'_1\sim\mathbf{u}}} \sum_{\substack{\mathbf{v}'_2\in\underline{\mathbf{V}'_2} \\ \mathbf{v}'_2\sim\mathbf{u}}} \left(\prod_{X\in\mathbf{V}_1} \Theta(\mathbf{v}[X] \mid \mathbf{uv}[\mathbf{Pa}(X)]) \right) \times \left(\prod_{X\in\mathbf{V}_2} \Theta(\mathbf{v}[X] \mid \mathbf{uv}[\mathbf{Pa}(X)]) \right)$$

Mais si $X \in \mathbf{V}_1$, alors $\Theta(\mathbf{v}[X] \mid \mathbf{uv}[\mathbf{Pa}(X)]) = \Theta(\mathbf{v}_1[X] \mid \mathbf{uv}'_1\mathbf{c}[\mathbf{Pa}(X)])$ ne dépend pas de $\mathbf{v}'_2$, puisque tous les parents de X sont dans $\mathbf{V}_1 \cup \mathbf{C} \cup \mathbf{U}$; et de même, si $X \in \mathbf{V}_2$, alors $\Theta(\mathbf{v}[X] \mid \mathbf{uv}[\mathbf{Pa}(X)])$ ne dépend pas de $\mathbf{v}'_1$. Donc

$$F_{\mathbf{V}}(\mathbf{u}) = \sum_{\substack{\mathbf{c}\in\underline{\mathbf{C}} \\ \mathbf{c}\sim\underline{\mathbf{u}}}} \left(\sum_{\substack{\mathbf{v}'_1\in\underline{\mathbf{V}'_1} \\ \mathbf{v}'_1\sim\mathbf{u}}} \prod_{X\in\mathbf{V}_1} \Theta(\mathbf{v}_1[X] \mid \mathbf{uv}'_1\mathbf{c}[\mathbf{Pa}(X)]) \right) \times \left(\sum_{\substack{\mathbf{v}'_2\in\underline{\mathbf{V}'_2} \\ \mathbf{v}'_2\sim\mathbf{u}}} \prod_{X\in\mathbf{V}_2} \Theta(\mathbf{v}_2[X] \mid \mathbf{uv}'_2\mathbf{c}[\mathbf{Pa}(X)]) \right)$$

On remarque aussi que si $\mathbf{v}_1 \in \underline{\mathbf{V}_1}$ et $\mathbf{v}_1 \sim \mathbf{u}[\mathbf{V}_1]\mathbf{c}$, alors forcément pour toute variable $X \in \mathbf{C}$, $\mathbf{v}_1[X]$ est fixé (et de même si $\mathbf{v}_2 \in \underline{\mathbf{V}_2}$ et $\mathbf{v}_2 \sim \mathbf{u}[\mathbf{V}_2]\mathbf{c}$, alors $\mathbf{v}_2[X]$ est fixé) d'où le résultat. ■

PREUVE (preuve de la proposition 11). — D'après la proposition 10, si $\mathbf{C}$ sépare $\mathbf{G}_1$ de $\mathbf{G}_2$ dans le graphe moralisé de $\mathcal{G}_{\mathrm{Anc}(\mathbf{U})}$, alors $\mathbf{G}_1 \perp\!\!\!\perp \mathbf{G}_2 \mid \mathbf{C}$, ce qui justifie le passage de la ligne (1) à la ligne (2) :

$$p(\mathbf{u}) = \sum_{\mathbf{c}\sim\mathbf{u}} p(\mathbf{u} \mid \mathbf{c})\, p(\mathbf{c}) \tag{1}$$

$$= \sum_{\mathbf{c}\sim\mathbf{u}} p(\mathbf{u}[\mathbf{G}_1] \mid \mathbf{c})\, p(\mathbf{u}[\mathbf{G}_2] \mid \mathbf{c})\, p(\mathbf{c}) \tag{2}$$

$$= \sum_{\mathbf{c}\sim\mathbf{u}} \frac{p(\mathbf{u}[\mathbf{G}_1]\mathbf{c})\, p(\mathbf{u}[\mathbf{G}_2]\mathbf{c})}{p(\mathbf{c})} \tag{3}$$

■

PREUVE (preuve du lemme 17). — Soit un ensemble de variables $\mathbf{V}$ et un réseau bayésien $\mathcal{G}$ tels que $\mathbf{V}$ ne soit pas décomposable dans $\mathcal{G}$, c'est-à-dire tel que le graphe moralisé du sous-graphe induit de $\mathbf{V}$ est complet. Montrons que $H_{\mathcal{G}_m}(\mathbf{V}) \leq 1$ ou $\mathbf{V} = \varnothing$.

Si $\mathbf{V} = \varnothing$, alors l'implication est vérifiée. Sinon, montrons qu'il existe une variable X tel que $X \in \mathbf{V}$ et $\mathbf{V} \subseteq \{X\} \cup \mathbf{Pa}(X)$.

Soit deux variables Y, Z de $\mathbf{V}$. Étant donné que Y et Z sont voisins dans le graphe moralisé du sous-graphe induit de $\mathbf{V}$, cela signifie que soit qu'ils sont les parents de X et que X apparient à $\mathbf{V}$, soit Y et Z sont voisins dans $\mathbf{V}$. Dans ce dernier cas, si Y est le parent de Z dans $\mathcal{G}$, et alors $Z = X$ (sinon $Y = X$).

Dans tous les cas, nous avons bien une variable X qui appartient à $\mathbf{V}$, et $\mathbf{V} \setminus \{X\} \subseteq \mathbf{Pa}(X)$.

Nous pouvons tout d'abord remarquer que X est la seule variable de $\mathbf{V}$ sans descendant dans $\mathbf{V}$, donc $\mathbf{F}_{\mathcal{G}}(\mathbf{V}) = \{X\}$. Ainsi, $H_{\mathcal{G}_m}(\mathbf{V}) = \max_{Y\in\mathbf{V}} d_{\mathcal{G}_m}(X, Y)$. Or, toutes les autres variables de $\mathbf{V}$ sont les parents de X.

Si $\mathbf{V} = \{X\}$, alors $H_{\mathcal{G}_m}(\mathbf{V}) = 0$. Si $\mathbf{V} \supset \{X\}$, alors $H_{\mathcal{G}_m}(\mathbf{V}) = 1$.

Au final, si $\mathbf{V}$ n'est pas décomposable, alors on a bien $H_{\mathcal{G}_m}(\mathbf{V}) \leq 1$ ou $\mathbf{V} = \varnothing$.

Réciproquement, supposons que $H_{\mathcal{G}_m}(\mathbf{V}) \leq 1$ ou $\mathbf{V} = \varnothing$.

Cas 1 : $\mathbf{V} = \varnothing$

Alors $\mathbf{V}$ n'est pas décomposable.

Cas 2 : $H_{\mathcal{G}_m}(\mathbf{V}) = 0$

Alors $|\mathbf{V}| = 1$ et $\mathbf{V}$ n'est pas décomposable.

Cas 3 : $H_{\mathcal{G}_m}(\mathbf{V}) = 1$

Montrons tout d'abord que dans ce cas, $|\mathbf{F}_G(\mathbf{V})| = 1$. Prouvons cela par l'absurde en supposant qu'il existe $X, Y \in \mathbf{V}$ sans descendant dans $\mathbf{V}$. Donc X n'est pas le père de Y ni inversement. De plus, X et Y n'ont pas d'enfant dans $\mathcal{G}_{\mathrm{Anc}(\mathbf{V})}$. Donc X et Y ne sont pas voisins dans le graphe moral $\mathcal{G}_m$, et donc $d_{\mathcal{G}_m}(X, Y) > 1$, ce qui contredit $H_{\mathcal{G}_m}(\mathbf{V}) = 1$. Il n'y a donc qu'une seule variable $X \in \mathbf{V}$ sans descendant dans $\mathbf{V}$. Étant donné que $H_{\mathcal{G}_m}(\mathbf{V}) = 1$, toutes les autres variables de $\mathbf{V}$ sont voisins de X.

Pour tout $Y \in \mathbf{V}$ différent de X, la distance entre X et Y vaut 1, ce qui signifie qu'il y a une arête entre X et Y. Or, il y a une arête dans le graphe moral $\mathcal{G}_m$ si, dans le réseau bayésien :

Cas 1 X est le parent de Y ou X et Y ont a un fils $Z \in \mathrm{Anc}(\mathbf{V})$. Or, on a supposé que X n'est l'ancêtre d'aucune autre variable de $\mathbf{V}$, donc ce cas est impossible.

Cas 2 X est le fils de Y.

Ainsi, X est le fils de toutes les autres variables de $\mathbf{V}$. Le graphe moralisé du sous-graphe induit de $\mathbf{V}$ est donc complet, ce qui signifie que $\mathbf{V}$ n'est pas décomposable. ■

PREUVE (preuve de la proposition 18). — Supposons que $\mathbf{V}$ ne soit pas décomposable dans $\mathcal{G}$ et montrons qu'il n'existe pas de partition admissible. Rappelons que le sous-graphe induit sur $\mathbf{V}$ est complet.

Supposons qu'il existe une partition admissible $\{\mathbf{G}_1, \mathbf{G}_2, \mathbf{C}\}$. On peut facilement vérifier que $\mathbf{G}_1 \cap \mathbf{V} \neq \varnothing$ et $\mathbf{G}_2 \cap \mathbf{V} \neq \varnothing$. Il existe donc $Y \in \mathbf{G}_1$ et $Z \in \mathbf{G}_2$. Or, Y et Z sont voisins, donc ils ne peuvent pas être séparés par $\mathbf{C}$. Donc une telle partition n'existe pas. ■

PREUVE (preuve du lemme 19). — Soit $\mathcal{G}$ un réseau bayésien, $\mathbf{V}$ un ensemble de variables. Alors, pour tout ensemble $\mathbf{E} \subseteq \mathrm{Anc}(\mathbf{V})$, s'il existe une variable $X \in \mathbf{F}_{\mathcal{G}}(\mathbf{V})$ telle que $X \notin \mathbf{E}$, alors :

$$|\mathrm{Anc}(\mathbf{E})| < |\mathrm{Anc}(\mathbf{V})|$$

X étant une variable qui n'a pas de descendant dans $\mathbf{V}$ (et donc dans $\mathbf{E}$) et que $X \notin \mathbf{E}$, $X \notin \mathrm{Anc}(\mathbf{E})$.

On sait donc que

- $\mathrm{Anc}(\mathbf{E}) \subseteq \mathrm{Anc}(\mathbf{V})$
- $X \notin \mathrm{Anc}(\mathbf{E})$
- $X \in \mathrm{Anc}(\mathbf{V})$

Nous pouvons donc en conclure que $|\mathrm{Anc}(\mathbf{E})| < |\mathrm{Anc}(\mathbf{V})|$. ■

PREUVE (preuve de la proposition 21). — Cette proposition résulte d'un résultat classique de statistiques : avec une fonction de perte 0-1 (c'est-à-dire qui compte 1 en cas d'échec, 0 en cas de succès, comme dans notre expérience), l'estimateur qui minimise

cette perte (c'est-à-dire le taux d'erreur) est l'estimateur du maximum a posteriori, qui recommande la valeur la plus probable.

Or, c'est exactement ce que fait l'Oracle. Donc l'Oracle maximise la probabilité de succès sur les autres méthodes, y compris celles évaluées par validation croisée. ■