La recommandation en configuration interactive

de Pierre-François Gimenez
encadré par Hélène Fargier et Jérôme Mengin,
à l'IRIT, Institut de Recherche en Informatique de Toulouse

Août 2015

IRIT/RT–2015–03–FR

Résumé – abstract

Résumé

Les configurateurs de produit en ligne se multiplient sur les plate-formes d'e-commerce et prennent de plus en plus d'importance. Ils deviennent pour les entreprises des interfaces privilégiées avec le client : une grande majorité des clients qui souhaitent acheter une voiture ont utilisé un configurateur afin de la choisir.

La recommandation dans le cadre des configurateurs constitue donc un enjeu majeur mais n'est pas encore très explorée par la recherche. Ce mémoire traite de nouvelles méthodes qui recommandent efficacement un utilisateur en maximisant son taux de satisfaction ou en recherchant un compromis entre les désirs de l'utilisateur et de l'entreprise.

Les algorithmes de ce mémoire sont efficaces : ils sont aptes à une utilisation en ligne et possèdent un taux de réussite proche du maximum théorique.

Abstract

Online configurators are getting more and more common because they are at a privileged position between the customer and the company. In fact, most customers choose the car they want with an online configurator before actually purchasing it.

Recommendation for configurators is an emerging issue. This research report deals with new recommendation methods for maximizing customer's satisfaction or for finding a compromise between a customer and the company.

The algorithms presented in this report are efficient: they are suitable for online configuration and have a near-optimal success-rate.

Table des matières

Introduction

Le e-commence tente de faire correspondre les besoins et les préférences de clients aux possibilités et aux objectifs de l'entreprise. Alors qu'une grande partie des produits disposent de très peu de variantes (comme le coloris), d'autres produits sont hautement configurables avec un très grand nombre de possibilités, comme des ordinateurs fixes ou des voitures. Renault, par exemple, peut produire environ 10^{27} voitures différentes.

Néanmoins, alors que les conditions sont remplies pour que le client trouve le produit qui puisse le satisfaire, 60% des clients parcourent un catalogue en ligne sans trouver ce qu'ils cherchent. C'est donc naturellement que sont apparus les configurateurs, des services en ligne permettant à un client de configurer les produits qu'il souhaite acheter. Un des enjeux des configurateurs est la recommandation de valeurs, de manière à ce que les choix effectués conviennent aux clients, tout en prenant en compte les contraintes de l'objet à configurer.

C'est dans ce contexte que naquit le projet ANR BR4CP (*Business Recommendation for Configurable Products*), qui regroupe des laboratoires et des industriels. Son objectif est d'adapter les travaux des systèmes de recommandation et de modélisation des préférences de l'utilisateur à la configuration en ligne, qui nécessite des algorithmes rapides sur des espaces combinatoires.

Ce mémoire se basera en particulier sur un problème de recommandation de valeurs lors d'une session de configuration.

Dans le chapitre 1, nous commencerons par poser le contexte et la problématique de recommandation. Le chapitre 2 s'attardera sur un ensemble d'algorithmes de recommandation basés sur un calcul de fréquences. Le chapitre 3 traitera des méthodes utilisant des réseaux bayésiens. Le chapitre 4 s'intéressera à la manière d'implémenter efficacement les algorithmes des chapitres 2 et 3 grâce aux langages de compilation. Le chapitre 5 traitera des expériences, détaillant protocole et résultats. Enfin, des résultats préliminaires portant sur des sujets connexes ainsi que des idées n'ayant pas encore été implémentées seront présentés chapitre 6.

Chapitre 1

Contexte

1 Configuration

1.1 Projet BR4CP

Le projet ANR Blanc « BR4CP » (*Business Recommendation for Configurable Products*) réunit des industriels : Cameleon software, IBM et Renault, et des laboratoires de recherche : le Centre de Recherche en Informatique de Lens (Lens), le Laboratoire en Informatique et Robotique et Micro électronique de Montpellier (LIRMM) et l'Institut de Recherche en Informatique de Toulouse (IRIT).

Son ambition est d'améliorer les systèmes de configuration interactive existants sous plusieurs aspects dont la recommandation. Ce projet ANR se positionne donc dans deux branches du domaine du e–commerce : la configuration interactive et la recommandation.

C'est dans ce cadre de ce projet que Renault nous a fourni une problématique réelle ainsi que des données sur lesquelles s'appuient ces travaux.

1.2 Les trois classes de produits recommandables

Il existe principalement trois classes de systèmes de ventes qui correspondent à trois classes de produits.

Les produits standardisés : ce sont des objets standardisés (c'est-à-dire peu ou pas personnalisés) comme des livres, des habits, des films... Ils sont les plus courants dans l'e-commerce. Parmi les plate-formes de vente célèbre, on peut citer Amazon. La recommandation se base sur des produits proches ou fréquemment achetés ensemble.

Les produits combinés : ces produits sont composés de plusieurs sous-produits. C'est le cas, par exemple, des agences de voyage qui proposent de choisir le moyen de transport, l'hôtel... La recommandation

se base alors sur un formulaire rempli par le client afin de connaître ses préférences [FPV11].

Les produits complexes : ces produits sont être conçus grâce à des paramètres soumis à des contraintes [Stu97]. Contrairement aux précédents produits, la taille du catalogue est combinatoire. Renault, par exemple, peut produire environ 10^{27} voitures différentes.

Ce sont de ces produits complexes uniquement dont on parlera durant toute la suite de ce mémoire.

1.3 Contraintes

Comme il a été écrit précédemment, les produits dont il est question sont caractérisés par les valeurs de ses paramètres. Néanmoins, toutes les combinaisons de valeurs ne forment pas un objet licite. Par exemple, Renault interdit pour des raisons évidentes à une voiture d'avoir les paramètres « décapotable » et « présence d'un toit ouvrant électrique » vrais simultanément.

Ces contraintes sont exprimées sous la forme de formules booléennes. Il est nécessaire, pour permettre au client d'effectuer les actions présentées précédemment, de résoudre un problème de satisfaction de contraintes, un CSP (*Constraint Satisfaction Problem*).

Il existe des algorithmes, appelés solveurs de CSP, qui permettent de résoudre ces problèmes. Ces solveurs forment un problème indépendant de la recommandation et ne seront pas étudiés. On supposera disposer d'un solveur de CSP durant la suite de ce mémoire. Le solveur utilisé sera présenté lors du chapitre 4.

1.4 La session de configuration

Afin de permettre à un client de choisir un produit dans un espace combinatoire, on lui permet de configurer petit à petit son produit lors d'une session de configuration. On appelle alors configurateur le logiciel permettant d'effectuer cette session de configuration.

Une session de configuration permet à l'utilisateur de choisir un à un les paramètres du produit qu'il désire. Dans le cadre de la configuration d'une voiture, ces paramètres peuvent être la couleur de la voiture, son moteur, sa gamme... L'utilisateur est généralement libre quant à l'ordre dans lequel il affecte les paramètres.

Parmi ses actions possibles, le client peut :

- choisir la valeur d'un paramètre ;
- annuler son choix de la valeur d'un paramètre ;

- demander au système les valeurs d'un paramètre qui, tout en respectant ses choix précédents, satisfont les contraintes ;
- demander les valeurs des paramètres qu'il n'a pas encore fixés qui minimisent le coût du produit ;
- demander la recommandation d'un paramètre.
- demander la recommandation d'une valeur d'un paramètre ;

2 Recommandation

2.1 Recommandation de paramètre et de valeur

Il existe deux différentes recommandations en configuration interactive :

recommandation de paramètre : c'est la recommandation qui suggère, par exemple, de s'intéresser au paramètre « couleur de la voiture ».

recommandation de valeur : c'est la recommandation qui suggère, pour un paramètre préalablement choisi, une de ses valeurs, par exemple, « couleur bleue ».

Dans le cadre du problème posé par Renault, nous ne nous intéresserons qu'à la recommandation de valeur.

2.2 Exemple de recommandation de valeurs en configuration

Afin de présenter la recommandation en configuration, rien ne vaut un exemple. Il s'agit de la recommandation d'une voiture configurable avec seulement trois paramètres. Les cas réels peuvent posséder bien plus de paramètres.

« Utilisateur : Me recommandez-vous de choisir un toit ouvrant ?

— Recommandeur : Oui, je vous recommande un toit ouvrant.

— Utilisateur : Non, je vais plutôt prendre sans toit ouvrant. Devrais-je prendre un système de climatisation ?

— Recommandeur : Sachant que vous ne voulez pas de toit ouvrant, je vous recommande la climatisation.

— Utilisateur : Très bien. Quelle gamme de prix me correspond le plus ?

— Recommandeur : Sachant que vous voulez une voiture sans toit ouvrant avec climatisation, les contraintes font que vous êtes obligé de choisir une voiture de milieu de gamme.

— Utilisateur : Je prends alors une voiture de milieu de gamme. »

2.3 Problématique

La problématique de recommandation abordée dans ce mémoire provient de Renault, qui participe au projet BR4CP. Dans ce problème, les produits sont des voitures et on a accès à un historique de ventes. Cet historique contient des exemples de configurations de voitures. Cet historique ne contient que les configurations finales choisies par les utilisateurs, et en particulier, on ne sait pas dans en quel ordre ont été choisie les différentes valeurs.

Dans ce problème, l'opération de recommandation est définie de la manière suivante.

Entrées :

- les valeurs déjà fixées par l'utilisateur
- la variable dont il faut recommander une valeur
- l'historique de ventes

Sortie : une valeur pour la variable demandée.

Contrairement à d'autres systèmes de recommandation qui renvoient une liste ordonnée de valeurs recommandées, le recommandeur ne doit retourner qu'une seule valeur. Une extension du problème qui puisse retourner une liste de valeurs est étudiée au chapitre 6.

2.4 Notations

Afin de facilité la lecture de ce mémoire, quelques notations seront adoptées.

- On appelle *variable* un paramètre qui puisse être affecté par l'utilisateur lors d'une session de configuration. Une variable est notée V avec un indice, comme V_1.
- L'ensemble des variables d'un problème de configuration est noté $\mathcal{V}$.
- L'ensemble des variables affectées par l'utilisateur est noté $\mathcal{V}_{affectees} \subseteq \mathcal{V}$. On note n le nombre de variables, c'est-à-dire $n \equiv Card(\mathcal{V})$.
- La variable pour laquelle on cherchera à recommander une valeur est notée $V_{reco} \in \mathcal{V}$. On suppose de plus que $V_{reco} \notin \mathcal{V}_{affectees}$.
- L'ensemble des valeurs autorisées par les contraintes pour V_{reco} sachant les valeurs des variables de $\mathcal{V}_{affectees}$ est noté $val_{\mathcal{V}_{affectees}}(V_{reco})$.

2.5 Objectifs

L'objectif de ces travaux était de rechercher et d'évaluer des méthodes de recommandation. En particulier, les méthodes proposées devaient :

- être calculables en ligne, afin que l'utilisateur du configurateur n'abandonne pas la configuration. Ceci impose d'avoir des algorithmes qui ne soient pas NP-complets ou NP-difficiles, car ceux-ci ne donnent aucune assurance sur leur temps de calcul. Néanmoins, des calculs préalables, hors-ligne, sont possibles sans contraintes de temps.

- posséder des taux de satisfaction élevés, c'est-à-dire maximiser la proportion de recommandations qui sont effectivement suivies par l'utilisateur.

  Le protocole d'évaluation des algorithmes est présenté chapitre 5.

Chapitre 2

Recommandation par calcul de fréquences

1 Introduction

1.1 Une recommandation naïve mais limitée

Afin de recommander une valeur, on dispose d'un ensemble d'apprentissage constitué d'exemples choisis par des utilisateurs précédents. Lorsqu'on doit recommander une valeur en connaissant certains choix de l'utilisateur, une idée intuitive est de calculer le choix le plus souvent pris par les utilisateurs qui ont fait les mêmes choix jusque là. Cette recommandation peut s'exprimer de la manière suivante : « 80 % des personnes qui ont fait les mêmes choix que vous ont choisi... ».

On peut néanmoins rencontrer un problème, notamment lorsque beaucoup de variables sont fixées : il n'y a parfois plus aucun exemple qui correspond aux choix de l'utilisateur, auquel cas le calcul précédent n'est plus réalisable.

D'une manière générale, plus le nombre de variable fixées est grand, plus le nombre d'exemples qui seront compatibles avec ces valeurs sera faible. C'est-à-dire que ce problème de pénurie d'exemples sera plus susceptible d'arriver en fin de recommandation. Ceci dépend grandement de la taille de l'ensemble d'apprentissage : plus on possède d'exemples, plus ceux qui satisfont certaines valeurs seront nombreux.

Cette méthode n'est donc pas applicable dans des cas réels, mais pourra servir de bases aux algorithmes suivants.

1.2 Première solution : la classification naïve bayésienne

La classification automatique est la catégorisation d'objets, c'est-à-dire l'attribution de classes à des objets. Il peut s'agir par exemple de reconnaissance

de forme (on attribuerait la classe « cube » à une certaine image) ou de diagnostic (avec des classes « malade » et « sain »).

La recommandation de valeur en configuration peut s'interpréter comme une classification ; par exemple, sachant que l'utilisateur veut une voiture sportive et qu'on cherche à lui recommander une couleur de voiture, on pourrait lui attribuer une classe correspondant à la couleur (« bleu », « rouge », etc.) en fonction de ses choix précédents. Dans les deux cas, le nombre de classes est assez limité (typiquement de l'ordre de la dizaine).

Une configuration interactive telle que celle du problème de Renault peut alors s'interpréter comme une succession de recommandations et donc de classifications.

Une méthode classique de classification est la *classification naïve bayésienne*, présentée dans [LIT92], qui est expliquée ici.

Dans la classification naïve bayésienne, on cherche à estimer $P(C|X_1, X_2, ..., X_n)$ où C est la variable de la classe à estimer et X_1, X_2, .. X_n sont les variables auxquelles nous avons accès.

Or, d'après le théorème de Bayes :

$$\begin{aligned} P(C|X_1, X_2, ..., X_n) &= \frac{P(C) \times P(X_1, X_2, ..., X_n|C)}{P(X_1, X_2, ..., X_n)} \\ &\propto P(C) \times P(X_1, X_2, ..., X_n|C) \\ &\propto P(C) \times P(X_1|C) \times P(X_2|C, X_1) \times ... \\ &\quad \times P(X_n|C, X_1, X_2, ..., X_{n-1}) \end{aligned}$$

L'hypothèse qui sous-tend la classification naïve bayésienne est l'indépendance deux à deux des variables X_1, ..., X_n. Dans ce cas,

$$P(C|X_1, X_2, ..., X_n) \propto P(C) \times \prod_{i=1}^{n} P(X_i|C)$$

Ainsi, il suffit d'estimer n^2 tables, où n est le nombres de variables. Étant donné que le conditionnement ne porte que sur une seule variable au maximum, l'estimation se fait sur un ensemble plus grand que la méthode naïve décrite précédemment.

La limite de cette méthode provient des hypothèses d'indépendances conditionnelles. Ainsi, lorsqu'on veut recommander une valeur pour A, on suppose que B et C sont indépendants conditionnellement à A, ce qui signifie que si A est connu, B n'apporte aucune information sur C. Or, lorsqu'on veut recommander une valeur pour C, C dépend de B et de A. C'est-à-dire que dans ce cas, si A est est connu, B apporte toujours de l'information sur C. On peut donc constater que selon la variable pour laquelle on cherche à recommander une valeur, les hypothèses se contredisent.

1.3 Seconde solution : la restauration

Une autre solution à ce problème est la restauration de variable. La restauration de variable, introduit dans [Sch15], consiste en l'élimination d'une connaissance sur les choix fait par l'utilisateur durant cette session.

Posons par exemple V_s la variable booléenne « voiture sportive », V_d la variable booléenne « voiture décapotable » et V_c la variable discrète « couleur de la voiture ».

Supposons, dans le cadre d'un configurateur de voiture, qu'on cherche à recommander une valeur pour la variable $V_{reco} = V_s$ et qu'on connaisse les choix de l'utilisateur pour les variables V_c (valeur choisie : « bleue ») et V_d (valeur choisie : « non »). Si aucun utilisateur n'a précédemment choisi de voiture à la fois bleue et décapotable, alors on ne peut pas estimer la loi de V_s.

Néanmoins, on peut raisonnablement penser que le fait qu'une voiture soit décapotable influence la probabilité pour que celle-ci soit aussi sportive. Par contre, le lien entre couleur de la voiture et le fait qu'elle soit sportive ou non ne paraît pas évident.

Restaurer V_c, c'est approcher la vraisemblance que la voiture soit sportive sachant qu'elle est décapotable et bleue par la vraisemblance que la voiture soit sportive sachant qu'elle est décapotable, c'est-à-dire approcher $P(V_s|V_d, V_c)$ par $P(V_s|V_d)$, et ainsi "oublier" la valeur de V_c.

Ainsi, lorsqu'on cherchera à calculer la vraisemblance de V_s, on cherchera les exemples de voiture qui sont décapotables, peu importe leur couleur. Étant donné que la contrainte a été relaxée, il est probable que cette fois on dispose d'exemples qui permettent de calculer la vraisemblance de V_s. Bien sûr, plusieurs variables peuvent être restaurées si besoin est.

Au final, la restauration consiste donc à trouver un ensemble $\mathcal{V}_{restaurees} \subseteq \mathcal{V}_{affectees}$ et à estimer $P(V_{reco}|\mathcal{V}_{affectees})$ par $P(V_{reco}|\mathcal{V}_{affectees} \setminus \mathcal{V}_{restaurees})$.

2 Difficultés de la restauration

2.1 Compromis entre la précision et fidélité de l'estimation

Un problème qui surgit est la question du compromis entre la précision et la fidélité de l'estimation. Supposons par exemple qu'on cherche à recommander A et qu'on connaisse la valeur de B, et que A est indépendant de B. Alors, comme $P(A) = P(A|B)$, restaurer B ne modifiera pas la distribution de A ; par contre, restaurer B ne pourra qu'augmenter l'ensemble des exemples sur lequel on estimera cette distribution de A. Dans cet exemple, il n'y a que les avantages à restaurer B.

Supposons par contre que A (à recommander) et B (connue) soient faiblement dépendants. Or, dans les exemples qu'on possède, aucun ne vérifie à la fois la valeur de B et celle de A. Il est plus raisonnable de restaurer B

et d'estimer A que de ne pas pouvoir terminer les calculs. Dans ce cas-ci, l'estimation qu'on fait de A alors que B est restaurée est une estimation de $P(A)$, et pas de $P(A|B)$. Ainsi, l'estimation n'est pas fidèle à la vraie distribution ; on va seulement approcher $P(A|B)$ par $P(A)$.

Il faut donc établir un compromis entre avoir une estimation précise (qui s'obtient avec un nombre d'exemples important) et une estimation fidèle (qui s'obtient avec la perte, lors de la restauration, du minimum d'information possible).

Nous n'avons pas trouvé de solution exacte à ce problème, néanmoins nous présenterons trois heuristiques.

Ces heuristiques sont des fonctions $f_{\mathcal{V}_{reco},\mathcal{V}_{affectees}}(\mathcal{V}_{restaurees})$. L'ensemble $\mathcal{V}_{restaurees}$ qui maximise une heuristique est considérée comme la solution.

2.2 Une méthode exhaustive impossible

Cette utilisation des heuristiques souffre d'un autre problème : on ne peut pas en pratique vérifier $2^{Card(\mathcal{V}_{affectees})}$ ensembles possibles. De plus, comme la restauration se fait lorsque $Card(\mathcal{V}_{affectees})$ est grand (car plus il y a de contraintes, moins il y a d'exemples les satisfaisant), cette méthode n'est en pratique pas réalisable en temps limité.

Afin de palier à ce problème, on va utiliser une méthode plus rapide et non optimale de recherche du maximum d'une heuristique : la méthode gloutonne.

2.3 Restauration gloutonne

La méthode gloutonne, bien qu'elle ne soit pas optimale, permet par contre d'avoir une réponse rapidement. Cet algorithme choisit séquentiellement les variables à restaurer, et construit donc petit à petit l'ensemble $\mathcal{V}_{restaurees}$. En supposant que la méthode de notation ait une complexité temporelle constante, la complexité de la méthode gloutonne est quadratique en la taille de $Card(\mathcal{V}_{affectees})$ tandis que la méthode exhaustive est exponentielle.

Cet algorithme n'effectue pas la recommandation mais uniquement la restauration et renvoie $\mathcal{V}_{restaurees}$, l'ensemble des variables à restaurer.

3 Heuristique basée sur un seuil

3.1 Mesure d'indépendance

La restauration s'appuie sur la notion d'indépendance probabiliste : si A et B sont indépendants, alors le fait de restaurer B ne modifie pas la distribution de A. Ce résultat provient directement de la définition de la probabilité conditionnelle : $A \perp\!\!\!\perp B$ et $P(B) \neq 1$ et $P(B) \neq 0 \Leftrightarrow P(A|B) = P(A)$.

Algorithm 1 Restauration gloutonne

procedure RESTGLOUTON($\mathcal{V}_{affectees}, V_{reco}, f_{V_{reco},\mathcal{V}_{affectees}}$)
  $\mathcal{V}_{restaurees} \leftarrow \emptyset$
  $x \leftarrow f_{V_{reco},\mathcal{V}_{affectees}}(\mathcal{V}_{restaurees})$
  repeat
    $V \leftarrow \underset{V_i \in \mathcal{V}_{affectees} \setminus \mathcal{V}_{restaurees}}{\arg\max} f_{V_{reco},\mathcal{V}_{affectees}}(\mathcal{V}_{restaurees} \cup \{V_i\})$
    $x_{temp} \leftarrow f_{V_{reco},\mathcal{V}_{affectees}}(\mathcal{V}_{restaurees} \cup \{V\})$
    if $x_{temp} \geq x$ **then**
      $\mathcal{V}_{restaurees} \leftarrow \mathcal{V}_{restaurees} \cup \{V\}$
      $x \leftarrow x_{temp}$
      arret $\leftarrow$ `false`
    else
      arret $\leftarrow$ `true`
    end if
  until *arret*
  return $\mathcal{V}_{restaurees}$
end procedure

Une idée intuitive peut donc être de restaurer la ou les variables les plus indépendantes possibles de V_{reco}.

Afin d'estimer l'indépendance entre deux variables, on utilise des tests statistiques d'association, qui permette de mesurer la dépendance entre deux variables. Il existe différents tests d'association, donc voici quelques exemples.

Test du χ^2

Ce test a été présenté pour la première fois dans [Pea00].

Soient X et Y deux variables aléatoires supposées prendre un nombre fini de valeurs, I pour X, J pour Y. On dispose d'un échantillonnage de N données. Notons $Oind|ij$ l'effectif observé de données pour lesquelles X prend la valeur i et Y la valeur j. Sous l'hypothèse d'indépendance, on s'attend à une valeur espérée $Eind|ij$ définie comme suit :

$$T = \sum_{i,j} \frac{(O_{ij} - E_{ij})^2}{E_{ij}}$$

On montre que la loi de T suit asymptotiquement une loi du χ^2 à $(I-1)(J-1)$ degrés de liberté.

Test du χ^2 de Yates

Présenté dans [Yat34], il s'agit d'une correction du test du χ^2 dans le cas de deux variables binaires. Ce test donne dans une certaine mesure de meilleurs

résultats lorsqu'on a peu de données.

$$\chi^2_{\text{Yates}} = \sum_{i=1}^{N} \frac{(|O_i - E_i| - 0.5)^2}{E_i}$$

Test G

$$G = 2 \sum_i O_i \cdot \ln\left(\frac{O_i}{E_i}\right)$$

La somme est faite sur les E_i non nuls. Ce test suit asymptotiquement, sous l'hypothèse d'indépendance, une loi du χ^2 à $(I-1)(J-1)$ degrés de liberté. Néanmoins, l'écart à cette loi asymptotique est plus faible que pour T du test du χ^2 [HT12].

Tests basés sur la théorie de l'information

Il existe également des tests qui ne sont pas statistiques mais basés sur la théorie de l'information, comme par exemple le test de [PH14].

3.2 Seuil

Si on doit recommander une variable possédant k modalités possibles, on doit estimer $k-1$ valeurs (les k probabilités moins un degré de liberté car la somme des probabilités fait 1). On va alors procéder à une restauration jusqu'à obtenir $N \times (k-1)$ exemples. Ainsi, on disposera d'environ N exemples pour estimer chaque modalités.

Différentes valeurs de N ont été expérimentées (cf. résultats au chapitre 5) ; une valeur raisonnable est 50.

4 Cas binaire : heuristique basée sur l'erreur quadratique moyenne

4.1 Introduction

Cette section s'attelle à un examen plus formel du compromis entre fidélité et précision de l'estimation lors de la restauration, présenté page 14. Après une étude du problème, elle fournira une nouvelle heuristique. Néanmoins, cette heuristique ne fonctionne que pour des variables booléennes.

Pour cela, il est nécessaire d'introduire des notations. Supposons qu'on cherche à recommander une valeur pour une variable V_{reco} booléenne, c'est-à-dire qui ne peut prendre que deux valeurs possibles v et $\bar{v}$, sachant que les variables $\mathcal{V}_{affectees}$ ont déjà été affectées.

Cette variable V_{reco} suit donc une loi de Bernoulli de paramètre $p_{\mathcal{V}_{affectees}} = P(V_{reco} = v|\mathcal{V}_{affectees})$. Ce paramètre $p_{\mathcal{V}_{affectees}}$ est inconnu et on cherche à l'estimer.

Afin de quantifier la qualité d'un estimateur $\hat{p}$ pour un paramètre p, une méthode classique est d'estimer son EQM (erreur quadratique moyenne), qui est définie de la manière suivante :

$$\text{EQM}(\hat{p}|p) \equiv \mathbb{E}\left((\hat{p} - p)^2\right)$$

Un résultat classique montre que l'erreur quadratique moyenne peut également s'écrire :

$$\text{EQM}(\hat{p}|p) = \text{Biais}(\hat{p})^2 + \text{Var}(\hat{p})$$

où :

$$\text{Biais}(\hat{p}) \equiv E[\hat{p}] - p$$

et

$$\text{Var}(\hat{p}) \equiv E[(\hat{p} - E[\hat{p}])^2]$$

On retrouve le coût d'une perte de fidélité (le biais $\text{Biais}(\hat{p})$) et le coût d'une perte de précision (la variance $\text{Var}(\hat{p})$).

On peut associer à chaque ensemble $\mathcal{V}_{restaurees}$ une heuristique qui sera une estimation de l'erreur quadratique moyenne de l'estimateur associé à cet ensemble.

Ces erreurs quadratiques se calculent différemment selon qu'il y a ou non restauration. Nous allons donc d'abord étudier l'estimateur sans restauration puis l'estimateur avec restauration afin de pouvoir au final proposer une heuristique.

4.2 EQM de l'estimateur sans restauration

En l'absence de restauration, l'estimateur empirique est :

$$\hat{p}_{\mathcal{V}_{affectees}} = \frac{1}{n} \sum_{i=1}^{n} \delta_{v, v_{\mathcal{V}_{affectees}, i}}.$$

où δ est le symbole de Kronecker[1], et $v_{\mathcal{V}_{affectees}, i}$ est la valeur de V_{reco} dans l'exemple i qui satisfait les valeurs affectées pour $\mathcal{V}_{affectees}$. Cet estimateur compte donc la proportion d'exemples satisfaisant les valeurs affectées pour $\mathcal{V}_{affectees}$ pour lesquels V_{reco} vaut v.

[1] $\delta_{i,j} = \begin{cases} 1 & \text{si } i = j \\ 0 & \text{si } i \neq j \end{cases}$

Or, $\hat{p}_{\mathcal{V}_{affectees}}$ est l'estimateur empirique et n'est pas biaisé (c'est un résultat classique), c'est-à-dire que :

$$\text{Biais}(\hat{p}_{\mathcal{V}_{affectees}}) = 0$$

La variance de $\hat{p}_{\mathcal{V}_{affectees}}$ est :

$$\begin{aligned}
\text{Var}(\hat{p}_{\mathcal{V}_{affectees}}) &= \text{Var}\left(\frac{1}{n}\sum_{i=1}^{n}\delta_{v,v_{\mathcal{V}_{affectees},i}}\right) \\
&= \frac{1}{n^2}\text{Var}\left(\sum_{i=1}^{n}\delta_{v,v_{\mathcal{V}_{affectees},i}}\right) \\
&= \frac{1}{n^2}n\,\text{Var}(V_{reco}|\mathcal{V}_{affectees}) \\
&= \frac{\text{Var}(V_{reco}|\mathcal{V}_{affectees})}{n}
\end{aligned}$$

Or, comme V_{reco} suit une loi de Bernoulli,

$$\begin{aligned}
\text{Var}(V_{reco}|\mathcal{V}_{affectees}) &= \mathbb{E}\left[V_{reco}^2|\mathcal{V}_{affectees}\right] - \mathbb{E}[V_{reco}|\mathcal{V}_{affectees}]^2 \\
&= \mathbb{E}\left[V_{reco}|\mathcal{V}_{affectees}\right] - \mathbb{E}[V_{reco}|\mathcal{V}_{affectees}]^2 \text{ car } V_{reco} \in \{0,1\} \\
&= p_{\mathcal{V}_{affectees}} - p_{\mathcal{V}_{affectees}}^2 \\
&= p_{\mathcal{V}_{affectees}} \times (1 - p_{\mathcal{V}_{affectees}})
\end{aligned}$$

Ainsi :

$$\text{Var}(\hat{p}) = \frac{p_{\mathcal{V}_{affectees}} \times (1 - p_{\mathcal{V}_{affectees}})}{n}$$

Qu'on peut estimer par :

$$\widehat{Var}(\hat{p}_{\mathcal{V}_{affectees}}) = \frac{\hat{p}_{\mathcal{V}_{affectees}} \times (1 - \hat{p}_{\mathcal{V}_{affectees}})}{n}$$

Au final, on obtient :

$$\widehat{\text{EQM}}(\hat{p}_{\mathcal{V}_{affectees}}|p) = 0^2 + \frac{\hat{p}_{\mathcal{V}_{affectees}} \times (1 - \hat{p}_{\mathcal{V}_{affectees}})}{n}$$

4.3 EQM de l'estimateur avec restauration

Supposons maintenant qu'on cherche à recommander une valeur de V_{reco} après avoir restauré $\mathcal{V}_{restaurees} \subseteq \mathcal{V}_{affectees}$. La grandeur qui nous intéresse est :

$$p_{\mathcal{V}_{restaurees} \setminus \mathcal{V}_{affectees}} = P(V_{reco} = v|\mathcal{V}_{restaurees} \setminus \mathcal{V}_{affectees})$$

On définit de la même manière que précédemment son estimateur :

$$\hat{p}_{\mathcal{V}_{affectees} \setminus \mathcal{V}_{restaurees}} = \frac{1}{n} \sum_{i=1}^{n} \delta_{v, v_{\mathcal{V}_{affectees} \setminus \mathcal{V}_{restaurees}, i}}.$$

L'estimation de la variance de cette estimateur est analogue au cas précédent :

$$\widehat{Var}(\hat{p}_{\mathcal{V}_{affectees} \setminus \mathcal{V}_{restaurees}}) = \frac{\hat{p}_{\mathcal{V}_{affectees} \setminus \mathcal{V}_{restaurees}} \times (1 - \hat{p}_{\mathcal{V}_{affectees} \setminus \mathcal{V}_{restaurees}})}{n}$$

Néanmoins, on procède à présent à une estimation de $p_{\mathcal{V}_{affectees} \setminus \mathcal{V}_{restaurees}}$ et plus à une estimation de $p_{\mathcal{V}_{affectees}}$. Ainsi, même si $\hat{p}_{\mathcal{V}_{affectees} \setminus \mathcal{V}_{restaurees}}$ est un estimateur non biaisé de $p_{\mathcal{V}_{affectees} \setminus \mathcal{V}_{restaurees}}$, c'est un estimation a priori biaisé de $p_{\mathcal{V}_{affectees}}$ car on a a priori $p_{\mathcal{V}_{affectees} \setminus \mathcal{V}_{restaurees}} \neq p_{\mathcal{V}_{affectees}}$.

Nous allons donc devoir estimer ce biais. Pour cela, un estimateur est proposé.

$$\widehat{Biais}(\hat{p}_{\mathcal{V}_{affectees} \setminus \mathcal{V}_{restaurees}}) = \hat{p}_{\mathcal{V}_{restaurees}} - \hat{p}_{\emptyset}$$

On remarque que si $\mathcal{V}_{restaurees} = \emptyset$, on estime bien que $\widehat{Biais}(\hat{p}_{\mathcal{V}_{affectees}}) = 0$.

Cet estimateur revient, en fait, à supposer que les grandeurs suivantes sont proches :

$$\hat{p}_{\mathcal{V}_{restaurees}} - \hat{p}_{\emptyset} \simeq \hat{p}_{\mathcal{V}_{affectees}} - \hat{p}_{\mathcal{V}_{affectees} \setminus \mathcal{V}_{restaurees}}$$

Avant d'expliquer d'où vient cet estimateur, il faut remarquer que pour trouver un bon estimateur de ce biais, il faudrait pouvoir estimer son erreur quadratique moyenne. C'est-à-dire qu'il faudrait estimer une erreur quadratique moyenne afin d'en estimer une autre (celle de $\hat{p}_{\mathcal{V}_{affectees} \setminus \mathcal{V}_{restaurees}}$).

Il est raisonnable de ne pas chercher à justifier statistiquement l'efficacité de cet estimateur qui demanderait à justifier d'autres estimateurs, etc. Cet estimateur va donc être expliqué intuitivement.

Lorsqu'on restaure les variables $\mathcal{V}_{restaurees}$, on ne peut plus estimer $p_{\mathcal{V}_{affectees}}$ mais $p_{\mathcal{V}_{affectees} \setminus \mathcal{V}_{restaurees}}$. Afin d'estimer l'écart qu'il peut y avoir entre ce deux valeurs, l'écart apporté par la restauration de $\mathcal{V}_{restaurees}$ sur la loi de V_{reco}, on suppose que cet écart ne dépend pas trop des autres variables fixées.

Plus précisément, l'estimateur dit que l'influence qu'a la restauration de $\mathcal{V}_{restaurees}$ sur V_{reco} ne dépend pas de $\mathcal{V}_{affectees} \setminus \mathcal{V}_{restaurees}$. Cet estimateur a l'inconvénient de faire une relaxation importante du problème (puisque $\mathcal{V}_{affectees} \setminus \mathcal{V}_{restaurees}$ n'entre plus en compte) mais a une précision importante car il dispose de beaucoup d'exemples.

4.4 Heuristique

Au final, l'heuristique est l'opposé d'une estimation de l'erreur quadratique. Il est nécessaire de prendre l'opposé car, dans l'algorithme de la restauration gloutonne, on cherche à maximiser l'heuristique, alors que l'erreur quadratique moyenne est préférable quand elle est faible.

L'erreur quadratique est estimée par :

$$\widehat{\mathrm{EQM}}(\hat{p}_{\mathcal{V}_{affectees} \setminus \mathcal{V}_{restaurees}} | p) = (\hat{p}_{\mathcal{V}_{restaurees}} - \hat{p}_{\emptyset})^2 + \frac{\hat{p}_{\mathcal{V}_{affectees}} \times (1 - \hat{p}_{\mathcal{V}_{affectees}})}{n}$$

Néanmoins, on rappelle que cette heuristique n'est utilisable que si V_{reco} est binaire.

5 Heuristique basée sur un calcul d'entropie

Le compromis entre fidélité et précision d'une estimation présenté plus haut peut être résolu par un calcul d'entropie. Ce calcul, qui est basé sur la théorie des possibilités et développé dans l'article [SP15], permettrait de maximiser l'efficacité de la restauration.

Néanmoins, les résultats préliminaires ont été décevants, notamment en terme de vitesse d'exécution. Cette piste n'a donc pas été plus développée.

Chapitre 3

Recommandation par réseau bayésien

1 Définition et apprentissage de réseau bayésien

1.1 Motivation

Les réseaux bayésiens sont des modèles statistiques permettant de représenter des distributions de probabilité à l'aide d'un graphe orienté acyclique. Ils permettent représenter de manière compacte et facilement exploitable des distributions probabilistes. Ils sont utilisés avec succès dans les domaines de la classification automatique, de l'analyse de risques, de la détection de spam...

Un réseau bayésien permet en effet d'effectuer des calculs de probabilité, appelés *inférences*. Ceci motive l'expérimentation des réseaux bayésiens dans le contexte de la recommandation de valeur en configuration : puisqu'ils sont efficaces pour classifier, ils seront peut-être efficaces pour recommander.

1.2 Définition des réseaux bayésiens

Un réseau bayésien est un graphe orienté acyclique (*directed acyclic graph* ou DAG en anglais) qui détient l'information d'une probabilité jointe sous forme décomposée. Les nœuds du graphe sont des variables aléatoires. Ce réseau permet de représenter des dépendances entre variables (par la structure du graphe) et permet un calcul de probabilités (par des tables de probabilités).

Deux variables reliées par un arc sont directement dépendantes. Deux variables A et B qui ne sont pas reliées entre elles mais reliées à une troisième variable C sont indépendantes conditionnellement à C.

Cela signifie que si la valeur de C n'est pas connue, alors une information sur A nous donne une information sur B (et réciproquement). Par contre, si C est connue, alors A et B sont indépendantes.

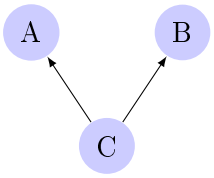

FIGURE 3.1 – Exemple de réseau bayésien à trois variables

Pour chaque variable on dispose de la loi de probabilité conditionnellement à la valeur de ses parents, ou de la loi de probabilité a priori si le nœud n'a pas de parent.

Il a été montré que chaque nœud X_i est indépendant des nœuds qui ne sont pas ses descendants, sachant ses parents [Pea89].

En notant Π_{X_i} l'ensemble des parents de X_i, la loi de Bayes nous dit que :

$$P(X_1, \ldots, X_n) = \prod_{i=1}^{n} P(X_i | \Pi_{X_i})$$

On rappelle que $n \equiv Card(\mathcal{V})$. Il existe des méthodes permettant de construire automatiquement un réseau bayésien qui encode une distribution de probabilité proche de celle qui suit un ensemble d'apprentissage. Cet apprentissage automatique se fait en deux étapes :

- l'apprentissage de la structure, qui concerne les relations de dépendance entre les variables qui composent l'objet ;
- l'apprentissage des paramètres probabilistes qui permettront ensuite d'effectuer les calculs de probabilité.

1.3 Apprentissage de la structure

L'apprentissage d'un réseau bayésien qui soit le plus probable a posteriori à partir de données est un problème NP-complet [Chi95].

Il existe deux principales approches pour l'apprentissage heuristique de réseau bayésien.

L'approche *search-and-score* : on recherche un réseau bayésien parmi tous les réseaux possibles qui maximise une fonction de score. Cette recherche peut être gloutonne, locale...

L'approche basée sur contraintes : cette approche construit le réseau en estimant les relations de dépendance conditionnelle avec des tests statistiques ou issus de la théorie de l'information.

Enfin, des méthodes hybrides existent.

1.4 Apprentissage des paramètres

L'apprentissage des paramètres dans un réseau bayésien recouvre l'apprentissage des probabilités conditionnelles. Chaque nœud dispose d'une table de probabilité conditionnellement à la valeur de ses parents.

Ces paramètres sont estimés de manière à maximiser la vraisemblance des données [NWL+07].

2 Application à la recommandation

2.1 Recommandation par réseau bayésien

Les réseaux bayésiens permettent de faire un calcul d'inférence, c'est-à-dire le calcul de la distribution d'une variable lorsque certaines variables (mais pas nécessairement toutes) sont connues. Le calcul d'inférence est un problème NP-complet [Coo90].

Nous n'avons trouvé aucune littérature sur la recommandation en configuration à l'aide de réseau bayésien. Néanmoins, les réseaux bayésiens sont très utilisés pour la classification, et la recommandation en configuration peut s'interpréter comme une classification.

La recommandation par réseau bayésien repose sur l'inférence de la variable V_{reco} à recommander conditionnellement aux valeurs des variables déjà fixées $\mathcal{V}_{affectees}$. On peut alors recommander la valeur la plus probable.

Cette méthode obtient de bons résultats mais a un problème de complexité : en effet, un algorithme NP-difficile ne garantit pas une réponse rapidement, alors que c'est souhaitable dans une application en ligne comme un configurateur.

2.2 Recommandation par réseaux bayésiens naïfs augmentés

Les réseaux bayésiens naïfs augmentés par arbre, présentés dans [FGG97], sont un cas particulier des réseaux bayésiens. Un réseau bayésien naïf augmenté par arbre de variable d'intérêt V_{reco} se construit en deux étapes :

- la construction d'un réseau bayésien sous forme d'un arbre sur toutes les variables sauf V_{reco} ;
- l'ajout de V_{reco} comme parent de tous les autres nœuds.

L'avantage de ces réseaux bayésiens naïfs augmentés par un arbre est que leur apprentissage et l'inférence sont polynomiaux.

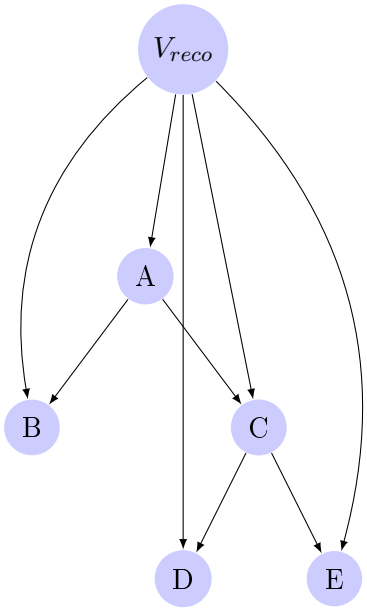

FIGURE 3.2 – Un réseau bayésien naïf augmenté par un arbre de variable d'intérêt V_{reco}

3 Restauration par d-séparation

3.1 Introduction

Comme cela a été dit précédemment, la structure d'un réseau bayésien est construite en fonction des dépendances entre les variables. Or, savoir si deux variables sont indépendantes ou non est très utile pour la restauration, car on sait que la restauration d'une variable indépendante à V_{reco} ne peut qu'améliorer la précision de la recommandation.

Afin d'utiliser un réseau bayésien et d'en déduire des relations d'indépendance, on utilise la notion de d-séparation.

3.2 La d-séparation

On définit la d-séparation dans un graphe orienté acyclique, telle que présentée dans [Pea89], sous la forme de deux règles.

Ces règles utilisent la notion de d-connectivité. On dit de deux nœuds qu'ils sont d-connectés si et seulement si ils ne sont pas d-séparés.

Dans les schémas suivants, $\mathcal{Z}$, l'ensemble des nœuds conditionnés (dont on connaît la valeur), est coloré en rouge. X est d-connecté à Y par l'ensemble $\mathcal{Z}$ est noté $\langle X|\mathcal{Z}|Y\rangle$.

On appelle collision tête-à-tête dans un chemin de A à B le nœud C appartenant à ce chemin pour lequel à la fois le nœud précédent et le nœud suivant dans ce chemin ont leur arc qui pointent vers C. C ne peut donc

être ni A (qui n'a pas de prédécesseur dans ce chemin) ni B (qui n'a pas de successeur dans ce chemin).

Blocage par conditionnement

A et B sont d-connectés s'il existe un chemin (sans prendre en compte les directions des arcs) sans collision tête à tête qui ne traverse aucun nœud de $\mathcal{Z}$.

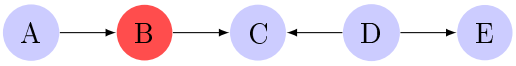

FIGURE 3.3 – Exemple pour le blocage par conditionnement. $\mathcal{Z} = \{B\}$

Dans cette figure, A est n'est pas d-connecté à C par rapport à $\mathcal{Z}$.

Conditionnement par collision

A et B sont d-connectés par rapport à $\mathcal{Z}$ s'il existe un chemin (sans prendre en compte les directions des arcs) dont les nœuds avec collision tête à tête appartiennent à $\mathcal{Z}$ ou possèdent un descendant qui appartient à $\mathcal{Z}$.

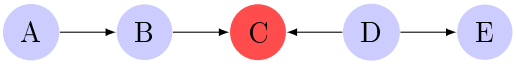

FIGURE 3.4 – Exemple pour le conditionnement par collision. $\mathcal{Z} = \{C\}$

Dans cette figure, tous les nœuds sont d-connectés deux à deux.

3.3 La propriété de Markov globale orientée

Les réseaux bayésiens vérifient la propriété de Markov globale orientée [Ric97], c'est-à-dire que :

$$\forall X, Y \in \mathcal{V}, \forall \mathcal{Z} \subseteq \mathcal{V} \setminus \{X, Y\} \, \langle X | \mathcal{Z} | Y \rangle \Rightarrow X \perp\!\!\!\perp Y \,|\, \mathcal{Z}$$

où $X \perp\!\!\!\perp Y \,|\, \mathcal{Z}$ est une indépendance conditionnelle définie par

$$X \perp\!\!\!\perp Y \,|\, \mathcal{Z} \iff P(X, Y \,|\, \mathcal{Z}) = P(X \,|\, \mathcal{Z}) \times P(Y \,|\, \mathcal{Z})$$

Ainsi, dans un réseau bayésien, si deux variables A et B sont d-séparées par un ensemble de variables $\mathcal{Z}$ alors A et B sont indépendants conditionnellement à $\mathcal{Z}$. Cette d-séparation permet habituellement de simplifier l'inférence en limitant le nombre de variables qui ont une incidence sur la distribution d'une variable d'intérêt.

Or, la restauration se base sur l'estimation d'indépendance. Dans ce cas, on peut imaginer avoir, en plus d'un algorithme de calcul de fréquences du chapitre 2, la structure d'un réseau bayésien appris afin d'effectuer une

restauration grâce au réseau bayésien. Dans cette idée, il n'y aurait aucune inférence bayésienne : l'utilité du réseau bayésien est de fournir une structure compacte de relations d'indépendance afin de choisir $\mathcal{V}_{restaurees}$. Le calcul de la valeur à recommander se fera comme dans le chapitre 2.

3.4 Application à la restauration

Bien que l'inférence soit NP-difficile, le calcul de d-séparation est linéaire [GVP13] en la taille du réseau bayésien. Il est donc possible de l'utiliser dans la recommandation par restauration : en effet, lorsqu'on connaît les valeurs des variables $\mathcal{V}_{affectees}$ précédemment fixées, on peut alors connaître les variables $\mathcal{V}_{restaurees} \subseteq \mathcal{V}_{affectees}$ qui pourraient être restaurées (c'est-à-dire retirées de $\mathcal{V}_{affectees}$) sans incidence sur la distribution de la variable d'intérêt.

Ainsi, on obtient $\mathcal{V}_{restaurees}$ le plus grand sous-ensemble de $\mathcal{V}_{affectees}$ tel que $\mathcal{V}_{restaurees}$ est d-séparé de la variable d'intérêt V_{reco} par $\mathcal{V}_{affectees} \setminus \mathcal{V}_{restaurees}$. Les variables de $\mathcal{V}_{restaurees}$ peuvent donc être restaurées sans que cela n'affecte la variable d'intérêt.

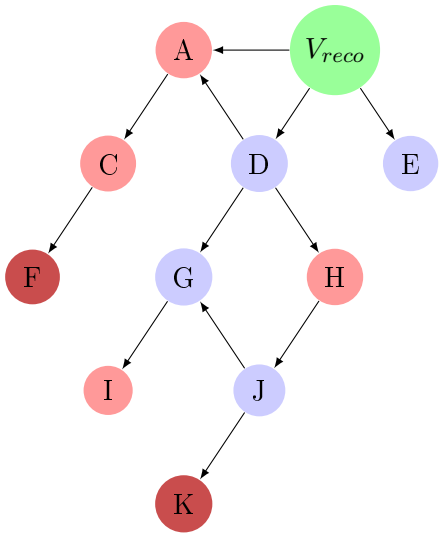

FIGURE 3.5 – Exemple de restauration par d-séparation

Dans cet exemple, on cherche à recommander V_{reco} en connaissant $\mathcal{V}_{affectees} = \{A, C, F, H, I, K\}$. On remarque que V_{reco} est d-séparé de F et K sachant $\mathcal{V}_{affectees} \setminus \{F, K\}$. On va donc restaurer les variables F et K, c'est-à-dire que $\mathcal{V}_{restaurees} = \{F, K\}$.

Chapitre 4

La compilation par SLDD

1 Les langages de compilation

1.1 Motivation

Nous avons présenté dans les chapitres 2 et 3 des algorithmes permettant de recommander une valeur pour V_{reco}. Néanmoins, certaines des opérations utilisées dans ces algorithmes peuvent être complexes ; par exemple, l'inférence dans un réseau bayésien est NP-difficile [Coo90], ce qui rend en l'état son application en ligne impossible.

Il existe néanmoins des structures de données pour lesquelles ces opérations sont plus rapides. Comme le temps d'exécution est un critère important afin d'évaluer ces algorithmes, nous avons recherché des structures de données adaptées.

Les structures de données étudiées afin de minimiser la complexité de certaines opérations s'appellent *langages de représentation* et la transformation de données vers un langage de représentation s'appelle la *compilation*.

De plus, on appelle les opérations qui modifient les données représentées des *transformation* et les autres opérations des *requêtes*.

Le sujet des langages de représentation étant vaste, nous nous limiterons à en présenter quelques uns. Le lecteur intéressé pourra consulter [DM02] pour une présentation plus approfondie des langages binaires et [FM07] pour les langages valués.

1.2 La carte de compilation

Le premier critère qui permet de comparer deux langages est leur expressivité. Un langage est dit *complet* s'il peut représenter toute proposition logique (dans le cas d'un langage binaire) ou toute fonction à domaine discret et à valeurs discrètes (dans le cas valué). Tous les langages qui seront décrits dans ce chapitre seront complets dans leur domaine respectif.

Tel que le présente [DM02], trois critères permettent de comparer deux langages complets :

- La compacité théorique. La compacité théorique d'un langage est relative à un autre. On dit d'un langage L_1 qu'il est au moins aussi compact qu'un langage L_2 (noté $L_1 \leq L_2$) si, pour toute fonction f, la taille de tout α représentant f dans L_1 est inférieure ou égale, à un polynôme près, à la taille d'un β représentant f dans L_2. On dit alors que L_1 est plus compact que L_2 (noté $L_1 < L_2$) si $L_1 \leq L_2$ et que $L_2 \not\leq L_1$.
- La complexité des requêtes ;
- La complexité des transformations.

On peut remarquer que la compacité théorique permet seulement de s'assurer qu'il ne peut pas y avoir explosion de taille en passant d'un langage à un autre. Néanmoins, même si L_1 est plus compact que L_2, cela ne signifie pas que taille de la représentation de f dans L_1 soit nécessairement plus petite que la représentation de f dans L_2.

Généralement, afin de diminuer la complexité de requêtes, on va complexifier le langage et le rendre moins compact ; l'inverse est aussi vrai : un langage compact aura tendance à avoir une mauvaise complexité temporelle sur les requêtes. C'est une illustration du compromis temps-espace qu'on retrouve souvent en algorithmique. Les transformations peuvent devenir également plus complexe si le langage est restrictif : en effet, il faut s'assurer que ce qui résulte de la transformation soit encore licite dans ce langage.

Ces critères de comparaison sont regroupés dans une *carte de compilation* [DM02], qui permet de comparer facilement des langages et choisir le plus adéquat.

2 Les diagrammes de décision

2.1 Définition des diagrammes de décision

Les diagrammes de décision forment une famille de langages. Un diagramme de décision est un graphe acyclique orienté ne possédant qu'une seule racine et comportant des nœuds de décision étiquetés par une variable. De chaque nœuds de décision sortent des arcs correspondant à une affectation possible de la variable de ce nœud. Les nœuds terminaux sont quant à eux étiquetés par une valeur.

On y définit la notion de chemin, qui est une suite d'arcs partant de la racine et finissant à un nœud terminal.

Enfin, un diagramme de décision est dit réduit si, pour chaque nœud de décision, les arcs qui en sortent ne pointent pas tous vers le même nœud. Sinon, le nœud peut être retiré sans perte d'information.

2.2 Définition du langage OBDD

Un diagramme de décision binaire ordonné (OBDD, *ordered binary decision diagram* [Bry86]), est un diagramme de décision :

binaire : les variables sont booléennes et donc les affectations possibles pour chaque variable sont « vrai » et « faux » (ce qui fait qu'un nœud de décision a deux arcs). Il n'y a que deux nœuds terminaux, étiquetés par les valeurs $\top$ (vrai) et $\bot$ (faux).

ordonné : quelque soit le chemin de la racine à un nœud terminal, l'ordre des variables rencontrées sera le même.

Il s'agit d'un langage booléen complet, qui peut représenter toute proposition logique.

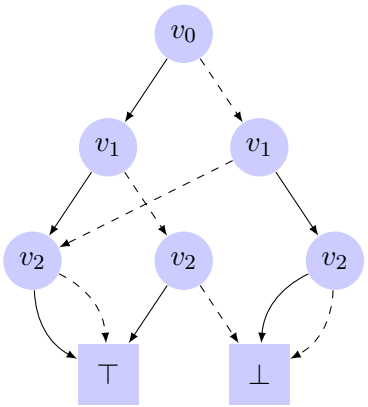

FIGURE 4.1 – OBDD non réduit représentant la fonction $(v_0 \wedge v_1) \vee (v_0 \wedge \bar{v_1} \wedge v_2) \vee (\bar{v_0} \wedge \bar{v_1})$. Les traits plein représentent $v_i = vrai$ et les traits discontinus $v_i = faux$

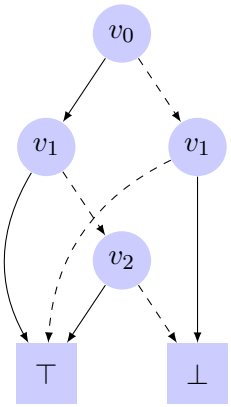

FIGURE 4.2 – Le même OBDD après réduction

2.3 Définition du langage ADD

Les ADD (*Algebraic Decision Diagrams*) sont une généralisation des OBDD à un domaine discret et non plus booléen [BFG+93] : les variables ainsi que le domaine d'arrivée sont discrets. Il peut donc y avoir plus de deux arcs qui sortent d'un nœud de décision. Les deux nœuds terminaux $\top$ et $\bot$ sont remplacés par un ou plusieurs nœuds terminaux valués.

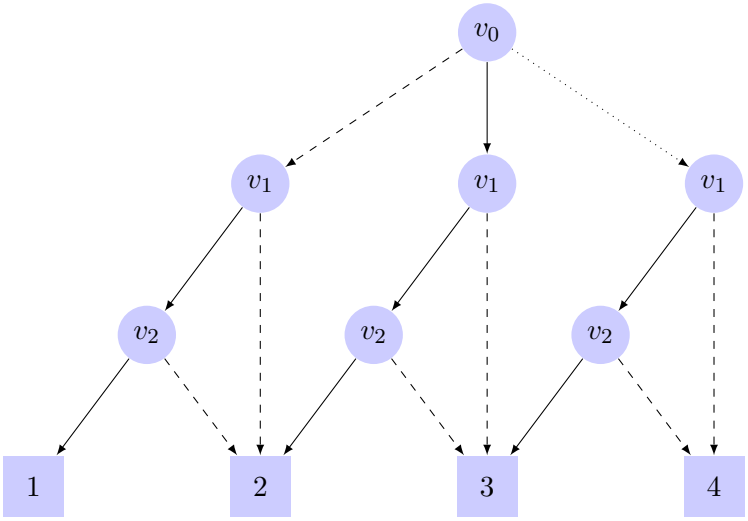

FIGURE 4.3 – ADD réduit représentant la fonction $v_0 - v_1 \times v_2 + 2$. Les traits discontinus représentent $v_i = 0$, les traits plein $v_i = 1$ et les pointillés $v_i = 2$.

2.4 Définition du langage SLDD

Le langage SLDD (*Semiring Labelled Decision Diagram*), introduit dans [Wil05], est un diagramme de décision valué, auquel est adjoint une structure de valuation $\mathcal{E} = \langle E, \otimes, \succ \rangle$. Dans un SLDD, il n'y a qu'un seul nœud terminal qui porte comme valeur l'élément neutre de $\otimes$ et ce sont les arcs qui sont valués. Le langage SLDD est plus compact que le langage ADD ([FMS13]).

Dans ce cas, pour un ensemble quelconque de valeurs des variables, la valeur de la fonction représentée par un SLDD est l'agrégation par l'opérateur $\otimes$ de chaque valeur étiquetée sur le chemin défini par les valeurs des variables.

De plus, un SLDD est dit normalisé si pour chaque nœud de décision, la plus petite valeur portée par les arcs qui en sortent est l'élément neutre de $\otimes$. Un *offset*, c'est-à-dire un arc valué pointant vers la racine, peut être ajouté.

On note en particulier $SLDD_+$ le sous-langage pour lequel $\otimes = +$ et $SLDD_\times$ le sous-langage pour lequel $\otimes = \times$. On peut montrer que $SLDD_+$ n'est pas plus compact que $SLDD_\times$ et que, réciproquement, $SLDD_\times$ n'est pas plus compact que $SLDD_+$ ([FMS13]) : suivant les problèmes, l'un peut

être plus succinct que l'autre.

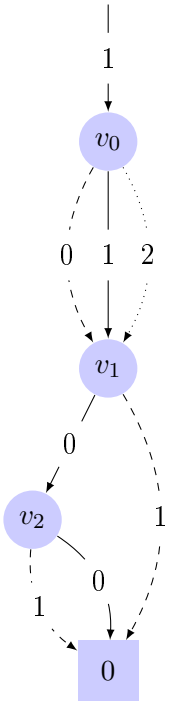

FIGURE 4.4 – $SLDD_+$ réduit et normalisé représentant la même fonction $v_0 - v_1 \times v_2 + 2$. Les traits discontinus représentent $v_i = 0$, les traits plein $v_i = 1$ et les pointillés $v_i = 2$.

Comme on peut le constater sur cet exemple, l'agrégation peut grandement réduire la taille du diagramme. Si la fonction à représenter est de nature additive, comme des coûts ou des prix, on utilise un de préférence $SLDD_+$. Si elle est de nature multiplicative, comme des probabilités, on utilise de préférence un $SLDD_\times$.

3 Quelques propriétés des SLDD

3.1 La compilation d'historique

Un historique tel que celui fournit par Renault peut être interprété comme une fonction à n variables (ces variables étant les mêmes que celles considérées lors de la configuration) qui donne, pour chaque configuration de voitures possible, le nombre de voitures vendues.

En tant que fonction discrète de plusieurs variables, elle peut être représentée par un SLDD de hauteur n. La nature additive du nombre de voitures nous permet intuitivement de penser à compiler l'historique sous la forme

d'un $SLDD_+$. En effet, on calcule des fréquences, ce qui revient à compter des exemples. Pour compter les voitures bleues, sans imposer le fait qu'elles aient ou non un toit ouvrant électrique, on va additionner le nombre de voitures bleues avec toit ouvrant électrique et le nombre de voitures bleues sans toit ouvrant électrique.

3.2 Les requêtes nécessaires au calcul de fréquences

Afin d'effectuer une recommandation rapide, il faut pouvoir effectuer certaines opérations rapidement :

- la transformation de conditionnement **CD**, qui affecte une valeur à une variable ;
- le déconditionnement, qui désaffecte une variable, ne peut pas être directement utilisé sur un SLDD (car il y aurait apparition d'information). On peut par contre simuler la désaffectation de V_0 en reprenant le SLDD compilé, qu'on aurait sauvegardé, et en réaffectant toutes les variables sauf V_0.
- la requête de comptage pondéré des produits, qui permet de savoir combien de voiture sont présentes dans un SLDD.

La requête de comptage pondérée n'est pas une requête classique. La dénomination « pondéré » vient du fait qu'on ne compte pas le nombre de voitures différentes (ce qui est une requête classique) mais compte les doublons.

Enfin, on peut y ajouter la requête de comptage de fréquences de la variable V_{reco} sachant $\mathcal{V}_{affectees}$ (on rappelle que $V_{reco} \notin \mathcal{V}_{affectees}$), qui peut s'exprimer en fonction des opérations ci-dessus :

1. conditionner toutes les variables de $\mathcal{V}_{affectees}$ aux valeurs choisies par l'utilisateur ;
2. compter le nombre d'exemples total N_{total}
3. pour chaque valeur v possible de V_{reco}, faire
   (a) conditionner V_{reco} à v
   (b) compter le nombre d'exemples N_v
   (c) déconditionner V_{reco}
4. déconditionner toutes les variables de $\mathcal{V}_{affectees}$
5. on peut alors estimer $P(V_{reco} = v \mid \mathcal{V}_{affectees})$ par $\frac{N_v}{N_{total}}$

Ces opérations sont suffisantes pour les algorithmes du chapitre 2 qui se basent sur le calcul de fréquences.

Enfin, les algorithmes de restauration nécessitent également de connaître le nombre d'exemples, ce qui est fait par la requête de comptage pondéré.

Or, d'après la carte de compilation du langage $SLDD_+$ [FMNS14], **CD** est réalisable en temps polynomial. Le comptage pondéré est lui linéaire en la taille du $SLDD_+$. On peut donc en déduire que la requête de comptage de fréquences est également réalisable en temps polynomial (cette requête peut s'apparenter à $+$**Marg**). Ces résultats mettent en l'avant l'adéquation du langage $SLDD_+$ avec les contraintes temporelles d'un configurateur en ligne.

3.3 Un SLDD pour résoudre des contraintes

Un réseau de contraintes est un ensemble de contraintes. Une contrainte peut être

souple : elle associe aux n-uplet qui la satisfont une valuation ψ ;

dure : elle associe aux n-uplet qui la satisfont l'élément absorbant de $\mathcal{E}$ ($+\infty$ pour un $SLDD_+$, 0 pour un $SLDD_\times$) [FMS14].

On peut interpréter le réseau de contraintes comme une fonction qui peut donc être représentée par un SLDD. Les requêtes alors utilisées sont **CD** (le conditionnement, c'est-à-dire l'affectation d'une variable) et **CO** (la vérification de cohérence globale, qui permet de savoir s'il existe au moins une solution).

3.4 Un SLDD pour inférer

Un $SLDD_\times$ peut être utilisé pour compiler un réseau bayésien [Sch15]. La complexité de l'inférence passe alors de NP-difficile à linéaire en la taille du $SLDD_\times$.

Chapitre 5

Expériences

1 Protocole

1.1 Motivation

La recommandation en configuration en ligne nécessite d'avoir des algorithmes à la fois rapides et efficaces. En effet, avoir une recommandation qui aide l'utilisateur et le but initialement recherché ; la contrainte temporelle provient du besoin de recommander parallèlement un grand nombre d'utilisateurs avec une puissance de calcul raisonnable.

Afin de mesurer la précision en recommandation d'un algorithme, on va décrire un algorithme « oracle » qui aura un taux de succès indépassable et constituera ainsi une borne supérieure.

La méthode de référence de l'état de l'art, la recommandation par inférence dans un réseau bayésien, sera comparée aux les autres algorithmes de ce mémoire.

1.2 Validation croisée

L'évaluation d'un algorithme de recommandation en configuration se fait par validation croisée. Il y a dix jeux de 10 000 observations (dont l'origine est présentée au chapitre 6). Il y a dix itérations. A chaque itération i, l'algorithme dispose d'exemples (qui proviennent de l'ensemble dit d'apprentissage) et devra faire des recommandations sur d'autres exemples (qui proviennent de l'ensemble dit d'apprentissage).

l'ensemble de test est le jeu de données i et l'ensemble d'apprentissage est composé des neuf autres ensembles.

Lors de l'apprentissage, l'algorithme dispose de produit entièrement configurés qui ont été sélectionnés par des utilisateurs. Ces produits satisfont les contraintes. Durant la validation, on utilise une session de configuration scénarisée. Ce scénario contient les ordres d'affectation des variables

pour chaque configuration. Le configurateur dispose également du fichier de contraintes.

Afin de focaliser les efforts sur la recommandation, nous fournissons pour chaque scénario et pour chaque variable les valeurs disponibles lorsque les choix précédents ont été fait conformément au scénario. Cela signifie qu'un algorithme qui n'est pas capable résoudre un CSP peut tout de même être évalué par ce protocole.

Enfin, on peut remarquer qu'on demande d'estimer une seule valeur et non une hiérarchie de préférences. Ce choix provient de la nature des données dont nous disposons, qui nous permettent de vérifier uniquement le premier choix. Néanmoins, tous les algorithmes présentés par la suite sont capables de fournir une recommandation constituée de plusieurs valeurs ordonnées par ordre de probabilité décroissante.

1.3 Cas triviaux

La recommandation de la valeur d'une variable s'effectue en connaissant les précédentes valeurs choisies par l'utilisateur dans cette session.

Du fait de la présence de contraintes, certaines valeurs de la variable à recommander peuvent être interdites. Par exemple, dans la configuration d'une voiture, le choix d'une voiture décapotable interdit le choix d'un toit ouvrant électrique.

Il peut ainsi arriver que la variable à recommander n'ait plus qu'une seule valeur licite. On nomme un tel cas un *cas trivial*.

Étant donné que l'algorithme de recommandation ne prend aucune décision lors d'un cas trivial, ces cas seront ignorés dans le calcul de son taux de réussite.

La proportion de cas triviaux varie selon les contraintes et les jeux de données. Par exemple, le taux de cas triviaux avec le jeu et les contraintes *small* est de 72 %.

Dans le cas où le configurateur cache à l'utilisateur les cas triviaux, le taux de succès ainsi calculé (cas triviaux ignorés) reflétera donc fidèlement le taux de réussite ressenti par l'utilisateur.

1.4 Jeux de données

Les algorithmes de recommandation disposent pour l'apprentissage des fichiers suivants :

contraintes.xml : il s'agit du fichier des contraintes que satisfont les produits. Toutes les algorithmes doivent recommander une valeur qui soit compatible avec les solutions (une telle valeur existe). C'est un fichier au format XCSP 2.1 (XML).

set0_scenario.xml ... set9_scenario.xml : ces fichiers de scénario sont utilisés uniquement lors de l'évaluation, et donnent l'ordre dans lequel les variables devront être recommandées. Ce sont des fichiers au format XML, qui suivent le format suivant :

```
<session>
        <affect var="v0" val="2" dom="0 1 2 3 4">
        <affect var="v4" val="4" dom="2 4">
</session>
```

Dans cet exemple, la session de configuration s'est faite sur deux variables. Tout d'abord la variable v_0 a été affectée à la valeur 2 ; les contraintes imposaient à v_0 de prendre une valeur parmi $\{0, 1, 2, 3, 4\}$. Après l'affectation de v_0, v_4 a été affectée à la valeur 4 ; les valeurs possibles pour v_4 sachant que $v_0 = 2$ étaient $\{2, 4\}$.

set0_exemples ... set9_exemples : ces fichiers contiennent les exemples d'apprentissage. Ce sont des fichiers au format XML, qui suivent le format suivant :

```
<exemple>
        <value var="v0" val="5">
        <value var="v1" val="8">
</exemple>
```

Dans cet exemple, v_0 a été affectée à la valeur 5 et v_1 à la valeur 8. Il n'y a pas d'informations sur l'ordre d'affectation : on ne sait pas si v_0 a été affectée avant ou après v_1.

1.5 Protocole d'évaluation

Input :

- les ensembles d'apprentissage
- le fichier de contraintes
- le scénario de test.

1. Traitement des ensembles d'apprentissage
2. Pour chaque session du scenario de test :
   (a) Pour chaque balise affect
      i. Calculer la valeur recommandée sachant le domaine des valeurs possibles et les précédents choix de l'utilisateur lors de la session.

ii. Mémoriser : trivial (le domaine des valeurs est réduit à un seul élément), échec ou succès.

Output :

- Le taux de succès (nombre de succès sur la somme du nombre de succès et d'échecs).
- Le taux de succès évolutif, c'est-à-dire le taux de succès en fonction du nombre de variables affectées dans la session

2 Algorithme « Oracle »

2.1 Idée intuitive

Voici sur un exemple simple l'idée intuitive qui sera détaillée dans les paragraphes suivants.

Supposons qu'on cherche à recommander une valeur pour le paramètre V_{reco} « toit ouvrant électrique » en connaissant l'ensemble des valeurs de $\mathcal{V}_{affectees}$.

Parmi toutes les personnes qui ont fait les mêmes choix pour $\mathcal{V}_{affectees}$, supposons que 70% des personnes prennent le toit ouvrant et 30% non.

Dans ce cas, quelle valeur faut-il recommander ? Plutôt le toit ouvrant, car on aura raison 7 fois sur 10 ; et c'est le meilleur ratio qu'on peut obtenir. Mais cela montre également qu'on ne peut pas avoir toujours raison, qu'il y a un taux de succès qu'on ne peut pas dépasser : quelle que soit la valeur qu'on recommande, il y a toujours une certaine probabilité non-négligeable que l'utilisateur choisisse une autre valeur.

Ce taux de succès maximal est donné par l'algorithme « Oracle » expliqué après. Il ne s'agit pas vraiment d'un algorithme de recommandation mais plutôt d'un moyen de calcul pour connaître le taux maximal qu'on ne pourra jamais dépasser.

2.2 Majorant du taux de succès

La recommandation peut s'interpréter comme un problème de classification. On cherche à en maximiser le taux de succès, c'est-à-dire à minimiser le taux d'erreur. Ce taux d'erreur peut-être interprété comme l'espérance de la fonction de perte 0/1, qui comptabilise 1 pour une erreur de prédiction (quelle qu'elle soit) et 0 pour une prédiction correcte.

$$\mathcal{L}_{01}(x, \hat{x}) = \begin{cases} 0 & \text{si } x = \hat{x} \\ 1 & \text{sinon} \end{cases}$$

Supposons qu'on connaisse la loi d'une variable discrète X et qu'on cherche à en faire une prédiction. Alors si la fonction de perte qu'on utilise est la fonction de perte $0/1$, la prédiction qui minimise cette perte est la valeur la plus probable.

En effet, pour une variable aléatoire X dont on connaît la loi $P(X)$:

$$\begin{aligned}\mathbb{E}[\mathcal{L}_{01}(x,\hat{x})] &= \sum_{x_i} P(x_i) \times \mathcal{L}_{01}(x_i,\hat{x}) \\ &= \sum_{x_i \neq \hat{x}} P(x_i) \\ &= 1 - P(\hat{x})\end{aligned}$$

Ainsi, minimiser $\mathbb{E}[\mathcal{L}_{01}(x,\hat{x})]$ revient à maximiser $P(\hat{x})$, et donc à choisir :

$$\hat{x} = \arg\max_{x_i} P(x_i)$$

En effet, P est supposé connu.

2.3 Algorithme oracle

Cela signifie que l'algorithme de recommandation qui maximise son taux de réussite en connaissant l'ensemble de test (et donc P) sur lequel on estime son erreur est l'estimateur du maximum de vraisemblance. On appelle cet algorithme *oracle*.

Or, lorsqu'un algorithme de recommandation ne connaît pas l'ensemble de test sur lequel on estime son erreur, ce qui est le cas lors de la validation croisée, il ne peut que faire moins bien que l'oracle car il dispose de moins d'informations que lui. En effet, dans ce cas on doit estimer P par $\tilde{P}$.

On peut donc en conclure que le taux de réussite des algorithmes de recommandation ne peut excéder le taux de réussite de l'oracle. Ceci ne veut pas non plus dire que le taux de réussite de l'oracle est atteignable (il ne l'est pas a priori). Nous avons donc un majorant du taux de succès moyen d'une session de recommandation.

3 Implémentations logicielles utilisées

3.1 Compilateur SALADD

Le compilateur SLDD utilisé est SALADD [Sch15]. Comme cela est expliqué dans le chapitre 4, les SLDD permettent le calcul de fréquences, la résolution de contraintes ainsi que l'inférence bayésienne.

3.2 Package BNlearn

Le package BNlearn pour le langage R fut utilisé pour l'apprentissage de réseau bayésien, en utilisant l'algorithme MMHC [TBA06], une méthode de référence. Il s'agit d'un algorithme d'apprentissage hybride : il cherche le squelette du graphe (c'est-à-dire une version non orientée du graphe) avec des tests d'association et l'oriente avec une recherche gloutonne.

4 Résultats

4.1 Introduction

Ces résultats ont été obtenus en utilisant une machine aux caractéristiques suivantes :

- 2 Processeurs AMD Opteron bi-coeur 2220 à 2,8Ghz
- 16 Go de RAM
- 110 Go de disque temporaire

Les programmes n'utilisaient qu'un seul cœur de cette machine.

Tous les résultats ont été générés à partir du jeu de données « small » de Renault[1]. Les grandeurs pour lesquelles il y avait moins de 1000 exemples ont été considérées comme non-signifiantes et ont été remplacées par $\star$.

ALEAT : un algorithme qui recommande équiprobablement une valeur possible.

ORACLE : l'oracle décrit ci-dessus.

DSEP50 : recommandation par restauration par d-séparation, puis avec une restauration à seuil (seuil vaut 50).

DSEP100 : recommandation par restauration par d-séparation, puis avec une restauration à seuil (seuil vaut 100).

SEUIL50 : recommandation par restauration par seuil, avec un seuil de 50.

SEUIL100 : restauration par d-séparation, avec un seuil de 100.

NAIF : recommandation par classification bayésienne naïve.

RB : recommandation par réseau bayésien.

[1] accessible sur http ://www.irit.fr/ Helene.Fargier/BR4CP/benches.html

	Durée moyenne d'une recommandation (ms)	Taux de réussite moyen (%)
ALEAT		44.76
NAIF	0.40	67.21
DSEP50	80.88	79.44
DSEP100	85.17	79.44
SEUIL50	98.62	79.42
SEUIL100	106.79	79.43
RB	117.05	79.46
ORACLE		80.01

TABLE 5.1 – Durée moyenne d'une recommandation

4.2 Interprétation

On peut constater que la classification bayésienne naïve se démarque des autres algorithmes à la fois par sa rapidité et son efficacité moyenne. Il s'agit d'une méthode à privilégier si la puissance de calcul est très faible devant le besoin.

Les autres méthodes offrent des résultats en taux de réussite élevés, puisqu'ils sont à moins de 1% du majorant obtenu par l'oracle. C'est alors le temps d'exécution qui devient différenciant.

On peut de plus observer qu'abaisser le seuil semble donner des résultats plus rapides sans perte mesurable d'efficacité.

La différence principale entre les algorithmes DSEP50 et DSEP100 et les algorithmes SEUIL50 et SEUIL100 est que ces premiers ont tendance à restaurer plus de valeurs, comme cela peut se voir dans le taux de réussite en fonction du nombre de restauration, où les algorithmes SEUIL50 et SEUIL100 restaurent rarement plus de 26 variables (table 5.3). Alors que DSEP50 et DSEP100 fonctionnent bien en restaurant entre 60% et 90% des variable, SEUIL50 et SEUIL100 fonctionnent mieux en restaurant entre 40% et 70% (table 5.4).

Il peut paraître naturel que plus $Card(\mathcal{V}_{affectees})$ est grand, plus on a d'information sur V_{reco} et donc plus on peut recommander facilement une valeur. Or, bien que tous les autres algorithmes ont un taux de succès qui croît avec $Card(\mathcal{V}_{affectees})$, la classification bayésienne naïve voit son taux de succès décroître (table 5.2).

$Card(\mathcal{V}_{affectees})$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
ORACLE	72.93	76.02	78.04	79.34	80.07	79.97	79.96	80.05	79.59	79.86	79.95	79.68	79.71	80.21	79.94	79.98
DSEP50	72.95	75.97	77.98	79.28	80.11	80.44	80.08	79.92	79.79	79.89	79.41	79.73	79.80	79.42	79.46	79.82
DSEP100	72.95	75.97	77.98	79.28	80.11	80.43	80.08	79.92	79.78	79.89	79.41	79.73	79.79	79.42	79.46	79.83
SEUIL50	72.95	75.97	77.98	79.28	80.11	80.45	80.09	79.92	79.77	79.91	79.40	79.73	79.77	79.42	79.46	79.77
SEUIL100	72.95	75.97	77.98	79.27	80.11	80.45	80.08	79.91	79.77	79.91	79.41	79.73	79.76	79.42	79.46	79.78
NAIF	64.18	74.46	77.23	78.58	78.63	77.72	76.02	74.72	73.15	71.77	70.32	69.16	68.33	67.33	65.78	65.70
RB	72.96	75.97	77.98	79.28	80.13	80.43	80.07	79.93	79.82	79.91	79.44	79.75	79.81	79.46	79.48	79.84

$Card(\mathcal{V}_{affectees})$	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
ORACLE	80.22	80.21	80.51	80.71	80.74	81.25	81.38	80.94	81.74	81.54	81.81	81.84	81.64	82.37	81.50	82.50
DSEP50	79.55	79.52	79.77	79.67	79.92	80.11	79.91	80.61	80.71	80.02	81.00	80.56	80.56	80.95	80.65	81.12
DSEP100	79.54	79.51	79.76	79.64	79.95	80.12	79.91	80.63	80.73	80.01	80.99	80.56	80.52	80.95	80.63	81.14
SEUIL50	79.56	79.57	79.73	79.65	79.93	80.13	79.97	80.57	80.67	80.00	80.93	80.56	80.53	80.95	80.59	81.13
SEUIL100	79.55	79.57	79.75	79.62	79.97	80.14	79.96	80.60	80.67	79.98	80.93	80.54	80.54	80.97	80.63	81.16
NAIF	64.30	63.67	63.33	62.96	62.53	62.40	62.09	61.37	61.98	61.65	61.21	61.45	61.01	60.79	61.22	60.68
RB	79.58	79.57	79.76	79.69	80.04	80.10	79.96	80.62	80.72	80.24	80.96	80.59	80.56	80.84	80.55	81.20

$Card(\mathcal{V}_{affectees})$	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
ORACLE	82.40	82.49	82.40	82.49	82.33	82.74	83.16	82.97	83.59	83.31	83.53	83.66	83.52	83.48	83.86	83.56
DSEP50	80.42	81.38	80.88	81.26	81.14	81.28	81.48	81.48	81.88	81.79	81.77	81.58	81.90	81.45	81.92	81.71
DSEP100	80.44	81.39	80.87	81.23	81.16	81.27	81.49	81.51	81.88	81.78	81.79	81.60	81.87	81.45	81.92	81.70
SEUIL50	80.31	81.25	80.80	81.15	81.07	81.19	81.35	81.51	81.78	81.73	81.77	81.60	81.70	81.42	81.81	81.71
SEUIL100	80.36	81.35	80.79	81.17	81.04	81.25	81.38	81.51	81.87	81.73	81.77	81.62	81.78	81.39	81.83	81.69
NAIF	60.46	59.84	59.58	60.68	59.99	60.38	60.29	59.51	60.45	60.62	60.14	59.99	60.47	60.21	59.83	60.39
RB	80.41	81.49	80.89	81.21	81.27	81.37	81.57	81.53	81.92	81.89	81.75	81.63	81.84	81.30	81.97	81.70

TABLE 5.2 – Taux de réussite en fonction de $Card(\mathcal{V}_{affectees})$

$Card(\mathcal{V}_{restaurees})$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
DSEP50	77.43	79.42	79.47	79.28	79.42	79.42	80.02	79.95	79.81	80.11	80.30	80.50	80.67	80.86	81.33	80.86
DSEP100	77.42	79.42	79.40	79.21	79.35	79.35	79.95	79.97	79.82	80.01	80.24	80.40	80.69	80.78	81.24	80.89
SEUIL50	79.10	87.07	82.20	84.35	83.94	83.93	85.13	85.51	85.76	85.10	84.47	82.72	83.31	81.89	83.21	85.48
SEUIL100	78.95	82.29	81.40	82.30	84.07	84.25	82.39	83.83	84.42	83.59	82.86	81.72	82.05	81.64	81.90	85.50

$Card(\mathcal{V}_{restaurees})$	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
DSEP50	81.45	81.54	81.48	81.64	82.23	82.22	82.35	82.11	82.17	82.07	82.53	82.54	81.99	81.96	81.78	82.53
DSEP100	81.41	81.58	81.59	81.76	82.17	82.13	82.38	82.09	82.25	81.99	82.70	82.60	82.24	81.93	81.79	82.66
SEUIL50	84.85	85.18	84.83	86.57	87.10	86.82	86.57	88.01	86.08	86.68	84.37	⋆	⋆	⋆	⋆	⋆
SEUIL100	84.85	85.07	85.73	87.20	86.78	87.31	87.11	87.98	86.82	85.50	84.48	⋆	⋆	⋆	⋆	⋆

$Card(\mathcal{V}_{restaurees})$	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
DSEP50	82.26	82.30	82.11	82.39	81.79	82.48	82.18	82.45	82.79	81.90	75.57	74.42	71.88	72.78	92.24	⋆
DSEP100	82.34	82.32	82.13	82.39	81.89	82.43	82.17	82.39	82.67	81.85	75.90	74.73	71.97	72.76	92.30	⋆
SEUIL50	⋆	⋆	⋆	⋆	⋆	⋆	⋆	⋆	⋆	⋆	⋆	⋆	⋆	⋆	⋆	⋆
SEUIL100	⋆	⋆	⋆	⋆	⋆	⋆	⋆	⋆	⋆	⋆	⋆	⋆	⋆	⋆	⋆	⋆

TABLE 5.3 – Taux de réussite en fonction du nombre de restaurations

$\frac{Card(\mathcal{V}_{restaurees})}{Card(\mathcal{V}_{affectees})}$	[0-10[	[10-20[	[20-30[	[30-40[	[40-50[	[50-60[	[60-70[	[70-80[	[80-90[	[90-100]
DSEP50	78.73	79.49	79.47	79.51	79.88	80.46	81.76	84.00	86.87	77.47
DSEP100	78.72	79.41	79.37	79.44	79.82	80.31	81.85	84.18	86.77	77.57
SEUIL50	79.62	90.55	84.13	81.07	85.45	86.81	84.15	$\star$	$\star$	$\star$
SEUIL100	79.48	88.02	82.51	79.83	86.01	86.59	84.29	79.95	$\star$	$\star$

TABLE 5.4 – Taux de réussite en fonction du taux de restauration

Nombre de valeurs possibles	2	3	4	5	6	7	8	12	16
Équiprobabilité	50	33.33	25	20	16.67	14.29	12.5	8.33	6.25
ORACLE	87.36	64.27	50.90	82.84	44.99	77.01	43.99	27.24	44.34
DSEP50	86.96	63.65	50.80	82.76	43.98	77.07	43.81	24.63	44.43
DSEP100	86.96	63.66	50.80	82.76	43.98	77.07	43.81	24.64	44.43
SEUIL50	86.96	63.63	50.80	82.76	43.98	77.07	43.81	24.47	44.43
SEUIL100	86.96	63.65	50.80	82.76	43.98	77.07	43.81	24.50	44.43
NAIF	72.79	60.00	50.74	82.69	43.78	77.14	43.94	19.62	44.43
RB	86.98	63.69	50.82	82.76	43.78	77.07	43.94	24.60	44.43

TABLE 5.5 – Taux de réussite en fonction du nombre de valeurs possibles

Chapitre 6

Perspectives

1 d-séparation dans un réseau de Markov

On a utilisé la d-séparation dans un réseau bayésien pour la restauration. Il existe une autre structure qui possède une propriété analogue : le réseau de Markov.

Les réseaux de Markov sont des graphes non orientés. Ils peuvent représenter des dépendances qui ne peuvent pas être représentées dans un réseau bayésien (comme les dépendances circulaires) mais la réciproque est vraie également.

Il pourrait s'agir d'une piste pour améliorer la restauration par d-séparation.

2 Problème des réseaux bayésiens avec relation déterministe

Les réseaux bayésiens ont été pensé pour encoder des distributions probabilistes. Néanmoins, lorsque les données d'apprentissage ont des relations déterministes, par exemple des contraintes dures qui interdisent des combinaisons de valeurs, les réseaux bayésiens perdent certaines de leurs propriétés.

Ce problème a été écarté lors de ces travaux mais méritent une attention particulière car les configurateurs manipulent souvent des variables qui satisfont un réseau de contraintes.

3 Calcul d'un ordre de préférence

Les algorithmes présents dans ce mémoire recommande une valeur, alors qu'il pourrait être intéressant pour l'utilisateur d'avoir plutôt une liste ordonnée de valeurs : la valeur la plus recommandée, puis une valeur un peu moins recommandée, etc. En fait, la recommandation d'une unique valeur dans ce

mémoire tient au fait que les données de Renault nous permettait d'évaluer de tels algorithmes.

Néanmoins, comme ces algorithmes calculent tous des probabilités, ils peuvent facilement renvoyer une liste ordonnée de valeurs, de la plus à la moins probable.

4 Critère d'arrêt de la restauration

La valeur du seuil utilisé dans la restauration est assez arbitraire. Par exemple, supposons qu'on possède 100 exemples sur lequel on veut estimer une variable A. Si on trouve 90 exemples avec A et 10 exemples avec $\bar{A}$, alors on pourra conclure que A est significativement plus probable que $\bar{A}$. Par contre, si sur 100 exemples, il y a 45 exemples avec A et 55 exemples avec $\bar{A}$, on pourrait être tenté d'effectuer une restauration supplémentaire afin d'avoir un écart plus significatif.

Il est possible d'estimer la signification de cet écart avec des tests statistiques : en effet, on cherche finalement à comparer deux moyennes. Néanmoins, ces tests se basent sur une estimation de la variance. Lors de nos tests, nous avons utilisé les SLDD qui permettent de calculer très rapidement une moyenne mais l'implémentation que nous avons utilisée ne permettait pas de calculer la variance.

Il existe néanmoins un cas particulier dans lequel la variance peut se déduire de l'espérance : le cas d'une variable de Bernoulli, booléenne, qui a pour espérance p et pour variance $p \cdot (1-p)$. Nous avons donc réalisé un test statistique permettant de savoir à quel point une moyenne p est significativement supérieure à 50 %. Si la différence est significative, alors il n'y a pas besoin de continuer la restauration.

Il s'agit du test t de Student, qui permet de comparer une espérance estimée à une espérance théorique (50 %) avec une variance empirique. En effet, la variable :

$$t = \frac{\overline{x} - 0,5}{s/\sqrt{n}},$$

suit alors approximativement une loi de Student avec $n-1$ degré de liberté, avec :

$$\overline{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

$$s = \overline{x}(1-\overline{x})$$

Cela nous fournit donc un critère d'arrêt. Les expériences montrent que ce critère d'arrêt accélère la recommandation au détriment d'une très légère baisse de performances. Néanmoins, lorsque le seuil de restauration est relativement bas (par exemple 50), ce critère d'arrêt est très peu utilisé.

5 Recommandation orientée en configuration

5.1 Motivation

La recommandation en configuration cherche à maximiser son taux de réussite, lorsque la valeur recommandée est effectivement choisie. Néanmoins, deux choses sont à prendre en compte :

- l'utilisateur peut être indécis, auquel cas plusieurs valeurs peuvent être à un même moment acceptables dans le sens où elles seraient acceptées par l'utilisateur si on les lui proposait. Un utilisateur qui configure une voiture et ne s'y connaît pas en technologies d'air conditionné pourrait par exemple systématiquement choisir ce qu'on lui recommande.
- l'entreprise qui vend le produit configuré peut dans ce cas préférer recommander certaines valeurs plutôt que d'autres. L'entreprise peut alors chercher à augmenter le prix du produit configuré, ou alors conseiller à l'utilisateur un produit qu'elle a déjà en stock.

La recommandation qui fait un compromis entre les préférences plus ou moins marquées de l'utilisateur et les préférences de l'entreprise est appelée *recommandation orientée*, car l'utilisateur est orienté vers certains produits plutôt que d'autres.

5.2 Classification et fonction de perte

En classification on cherche à estimer une classe c pour une instance. Néanmoins, toutes les erreurs de classification n'ont pas le même impact, la même gravité.

Par exemple, prenons le cas d'un programme qui dirige automatiquement une rame de métro et qui peut, à tout instant, décider d'accélérer, de ralentir ou de s'arrêter. Pour des raisons de sécurité évidentes, il est plus dangereux d'accélérer dans une situation où il aurait fallu ralentir que ralentir dans une situation où il aurait fallu accélérer.

Une telle asymétrie peut se représenter par une fonction de perte, qui associe à chaque couple $(c, \hat{c})$ un coût $\mathcal{L}(c, \hat{c}) \geq 0$ où habituellement $c = \hat{c} \Rightarrow \mathcal{L}(c, \hat{c}) = 0$.

5.3 Compromis de recommandation

On obtient généralement une distribution de probabilité sur une variable dont on souhaite recommander une valeur. Si on interprète ces probabilités (qui sont issues de calculs de fréquences empiriques) comme les probabilités que l'utilisateur choisisse un cas ou l'autre, alors on peut calculer, en fonction de la valeur recommandée l'espérance de la fonction de perte.

La valeur recommandée va être "attirée" par la volonté de l'utilisateur, car la fonction de perte est nulle en cas d'identité : si on est sûr que l'utilisateur veut la valeur x, alors on lui recommandera la valeur x car $\mathcal{L}(x, x) = 0$. Elle va aussi être en même temps "attiré" par les valeurs qui représentent un faible coût dans la fonction de perte (en cas d'équiprobabilité de la préférence d'une utilisateur, on choisira la valeur de moindre coût moyen).

Au final, il en sortira un compromis entre ce que désirent l'acheteur et le vendeur. Ce compromis peut balancer plus d'un côté ou de l'autre en fonction des valeurs de la fonction de perte, et on peut ainsi obtenir une recommandation faiblement ou fortement orientée.

6 Génération de prévision de ventes

6.1 Génération de prévision de ventes

Un problème sans rapport a priori avec la configuration est la génération de prévision de ventes. La génération de prévision de ventes consiste en la génération d'une liste de produits répondant à certains critères :

- elle doit s'appuyer sur les ventes récentes ;
- elle doit incorporer de nouvelles valeurs qui n'ont jamais été vendues (nouvelle option, nouvelle configuration possible, ...) mais dont des experts prévoient des taux de ventes.

La recommandation en configuration permet dans une certaine mesure de répondre à ces exigences.

6.2 Génération par filtrage

Les réseaux bayésiens peuvent générer des exemples selon leur distribution de probabilité. Le problème de ces exemples générés est qu'ils ne respectent a priori pas les contraintes, car ces contraintes se sont pas intégrées dans un réseau bayésien.

Une méthode de génération d'exemples satisfaisant les contraintes peut donc être de générer dans un premier lieu un nombre important d'exemples grâce à un réseau bayésien, puis de ne conserver que les exemples qui satisfont les contraintes.

Néanmoins, cela pose un problème : la proportion des exemples qui satisfont les contraintes peut être trop faible pour que le filtrage soit une technique efficace (par exemple, sur le jeu de donné *big*[1] du projet BR4CP, seulement un exemple sur des millions satisfait les contraintes).

[1] accessible sur http ://www.irit.fr/ Helene.Fargier/BR4CP/benches.html

De plus, étant donné que certains exemples sont rejetés, il n'y a plus de garanties que les exemples conservés après filtrage suivent toujours une distribution proche de celle du réseau bayésien.

6.3 Génération par correction d'historique

Dans le cadre du partenariat avec Renault, nous avons eu accès à des historiques de ventes. Malheureusement une partie ne satisfaisait pas les contraintes. Ceci s'explique par la modification des contraintes : une voiture vendue il y a un an peut, par exemple, ne plus satisfaire les normes écologiques ou alors ne pas avoir de toit ouvrant alors que celui-ci est devenu de série.

Nous avons donc cherché à corriger ces historiques, c'est-à-dire à les modifier les moins possibles de manière à ce qu'ils satisfassent les contraintes.

Pour cela, nous procédons à une simulation de configuration partielle. Lorsque, pour une variable, la valeur de l'historique n'est plus possible, on demande à un algorithme de recommandation la distribution de probabilité de cette variable. On tire alors, en respectant cette distribution, la valeur corrigée.

Algorithm 2 Génération d'exemples par correction d'historique

procedure CORRECTION($session, f_{possible}, reco$)
  $affectation \leftarrow$ liste vide
  for all $(variable, valeur) \in session$ **do**
    $possibles \leftarrow f_{possible}(variable, affectation)$ ▷ Récupération des valeurs possibles
    if $valeur \notin possibles$ **then** ▷ Correction nécessaire
      $proba \leftarrow reco(variable, affectation)$
      $valeur \leftarrow aleatoire(possibles, proba)$ ▷ Tirage d'une valeur selon une distribution
    end if
    $affectation \leftarrow affectation \cup (variable, valeur)$
  end for
  return *affectation*
end procedure

6.4 Génération par configuration simulée

Cette dernière méthode de génération de prévision de ventes s'inspire de la méthode précédente. Alors que la méthode de correction d'historiques n'utilise la recommandation que pour corriger certaines valeurs, on utilise ici le système de recommandation pour créer entièrement de nouveaux exemples suivant une distribution de probabilité proche de la loi des données originelles

et satisfaisant les contraintes. Cette simulation de configuration complète peut s'écrire algorithmiquement de la manière suivante.

Algorithm 3 Génération d'exemples par configuration simulée

procedure GENERECONFSIMU($ordre, f_{possible}$)
  $affectation \leftarrow$ liste vide
  for all $variable \in ordre$ **do** ▷ On respecte l'ordre des variables
    $possibles \leftarrow f_{possible}(variable, affectation)$ ▷ Récupération des valeurs possibles
    $proba \leftarrow reco(variable, affectation)$
    $valeur \leftarrow aleatoire(possibles, proba)$ ▷ Tirage d'une valeur selon une distribution
    $affectation \leftarrow affectation \cup (variable, valeur)$
  end for
  return *affectation*
end procedure

Conclusion

Les résultats obtenus sont :

- nos algorithmes ont une précision très proche de la méthode de référence tout en ayant un temps d'exécution très faible. De plus, nos algorithmes étant polynomiaux, on est assurés d'avoir une réponse rapide, contrairement à la méthode de référence.
- la mise en évidence d'un taux de réussite maximal qui permet de montrer que les algorithmes présentés sont, comme l'algorithme de référence, très précis (moins de 1% de différence !).

Ces résultats nous encouragent à poursuivre ces travaux et de prochains résultats seront probablement publiés.

Table des figures

Liste des tableaux

Bibliographie

[BFG+93] R. Iris Bahar, Erica A. Frohm, Charles M. Gaona, Gary D. Hachtel, Enrico Macii, Abelardo Pardo, and Fabio Somenzi. Algebraic decision diagrams and their applications. In *Proceedings of the 1993 IEEE/ACM International Conference on Computer-Aided Design, 1993, Santa Clara, California, USA, November 7-11, 1993*, pages 188–191, 1993.

[Bry86] Randal E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Trans. Computers*, 35(8) :677–691, 1986.

[Chi95] David Maxwell Chickering. Learning bayesian networks is NP-complete. In *Learning from Data - Fifth International Workshop on Artificial Intelligence and Statistics, AISTATS 1995, Key West, Florida, US, January, 1995. Proceedings.*, pages 121–130, 1995.

[Coo90] Gregory F. Cooper. The computational complexity of probabilistic inference using bayesian belief networks. *Artif. Intell.*, 42(2-3) :393–405, 1990.

[DM02] Adnan Darwiche and Pierre Marquis. A knowledge compilation map. *J. Artif. Intell. Res. (JAIR)*, 17 :229–264, 2002.

[FGG97] Nir Friedman, Dan Geiger, and Moisés Goldszmidt. Bayesian network classifiers. *Machine Learning*, 29(2-3) :131–163, 1997.

[FM07] Hélène Fargier and Pierre Marquis. On valued negation normal form formulas. In *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007*, pages 360–365, 2007.

[FMNS14] Hélène Fargier, Pierre Marquis, Alexandre Niveau, and Nicolas Schmidt. A knowledge compilation map for ordered real-valued decision diagrams. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada.*, pages 1049–1055, 2014.

[FMS13] Hélène Fargier, Pierre Marquis, and Nicolas Schmidt. Semiring labelled decision diagrams, revisited : Canonicity and spatial efficiency issues. In *IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013*, 2013.

[FMS14] Hélène Fargier, Pierre Marquis, and Nicolas Schmidt. Compacité pratique des diagrammes de décision valués. normalisation, heuristiques et expérimentations. *Revue d'Intelligence Artificielle*, 28(5) :571–592, 2014.

[FPV11] Boi Faltings, Pearl Pu, and Paolo Viappiani. Preference-based search using example-critiquing with suggestions. *CoRR*, abs/1110.0026, 2011.

[GVP13] Dan Geiger, Tom S. Verma, and Judea Pearl. d-separation : From theorems to algorithms. *CoRR*, abs/1304.1505, 2013.

[HT12] Peter Harremoës and Gábor E. Tusnády. Information divergence is more chi squared distributed than the chi squared statistics. *CoRR*, abs/1202.1125, 2012.

[LIT92] Pat Langley, Wayne Iba, and Kevin Thompson. An analysis of bayesian classifiers. In *Proceedings of the 10th National Conference on Artificial Intelligence. San Jose, CA, July 12-16, 1992.*, pages 223–228, 1992.

[NWL+07] Patrick Naïm, Pierre-Henri Wuillemin, Philippe Leray, Olivier Pourret, and Anna Becker. *Réseaux bayésiens*. Eyrolles, third edition, 2007.

[Pea00] Karl Pearson. On the criterion that a given system of deviations from the probable in the case of a correlated system of variables is such that can be reasonably supposed to have arisen from random sampling. *Philosophical Magazine*, 50 :157–175, 1900.

[Pea89] Judea Pearl. *Probabilistic reasoning in intelligent systems - networks of plausible inference*. Morgan Kaufmann series in representation and reasoning. Morgan Kaufmann, 1989.

[PH14] Shawn D. Pethel and Daniel W. Hahs. Exact test of independence using mutual information. *Entropy*, 16(5) :2839–2849, 2014.

[Ric97] Thomas Richardson. A characterization of markov equivalence for directed cyclic graphs. *Int. J. Approx. Reasoning*, 17(2-3) :107–162, 1997.

[Sch15] Nicolas Schmidt. *Compilation de préférences, application à la configuration de produit.* Thèse de doctorat, École doctorale sciences pour l'ingénieur, 2015.

[SP15] Mathieu Serrurier and Henri Prade. Entropy evaluation based on confidence intervals of frequency estimates : Application to the learning of decision trees. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, pages 1576–1584, 2015.

[Stu97] Markus Stumptner. An overview of knowledge-based configuration. *AI Commun.*, 10(2) :111–125, 1997.

[TBA06] Ioannis Tsamardinos, Laura E. Brown, and Constantin F. Aliferis. The max-min hill-climbing bayesian network structure learning algorithm. *Machine Learning*, 65(1) :31–78, 2006.

[Wil05] Nic Wilson. Decision diagrams for the computation of semiring valuations. In *IJCAI-05, Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence, Edinburgh, Scotland, UK, July 30-August 5, 2005*, pages 331–336, 2005.

[Yat34] F. Yates. Contingency tables involving small numbers and the chi-squared test. *Supplement to the Journal of the Royal Statistical Society*, 1(2) :pp. 217–235, 1934.