Apprentissage de préférences en espace combinatoire et application à la recommandation en configuration interactive

Pierre-François Gimenez

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par : *l'Université Toulouse 3 Paul Sabatier (UT3 Paul Sabatier)*

Présentée et soutenue le *10 octobre 2018* **par :**

Pierre-François Gimenez

Apprentissage de préférences en espace combinatoire et application à la recommandation en configuration interactive

JURY

Jean-Marc Astesana	Ingénieur de recherche Groupe Renault	Membre invité
Hélène Fargier	Directrice de recherche CNRS IRIT – Université Toulouse III	Directrice de thèse
Christophe Gonzales	Professeur des universités LIP6 – UPMC	Rapporteur
Eyke Hüllermeier	Professeur des universités Paderborn University	Rapporteur
Jérôme Mengin	Maître de conférence IRIT – Université Toulouse III	Directeur de thèse
Régis Sabbadin	Directeur de recherche INRA, Unité MIA	Membre invité
Élise Vareilles	Maître-assistant IMT Mines d'Albi	Examinatrice
Bruno Zanuttini	Professeur des universités GREYC – Université de Caen	Président

École doctorale et spécialité :
MITT : Domaine STIC : Intelligence Artificielle
Unité de Recherche :
Institut de Recherche en Informatique de Toulouse (UMR 5505)
Directeur(s) de Thèse :
Hélène Fargier et *Jérôme Mengin*
Rapporteurs :
Christophe Gonzales et *Eyke Hüllermeier*

Résumé

L'analyse et l'exploitation des préférences interviennent dans de nombreux domaines, comme l'économie, les sciences sociales ou encore la psychologie. Depuis quelques années, c'est l'*e-commerce* qui s'intéresse au sujet dans un contexte de personnalisation toujours plus poussée. Notre étude s'est portée sur la représentation et l'apprentissage de préférences sur des objets décrits par un ensemble d'attributs. Ces espaces combinatoires sont immenses, ce qui rend impossible en pratique la représentation *in extenso* d'un ordre de préférences sur leurs objets. C'est pour cette raison que furent construits des langages permettant de représenter de manière compacte des préférences sur ces espaces combinatoires. Notre objectif a été d'étudier plusieurs langages de représentation de préférences et l'apprentissage de préférences.

Nous avons développé deux axes de recherche. Le premier axe est l'algorithme DRC, un algorithme d'inférence dans les réseaux bayésiens. Alors que les autres méthodes d'inférence utilisent le réseau bayésien comme unique source d'information, DRC exploite le fait qu'un réseau bayésien est souvent appris à partir d'un ensemble d'objets qui ont été choisis ou observés. Ces exemples sont une source d'information supplémentaire qui peut être utilisée lors de l'inférence. L'algorithme DRC, de ce fait, n'utilise que la structure du réseau bayésien, qui capture des indépendances conditionnelles entre attributs et estime les probabilités conditionnelles directement à partir du jeu de données. DRC est particulièrement adapté à une utilisation dans un contexte où les lois de probabilité évoluent mais où les indépendances conditionnelles ne changent pas.

Le second axe de recherche est l'apprentissage de k-LP-trees à partir d'exemples d'objets vendus. Nous avons défini formellement ce problème et introduit un score et une distance adaptés. Nous avons obtenu des résultats théoriques intéressants, notamment un algorithme d'apprentissage de k-LP-trees qui converge avec assez d'exemples vers le modèle cible, un algorithme d'apprentissage de LP-tree linéaire optimal au sens où il minimise notre score, ainsi qu'un résultat sur le nombre d'exemples suffisants pour apprendre un « bon » LP-tree linéaire : il suffit d'avoir un nombre d'exemples qui dépend logarithmiquement du nombre d'attributs du problème.

Enfin, une contribution expérimentale évalue différents langages dont nous apprenons des modèles à partir d'historiques de voitures vendues. Les modèles appris sont utilisés pour la recommandation de valeur en configuration interactive de voitures Renault. La configuration interactive est un processus de construction de produit où l'utilisateur choisit successivement une valeur pour chaque attribut. Nous évaluons la précision de la recommandation, c'est-à-dire la proportion des recommandations qui auraient été acceptées, et le temps de recommandation ; de plus, nous examinons les différents paramètres qui peuvent influer sur la qualité de la recommandation. Nos résultats sont concluants : les méthodes que nous avons évaluées, qu'elles proviennent de la littérature ou de nos contributions théoriques, sont bien assez rapides pour être utilisées en ligne et ont une précision très élevée, proche du maximum théorique.

Abstract

The analysis and the exploitation of preferences occur in multiple domains, such as economics, humanities and psychology. E-commerce got interested in the subject a few years ago with the surge of product personalisation. Our study deals with the representation and the learning of preferences on objects described by a set of attributes. These combinatorial spaces are huge, which makes the representation of an ordering in extenso intractable. That's why preference representation languages have been built: they can represent preferences compactly on these huge spaces. In this dissertation, we study preference representation languages and preference learning.

Our work focuses on two approaches. Our first approach led us to propose the DRC algorithm for inference in Bayesian networks. While other inference algorithms use the sole Bayesian network as a source of information, DRC makes use of the fact that Bayesian networks are often learnt from a set of examples either chosen or observed. Such examples are a valuable source of information that can be used during the inference. Based on this observation, DRC uses not only the Bayesian network structure that captures the conditional independences between attributes, but also the set of examples, by estimating the probabilities directly from it. DRC is particularly adapted to problems with a dynamic probability distribution but static conditional independences.

Our second approach focuses on the learning of k-LP-trees from sold items examples. We formally define the problem and introduce a score and a distance adapted to it. Our theoretical results include a learning algorithm of k-LP-trees with a convergence property, a linear LP-tree algorithm minimising the score we defined and a sample complexity result: a number of examples logarithmic in the number of attributes is enough to learn a "good" linear LP-tree.

We finally present an experimental contribution that evaluates different languages whose models are learnt from a car sales history. The models learnt are used to recommend values in interactive configuration of Renault cars. The interactive configuration is a process in which the user chooses a value, one attribute at a time. The recommendation precision (the proportion of recommendations that would have been accepted by the user) and the recommendation time are measured. Besides, the parameters that influence the recommendation quality are investigated. Our results are promising: these methods, described either in the literature or in our contributions, are fast enough for an on-line use and their success rate is high, even close to the theoretical maximum.

Remerciements

Mes premières pensées vont vers les deux personnes qui m'ont encadré, soutenu et tant appris pendant ces trois ans, mes directeurs de thèse Hélène et Jérôme. Merci Hélène, ton exigence m'a rendu plus organisé et plus rigoureux. Merci Jérôme, pour tout ce temps pris à discuter nos idées et à vérifier mes preuves.

Je remercie Mme et MM. Jean-Marc Astesana, Christophe Gonzales, Eyke Hüllermeier, Régis Sabbadin, Élise Vareilles et Bruno Zanuttini pour avoir accepté de faire partie de mon jury de thèse.

Durant ces trois ans, j'ai eu la chance de rencontrer des chercheurs qui m'ont accompagné et conseillé sur l'enseignement, la vulgarisation et l'orientation post-thèse. Je pense notamment à Florence, Fred, Jonathan et Marie-Christine qui m'ont beaucoup apporté. Mes remerciements s'adressent particulièrement à Mathieu, dont les conversations ont initié mes travaux sur les LP-trees.

Je tiens également à remercier ceux qui ont été, à un moment ou à un autre, mes colocataires au bureau 302 : Nicolas [1], Nicolas [2], Rémi, Zeineb, Julien, Victor et Mickaël, ainsi que tous les stagiaires et doctorants [3] qui ont croisé ma route, notamment Chloé, Céline, Sébastien, Fabien, Estelle et Élise.

Enfin, je ne pourrais pas terminer sans remercier ma famille, mes frères, mes parents et bien sûr Manon ♡. Cette thèse vous est dédiée.

Quoi, tu es encore là ? Mais les remerciements sont finis. Hein ? Je t'ai oublié·e à tort ? Oh non, je suis désolé… je t'amène des bonbons la prochaine fois.

1. celui qui est une *hotline* très efficace pour son compilateur SALADD [Sch15b] et, accessoirement, un excellent ami.

2. celui qui, non content d'écrire sa thèse, a créé en parallèle la librairie python à succès « Surprise » [Hug17].

3. sauf un.

Table des matières

Page

1 Introduction **1**

I Contexte **5**

2 Langages de représentation de préférences en espace combinatoire **7**
2.1 Contexte et notations 8
2.1.1 Relations de préférences 8
2.1.2 Préférences sur espace combinatoire 12
2.2 Langages de représentation de préférence 15
2.2.1 Famille des CP-nets 15
2.2.2 Famille des LP-trees 23
2.2.3 GAI-net et GAI-tree 29
2.2.4 VCSP 31
2.2.5 Langages basés sur la logique propositionnelle 33
2.2.6 Réseaux bayésiens 34
2.2.7 Autres langages de représentation de préférences 36
2.3 Requêtes et complexité 37
2.3.1 Requêtes d'optimisation 37
2.3.2 Requêtes de comparaison 37
2.3.3 Complexité des requêtes 37

3 Apprentissage automatique de modèles graphiques de représentation de préférences **39**
3.1 Notions d'apprentissage automatique 40
3.1.1 Problèmes classiques d'apprentissage 40
3.1.2 Apprenabilité et *sample complexity* en classification binaire 41
3.1.3 Compromis biais-variance 44
3.2 Problèmes d'apprentissage de préférences 45
3.2.1 Ordonner des objets à partir de comparaisons 45
3.2.2 Ordonner des objets à partir de notes 46
3.2.3 Ordonner des étiquettes 46
3.2.4 Apprentissage de préférences par élicitation 47

3.3 Apprentissage de modèles graphiques 47
3.3.1 Apprentissage de CP-net 48
3.3.2 Apprentissage de LP-tree 49
3.3.3 Apprentissage de GAI-net 50
3.3.4 Apprentissage de réseau bayésien 51
3.4 Apprenabilité des langages de représentation de préférences . . . 53
3.4.1 L'apprentissage à partir de comparaisons, une classification binaire cachée 53
3.4.2 Dimension VC de quelques langages de représentation de préférence 53

II Contributions théoriques à l'apprentissage et à l'inférence 55

4 DRC : inférence en réseau bayésien par estimation directe 57
4.1 Méthodes d'inférences 58
4.1.1 Requêtes d'inférence 58
4.1.2 Calcul de marginale : approche naïve 58
4.1.3 Inférence exacte 59
4.1.4 Inférence stochastique 59
4.1.5 Recursive conditioning 59
4.2 Direct Recursive Conditioning 62
4.2.1 Motivation 62
4.2.2 Calcul d'indépendances conditionnelles 64
4.2.3 *Direct Recursive Conditioning* 65
4.2.4 Construction d'un arbre de décomposition ternaire 69
4.2.5 Construction de partitions admissibles 72
4.3 Conclusion 76

5 Apprentissage de k-LP-trees à partir d'exemples positifs 77
5.1 Problème d'apprentissage à partir d'exemples positifs 78
5.1.1 Présentation du problème 78
5.2 Distance et score 79
5.2.1 Distances usuelles entre deux ordres 79
5.2.2 *Ranking loss* 80
5.2.3 Rang moyen empirique 82
5.2.4 La vraisemblance 84
5.3 Apprentissage heuristique de k-LP-tree 86
5.3.1 Apprentissage des tables de préférences 86
5.3.2 Heuristique d'étiquetage de nœuds 89
5.3.3 Heuristique de scission 98
5.3.4 Complexité temporelle 98
5.3.5 Élagage pour réduire le surapprentissage 99
5.4 Apprentissage optimal de LP-tree linéaire 100
5.5 *Sample complexity* des LP-trees linéaires binaires 103
5.6 Expérimentations 106
5.6.1 Protocole 106
5.6.2 Résultats 107
5.7 Perspectives 109

III Application à la recommandation en configuration interactive 111

6 Recommandation en configuration interactive 113
6.1 Les familles de systèmes de recommandation 114
6.1.1 Recommandation par filtrage collaboratif 114
6.1.2 Recommandation basée sur le contenu 114
6.1.3 Recommandation basée sur la connaissance 115
6.2 Configuration interactive 115
6.2.1 Prise en compte des contraintes de faisabilité 116
6.2.2 Recommandation en configuration interactive 117
6.2.3 Données disponibles dans les problèmes Renault 118
6.3 Méthodes de recommandation 118
6.3.1 Réseau bayésien naïf 118
6.3.2 k plus proches voisins 119
6.3.3 Recommandation à partir de l'extension préférée 122
6.3.4 Recommandation la plus probable à partir d'un réseau bayésien 123
6.4 Résumé 123

7 Expérimentations sur la recommandation de valeurs en configuration interactive 125
7.1 Étude de cas : Renault 126
7.2 Protocole 127
7.2.1 Oracle 128
7.3 Expériences avec les algorithmes de la littérature 129
7.3.1 Influence de la taille du voisinage 130
7.3.2 Vaut-il mieux apprendre avec les exemples infaisables ? . . 131
7.3.3 Évolution du taux d'erreur 133
7.3.4 Temps de recommandation en fonction du nombre d'attributs assignés 135
7.3.5 Vaut-il mieux utiliser des clusters ? 136
7.3.6 Influence de la taille du jeu d'apprentissage 139
7.3.7 Influence des contraintes 140
7.3.8 Conclusion 142
7.4 Expériences relatives à DRC 142
7.4.1 Taux de succès avec mise à jour de l'historique 144
7.5 Expériences relatives aux LP-trees 146
7.6 Conclusion 149
7.6.1 Évaluation des algorithmes de la littérature 150
7.6.2 L'avantage de DRC 150
7.6.3 Efficacité réelle des LP-trees 150

8 Conclusion 151

Bibliographie 153

A Définitions de la théorie des graphes 169

Chapitre 1

Introduction

La représentation, l'apprentissage et le raisonnement sur les préférences forment un sujet de recherche qui intervient dans un grand nombre de problèmes. Voici quelques exemples de domaines dans lesquels les préférences sont étudiées :

- la décision sous incertitude prend en compte les préférences sur les différentes conséquences que des actions peuvent avoir ;
- en économie, on cherche à modéliser les préférences d'un consommateur rationnel ;
- en sciences sociales, les processus de vote ont pour objectif d'agréger un ensemble de préférences individuelles en une préférence globale de groupe ;
- en psychologie, les préférences sont étudiées de manière expérimentale.

L'étude des préférences est également au centre d'un certain nombre d'applications. Par exemple, la personnalisation de services dans l'*e-commerce*, et notamment la recommandation de produits, s'appuie sur les préférences des utilisateurs.

Problématique

En tant qu'internautes, nous sommes de plus en plus habitués aux recommandations des plates-formes en ligne, qu'elles viennent par exemple de Netflix ou d'Amazon. Dans le cas de Netflix, les vidéos recommandées et acceptées par les utilisateurs formaient en 2016 près de 80 % du total des vidéos visionnées sur la plate-forme [GUH16].

Le problème est complètement différent du cas de Netflix dès lors qu'on s'intéresse à des alternatives définies par un ensemble d'attributs discrets. On retrouve ce type d'alternatives dans un grand nombre de situations :

- des trajets en avion, définis par leurs aéroports de départ et d'arrivée, la durée du trajet, la présence ou non de correspondance, leur prix, la compagnie aérienne... ;

- des voitures, définies par une marque, un modèle, une couleur, un moteur et toutes les options... ;
- des cuisines, définies par les matériaux des sols et des murs, la présence d'un îlot ou non, le bois des meubles, le matériau et la couleur du plan de travail...

Ces ensembles de voitures, de trajets aériens et de cuisines sont combinatoires et le nombre de combinaisons est immense dès qu'il y a plus que quelques attributs. Cette constatation est le point de départ de cette thèse : comment recommander efficacement en espace combinatoire ?

Étude des préférences en espace combinatoire

On peut considérer trois axes à l'étude des préférences en espace combinatoire : leur représentation, leur apprentissage et leur exploitation.

La représentation des préférences : on considère généralement pouvoir capturer les préférences sous la forme de relations d'ordre, partielles ou complètes. Grâce aux langages de représentation de préférences, on peut représenter ces relations de manière compacte.

L'apprentissage de préférences : l'apprentissage automatique de préférences transforme des données brutes en une relation d'ordre représentée à l'aide d'un modèle exprimé dans un langage de préférences.

L'exploitation des préférences : la manière d'exploiter le modèle appris varie selon les applications ; dans notre cas, l'objectif est de produire une recommandation afin de guider un utilisateur dans la configuration d'un produit.

Notre principale problématique fut : *comment utiliser un ensemble d'objets choisis par des utilisateurs précédents, par exemple un historique de ventes, pour produire et exploiter un modèle de préférences ?* Cette problématique s'est déclinée en deux questions distinctes :

- cet ensemble d'objets choisis peut-il être utilisé lors de l'inférence en réseau bayésien ?
- ces objets choisis peuvent-ils servir à apprendre des préférences à l'aide d'un langage qui ne représente pas une distribution de probabilité ?

Plan de thèse et contributions

La première partie de la thèse est consacrée au contexte de nos travaux. Elle présente différents langages de représentation des préférences et introduit des notions d'apprentissage qui seront utiles par la suite.

La seconde partie de la thèse contient deux contributions théoriques sur l'apprentissage et l'exploitation de préférences.

La première contribution théorique est une méthode d'inférence dans les réseaux bayésiens nommée DRC (pour *Direct Recursive Conditioning*), publiée dans la Revue d'Intelligence Artificielle (RIA) [FGM18b]. En exploitant des

informations supplémentaires, nous ouvrons la voie à des algorithmes d'inférence potentiellement plus rapides et plus précis.

La seconde contribution théorique concerne l'apprentissage de modèles de représentation ordinaux à partir d'historiques de ventes : nous introduisons un nouveau problème d'apprentissage, adapté à une utilisation en situation réelle, et nous présentons un algorithme d'apprentissage dont nous étudions les propriétés. De plus, nous avons réalisé des expériences avec des données simulées afin de valider notre approche. Ces travaux ont été présentés lors de la conférence AAAI [FGM18a].

La troisième et dernière partie de la thèse présente la contribution expérimentale. Il s'agit d'une analyse des performances empiriques de nos contributions théoriques lorsqu'elles sont appliquées à la recommandation en configuration interactive. Il s'agit d'un problème provenant de l'industrie et qui nous permet donc d'évaluer, sur des données réelles, la qualité de nos outils. Ces expérimentations ont été publiées au *workshop* international de configuration [FGM16].

Première partie

Contexte

Chapitre 2

Langages de représentation de préférences en espace combinatoire

Sommaire

2.1 Contexte et notations 8
2.1.1 Relations de préférences 8
2.1.2 Préférences sur espace combinatoire 12
2.2 Langages de représentation de préférence 15
2.2.1 Famille des CP-nets 15
2.2.2 Famille des LP-trees 23
2.2.3 GAI-net et GAI-tree 29
2.2.4 VCSP 31
2.2.5 Langages basés sur la logique propositionnelle 33
2.2.6 Réseaux bayésiens 34
2.2.7 Autres langages de représentation de préférences . . 36
2.3 Requêtes et complexité 37
2.3.1 Requêtes d'optimisation 37
2.3.2 Requêtes de comparaison 37
2.3.3 Complexité des requêtes 37

Les préférences sont le plus souvent capturées par des relations binaires qui permettent d'exprimer le fait qu'un utilisateur préfère un objet à un autre, qu'il est indifférent entre deux alternatives ou qu'il lui est impossible de comparer deux objets.

Néanmoins, dans le cas de produits configurables qui combinent un ensemble de composants ou d'options personnalisables par l'utilisateur, le nombre d'objets différents est exponentiel en le nombre de paramètres, ce qui rend la description en extension d'une relation binaire impossible en pratique. Afin de représenter de manière compacte ces relations binaires, on utilise des langages de représentation de préférences.

Ce chapitre va tout d'abord introduire les relations binaires et les différentes notions d'ordre qui existent. Puis nous présenterons plusieurs langages de représentation de préférences avant de détailler différentes requêtes qui pourront être utilisées sur des modèles de préférences.

2.1 Contexte et notations

Pour capturer les préférences d'un utilisateur, on utilise des relations binaires. Le lecteur pourra consulter [BV03] pour avoir plus de détails sur ce que nous allons présenter.

2.1.1 Relations de préférences

Relation binaire

Définition 2.1.1 (Relation binaire)**.** *Une relation binaire $\mathcal{R}$ sur un ensemble fini A est un sous-ensemble du produit cartésien $A \times A$, c'est-à-dire en ensemble de paires ordonnées $(a, b) \in A \times A$. On écrira $a \mathcal{R} b$ si le couple (a, b) appartient à $\mathcal{R}$. Si au contraire (a, b) n'appartient pas à $\mathcal{R}$, on écrira $a \neg\mathcal{R} b$.*

Dans notre cas, $\mathcal{R}$ sera une relation de préférence interprétée de la manière suivante : $a \mathcal{R} b$ signifie que a est au moins aussi préféré que b.

Plusieurs propriétés fondamentales vont nous permettre de caractériser plus précisément ces relations binaires.

Définition 2.1.2 (Propriétés d'une relation binaire)**.** *Une relation binaire $\mathcal{R}$ est dite :*

réflexive : $a \mathcal{R} a$

complète : $a \mathcal{R} b$ *ou* $b \mathcal{R} a$

antisymétrique : $a \mathcal{R} b$ *et* $b \mathcal{R} a \Rightarrow a = b$

symétrique : $a \mathcal{R} b \Rightarrow b \mathcal{R} a$

asymétrique : $a \mathcal{R} b \Rightarrow b \neg\mathcal{R} a$

transitive : $a \mathcal{R} b$ *et* $b \mathcal{R} c \Rightarrow a \mathcal{R} c$

pour tout $a, b, c \in A$.

On peut à présent s'interroger sur les propriétés qu'une relation de préférence peut avoir.

- Toute alternative est au moins aussi bonne qu'elle-même, donc une relation de préférence devrait être réflexive.
- On peut être indifférents à deux alternatives a et b différentes, c'est-à-dire avoir $a \mathcal{R} b$ et $b \mathcal{R} a$ sans avoir $a = b$.

- La transitivité est une propriété nécessaire à la rationalité de nos choix. C'est de plus une propriété très pratique pour apprendre des préférences, car savoir que a est préféré à b et que b est préféré à c nous permet de savoir directement que a est préféré à c. [1]

Pour ces raisons, on capture les préférences sous la forme d'une relation binaire réflexive et transitive, ce qu'on appelle un préordre.

Préordre

Définition 2.1.3 (Préordre)**.** *Une relation binaire $\mathcal{R}$ est un préordre si :*

- *$\mathcal{R}$ est réflexive ;*
- *$\mathcal{R}$ est transitive.*

Cette définition implique qu'il y a quatre possibilités entre deux éléments $a, b \in A$:

- $a \mathcal{R} b$ et $b \neg\mathcal{R} a$. Dans ce cas, a est strictement préféré à b.
- $a \neg\mathcal{R} b$ et $b \mathcal{R} a$. Dans ce cas, b est strictement préféré à a.
- $a \mathcal{R} b$ et $b \mathcal{R} a$. Il y a indifférence entre a et b.
- $a \neg\mathcal{R} b$ et $b \neg\mathcal{R} a$. Dans ce cas, a et b sont incomparables.

Les préférences que nous chercherons à représenter correspondent à des cas particuliers des préordres : les préordres complets, les ordres totaux et les ordres partiels.

Préordre complet

Définition 2.1.4 (Préordre complet)**.** *Une relation binaire $\mathcal{R}$ est un préordre complet si :*

- *$\mathcal{R}$ est réflexive ;*
- *$\mathcal{R}$ est transitive ;*
- *$\mathcal{R}$ est complète.*

Les préordres complets sont notamment ceux que nous rencontrerons lorsqu'une relation de préférence est représentée par une fonction à valeurs numériques, comme le montre la proposition suivante.

Proposition 2.1.1 ([Deb64])**.** *Soit $\mathcal{R}$ un préordre complet sur A, un ensemble fini. Il existe une fonction $g : A \to \mathbb{R}$ tel que, pour tout $a, b \in A$:*

$$a \mathcal{R} b \Leftrightarrow g(a) \geq g(b)$$

1. Néanmoins, des expériences montrent que nos préférences ne sont pas toujours transitives, ce qui peut nous amener à faire des choix irrationnels [Fis91].

Si g satisfait cette proposition pour $\mathcal{R}$, on dit que g est une représentation numérique de $\mathcal{R}$. Connaître g nous permet de retrouver $\mathcal{R}$: pour savoir si $a \mathcal{R} b$, il suffit de vérifier $g(a) \geq g(b)$. On peut remarquer que cette représentation numérique n'est pas unique ; par exemple si g est une représentation numérique de $\mathcal{R}$, alors $2g$ en est aussi une.

Les préordres complets sont donc des préordres où il n'y a pas d'incomparabilité possible entre deux éléments.

Ordre total

Définition 2.1.5 (Ordre total)**.** *Une relation binaire $\mathcal{R}$ est un ordre total si :*

- *$\mathcal{R}$ est réflexive ;*
- *$\mathcal{R}$ est transitive ;*
- *$\mathcal{R}$ est complète ;*
- *$\mathcal{R}$ est antisymétrique.*

Ces structures de préférences ordonnent toutes les alternatives de la préférée à la moins préférée, sans qu'il y ait d'indifférence. De ce fait, les ordres totaux peuvent également se représenter sous forme numérique, via ce théorème :

Proposition 2.1.2. *Soit $\mathcal{R}$ un ordre total sur A, un ensemble fini. Il existe une fonction $g : A \to \mathbb{R}$ tel que, pour tout $a, b \in A$:*

$$\begin{cases} a \mathcal{R} b \Leftrightarrow g(a) \geq g(b) \\ g(a) = g(b) \Rightarrow a = b \end{cases}$$

Les ordres totaux sont donc des préordres où il ne peut y avoir ni incomparabilité ni indifférence entre deux éléments différents. On peut de plus y définir une fonction de rang de la manière suivante :

Définition 2.1.6 (Fonction rang d'un ordre total)**.** *Soit $\mathcal{R}$ un ordre total. On définit la fonction* rank *de $\mathcal{R}$ de la façon suivante :*

$$\mathsf{rank}(\mathcal{R}, \mathbf{o}) = |\{\mathbf{o}' : \mathbf{o}' \mathcal{R} \mathbf{o}\}|$$

L'objet de rang 1 est le plus préféré (car cet objet est en relation avec lui-même uniquement), l'objet de rang 2 le second plus préféré, etc.

Exemple 2.1.1. Soit $\mathcal{R}$ un ordre total. Alors la fonction $-\mathsf{rank}$ est une représentation numérique de $\mathcal{R}$. Par transitivité on obtient que :

$$a \mathcal{R} b \Leftrightarrow \mathsf{rank}(a) \leq \mathsf{rank}(b)$$

De plus, on peut prouver que :

$$\mathsf{rank}(a) = \mathsf{rank}(b) \Rightarrow a = b$$

On en déduit bien que :

$$\begin{cases} a \ \mathcal{R} \ b \Leftrightarrow -\mathsf{rank}(a) \geq -\mathsf{rank}(b) \\ -\mathsf{rank}(a) = -\mathsf{rank}(b) \Rightarrow a = b \end{cases}$$

Nous nous intéresserons aussi à une généralisation des ordres totaux : les ordres partiels.

Ordre partiel

Définition 2.1.7 (Ordre partiel)**.** *Une relation binaire $\mathcal{R}$ est un ordre partiel si :*

- *$\mathcal{R}$ est réflexive ;*
- *$\mathcal{R}$ est transitive ;*
- *$\mathcal{R}$ est antisymétrique.*

En d'autres mots, un ordre partiel impose à toute paire (a, b) où $a \neq b$ d'être dans l'une de ces trois possibilités :

- a est strictement préféré à b ;
- b est strictement préféré à a ;
- a et b sont incomparables.

Cela signifie que, contrairement aux préordres, il peut y avoir incomparabilité mais pas d'indifférence entre deux éléments différents.

Il sera également utile de pouvoir vérifier si un ordre total est « compatible » avec un ordre partiel, c'est-à-dire si l'ordre total ne contredit jamais l'ordre partiel. Plus formellement :

Définition 2.1.8 (Compatibilité)**.** *Soient $\mathcal{R}$ et $\mathcal{R}'$ deux préordres. On dit que $\mathcal{R}$ est compatible avec $\mathcal{R}'$ si, pour tout $a, b \in A$, $a \ \mathcal{R}' \ b \Rightarrow a \ \mathcal{R} \ b$.*

Définissons à présent l'union et l'intersection de relations.

Définition 2.1.9 (Union de relations)**.** *Soient deux relations binaires $\mathcal{R}_1$ et $\mathcal{R}_2$. On définit la relation $\mathcal{R}_1 \cup \mathcal{R}_2$ de la manière suivante :*

$$a \ \mathcal{R}_1 \cup \mathcal{R}_2 \ b \Leftrightarrow a \ \mathcal{R}_1 \ b \text{ ou } a \ \mathcal{R}_2 \ b$$

Définition 2.1.10 (Intersection de relations)**.** *Soient deux relations binaires $\mathcal{R}_1$ et $\mathcal{R}_2$. On définit la relation $\mathcal{R}_1 \cap \mathcal{R}_2$ de la manière suivante :*

$$a \ \mathcal{R}_1 \cap \mathcal{R}_2 \ b \Leftrightarrow a \ \mathcal{R}_1 \ b \text{ et } a \ \mathcal{R}_2 \ b$$

Le théorème suivant permet de donner une nouvelle interprétation aux ordres partiels.

Proposition 2.1.3. *Soit $\mathcal{R}$ un ordre partiel. Si on note T l'ensemble des ordres totaux compatibles avec $\mathcal{R}$, alors $\mathcal{R} = \bigcap_{\mathcal{T} \in T} \mathcal{T}$.*

Notations usuelles

Nous introduisons des notations usuelles pour les préordres que nous utiliserons dans la suite de cette thèse.

Soit $\mathcal{R}$ un préordre. On utilisera la notation $\succeq_{\mathcal{R}}$, définie de la manière suivante :

$$a \succeq_{\mathcal{R}} b \text{ ssi } a \mathcal{R} b$$

Deux notions supplémentaires vont nous être utiles : la partie symétrique et la partie asymétrique d'un préordre.

Définition 2.1.11 (Partie symétrique)**.** *Soit $\mathcal{R}$ un préordre. On définit la* partie symétrique *de $\mathcal{R}$, notée $\approx_{\mathcal{R}}$, de la manière suivante :*

$$a \approx_{\mathcal{R}} b \textit{ ssi } a \mathcal{R} b \textit{ et } b \mathcal{R} a$$

Définition 2.1.12 (Partie asymétrique)**.** *Soit $\mathcal{R}$ un préordre. On définit la* partie asymétrique *de $\mathcal{R}$, notée $\succ_{\mathcal{R}}$, de la manière suivante :*

$$a \succ_{\mathcal{R}} b \textit{ ssi } a \mathcal{R} b \textit{ et } b \neg\mathcal{R} a$$

Afin d'alléger les notations, nous omettrons l'indice $\mathcal{R}$ aux symboles $\succeq$, $\approx$ et $\succ$ lorsqu'il n'y aura pas d'ambiguïté.

Tableau récapitulatif

Voici un tableau récapitulant, pour chaque type de relation binaire que nous venons de voir, si l'indifférence entre deux éléments différents et l'incomparabilité peuvent être représentées.

	Indifférence possible	Incomparabilité possible
Préordre	Oui	Oui
Préordre complet	Oui	Non
Ordre total	Non	Non
Ordre partiel	Non	Oui

TABLE 2.1 – Tableau récapitulatif de ce que peuvent représenter les différentes relations binaires.

2.1.2 Préférences sur espace combinatoire

Notations

Nous n'avons fait jusque là aucune hypothèse sur l'ensemble A ; on peut donc imaginer des relations de préférences sur un ensemble de livres, d'artistes ou encore de voitures. Dans notre cas, ce sont les préférences sur espace combinatoire qui nous intéressent. Les notations suivantes seront utilisées tout au long de cette thèse.

Soit un ensemble $\mathcal{X}$ de n attributs discrets, chaque attribut (aussi appelé variable) X prenant ses valeurs dans un domaine $\underline{X}$. On s'intéresse aux relations de préférences définies sur l'ensemble des vecteurs $\mathbf{o} \in \prod_{X \in \mathcal{X}} \underline{X}$; nous notons

leur ensemble $\underline{\mathcal{X}}$. Ces éléments $\mathbf{o}$ portent plusieurs noms dans la littérature : alternative, objet, instanciation...

Si $\mathbf{U}$ est un sous-ensemble de $\mathcal{X}$, alors $\underline{\mathbf{U}}$ désigne l'ensemble des instanciations partielles $\prod_{X \in \mathbf{U}} \underline{X}$. Dans la suite de cette thèse, nous utiliserons la notation suivante : les lettres majuscules en caractères gras représenteront des ensembles d'attributs et nous noterons une instanciation d'un de ces ensembles par la lettre minuscule correspondante, par exemple $\mathbf{u} \in \underline{\mathbf{U}}$. Nous réserverons les minuscules maigres (non-gras) pour les valeurs d'un seul attribut, par exemple $x \in \underline{X}$.

Si $\mathbf{U}$ et $\mathbf{V}$ sont deux ensembles d'attributs, 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 $\mathbf{u}[\mathbf{U} \cap \mathbf{V}] = \mathbf{v}[\mathbf{U} \cap \mathbf{V}]$, ce qu'on écrira $\mathbf{u} \sim \mathbf{v}$. 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 attributs de $\mathbf{V} \setminus \mathbf{U}$ (ou, de manière équivalente, $\mathbf{uv}$ étend $\mathbf{v}$ avec les valeurs de $\mathbf{u}$ sur les attributs de $\mathbf{U} \setminus \mathbf{V}$).

Exemple 2.1.2. Afin de faciliter l'explication des différentes notions présentées dans cette thèse, voici un exemple de problème qui sera utilisé tout au long des chapitres.

On s'intéresse aux préférences sur différentes configurations de dîner. Trois attributs permettent de configurer ce dîner :

Le plat principal (P) : soit de la viande, soit du poisson.

Le vin (V) : soit du vin blanc, soit du vin rouge.

Le dessert (D) : soit un dessert au chocolat, soit une salade de fruits.

Il y a donc en tout huit dîners possibles.

Lorsqu'une personne indique « *je préfère le repas viande/vin rouge/chocolat au repas viande/vin rouge/fruit* », elle exprime une préférence entre deux menus.

Néanmoins, on a également tendance à exprimer nos préférences de manière plus générale, par exemple en disant : « *je préfère les repas avec dessert au chocolat plutôt qu'avec salade de fruits* ». Cet énoncé ne compare pas deux menus comme précédemment et peut être interprété de plusieurs manières. Nous allons présenter deux interprétations : les préférences *ceteris paribus* et les relations d'importance.

Indépendance préférentielle

Examinons de plus près la phrase « *je préfère les repas avec dessert au chocolat plutôt qu'avec salade de fruits* ». Une première manière d'interpréter cette phrase est de comprendre que, pour un même menu, l'utilisateur préfère sa variante chocolatée à sa variante fruitée.

Ainsi :

- il préfère le repas *viande/vin rouge/chocolat* à *viande/vin rouge/fruit*;
- on n'a aucune information sur la préférence entre *viande/vin rouge/chocolat* et *poisson/vin blanc/fruit*.

Autrement dit, toutes choses étant égales par ailleurs, il préfère le chocolat aux fruits. Ces préférences sont qualifiées de *ceteris paribus* (signifiant « toutes choses

égales par ailleurs »). Ces préférences ont l'avantage d'être compréhensibles intuitivement et d'avoir été étudiées dès les années 1960 [VW63]. Cette propriété est formalisée avec la notion d'indépendance préférentielle inconditionnelle.

Définition 2.1.13 (Indépendance préférentielle inconditionnelle)**.** *Soit* $\{\mathbf{X}, \mathbf{U}\}$ *une partition de* $\mathcal{X}$*. Soit* $\succ$ *la partie asymétrique d'un préordre quelconque sur* $\mathcal{X}$*. On dit que* $\mathbf{X}$ *est (inconditionnellement) préférentiellement indépendant de* $\mathbf{U}$ *pour* $\succ$ *s'il existe une relation d'ordre totale* $\succ_{\mathbf{X}}$ *définie sur* $\underline{\mathbf{X}}$ *telle que, pour tout* $\mathbf{x}, \mathbf{x}' \in \underline{\mathbf{X}}$*,* $\mathbf{u} \in \underline{\mathbf{U}}$ *:*

$$\mathbf{x} \succ_{\mathbf{X}} \mathbf{x}' \Rightarrow \mathbf{xu} \succ \mathbf{x}'\mathbf{u}$$

Afin d'exprimer des indépendances plus fines, la notion d'indépendance préférentielle conditionnelle a été introduite.

Définition 2.1.14 (Indépendance préférentielle conditionnelle)**.** *Soit* $\{\mathbf{X}, \mathbf{Y}, \mathbf{U}\}$ *une partition de* $\mathcal{X}$*. Soit* $\succ$ *la partie asymétrique d'un préordre quelconque sur* $\mathcal{X}$*. On dit que* $\mathbf{X}$ *est préférentiellement indépendant de* $\mathbf{Y}$ *conditionnellement à* $\mathbf{u} \in \underline{\mathbf{U}}$ *pour* $\succ$ *s'il existe une relation d'ordre totale* $\succ_{\mathbf{X}}$ *définie sur* $\underline{\mathbf{X}}$ *tels que, pour tout* $\mathbf{x}, \mathbf{x}' \in \underline{\mathbf{X}}$*,* $\mathbf{y} \in \underline{\mathbf{Y}}$ *:*

$$\mathbf{x} \succ_{\mathbf{X}} \mathbf{x}' \Rightarrow \mathbf{xyu} \succ \mathbf{x}'\mathbf{yu}$$

L'indépendance préférentielle conditionnelle permet de capturer l'énoncé suivant : « *pour les repas avec de la viande, je préfère le vin rouge au vin blanc (toutes autres choses étant égales par ailleurs), alors que pour les repas avec du poisson, je préfère le vin blanc au vin rouge (toutes autres choses étant égales par ailleurs)* ».

Importance conditionnelle et inconditionnelle

Examinons de plus près la phrase « *je préfère les repas avec dessert au chocolat plutôt qu'avec salade de fruits* ». On peut la comprendre de la manière suivante : « *quelque soit le repas avec un dessert au chocolat, je le préfère à n'importe quel repas avec une salade de fruits* ». C'est ce que la notion d'importance inconditionnelle, définie par [BDS06], va nous permettre de formaliser.

Définition 2.1.15 (Importance inconditionnelle)**.** *Soient* $\{\mathbf{X}, \mathbf{Y}, \mathbf{U}\}$ *une partition de* $\mathcal{X}$*. Soit* $\succ$ *la partie asymétrique d'un préordre quelconque sur* $\mathcal{X}$*. On dit que* $\mathbf{X}$ *est inconditionnellement plus important que* $\mathbf{Y}$ *pour* $\succ$*, ce qu'on note* $\mathbf{X} \triangleright \mathbf{Y}$*, s'il existe une relation d'ordre totale* $\succ_{\mathbf{X}}$ *définie sur* $\underline{\mathbf{X}}$ *telle que, pour tout* $\mathbf{x}, \mathbf{x}' \in \underline{\mathbf{X}}$*,* $\mathbf{y}, \mathbf{y}' \in \underline{\mathbf{Y}}$*,* $\mathbf{u} \in \underline{\mathbf{U}}$ *:*

$$\mathbf{x} \succ_X \mathbf{x}' \Rightarrow \mathbf{xyu} \succ \mathbf{x}'\mathbf{y}'\mathbf{u}$$

Nous noterons $\mathbf{X} \triangleright \mathbf{Y}(\succ_{\mathbf{X}})$ la relation de préférence minimale telle que $\mathbf{X}$ soit inconditionnellement plus important que $\mathbf{Y}$ par rapport à la relation $\succ_{\mathbf{X}}$, i.e. :

$$\mathbf{xyu}\ \mathbf{X} \triangleright \mathbf{Y}(\succ_{\mathbf{X}})\ \mathbf{x}'\mathbf{y}'\mathbf{u} \text{ ssi } \mathbf{x} \succ_X \mathbf{x}'$$

Il existe également une version conditionnelle :

Définition 2.1.16 (Importance conditionnelle)**.** *Soient* $\mathbf{X}$*,* $\mathbf{Y}$ *et* $\mathbf{Z}$ *trois sous-ensembles deux à deux disjoints de* $\mathcal{X}$ *et* $\mathbf{U}$ *l'ensemble des autres attributs de* $\mathcal{X}$*, i.e.* $\mathbf{U} = \mathcal{X} \setminus (\mathbf{Z} \cup \mathbf{X} \cup \mathbf{Y})$*. Soit* $\succ$ *la partie asymétrique d'un préordre quelconque sur* $\mathcal{X}$*. On dit* $\mathbf{X}$ *est plus important que* $\mathbf{Y}$ *conditionnellement à* $\mathbf{z} \in \underline{\mathbf{Z}}$ *pour* $\succ$*, noté* $\mathbf{X} \triangleright_{\mathbf{z}} \mathbf{Y}$*, s'il existe une relation d'ordre totale* $\succ_{\mathbf{X}}$ *définie sur* $\underline{\mathbf{X}}$ *telle que, pour tout* $\mathbf{x}, \mathbf{x}' \in \underline{X}$*,* $\mathbf{y}, \mathbf{y}' \in \underline{Y}$*,* $\mathbf{u} \in \underline{\mathbf{U}}$ *:*

$$\mathbf{x} \succ_{\mathbf{X}} \mathbf{x}' \Rightarrow \mathbf{xyzu} \succ \mathbf{x}'\mathbf{y}'\mathbf{zu}$$

De la même manière que précédemment, nous noterons $\mathbf{X} \triangleright_{\mathbf{z}} \mathbf{Y}(\succ_{\mathbf{X}})$ la relation de préférence minimale telle que $\mathbf{X}$ soit inconditionnellement plus important que $\mathbf{Y}$ par rapport à la relation $\succ_{\mathbf{X}}$, i.e. :

$$\mathbf{xyzu}\ \mathbf{X} \triangleright_{\mathbf{z}} \mathbf{Y}(\succ_{\mathbf{X}})\ \mathbf{x}'\mathbf{y}'\mathbf{zu} \text{ ssi } \mathbf{x} \succ_X \mathbf{x}'$$

Ces concepts permettent généralement une représentation fidèle des comportements humains [GG96].

2.2 Langages de représentation de préférence

Le nombre d'alternatives est exponentiel en n, le nombre d'attributs dans $\mathcal{X}$, et le nombre d'ordres totaux est factoriel en ce nombre d'alternatives. De ce fait, il est souvent impraticable de représenter en extension une relation de préférence. C'est pour cela qu'ont été développés des langages de représentation de préférences. Comme il est improbable que nous ayons dans notre cerveau un ordonnancement explicite et exhaustif de tous nos goûts, c'est qu'il existe probablement une certaine manière de représenter, de façon compacte, nos préférences.

Les langages de représentation de préférence permettent de représenter de manière compacte des relations de préférences. Il existe différents langages, adaptés chacun à certaines formes de préférences. Il en existe deux grandes familles :

- La première famille, celle des langages de représentation ordinaux, fait très peu d'hypothèses sur les préférences à apprendre.
- La seconde famille, celle des langages de représentation numériques, représente des préordres complets à l'aide d'une fonction d'utilité qui a parfois une interprétation quantitative (par exemple une probabilité). En effet, la proposition 2.1.1 montre que tout préordre complet $\succeq$ peut être représenté sous forme d'une fonction d'utilité $u : \underline{\mathcal{X}} \mapsto \mathbb{R}$, de sorte que $\mathbf{o} \succeq \mathbf{o}'$ si et seulement si $u(\mathbf{o}) \geq u(\mathbf{o}')$.

Plusieurs de ces langages s'appuient sur la théorie des graphes. L'annexe A en rappelle les principales notions. Par la suite, on notera $\mathbf{Pa}(N)$ l'ensemble des parents du nœud N et $\mathbf{Anc}(N)$ l'ensemble des ancêtres du nœud N.

2.2.1 Famille des CP-nets

Un langage de représentation adapté à la représentation de préférences *ceteris paribus*, les CP-nets (*conditional preference network*), a été introduit dans

[BBD+04]. Les CP-nets permettent d'exprimer en particulier des indépendances préférentielles conditionnelles.

Nous allons présenter formellement les CP-nets ainsi que certaines de leurs variantes.

CP-net

Un CP-net est composé de deux parties : un graphe qui représente des indépendances préférentielles conditionnelles et des tables de préférences conditionnelles. Ces tables sont définies formellement de la manière suivante :

Définition 2.2.1 (Table de préférence conditionnelle)**.** *Soient* $\mathbf{X}, \mathbf{P}$ *deux ensembles d'attributs disjoints de* $\mathcal{X}$*. Une table de préférence conditionnelle associe à tout* $\mathbf{p} \in \underline{\mathbf{P}}$ *un ordre total* $\succ_{\mathbf{X}}$ *sur le domaine* $\underline{\mathbf{X}}$*, noté* $\mathsf{CPT}_{\mathbf{X}}(\mathbf{p})$*.*

Chaque table de préférence peut être décomposée comme un ensemble de règles de préférences, définies ainsi :

Définition 2.2.2 (Règle de préférence)**.** *Soit* $\mathsf{CPT}_{\mathbf{X}}$ *une table de préférence conditionnelle définie sur* $\underline{\mathcal{P}}$*. Chaque paire* $(\mathbf{p}, \mathsf{CPT}_{\mathbf{X}}(\mathbf{p}))$ *est appelée une règle, et sera notée de la manière suivante :*

$$\mathbf{p} \ : \ \mathsf{CPT}_{\mathbf{X}}(\mathbf{p})$$

Si $\mathbf{P} = \varnothing$*, la table de préférence conditionnelle se réduit à une seule règle et n'a pour image qu'un seul ordre* $\succ_{\mathbf{X}}$*, on la notera* $\top : \succ_{\mathbf{X}}$ *ou juste* $\succ_{\mathbf{X}}$*.*

Exemple 2.2.1. Voici un exemple de table de préférence conditionnelle :

viande : *vin rouge* > *vin blanc*
poisson : *vin blanc* > *vin rouge*

Maintenant que les tables de préférences conditionnelles ont été définies, nous pouvons définir les CP-nets.

Définition 2.2.3 (CP-net)**.** *Un CP-net sur les attributs* $\mathcal{X} = \{X_1, \ldots, X_n\}$ *est un graphe dirigé avec n nœuds, chaque nœud étant étiqueté par un attribut différent. À chaque nœud est associée une table de préférence conditionnelle* CPT_{X_i} *qui associe à chaque instanciation* $\mathbf{u}$ *des parents de* X_i *un ordre total sur les valeurs de* $\underline{X_i}$*.*

Exemple 2.2.2. Le CP-net de la figure 2.1 capture la préférence suivante :

- Je préfère la salade de fruit au dessert au chocolat.
- Je préfère la viande au poisson.
- Pour les repas avec poisson, je préfère le vin blanc au vin rouge, alors que pour les repas avec viande, je préfère le vin rouge au vin blanc.

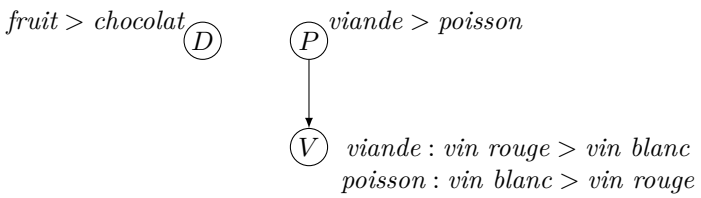


FIGURE 2.1 – Un exemple de CP-net

Sémantique des CP-nets

Afin de détailler la sémantique des CP-nets, il nous faut introduire la notion de *swap* (également appelé *flip*).

Définition 2.2.4 (Swap)**.** *Un* swap *est une paire d'objets* $\{\mathbf{o}, \mathbf{o}'\}$ *qui ne diffèrent que par la valeur d'un seul attribut, appelé attribut* swappé*. Autrement dit, il existe* $X \in \mathcal{X}$*,* $\mathbf{u} \in \underline{\mathcal{X} \setminus \{X\}}$ *et* $x, x' \in \underline{X}, x \neq x'$*, tels que* $\mathbf{o} = \mathbf{u}x$ *et* $\mathbf{o}' = \mathbf{u}x'$*.*

Un CP-net $\mathcal{C}$ ordonne tout *swap* $\{\mathbf{u}x, \mathbf{u}x'\}$ (où $X \in \mathcal{X}$ et $\mathbf{U} = \mathcal{X} \setminus \{X\}$) de la manière suivante : dans la table de préférence conditionnelle du nœud étiqueté par X, il y a une seule règle compatible avec $\mathbf{u}$. Cette règle $\succ_X$ ordonne complètement les valeurs de X. On définit alors $\succ_\mathcal{C}$ de la manière suivante : $\mathbf{u}x \succ_\mathcal{C} \mathbf{u}x'$ ssi $x \succ_X x'$.

Étant donné que les règles des tables de préférences conditionnelles dépendent de la valeur de leurs parents, un arc depuis un nœud X_1 vers un nœud X_2 signifie que la préférence sur X_2 dépend de la valeur de X_1.

Exemple 2.2.3. Le CP-net de la figure 2.1 ordonne (entre autres) deux *swaps* comme suit :

- *fruit/viande/vin blanc* $\succ_\mathcal{C}$ *chocolat/viande/vin blanc*
- *chocolat/poisson/vin blanc* $\succ_\mathcal{C}$ *chocolat/poisson/vin rouge*

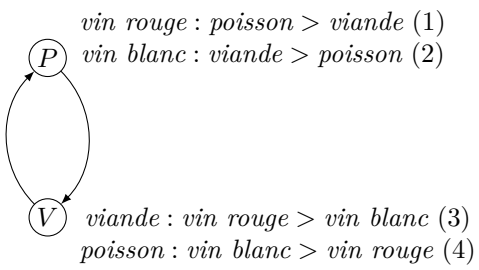
Définition 2.2.5 (Satisfaction d'un CP-net par un ordre total)**.** *On dit qu'un ordre total* $\succ$ *satisfait un CP-net* $\mathcal{C}$ *si, pour tous les swaps* $\{\mathbf{o}, \mathbf{o}'\}$ *:*

$$\mathbf{o} \succ_\mathcal{C} \mathbf{o}' \Rightarrow \mathbf{o} \succ \mathbf{o}'$$

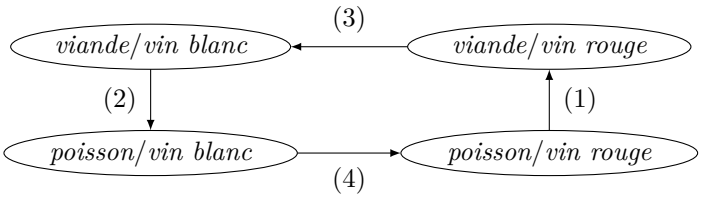
Proposition 2.2.1 ([BBD+04])**.** *L'ordre total* $\succ$ *satisfait un CP-net* $\mathcal{C}$ *si et seulement si, pour chaque succession de swaps* $\{\mathbf{s}_i\}_{i=1}^k$ *tels que* $\mathbf{s}_1 \succ_\mathcal{C} \mathbf{s}_2 \succ_\mathcal{C} \ldots \succ_\mathcal{C} \mathbf{s}_k$*, alors* $\mathbf{s}_1 \succ \mathbf{s}_k$*.*

Définition 2.2.6 (Satisfiabilité d'un CP-net)**.** *Un CP-net est dit satisfiable s'il existe au moins un ordre total qui le satisfait.*

Exemple 2.2.4. Le CP-net suivant n'est pas satisfiable (l'attribut « Dessert » a été omis par simplicité).



En effet, ce CP-net représente les préférences suivantes :



Or, nous obtenons un cycle dans les *swaps* :

$$\textit{viande/vin rouge} \succ_{\mathcal{C}} \textit{viande/vin blanc} \succ_{\mathcal{C}} \textit{poisson/vin blanc} \succ_{\mathcal{C}} \textit{poisson/vin rouge} \succ_{\mathcal{C}} \textit{viande/vin rouge}$$

Il n'existe donc aucun ordre total qui satisfasse ce CP-net.

En utilisant la proposition 2.1.3, on peut définir l'ordre partiel représenté par un CP-net satisfiable comme étant l'intersection de tous les ordres qui satisfont ce CP-net. De ce fait, la présence d'incomparabilités dans l'ordre partiel représenté par un CP-net est plus due à une limitation technique de l'expressivité des CP-nets qu'à une réelle volonté de vouloir représenter l'incomparabilité.

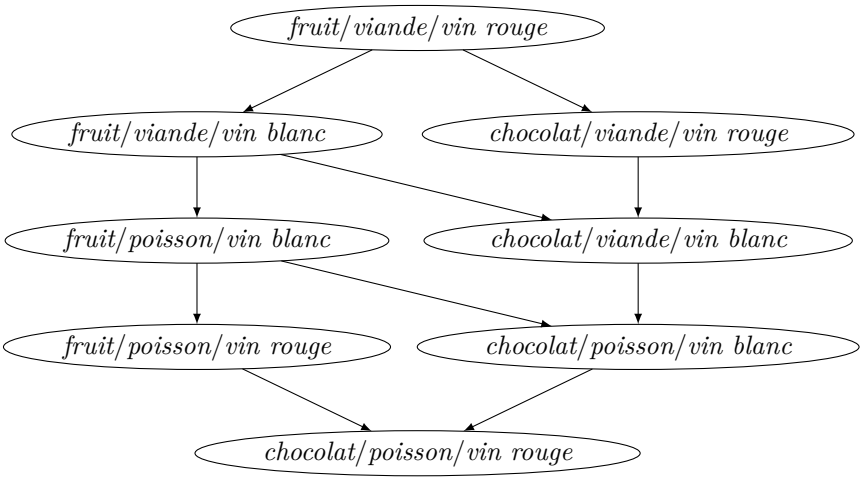
Enfin, les CP-nets sont bien adaptés à la représentation d'indépendances préférentielles, comme le montre la proposition suivante.

Proposition 2.2.2 ([BBD+04])**.** *Soient $\mathcal{C}$ un CP-net satisfiable, et X un attribut de $\mathcal{X}$. Si on note $\mathbf{Pa}(X)$ l'ensemble des attributs qui étiquettent les parents de X dans $\mathcal{C}$ et $\mathbf{u}$ le tuple de leur valeurs, alors $\{X\}$ est préférentiellement indépendant de $\mathcal{X} \setminus (\{X\} \cup \mathbf{Pa}(X))$ conditionnellement à $\mathbf{u}$.*

Exemple 2.2.5. La relation de préférences du CP-net de la figure 2.1 implique que *fruit/viande/vin rouge* est préféré à *chocolat/poisson/vin blanc*. En effet :

- $\textit{fruit/viande/vin rouge} \succ_{\mathcal{C}} \textit{cchocolat/viande/vin rouge}$
- $\textit{chocolat/viande/vin rouge} \succ_{\mathcal{C}} \textit{chocolat/viande/vin blanc}$
- $\textit{chocolat/viande/vin blanc} \succ_{\mathcal{C}} \textit{chocolat/poisson/vin blanc}$

Par contre, ce même CP-net ne permet pas de comparer les repas *chocolat/viande/vin rouge* et *fruit/poisson/vin blanc*. Une représentation graphique de l'ordre partiel de ce CP-net est présentée à la figure suivante.



CP-net acyclique

Les CP-nets acycliques ont été introduit par [BBD+04] qui remarque que cette classe de CP-net est à la fois intuitive [DBS01] et dispose de propriétés intéressantes. Les CP-nets acycliques sont définis formellement de la manière suivante :

Définition 2.2.7 (CP-net acyclique)**.** *Un CP-net acyclique est un CP-net dont le graphe ne contient aucun circuit.*

Une propriété fondamentale des CP-nets acycliques est leur satisfiabilité.

Proposition 2.2.3 (Satisfiabilité des CP-nets acycliques [BBD+04])**.** *Tous les CP-nets acycliques sont satisfiables.*

Le CP-net de la figure 2.1 est un CP-net acyclique.

CP-net séparable

Les CP-nets séparables, introduits par [LM09], constituent un cas particulier des CP-nets.

Définition 2.2.8 (CP-net séparable)**.** *Un CP-net est dit séparable si son graphe n'a pas d'arcs.*

Les CP-nets séparables sont bien plus restreints dans les préférences qu'ils peuvent représenter. De plus, on peut remarquer qu'un CP-net séparable est nécessairement acyclique, donc satisfiable. La figure 2.2 donne un exemple de CP-net séparable.

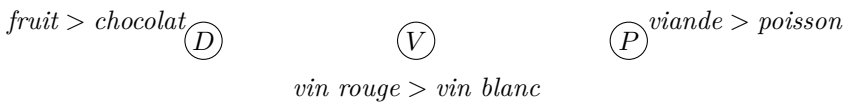


FIGURE 2.2 – Exemple de CP-net séparable

TCP-net

Les CP-nets permettent de représenter facilement des indépendances préférentielles entre attributs. Considérons par exemple le CP-net de la figure 2.1 : les préférences sur le dessert sont indépendantes de la valeur du plat principal. Le dîner préféré selon ce CP-net est *fruit/viande/vin rouge*. Néanmoins, ce CP-net ne nous permet pas de savoir quel dîner est préféré entre *fruit/poisson/vin rouge* et *chocolat/viande/vin rouge* car ces deux dîners ont chacun, par rapport au dîner préféré, un attribut dont la valeur a été déclassé (poisson au lieu de viande pour le premier, chocolat au lieu de fruit pour le second). Les TCP-nets, introduits par [BD02, BDS06], sont une extension des CP-net qui augmente l'expressivité des CP-nets en y adjoignant la représentation d'importance entre attributs.

Définition 2.2.9 (TCP-net)**.** *Un TCP-net est défini par :*

- *Un CP-net, composé d'un ensemble de n nœuds étiquetés chacun par un attribut différent de $\mathcal{X}$, d'arcs de préférence conditionnelle et de tables de préférences conditionnelles ;*
- *Des arcs d'importance inconditionnelle ; un arc d'importance inconditionnelle de X vers Y indique que X est inconditionnellement plus important que Y, i.e. $\{X\} \triangleright \{Y\}$;*
- *Des arêtes d'importance conditionnelle ; il y a une arête d'importance conditionnelle entre X et Y si et seulement si il existe $\mathbf{z}, \mathbf{z}'$ tel que $\{X\} \triangleright_{\mathbf{z}} \{Y\}$ et $\{Y\} \triangleright_{\mathbf{z}'} \{X\}$.*
- *De tables associées à chaque arête d'importance conditionnelle et qui associent aux valeurs de $\mathbf{z} \in \underline{\mathbf{Z}}$, $\mathbf{Z} \subseteq \mathcal{X} \setminus \{X, Y\}$ une relation d'importance entre X et Y conditionnellement à $\succ_{\mathbf{z}}$ (c'est-à-dire soit $X \triangleright_{\mathbf{z}} Y$, soit $Y \triangleright_{\mathbf{z}} X$).*

Sémantique des TCP-nets

Un ordre total $\succeq$ satisfait un TCP-net s'il est compatible avec toutes les indépendances préférentielles et toutes les relations d'importance conditionnelle et inconditionnelle de ce TCP-net. Tout comme pour les CP-nets, le préordre que représente un TCP-net est l'intersection de tous les ordres qui le satisfont.

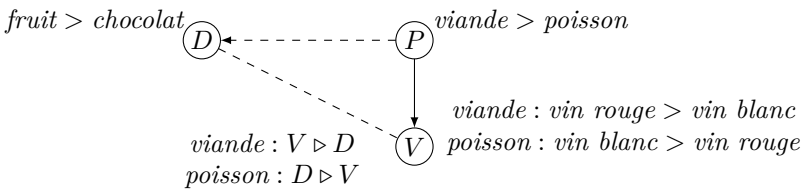
Exemple 2.2.6. Nous allons reprendre le CP-net de la figure 2.1 et le transformer en TCP-net en ajoutant les informations suivantes :

- j'accorde une plus grande importance au plat principal qu'au dessert
- pour les repas avec *viande*, j'accorde une plus grande importance au

vin qu'au dessert

- pour les repas avec *poisson*, j'accorde une plus grande importance au dessert qu'au vin

Ce TCP-net est présenté à la figure suivante :



Par rapport au CP-net de la figure 2.1, de nouvelles comparaisons sont possibles :

- $chocolat/viande/vin\ rouge \succeq fruit/poisson/vin\ rouge$ car $P \triangleright D$
- $chocolat/viande/vin\ blanc \succeq fruit/poisson/vin\ blanc$ car $P \triangleright D$
- $chocolat/viande/vin\ rouge \succeq fruit/viande/vin\ blanc$ car $V \triangleright_{viande} D$
- $fruit/poisson/vin\ rouge \succeq chocolat/poisson/vin\ blanc$ car $D \triangleright_{poisson} V$

Il se trouve qu'avec ces informations supplémentaires, ce TCP-net représente un ordre total :

$$fruit/viande/vin\ rouge \succ chocolat/viande/vin\ rouge \succ fruit/viande/vin\ blanc \succ chocolat/viande/vin\ blanc \succ fruit/poisson/vin\ blanc \succ fruit/poisson/vin\ rouge \succ chocolat/poisson/vin\ blanc \succ chocolat/poisson/vin\ rouge$$

PCP-net

Les CP-nets ne peuvent pas incorporer d'incertitude sur les préférences. Or, l'incertitude peut être utile dans la représentation des préférences afin de représenter les limitations de la connaissance. C'est dans ce contexte que [Cor12] et [BZFM13] introduisent les *PCP-net* (*Probabilistic CP-nets*), un langage qui étend les CP-nets en leur adjoignant une nature probabiliste.

Définition 2.2.10 (PCP-net)**.** *Un PCP-net est un CP-net auxquels sont adjoints :*

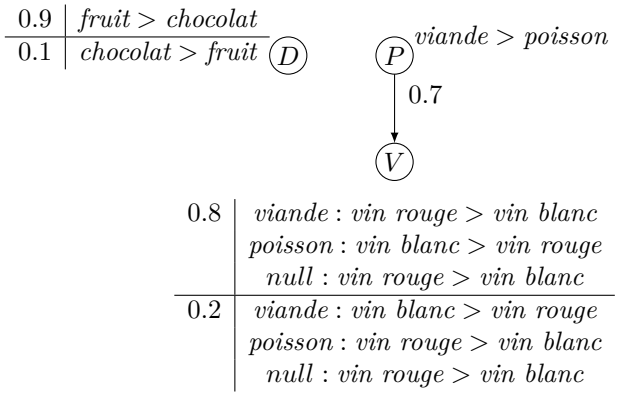
- *une probabilité d'existence, associée à chaque arc de dépendances ;*
- *une distribution de probabilité sur les ordres de préférence, associée à chaque nœud.*

Un PCP-net représente une distribution de probabilité sur l'ensemble des CP-nets de la manière suivante : la probabilité d'un CP-net selon un certain PCP-net est le produit des probabilités des arcs de ce CP-net et des probabilités de ses tables de préférences. L'exemple suivante illustre ce calcul.

Exemple 2.2.7. Nous allons étendre le CP-net de la figure 2.1 en y incluant le fait que :

- Il est probable ($p = 0.7$) que la préférence sur le vin dépende du plat principal. Si ce n'est pas le cas, on suppose que le vin rouge est inconditionnellement préféré au vin blanc.
- Les préférences sur le dessert ont 10 % de chance d'être erronées ;
- Les préférences sur le plat principal sont certaines ;
- Les préférences sur le vin ont 20 % de chance d'être erronées ;

Ce PCP-net est présenté dans la figure suivante.



On peut calculer la probabilité du CP-net $\mathcal{C}$ de la figure 2.1 selon ce TCP-net :

$$\begin{aligned} p(\mathcal{C}) &= p(\text{arc entre } P \text{ et } V)p(\mathsf{CPT}_D)p(\mathsf{CPT}_V)p(\mathsf{CPT}_P) \\ &= 0.7 \cdot 0.9 \cdot 0.7 \cdot 1 \\ &= 0.504 \end{aligned}$$

Autres variantes des CP-nets

Il existe d'autres variantes des CP-nets :

- les UCP-nets [BBB01] sont une variation des CP-nets où les tables de préférence conditionnelle sont remplacées par des fonctions d'utilité inspirées des GAI-nets.
- les CP-nets hiérarchiques [MC07], qui peuvent exprimer des préférences sur des domaines continus ;

- les WCP-nets [WZS+12], qui pondèrent les préférences conditionnelles et les attributs afin de pouvoir comparer toute paire d'objets ;

- les mCP-nets [RVW], une extension des CP-nets conçue pour représenter les préférences de plusieurs agents et dont la sémantique est basée sur un vote des différents agents.

2.2.2 Famille des LP-trees

Les arbres de préférences lexicographiques (ou *LP-tree* pour *lexicographic preference tree*) ont été proposés par [BCL+10] et permettent de représenter aisément des relations d'importance ; plus précisement, ils généralisent les ordres lexicographiques. Nous allons présenter formellement les LP-trees ainsi que certaines de leurs variantes.

LP-tree

Définition 2.2.11 (LP-tree)**.** *Un LP-tree se compose de deux parties :*

- *un arbre enraciné indiquant l'importance relative des attributs. Chaque nœud de cet arbre est étiqueté par un attribut de $\mathcal{X}$. Aucun attribut ne peut apparaître deux fois dans une branche. Pour chaque nœud étiqueté par un attribut X :*

  - *soit c'est une feuille de l'arbre ;*
  - *soit il a une seule arête sortante sans étiquette ;*
  - *soit il a $|\underline{X}|$ arêtes sortantes, chacune étiquetée par une valeur différente de X.*

- *des tables de préférences conditionnelles contenant pour chaque nœud N un ensemble de règles noté CPT_N.*

Pour un nœud N, on note $\mathbf{Anc}(N)$ l'ensemble des étiquettes des nœuds au-dessus de N. Les valeurs des attributs qui étiquettent le chemin depuis la racine jusqu'à N influencent la préférence à N. On note par $\mathbf{Inst}(N)$ l'ensemble des nœuds au-dessus de N avec des arêtes sortantes étiquetées et par $\mathbf{inst}(N)$ l'ensemble des valeurs de ces arêtes entre la racine et N (donc $\mathbf{inst}(N) \in \underline{\mathbf{Inst}(N)}$).

Si X est l'attribut qui étiquette N, alors la table de préférence conditionnelle CPT_N associe à tout tuple de valeurs $\mathbf{u}$ de $\mathbf{U} \subseteq \mathbf{Anc}(N) \setminus \mathbf{Inst}(N)$ (c'est-à-dire un sous-ensemble des attributs non affectés) une relation d'ordre total $\succ_{\mathbf{u}}$ sur les valeurs $\underline{X}$.

Exemple 2.2.8. Un LP-tree est présenté à la figure 2.3. Soit N le nœud en bas à gauche étiqueté par P : on a $\mathbf{Anc}(N) = \{D, V\}$, $\mathbf{Inst}(N) = \{D\}$ et $\mathbf{inst}(N) = \textit{fruit}$.

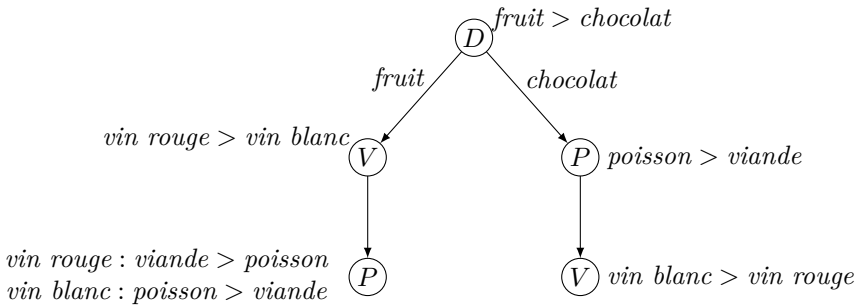


FIGURE 2.3 – Un LP-tree

Sémantique des LP-trees

[BCL$^+$10] propose une démarche constructive pour comparer deux alternatives. Afin de mettre en avant les liens étroits entre les LP-trees et la notion d'importance conditionnelle que nous avons vue plus tôt, nous définissons autrement la sémantique des LP-trees. Nous prouverons par la suite l'équivalence entre notre approche et l'approche constructive.

Rappelons tout d'abord que $\mathbf{X} \rhd_{\mathbf{z}} \mathbf{Y}(\succ_{\mathbf{X}})$ définit une relation de préférence (cf. la section 2.1.2). Nous allons associer à chaque nœud N la relation d'importance pour laquelle l'attribut à ce nœud est plus important que tous les attributs plus bas dans l'arbre. La sémantique que nous proposons est donc la suivante. Soit $\mathcal{L}$ un LP-tree. Alors la partie asymétrique $\succ_{\mathcal{L}}$ de la relation binaire représentée par $\mathcal{L}$ est définie de la manière suivante, où $X(N)$ est une notation pour l'étiquette du nœud N :

$$\succ_{\mathcal{L}} = \bigcup_{\substack{N \in \mathcal{L} \\ \mathbf{u}:\succ_{X(N)} \in \mathsf{CPT}_N}} (\{X(N)\} \rhd_{\mathbf{u},\mathbf{inst}(N)} \mathcal{X} \setminus (\{X(N)\} \cup \mathbf{Anc}(N)))(\succ_{X(N)})$$

Autrement dit, la partie asymétrique de la relation d'ordre représentée par un LP-tree est l'union des relations d'importance conditionnelle définies à chacun de ses nœuds. En adjoignant une partie symétrique qui met en relation tout objet avec lui-même, la relation d'ordre représentée par un LP-tree sera réflexive. Autrement dit :

Définition 2.2.12 (Sémantique d'un LP-tree)**.** *La relation d'ordre $\mathcal{R}$ représentée par un LP-tree $\mathcal{L}$ est définie de la manière suivante :*

$$\mathbf{o}\ \mathcal{R}\ \mathbf{o}' \textit{ ssi } \mathbf{o} \succ_{\mathcal{L}} \mathbf{o}' \textit{ ou } \mathbf{o} = \mathbf{o}'$$

Montrons tout d'abord que cette relation d'ordre est un ordre partiel.

Lemme 2.2.1. *La relation binaire représenté par un LP-tree est un ordre partiel.*

Démonstration. Soient $\mathcal{L}$ un LP-tree et $\mathcal{R}$ la relation de préférence qu'il représente. Pour montrer que $\mathcal{R}$ est un ordre partiel, nous devons montrer que $\mathcal{R}$ est réflexive, transitive et antisymétrique.

(*Réflexivité*) Pour tout $\mathbf{o} \in \underline{\mathcal{X}}$, $\mathbf{o}\ \mathcal{R}\ \mathbf{o}$, donc $\mathcal{R}$ est réflexive.

(*Transitivité*) Soient $\mathbf{o}$, $\mathbf{o}'$ et $\mathbf{o}''$ trois objets de $\mathcal{X}$ tels que $\mathbf{o}\ \mathcal{R}\ \mathbf{o}'$ et $\mathbf{o}'\ \mathcal{R}\ \mathbf{o}''$. Montrons que $\mathbf{o}\ \mathcal{R}\ \mathbf{o}''$.

Traitons d'abord le cas où $\mathbf{o} = \mathbf{o}'$. Dans ce cas, par hypothèse, $\mathbf{o}'\ \mathcal{R}\ \mathbf{o}''$, donc $\mathbf{o}\ \mathcal{R}\ \mathbf{o}''$, ce qui conclut ce cas. Le cas où $\mathbf{o}' = \mathbf{o}''$ est similaire. Concernant le cas $\mathbf{o} = \mathbf{o}''$, étant donné que $\mathcal{R}$ est réflexive on sait que $\mathbf{o}\ \mathcal{R}\ \mathbf{o}$; on peut en déduire que $\mathbf{o}\ \mathcal{R}\ \mathbf{o}''$, ce qui conclut ce cas.

Supposons par la suite que $\mathbf{o}, \mathbf{o}'$ et $\mathbf{o}''$ sont distincts. Étant donné que $\mathbf{o}$ et $\mathbf{o}'$ sont distincts, $\mathbf{o}\ \mathcal{R}\ \mathbf{o}'$ implique qu'il existe un nœud N étiqueté par X et une règle $\mathbf{u} :\succ_X$ tels que $X \triangleright_{\mathbf{u},\mathbf{inst}(N)} \mathcal{X} \setminus (\{X\} \cup \mathbf{Anc}(N))(\succ_X)$ implique que $\mathbf{o}\ \mathcal{R}\ \mathbf{o}'$. De même, il existe un nœud N' étiqueté par Y et une règle $\mathbf{v} :\succ_Y$ tels que $Y \triangleright_{\mathbf{v},\mathbf{inst}(N')} \mathcal{X} \setminus (\{Y\} \cup \mathbf{Anc}(N'))(\succ_Y)$ implique que $\mathbf{o}'\ \mathcal{R}\ \mathbf{o}''$.

Montrons que N et N' appartiennent à la même branche. Prouvons cela par l'absurde en supposant que N et N' appartiennent à deux branches différentes et notons N'' leur dernier ancêtre commun, étiqueté par l'attribut Z. N'' a donc au moins deux arêtes sortantes : notons z l'étiquette qui mène à la branche de N et z' l'étiquette qui mène à la branche de N'.

Étant donné que $\mathbf{o}'$ peut être comparé dans au nœud N, cela signifie que $\mathbf{o}'$ est compatible avec $\mathbf{inst}(N)$, et en particulier que $\mathbf{o}'[Z] = z$. Or, $\mathbf{o}'$ peut également être comparé dans N' : le même raisonnement nous montre que $\mathbf{o}'[Z] = z'$. Étant donné que z et z' sont distincts, nous aboutissons à une contradiction. Cela prouve donc que N et N' sont dans la même branche.

Supposons sans perdre de généralité que N est un ancêtre de N'. Montrons que la relation d'importance conditionnelle en N permet de comparer $\mathbf{o}$ et $\mathbf{o}''$.

Étant donné que $\mathbf{o}$ et $\mathbf{o}'$ ont été comparés avec une relation d'importance du nœud N et la règle $\mathbf{u} :\succ_X$, cela signifie que $\mathbf{o}[X] \neq \mathbf{o}'[X]$ et que $\mathbf{o}[X] \succ_X \mathbf{o}'[X]$. De plus, comme N est un ancêtre de N', l'étiquette de N (qu'on a notée X) fait partie des ancêtres de N', i.e. $X \in \mathbf{Anc}(N')$, donc X ne fait pas partie des attributs par rapport auxquelles l'étiquette de N' (qu'on a noté Y) est plus importante. Autrement dit, si $\mathbf{o}'$ et $\mathbf{o}''$ ont pu être comparés avec la relation d'importance inconditionnelle de N', c'est que $\mathbf{o}'[X] = \mathbf{o}''[X]$. De la même manière, en notant $\mathbf{U}$ l'ensemble d'attributs sur lequel est définie la règle $\mathbf{u} :\succ_X$ de N, $\mathbf{Inst}(N) \cup \mathbf{U} \subseteq \mathbf{Inst}(N')$ donc $\mathbf{inst}(N) = \mathbf{o}[\mathbf{Inst}(N)] = \mathbf{o}''[\mathbf{Inst}(N)]$ et $\mathbf{u} = \mathbf{o}[\mathbf{U}] = \mathbf{o}''[\mathbf{U}]$.

Au final, $\mathbf{o}[X] \succ_X \mathbf{o}''[X]$, $\mathbf{o}[\mathbf{Inst}(N)] = \mathbf{o}''[\mathbf{Inst}(N)]$ et $\mathbf{u} = \mathbf{o}[\mathbf{U}] = \mathbf{o}''[\mathbf{U}]$, donc la règle $\mathbf{u} :\succ_X$ de CPT_N s'applique : $\mathbf{o}\ \mathcal{R}\ \mathbf{o}''$. Nous avons donc prouvé la transitivité de $\mathcal{R}$.

(*Antisymétrie*) Soient $\mathbf{o}$ et $\mathbf{o}'$ tels que $\mathbf{o}\ \mathcal{R}\ \mathbf{o}'$ et $\mathbf{o}'\ \mathcal{R}\ \mathbf{o}$. Pour prouver l'antisymétrie, nous cherchons à prouver que $\mathbf{o} = \mathbf{o}'$. Il y a deux possibilités : soit $\mathbf{o} = \mathbf{o}'$, soit il existe deux nœuds distincts N et N' (étiqueté respectivement par X et Y) dont une relation d'importance conditionnelle compare différemment $\mathbf{o}$ et $\mathbf{o}'$ (resp. avec la règle $\mathbf{u} :\succ_X$ et $\mathbf{v} :\succ_Y$). Montrons que cette seconde possibilité aboutit à une contradiction.

En appliquant le même raisonnement que précédemment, on peut montrer que N et N' appartiennent à la même branche. Supposons sans perdre de généralité que N soit un ancêtre de N', i.e. $X \in \mathbf{Anc}(N')$. L'attribut Y de N' n'est donc pas plus important que X, ce qui signifie que si une règle de N' peut comparer $\mathbf{o}$ et $\mathbf{o}'$, c'est que $\mathbf{o}[X] = \mathbf{o}'[X]$. Or, étant donné que N permet de comparer $\mathbf{o}$ et $\mathbf{o}'$, cela signifie que $\mathbf{o}[X] \neq \mathbf{o}[X']$. Nous aboutissons donc à une contradiction : N et N' ne peuvent être distincts.

Donc nécessairement, $\mathbf{o} = \mathbf{o}'$, ce qui prouve l'antisymétrie de $\mathcal{R}$.

Étant donné que la relation $\mathcal{R}$ est réflexive, transitive et antisymétrique, il s'agit d'un ordre partiel. □

La proposition suivante montre l'équivalence entre notre sémantique et la sémantique de [BCL+10], qui peut être écrite sous la forme de l'algorithme 1.

Algorithm 1: Compare deux alternatives selon un LP-tree

Input: $\mathcal{L}$ un LP-tree, $\mathbf{o}$ et $\mathbf{o}'$ deux objets
Output: Renvoie *vrai* si $\mathcal{L}$ satisfait $\mathbf{o} \succeq \mathbf{o}'$, *faux* sinon
Algorithm `CompareInLPTree`$(\mathcal{L}, \mathbf{o}, \mathbf{o}')$

```
1  if o = o' then
2      return vrai
3  N ← la racine de L
4  X ← l'attribut qui étiquette N
5  >_X ← l'unique règle de CPT_N compatible avec o et o'
6  if o[X] ≠ o'[X] then
7      return o[X] >_X o'[X]
8  else if N est une feuille then
       // Possible seulement si le LP-tree n'est pas complet
9      return faux
10 else if N a un unique enfant N' then
11     L' ← le sous-arbre de L dont N' est la racine
12     return CompareInLPTree(L', o, o')
13 else
14     N' ← l'enfant de N par l'arête sortante étiquetée par o[X]
15     L' ← le sous-arbre de L dont N' est la racine
16     return CompareInLPTree(L', o, o')
```

Proposition 2.2.4. *Soit un LP-tree et $\succeq$ l'ordre partiel qu'il représente d'après la sémantique définie en 2.2.12. Alors, pour tout couple $\mathbf{o}, \mathbf{o}'$:*

$$\mathbf{o} \succeq \mathbf{o}' \Leftrightarrow \texttt{CompareInLPTree}(\mathcal{L}, \mathbf{o}, \mathbf{o}') \textit{ est vrai}$$

Démonstration. Le cas où $\mathbf{o} = \mathbf{o}'$ est trivial. Supposons pour la suite que $\mathbf{o} \neq \mathbf{o}'$.

($\Leftarrow$) Prouvons tout d'abord que si `CompareInLPTree`$(\mathcal{L}, \mathbf{o}, \mathbf{o}')$ retourne *vrai*, alors $\mathbf{o} \succeq \mathbf{o}'$. Étant donné que $\mathbf{o}$ et $\mathbf{o}'$ sont distincts, si `CompareInLPTree`$(\mathcal{L}, \mathbf{o}, \mathbf{o}')$ est vrai, c'est qu'il existe une règle $\mathbf{u} :\succ_X$ associée à un nœud N telle que $\mathbf{o}[X] \succ_X \mathbf{o}'[X]$ (ligne 7). Cette même règle se retrouve dans $\succ_{\mathcal{L}}$ et implique que $\mathbf{o} \succeq \mathbf{o}'$

($\Rightarrow$) Prouvons que si $\mathbf{o} \succeq \mathbf{o}'$, alors `CompareInLPTree`$(\mathcal{L}, \mathbf{o}, \mathbf{o}')$ est vrai. Étant donné que $\mathbf{o}$ et $\mathbf{o}'$, si $\mathbf{o} \succeq \mathbf{o}'$, c'est qu'il existe une règle $\mathbf{u} :\succ_X$ associée à un nœud N telle que $(\{X\} \triangleright_{\mathbf{u},\mathbf{inst}(N)} \mathcal{X} \setminus (\{X\} \cup \mathbf{Anc}(N)))(\succ_X)$ implique que $\mathbf{o} \succeq \mathbf{o}'$, autrement dit $\mathbf{o}[X] \succ_X \mathbf{o}'[X]$. Montrons que lors de son exploration de l'arbre, l'algorithme 1 aboutit au nœud N.

L'algorithme 1 est un parcours d'arbre ; à chaque embranchement, il choisit l'unique sous-arbre compatible avec $\mathbf{o}$ et $\mathbf{o}'$ et s'arrête lorsqu'il a atteint une feuille ou lorsqu'il a pu comparer $\mathbf{o}$ et $\mathbf{o}'$. Ainsi, étant donné que tous les

nœuds N' que l'algorithme 1 n'explore pas ne sont pas compatibles avec $\mathbf{o}$ ou $\mathbf{o}'$ (i.e. $\mathbf{inst}(N') \neq \mathbf{o}[\mathbf{Inst}(N')]$ ou $\mathbf{inst}(N') \neq \mathbf{o}'[\mathbf{Inst}(N')]$), il va explorer tous les nœuds N'' compatibles avec $\mathbf{o}$ et $\mathbf{o}'$ (i.e. $\mathbf{inst}(N'') = \mathbf{o}[\mathbf{Inst}(N'')] = \mathbf{o}'[\mathbf{Inst}(N'')]$).

Or, le nœud N dont une règle a permis de comparer $\mathbf{o}$ et $\mathbf{o}'$ provient d'une relation d'importance conditionnelle sachant $\mathbf{inst}(N)$: si cette règle a pu comparer $\mathbf{o}$ et $\mathbf{o}'$, c'est que $\mathbf{inst}(N) = \mathbf{o}[\mathbf{Inst}(N)] = \mathbf{o}'[\mathbf{Inst}(N)]$. Donc l'algorithme 1 va explorer le nœud N et, en vérifiant que $\mathbf{o}[X] \succ_X \mathbf{o}'[X]$, retourner *vrai*. □

Exemple 2.2.8 (Suite de l'exemple précédent)**.** La relation de préférence représentée par le LP-tree de la figure 2.3 est :

$$\begin{aligned} &\textit{fruit/vin rouge/viande} \succ \textit{fruit/vin rouge/poisson} \succ \\ &\quad \textit{fruit/vin blanc/poisson} \succ \textit{fruit/vin blanc/viande} \succ \\ &\quad \textit{chocolat/vin blanc/poisson} \succ \textit{chocolat/vin rouge/poisson} \succ \\ &\quad \textit{chocolat/vin blanc/viande} \succ \textit{chocolat/vin rouge/viande} \end{aligned}$$

Définition 2.2.13 (LP-tree complet)**.** *Soit $\mathcal{L}$ un LP-tree. Une branche de $\mathcal{L}$ est complète si et seulement si elle contient tous les attributs de $\mathcal{X}$. Si toutes les branches de $\mathcal{L}$ sont complètes, alors $\mathcal{L}$ est dit lui-même complet.*

Proposition 2.2.5 ([BH12])**.** *Un LP-tree représente un ordre total si et seulement si son arbre est complet.*

Dans un LP-tree complet, étant donné que l'ordre représenté est un ordre total, chaque alternative a un rang. [LMX12] a montré que ce rang peut être calculé en temps polynomial et que, pour un rang donné, trouver l'alternative qui y correspond peut être effectué en temps polynomial.

Les LP-trees non-complets ont été spécifiquement étudiés dans l'article [LT15]. Par la suite, sauf mention contraire, nous nous intéresserons uniquement aux LP-trees complets.

LP-tree linéaire

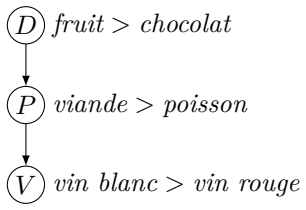
Définition 2.2.14 (LP-tree linéaire)**.** *Un LP-tree linéaire (également appelé LPM pour Lexicographic Preference Model par [YWLd10]) est un LP-tree avec une unique branche et dont chaque table de préférences conditionnelles ne contient qu'une seule règle.*

Il s'agit d'une forte restriction en termes d'expressivité : les LP-trees linéaires représentent les ordres lexicographiques usuels. Dans l'article de [BCL$^+$10], ils correspondent aux LP-tree avec des préférences locales inconditionnelles et une relation d'importance inconditionnelle.

Un LP-tree linéaire est présenté à la figure 2.4.

k-LP-tree

[BH12, BHKG17] proposent d'étendre l'expressivité des LP-trees en autorisant l'étiquetage d'un nœud par un ensemble d'attributs $\mathbf{X}$ considéré comme



$$\begin{aligned} &\textit{fruit/viande/vin blanc} \succ \textit{fruit/viande/vin rouge} \succ \\ &\quad \textit{fruit/poisson/vin blanc} \succ \textit{fruit/poisson/vin rouge} \succ \\ &\quad \textit{chocolat/viande/vin blanc} \succ \textit{chocolat/viande/vin rouge} \succ \\ &\quad \textit{chocolat/poisson/vin blanc} \succ \textit{chocolat/poisson/vin rouge} \end{aligned}$$

FIGURE 2.4 – Un LP-tree linéaire et la relation de préférence qu'il représente

un seul attribut à plusieurs dimensions : les règles dans la table de préférence conditionnelle définissent des ordres totaux sur $\underline{\mathbf{X}}$, c'est-à-dire sur le produit cartésien des domaines des attributs $\mathbf{X}$. Notons que, de ce fait, n'importe quel ordre total $\mathcal{R}$ peut en principe être représenté par un tel LP-tree, possiblement en étiquetant la racine avec l'ensemble des attributs $\mathcal{X}$ et en lui associant la table de préférence qui contient pour unique règle $\top : \mathcal{R}$.

On peut restreindre la taille de ces LP-trees en fixant le nombre maximal d'attributs qui peuvent étiqueter un nœud. On note k-LP-tree un LP-tree dont les nœuds sont étiquetés par au plus k attributs. Par exemple, la figure 2.5 montre un 2-LP-tree dont les préférences ne peuvent pas être exprimées avec un LP-tree classique.

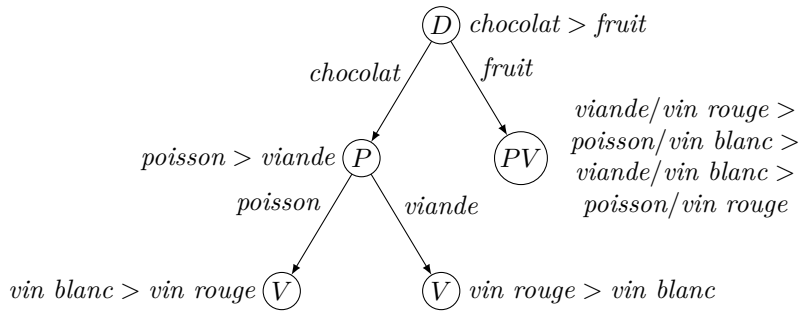
Forêts de LP-trees partiels

Dans sa thèse, [Liu16] propose de représenter une relation de préférence par une forêt (c'est-à-dire une collection) de LP-trees partiels (i.e. non-complets). Chaque LP-tree partiel représente un ordre partiel. Pour agréger ces ordres partiels, [Liu16] propose d'utiliser la méthode de Condorcet : en notant $N(\mathbf{o}, \mathbf{o}')$ le nombre de LP-trees partiels selon lesquels $\mathbf{o} \succ \mathbf{o}'$, on peut comparer deux objets $\mathbf{o}$ et $\mathbf{o}'$ à l'aide d'une forêt de LP-trees partiels de la manière suivante :

- si $N(\mathbf{o}, \mathbf{o}') > N(\mathbf{o}', \mathbf{o})$, la forêt représente $\mathbf{o} \succ \mathbf{o}'$;
- si $N(\mathbf{o}, \mathbf{o}') < N(\mathbf{o}', \mathbf{o})$, la forêt représente $\mathbf{o}' \succ \mathbf{o}$;
- si $N(\mathbf{o}, \mathbf{o}') = N(\mathbf{o}', \mathbf{o})$, la forêt représente $\mathbf{o} \approx \mathbf{o}'$.

Le problème avec ce langage est que l'ordre représenté peut contenir des cycles (de longueur supérieure à 1) du fait du paradoxe de Condorcet [Con85].

Les langages suivants sont des langages de représentation numériques. Les langages de représentation numériques s'appuient sur la proposition 2.1.2 selon laquelle un préordre complet peut être représentée par une fonction réelle appelée *fonction d'utilité*, ou encore *ranking* (notée le plus souvent u). Ces langages



$$chocolat/poisson/vin\ blanc \succ chocolat/poisson/vin\ rouge \succ chocolat/viande/vin\ rouge \succ chocolat/viande/vin\ blanc \succ fruit/viande/vin\ rouge \succ fruit/poisson/vin\ blanc \succ fruit/viande/vin\ blanc \succ fruit/poisson/vin\ rouge$$

FIGURE 2.5 – Un 2-LP-tree et la relation de préférence qu'il représente

cherchent à représenter de manière compacte une fonction d'utilité, ce qui passe par une décomposition de la fonction de représentation en sous-fonctions en exploitant différentes indépendances. Parfois, le préordre complet sera représenté par une fonction de *pénalité* p : alors que les objets préférés maximisent la fonction d'utilité, ils minimisent la fonction de pénalité. Autrement dit, $-p$ peut être considérée comme une fonction d'utilité.

2.2.3 GAI-net et GAI-tree

Les GAI-net décomposent la fonction d'utilité en une somme de sous-fonctions définies chacune sur un sous-ensemble des attributs. Plus précisément, les GAI-net se basent sur une notion introduite par [Fis70] : l'indépendance additive généralisée (en anglais *Generalized Additive Independence*, ou GAI).

Définition 2.2.15 (Indépendance additive généralisée)**.** *Soient $\mathbf{Z}_1, \ldots, \mathbf{Z}_k$ des sous-ensembles de $\mathcal{X}$ tels que $\mathcal{X} = \cup_{i=1}^{k} \mathbf{Z}_i$. On dit qu'il y a indépendance additive généralisée entre $\mathbf{Z}_1, \ldots, \mathbf{Z}_k$ pour un ordre total $\mathcal{R}$ si et seulement s'il existe des fonctions réelles $u_i : \mathbf{Z}_i \mapsto \mathbb{R}$ telles que la fonction u, définie par*

$$u(\mathbf{o}) = \sum_{i=1}^{k} u_i(\mathbf{o}[\mathbf{Z}_i]) \text{ pour tout } \mathbf{o} \in \underline{\mathcal{X}}$$

soit strictement croissante par rapport à $\mathcal{R}$. On parle alors de décomposition GAI pour u.

Exemple 2.2.9. Soient par exemple les sous-ensembles suivants :

- $\mathbf{Z}_1 = \{D\}$

- $\mathbf{Z}_2 = \{V, P\}$

On définit deux fonctions d'utilité u_1 et u_2 respectivement sur $\underline{\mathbf{Z}_1}$ et $\underline{\mathbf{Z}_2}$.

$\underline{\mathbf{Z}_1}$	*chocolat*	*fruit*
u_1	130	0

$\underline{\mathbf{Z}_2}$	*vin blanc/poisson*	*vin blanc/viande*
u_2	6	4

$\underline{\mathbf{Z}_2}$	*vin rouge/poisson*	*vin rouge/viande*
u_2	2	8

On peut alors calculer l'utilité $u(\mathbf{o})$ d'un objet $\mathbf{o}$ à partir des fonctions d'utilité u_1 et u_2 de la manière suivante : $u(\mathbf{o}) = u_1(\mathbf{o}[\mathbf{Z}_1]) + u_2(\mathbf{o}[\mathbf{Z}_2])$.

$$\begin{cases} u(\textit{fruit/viande/vin rouge}) = 0 + 8 = 8 \\ u(\textit{fruit/viande/vin blanc}) = 0 + 4 = 4 \\ u(\textit{fruit/poisson/vin rouge}) = 0 + 2 = 2 \\ u(\textit{fruit/poisson/vin blanc}) = 0 + 6 = 6 \\ u(\textit{chocolat/viande/vin rouge}) = 130 + 8 = 138 \\ u(\textit{chocolat/viande/vin blanc}) = 130 + 4 = 134 \\ u(\textit{chocolat/poisson/vin rouge}) = 130 + 2 = 132 \\ u(\textit{chocolat/poisson/vin blanc}) = 130 + 6 = 136 \end{cases}$$

On en déduit la relation de préférence suivante :

$$\begin{aligned} &\textit{chocolat/viande/vin rouge} \succ \textit{chocolat/poisson/vin blanc} \succ \\ &\quad \textit{chocolat/viande/vin blanc} \succ \textit{chocolat/poisson/vin rouge} \succ \\ &\quad\quad \textit{fruit/viande/vin rouge} \succ \textit{fruit/poisson/vin blanc} \succ \\ &\quad\quad\quad \textit{fruit/viande/vin blanc} \succ \textit{fruit/poisson/vin rouge} \end{aligned}$$

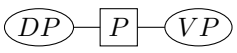
Les GAI-net forment un modèle graphique introduit par [GP04] permettant de représenter des indépendances GAI. Ils sont inspirés des arbres de jonction développés pour les réseaux bayésiens par [LS88]. N'importe quelle décomposition GAI peut être représentée sous la forme d'un GAI-net.

Définition 2.2.16 (GAI-net)**.** *Soient $\mathbf{Z}_1, \ldots, \mathbf{Z}_k$ des sous-ensembles de $\mathcal{X}$ tels que $\mathcal{X} = \cup_{i=1}^{k} \mathbf{Z}_i$ et tels qu'il y ait une indépendance additive généralisée entre $\mathbf{Z}_1, \ldots, \mathbf{Z}_k$ pour $\mathcal{R}$. Autrement dit, il existe une fonction d'utilité $u = \sum_{i=1}^{k} u_i$ soit strictement croissante par rapport à $\mathcal{R}$ pour laquelle chaque u_i est définie sur l'ensemble $\underline{\mathbf{Z}_i}$. Alors un GAI-net représentant u est un graphe non orienté (V, E) tel que :*

- $V = \{\mathbf{Z}_1, \ldots, \mathbf{Z}_k\}$
- *Pour chaque arête $\{\mathbf{Z}_i, \mathbf{Z}_j\} \in E$, $\mathbf{Z}_i \cap \mathbf{Z}_j \neq \varnothing$. De plus, pour tout paire de sommets $\mathbf{Z}_i, \mathbf{Z}_j$ tel que $\mathbf{T}_{ij} = \mathbf{Z}_i \cap \mathbf{Z}_j \neq \varnothing$, il existe un chemin dans G entre $\mathbf{Z}_i$ et $\mathbf{Z}_j$ dont les nœuds contiennent tous les attributs de $\mathbf{T}_{ij}$.*

De plus, chaque arête $\{\mathbf{Z}_i, \mathbf{Z}_j\}$ est étiquetée par l'ensemble non vide $\mathbf{Z}_i \cap \mathbf{Z}_j$.

Exemple 2.2.9 (Suite de l'exemple)**.** Le GAI-net qui représente les indépendances GAI de l'exemple précédent est présenté à la figure suivante, où les nœuds sont dessinés avec des ellipses et l'étiquette de l'arc est dessiné avec un rectangle.



Un GAI-tree est un GAI-net dont le graphe est un arbre. Comme le montre [GP04], tout GAI-net peut être représenté sous la forme d'un GAI-tree (la transformation, similaire à celle des arbres de jonction des réseaux bayésiens, est NP-difficile [Kjæ90]). L'avantage des GAI-trees est que les requêtes ont une complexité plus faible.

2.2.4 VCSP

Les problèmes de satisfaction de contraintes (CSP, pour *Constraint Satisfaction Problem* sont des problèmes dans lesquels on cherche des combinaisons de valeurs qui satisfont un ensemble de contraintes. Une contrainte peut refléter, par exemple, l'impossibilité d'affecter à deux cours différents la même salle au même moment. Nous invitons le lecteur à lire [RVBW06] pour une vue d'ensemble approfondie sur le sujet.

Formellement, un CSP est défini de la manière suivante.

Définition 2.2.17 (CSP)**.** *Un CSP est un triplet* $(\mathcal{X}, D, C)$ *:*

- $\mathcal{X}$ *est l'ensemble des attributs du problème ;*
- D *est l'ensemble des domaines de ces attributs, i.e. pour tout* $X \in \mathcal{X}$*,* $D(X) = \underline{X}$ *;*
- C *est l'ensemble des contraintes. Une contrainte* c *est un couple* $(\mathbf{X}_c, R_c)$ *listant les tuples* $R_c \subseteq \underline{\mathbf{X}_c}$ *autorisés pour l'ensemble d'attributs* $\mathbf{X}_c \subseteq \mathcal{X}$*.*

On dit qu'un objet $\mathbf{o} \in \underline{\mathcal{X}}$ est *solution* d'un CSP si $\mathbf{o}$ vérifie toutes les contraintes de ce CSP, c'est-à-dire si, pour toutes les contraintes $(\mathbf{X}_c, R_c)$, $\mathbf{o}[\mathbf{X}_c] \in R_c$.

Hormis les préférences binaires telles que « j'aime »/« je n'aime pas », les préférences ne font pas partie du formalisme CSP, mais peuvent y être naturellement introduites. Les préférences, en effet, expriment des *desiderata* qui ne sont pas indispensables. Par exemple, les étudiants peuvent préférer avoir un emploi du temps dans lequel les distances entre deux salles successives sont faibles, afin de ne pas traverser le campus entre chaque cours. [MRS06] remarque que les préférences peuvent être modélisées sous la forme de contraintes « souples » : la contrainte d'avoir des emplois du temps avec salles proches doit être respectée autant que possible mais peut être violée.

[SFV95] introduit une extension du formalisme CSP, appelée VCSP (*valued constraint satisfaction problem*), qui permet d'incorporer des contraintes souples.

Définition 2.2.18 (VCSP)**.** *Un VCSP est défini par un CSP* $(\mathcal{X}, D, C)$*, une structure de valuation* $(E, \circledast, \succ)$ *et une application* ϕ*, où :*

- E *désigne l'ensemble des degrés que peut recevoir une contrainte ;*

- $\circledast$ *est un opérateur permettant de manipuler les degrés des contraintes ;*
- $\succ$ *est un préordre complet permettant de comparer ces degrés ;*
- $\phi : C \mapsto E$ *est une application qui associe à chaque contrainte* c *un coût de violation* $\phi(c)$*.*

La fonction de pénalité p est alors définie par :

$$p(\mathbf{o}) = \underset{\substack{c \in C \\ \mathbf{o} \text{ viole } c}}{\circledast} \phi(c)$$

Pour comparer deux valeurs de pénalité, on utilise l'opérateur $\succ$ de la structure de valuation ; on cherche à *minimiser* cette pénalité. La sémantique d'un VCSP en tant que langage de représentation de préférence est donc :

$$\mathbf{o}' \succeq \mathbf{o} \Leftrightarrow p(\mathbf{o}') \preceq p(\mathbf{o})$$

Les VCSP permettent d'exprimer différents langages selon la structure de valuation utilisée ; en voici quelques exemples. Le lecteur pourra lire [BMR+99] pour une vue d'ensemble du sujet.

CSP pondéré

[FW92] introduit les *PCSP* (CSP partiels), plus tard nommés *WCSP* (CSP pondérés), qui associent un poids à chaque contrainte. Ils peuvent être représentés sous la forme d'un VCSP avec la structure de valuation $(\mathbb{N} \cup \{+\infty\}, +, >)$. Autrement dit, la fonction de pénalité d'un WCSP s'exprime de la manière suivante :

$$p(\mathbf{o}) = \sum_{\substack{c \in C \\ \mathbf{o} \text{ viole } c}} \phi(c)$$

Le formalisme des WCSP ressemble à celui des GAI-nets vus précédemment, à ceci près qu'ils intègrent en plus la valuation $+\infty$ qui correspond aux contraintes dures (bien que [GPQ08] propose une notion équivalente pour les GAI-nets). Ces deux formalismes expriment la même idée et ont été introduits dans des contextes différents : alors que les GAI-nets ont été introduits pour agréger des fonctions d'utilité, les WCSP recherchent des situations minimisant le coût des contraintes violées.

CSP flou

En se basant sur les travaux de [Lan91] qui montrent que la théorie des possibilités peut être utilisée pour exprimer des contraintes souples, [DFP93] introduit les *Fuzzy CSP* (CSP flous) qui définissent une distribution de possibilité sur l'espace des objets. Autrement dit, à chaque objet est associé un degré qui estime à quel point cet objet satisfait toutes les contraintes. Pour cela, [DFP93] associe à chaque contrainte souple une priorité entre 0 et 1. Les CSP flous peuvent être représentés sous la forme d'un VCSP avec la structure de valuation $([0, 1], \min, <)$.

CSP lexicographique

[FLS93] introduit les *Lexicographic CSP* qui ordonnent les contraintes de préférence : deux solutions satisfaisant les contraintes dures sont comparées en vérifiant une à une, de manière lexicographique, les contraintes de préférences. Ils peuvent être représentés sous la forme d'un VCSP avec une structure de valuation utilisant un ordre lexicographique.

CSP probabiliste

[FL93] introduit les *Probabilistic CSP* qui suppose ne pas connaître complètement le vrai problème de satisfaction de contraintes ; ils associent à chaque contraintes une probabilité d'être réellement présente dans le problème. À chaque solution est associée un degré de probabilité de satisfaire les contraintes ; la présence d'une contrainte étant supposée indépendante de la présence des autres, le degré de probabilité d'une solution est simplement le produit des probabilités des contraintes qu'elle viole. Les CSP probabilistes peuvent être représentés sous la forme d'un VCSP avec la structure de valuation $([0,1], (x,y) \mapsto x + y - xy, >)$.

2.2.5 Langages basés sur la logique propositionnelle

Il est également possible d'exprimer des préférences avec des logiques. Nous allons présenter quelques langages basés sur la logique propositionnelle, étudiés dans [CLLM04]. Dans ce cas, on considère que les attributs de $\mathcal{X}$ sont à domaine booléen (quitte à les dichotomiser en ajoutant d'autres attributs) et sont donc des variables propositionnelles. Chaque objet $\mathbf{o} \in \underline{\mathcal{X}}$ est alors considéré comme une interprétation.

Ces langages booléens s'appuient sur un ensemble de formules propositionnelles G_i (pour *goal*, objectif). À chaque formule peut être associé un poids, une priorité, ou d'autres informations selon les langages. On note $\mathbf{o} \models G_i$ si l'interprétation $\mathbf{o}$ est un modèle de la formule G_i.

Par pénalités

Dans ce langage introduit par [Han92], chaque formule G_i est associée à une pénalité α_i. Pour calculer la pénalité d'un objet $\mathbf{o}$, il suffit d'additionner les pénalités des objectifs qu'il ne remplit pas :

$$p(\mathbf{o}) = \sum_{\substack{G_i \\ \mathbf{o} \not\models G_i}} \alpha_i$$

Par distance aux objectifs

L'approche par pénalité pénalise les objets en fonction des objectifs qu'ils ne remplissent pas. Néanmoins, entre deux objets $\mathbf{o}$ et $\mathbf{o}'$ qui satisfont les mêmes objectifs, un utilisateur peut tout de même avoir une préférence pour l'objet le plus « proche » de remplir une formule G_i. C'est l'approche apportée par [LL01].

Soit $d(\mathbf{o}, \mathbf{o}')$ la distance de Hamming entre deux objets $\mathbf{o}$ et $\mathbf{o}'$. Pour chaque formule G, on définit alors la distance d'un objet $\mathbf{o}$ à G de la manière suivante :

$$d(\mathbf{o}, G) = \min_{\mathbf{o}' \models G} d(\mathbf{o}, \mathbf{o}')$$

Dans ce cas, la pénalité de $\mathbf{o}$ est définie par :

$$p(\mathbf{o}) = \sum_{G_i} \alpha_i d(\mathbf{o}, G_i)$$

Autres approches logiques

Il existe également d'autres représentations de préférences basées sur une logique, parmi lesquelles nous pouvons citer :

- les objectifs prioritisés [BCD+93, Bre89], qui ajoute un ordre lexicographique aux objectifs ;
- les logiques conditionnelles [Bou94], qui identifient des contextes dans lesquelles les objectifs sont remplis ;
- les préférences *ceteris paribus* [WD91], qui permettent d'exprimer les préférences *ceteris paribus* que nous avons vues précédemment.

2.2.6 Réseaux bayésiens

Les réseaux bayésiens, introduits par [Pea89], permettent de représenter de manière compacte une distribution de probabilité sur un espace combinatoire, en utilisant les indépendances probabilistes existant entre certains attributs d'un problème donné.

Ils n'ont pas été introduits dans un objectif de représentation de préférences et c'est pour cela que nous leur avons réservé une section à part. Comme nous allons le voir, les réseaux bayésiens peuvent facilement être adaptés à la représentation de préférences.

Définition et propriétés

Définition 2.2.19 (Réseau bayésien)**.** *Un réseau bayésien sur l'ensemble d'attributs $\mathcal{X}$ est une paire $(\mathcal{G}, \Theta)$ où*

- *$\mathcal{G}$ est un graphe acyclique orienté dont les sommets sont les attributs de $\mathcal{X}$.*
- *Θ est un ensemble de lois de probabilité conditionnelles : pour chaque attribut $X \in \mathcal{X}$, $\Theta(X \mid \mathbf{Pa}(X))$ dénote la loi de probabilité de X sachant les valeurs des parents de X. Si tous les attributs sont discrets, alors pour tout X, $\Theta(X \mid \mathbf{Pa}(X))$ est une table de probabilités conditionnelles. De plus, si $x \in \underline{X}$, $\mathbf{u} \in \mathbf{Pa}(X)$, 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 sur $\underline{\mathcal{X}}$. Dans notre cas, il représente une distribution de probabilité sur les alternatives. La probabilité d'une alternative $\mathbf{o} \in \underline{\mathcal{X}}$ est :

$$p(\mathbf{o}) = \prod_{X \in \mathcal{X}} \Theta(\mathbf{o}[X] \mid \mathbf{o}[\mathbf{Pa}(X)]) \tag{2.1}$$

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 certains attributs ou groupes d'attributs.

Définition 2.2.20 (Indépendance conditionnelle)**.** *Étant donnée une distribution de probabilité p sur $\underline{\mathcal{X}}$, 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 2.2.6. *Étant donné un réseau bayésien $(\mathcal{G}, \Theta)$ sur $\mathcal{X}$, tout attribut X est indépendant de ses non-descendants sachant ses parents. Formellement, si $\mathbf{N}$ désigne l'ensemble des attributs Y telles qu'il n'y a pas de chemin dans $\mathcal{G}$ de X à Y, alors $X \perp\!\!\!\perp \mathbf{N} \mid \mathbf{Pa}(X)$.*

Les réseaux bayésiens sont fondés sur cette notion d'indépendance conditionnelle, car c'est elle qui permet de représenter une loi de probabilité sous forme compacte.

Un des grands avantages des réseaux bayésiens est leur pouvoir de représentation. En effet :

Proposition 2.2.7 ([Pea89])**.** *Toute loi de probabilité discrète qui ne s'annule pas peut être représentée par un réseau bayésien.*

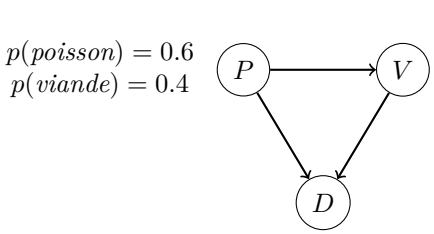
C'est l'exploitation de ces indépendances conditionnelles qui a fait le succès des réseaux bayésiens, puisqu'elles permettent de simplifier des calculs de probabilités marginales, et peuvent être détectées par des tests statistiques lorsqu'on veut apprendre un réseau bayésien à partir d'un jeu de données.

Lors de l'apprentissage du réseau bayésien, on réalise en effet des tests d'indépendance conditionnelle sur les données d'apprentissage pour construire un réseau bayésien dont la structure les respecte. Lors d'un calcul de probabilité à partir d'un réseau bayésien, les indépendances conditionnelles décrites par la structure du réseau bayésien permettent de décomposer les calculs en sous-calculs, ce qui permet d'aboutir à l'équation (2.1).

Adaptation à la représentation de préférences

Un réseau bayésien représente une distribution de probabilité sur l'espace $\underline{\mathcal{X}}$. En nous inspirant des langages de représentation numériques, nous pouvons définir le préordre complet représenté par un réseau bayésien de la manière suivante : $\mathbf{o} \succeq \mathbf{o'}$ si et seulement si $p(\mathbf{o}) \geq p(\mathbf{o'})$.

Exemple 2.2.10. Un exemple de réseau bayésien est représenté sur la figure suivante.



$$p(fruit \mid poisson, vin\ blanc) = 0.8$$
$$p(chocolat \mid poisson, vin\ blanc) = 0.2$$
$$p(fruit \mid poisson, vin\ rouge) = 0.4$$
$$p(chocolat \mid poisson, vin\ rouge) = 0.6$$
$$p(fruit \mid viande, vin\ blanc) = 0.7$$
$$p(chocolat \mid viande, vin\ blanc) = 0.3$$
$$p(fruit \mid viande, vin\ rouge) = 0.3$$
$$p(chocolat \mid viande, vin\ rouge) = 0.7$$

On souhaite comparer le dîner *viande/vin rouge/chocolat* et le dîner *poisson/vin rouge/fruit*. Pour cela, on calcule la probabilité de chaque menu :

$$\begin{aligned} p(viande/vin\ rouge/chocolat) &= \Theta(viande)\Theta(vin\ rouge \mid viande) \\ &\quad \times \Theta(chocolat \mid viande, vin\ rouge) \\ &= 0.4 \cdot 0.9 \cdot 0.7 = 0.252 \end{aligned}$$

$$\begin{aligned} p(poisson/vin\ rouge/fruit) &= \Theta(poisson)\Theta(vin\ rouge \mid poisson) \\ &\quad \times \Theta(fruit \mid poisson, vin\ rouge) \\ &= 0.6 \cdot 0.2 \cdot 0.4 = 0.048 \end{aligned}$$

Comme $p(viande/vin\ rouge/chocolat) > p(poisson/vin\ rouge/fruit)$, on en conclut que le menu *viande/vin rouge/chocolat* est strictement préféré au menu *poisson/vin rouge/fruit*.

2.2.7 Autres langages de représentation de préférences

Il existe d'autres langages de représentation de représentations, dont voici quelques exemples.

- les PSDD (*Probabilistic Sentential Decision Diagram* [KdBCD14]) permettent de modéliser une distribution de probabilité sur les modèles d'une théorie propositionnelle.
- Les cp-theories (*conditional preference theories*) introduits par [Wil11] forment un langage non-graphique de représentation de préférences. C'est un langage particulièrement expressif : il peut représenter tout ordre représentable par un CP-net, un TCP-net ou un LP-tree.
- Les réseaux de préférences possibilistes [BADGP14] forment un langage de représentation graphique basé sur la théorie des possibilités.

2.3 Requêtes et complexité

Une fois un modèle de préférence établi ou appris, on peut le questionner au moyen de requêtes. Nous allons considérer deux catégories de requêtes : les requêtes d'optimisation et les requêtes de comparaison.

2.3.1 Requêtes d'optimisation

Les requêtes d'optimisation recherche un objet préféré selon préordre $\succeq$.

Meilleure extension de u : déterminer s'il existe retourner un objet $\mathbf{o} \in \underline{\mathcal{X}}$ tel que $\mathbf{o}$ soit compatible avec l'instanciation partielle $\mathbf{u}$ et qu'il n'existe aucun $\mathbf{o}' \in \underline{\mathcal{X}}$ compatible avec $\mathbf{u}$ tel que $\mathbf{o}' \succ \mathbf{o}$.

k-meilleurs : retourner un ensemble M de k objets distincts tel que pour tout $\mathbf{o} \in \underline{\mathcal{X}} \setminus M$, $\mathbf{o}' \in M : \mathbf{o} \not\succ \mathbf{o}'$.

La requête des k-meilleurs a été peu étudiée pour les ordres partiels.

2.3.2 Requêtes de comparaison

Ces requêtes de comparaison, introduites par [BBD+04], ont pour objectif, comme leur nom l'indique, de comparer deux alternatives $\mathbf{o}$ et $\mathbf{o}'$ selon un préordre $\succeq$.

Requête de dominance : déterminer si $\mathbf{o} \succeq \mathbf{o}'$. Cela signifie que pour tous les ordres totaux $\succeq'$ compatibles avec $\succeq$, $\mathbf{o} \succeq' \mathbf{o}'$.

Requête d'ordonnancement : déterminer si $\mathbf{o}' \not\succeq \mathbf{o}$. Cela signifie qu'il existe un ordre total $\succeq'$ compatible avec $\succeq$ tel que $\mathbf{o} \succeq' \mathbf{o}'$.

Dans le cas des préordres complets (comme les préordres complets et les ordres totaux), les requêtes de dominance et d'ordonnancement sont liées car $\mathbf{o} \not\succeq \mathbf{o}' \Rightarrow \mathbf{o}' \succeq \mathbf{o}$. La requête de dominance est celle qui nous intéressera dans le cas des préordres complets et des ordres totaux.

2.3.3 Complexité des requêtes

Les deux tables 2.2 et 2.3 récapitulent les connaissances actuelles sur les classes de complexité de ces requêtes selon les langages de représentation.

	Meilleure extension
k-LP-tree partiel	P
CP-net	NP-easy[a] [BD03]
CP-net acyclique	P [BBD$^+$04]

	Dominance	Ordonnancement
k-LP-tree partiel	P	P
CP-net	PSPACE-c [GLTW08]	?
CP-net acyclique	NP-c[b] [BBD$^+$04]	P [BBD$^+$04]

[a] Cette requête n'a un sens que si le CP-net est satisfiable. Tester la satisfiabilité d'un CP-net est PSPACE-complet [GLTW08].
[b] Mais on dispose d'un algorithme polynomial pour les CP-nets dont le graphe est un poly-arbre [BBD$^+$04] et d'un algorithme linéaire pour les CP-nets dont le graphe est un arbre enraciné [BZFM13].

TABLE 2.2 – Complexité des requêtes pour les langages exprimant des ordres partiels

	Meilleure extension	k-meilleurs	Dominance
k-LP-tree complet	P	P	P
Réseau bayésien	NP-hard [Shi94]	NP-hard [SG94]	P
GAI-tree	NP-hard[a] [GPQ08]	NP-hard [GPQ08]	P
UCP-net	P [BBB01]	NP-easy	P
VCSP	NP-hard[b]	NP-hard[c]	P

[a] [GPQ08] propose un algorithme en $\sum_i |\underline{\mathbf{Z}_i}|$, qui est dans le pire cas exponentiel en n.
[b] Car résoudre un CSP est déjà un problème NP-hard.
[c] Étant donné que la vérification d'une solution est P et que trouver les k-meilleurs est au moins aussi difficile que trouver la meilleure solution.

TABLE 2.3 – Complexité des requêtes pour les langages exprimant des préordres complets ou des ordres totaux

Chapitre 3

Apprentissage automatique de modèles graphiques de représentation de préférences

Sommaire

3.1 Notions d'apprentissage automatique 40
3.1.1 Problèmes classiques d'apprentissage 40
3.1.2 Apprenabilité et *sample complexity* en classification binaire 41
3.1.3 Compromis biais-variance 44
3.2 Problèmes d'apprentissage de préférences 45
3.2.1 Ordonner des objets à partir de comparaisons 45
3.2.2 Ordonner des objets à partir de notes 46
3.2.3 Ordonner des étiquettes 46
3.2.4 Apprentissage de préférences par élicitation 47
3.3 Apprentissage de modèles graphiques 47
3.3.1 Apprentissage de CP-net 48
3.3.2 Apprentissage de LP-tree 49
3.3.3 Apprentissage de GAI-net 50
3.3.4 Apprentissage de réseau bayésien 51
3.4 Apprenabilité des langages de représentation de préférences 53
3.4.1 L'apprentissage à partir de comparaisons, une classification binaire cachée 53
3.4.2 Dimension VC de quelques langages de représentation de préférence 53

L'apprentissage automatique, aussi appelé apprentissage artificiel et connu sous l'anglicisme *machine learning*, est le domaine de l'intelligence artificielle qui « englobe toute méthode permettant de construire un modèle de la réalité à partir de données, soit en améliorant un modèle partiel ou moins général, soit en créant complètement le modèle » [CM11]. Alors que pendant longtemps les logiciels s'appuyaient exclusivement sur la connaissance d'experts du domaine, nous disposons à présent de méthodes qui peuvent exploiter automatiquement des données pour détecter des motifs ou des règles. De nos jours, l'apprentissage automatique se cache derrière plusieurs succès de l'intelligence artificielle, comme la vision par ordinateur, les jeux, la détection de manœuvres frauduleuses et, bien sûr, la personnalisation de notre environnement numérique.

Ce chapitre donne des notions fondamentales d'apprentissage automatique puis décrit les problèmes classiques qui se posent dans le domaine de l'apprentissage de préférences. Enfin, nous répertorierons et présenterons certains algorithmes d'apprentissage de modèles graphiques de préférences.

3.1 Notions d'apprentissage automatique

3.1.1 Problèmes classiques d'apprentissage

Les multiples facettes de la notion d'apprentissage rendent ce domaine particulièrement vaste. Les problèmes les plus populaires en apprentissage de préférence sont des problèmes d'apprentissage *supervisé* où les données d'apprentissage sont composées d'observations étiquetées. Nous allons présenter trois familles de problèmes d'apprentissage supervisé qui se retrouvent dans l'apprentissage de préférences : la *classification*, l'apprentissage *à partir d'exemples positifs* et l'apprentissage par *élicitation* (aussi appelé apprentissage actif). [CM11] constitue un ouvrage de référence sur le sujet qui ravira certainement le lecteur intéressé.

Classification

La classification regroupe les problèmes d'apprentissage dont les données sont des couples $(\mathbf{x}, y)$ où $\mathbf{x}$ est une observation et y est la classe de cette observation, i.e. la sortie désirée. Par exemple, pour apprendre à reconnaître des chiffres écrits à la main (une tâche de vision par ordinateur), l'observation $\mathbf{x}$ est une image (le chiffre écrit) et la classe y, le chiffre en lui-même (8, par exemple).

La classification est très répandue ; voici quelques exemples classiques :

- la reconnaissance de formes, où une observation est une image et sa classe l'objet à reconnaître ;
- la détection de *spam*, où une observation est le texte d'un e-mail et sa classe est soit « *spam* », soit « non-*spam* » ;
- la reconnaissance vocale, où une observation est un son et sa classe, le mot à reconnaître ;

À partir d'un ensemble de couples $\{(\mathbf{x}_i, y_i)\}_i$, l'objectif est de construire un modèle qui permette de prédire les classes pour des observations données. Dans le cas de la reconnaissance de chiffres manuscrits, une fois avoir entraîné le système sur des images accompagnées de leur classe, on peut interroger le modèle appris sur de nouvelles images dont il prédira les classes.

Apprentissage à partir d'exemples positifs

Dans les problèmes d'apprentissage à partir d'exemples positifs, les données sont un jeu d'observations $\{\mathbf{x}_i\}_i$ tirées à partir d'une loi de probabilité. L'objectif peut être par exemple d'apprendre un modèle qui représente cette loi de probabilité.

Apprentissage actif : l'élicitation

L'apprentissage actif, également appelé élicitation, est un problème d'apprentissage introduit par [Ang88]. L'apprentissage actif tire son nom du fait qu'il se base sur l'interrogation de l'environnement (typiquement un utilisateur). Par exemple, l'algorithme d'apprentissage peut demander à l'utilisateur s'il préfère une voiture électrique bleue à une voiture essence rouge.

Contrairement aux deux problèmes précédents, l'élicitation ne dispose d'aucune donnée initiale. L'objectif est alors d'apprendre un modèle caché $\mathcal{M}'$ en posant le moins de questions possible.

3.1.2 Apprenabilité et *sample complexity* en classification binaire

L'étude théorique de l'apprenabilité s'intéresse à la « facilité » d'apprentissage de certains modèles, et notamment au nombre d'exemples nécessaires pour apprendre un modèle correctement. Ces notions ont été formalisées à l'aide de la dimension de Vapnik-Chervonenkis [VC15] et de l'apprentissage *Probably Approximately Correct* (PAC) [Val84].

Ces notions ont été introduites pour la classification binaire, c'est-à-dire avec uniquement deux classes, classiquement notées $+$ et $-$. Pour cette raison, nous allons faire un détour pour présenter les notions essentielles de la classification binaire avant de les adapter à l'apprentissage de modèles de préférences.

Vocabulaire de classification binaire

En classification binaire, il n'y a que deux classes notées $+$ et $-$. La classification binaire se retrouve dans de nombreux problèmes où on cherche à apprendre à reconnaître un certain concept. Par exemple, si on cherche à reconnaître les mangues mûres, la classe $+$ peut signifier « mûre » et la classe $-$, « pas mûre ».

En classification, on fait souvent l'hypothèse qu'il existe une certaine fonction cible qui représente le concept à apprendre et qui associe à chaque observation sa classe. L'objectif est ici d'apprendre une fonction qui soit la plus « proche » possible de cette fonction cible.

Pour des raisons que nous détaillerons par la suite, nous allons apprendre une fonction de $\underline{\mathbf{X}}$ dans $\underline{Y}$ parmi un ensemble de fonctions possibles défini à l'avance. En classification, on appellera cet ensemble *l'ensemble d'hypothèses*, et chacun de ses éléments sera appelé une *hypothèse*.

Risque empirique et risque réel

Un algorithme d'apprentissage supervisé (comme c'est le cas en classification), dispose d'un ensemble d'exemples $\mathcal{E} = \{(\mathbf{x}_i, y_i)\}_i$. Afin d'évaluer chaque hypo-

thèse, on peut calculer son risque empirique, qui est la proportion d'observations de $\mathcal{E}$ qu'elle classe mal.

Définition 3.1.1 (Risque empirique)**.** *Soient $\mathcal{E}$ un ensemble d'exemples et h une hypothèse. Le risque empirique de h est :*

$$e_{\mathcal{E}}(h) = \frac{1}{|\mathcal{E}|} \sum_{(\mathbf{x},y)\in\mathcal{E}} \mathbf{1}_{h(\mathbf{x})\neq y}$$

Si la fonction cible f fait partie de l'ensemble d'hypothèses, et que les exemples ne sont pas bruités (c'est-à-dire si, pour chaque exemple $(\mathbf{x}, y)$, on ait bien $f(\mathbf{x}) = y$), alors il existe une hypothèse dont le risque empirique est nul.

Néanmoins, notre objectif final est d'apprendre une hypothèse qui corresponde au plus près à la fonction cible sur toutes les observations, et pas uniquement sur les exemples fournis. En effet, il est parfaitement possible de trouver une hypothèse dont le risque empirique soit très faible mais qui ait de piètres facultés de généralisation, c'est-à-dire qui aurait de mauvaises performances sur d'autres observations.

C'est pour cela que la mesure qu'on cherche *in fine* à minimiser est le risque *réel*. C'est l'erreur moyenne, c'est-à-dire la proportion d'exemples que h classerait mal, si les exemples sont tirés à partir d'une loi de probabilité p.

Définition 3.1.2 (Risque réel)**.** *Soient h une hypothèse, f la fonction cible. Le risque réel de h, pour une distribution de probabilité p de tirage sur les observations données, est :*

$$e(h) = \sum_{\mathbf{x}} p(\mathbf{x}) \mathbf{1}_{h(\mathbf{x})\neq f(\mathbf{x})} = \mathbb{E}_p[\mathbf{1}_{h(\mathbf{x})\neq f(\mathbf{x})}]$$

Maintenant que les notions de risques ont été formalisées, nous allons pouvoir définir l'apprentissage PAC qui définit l'apprenabilité d'un ensemble d'hypothèses.

Apprentissage *Probably Approximately Correct* (PAC)

Dans son article fondateur [Val84], Valiant propose une théorie de l'apprenabilité et introduit la notion d'apprentissage PAC, en anglais *Probably Approximately Correct*, c'est-à-dire apprentissage probablement approximativement correct. Étudions de plus près ce nom :

Approximativement correct : nous ne cherchons pas à apprendre la fonction cible (ce qui n'est pas toujours possible) ; néanmoins, on veut que l'hypothèse apprise h soit approximativement correcte, c'est-à-dire que le risque réel de h soit faible.

Probablement : la nature aléatoire du tirage des exemples fait qu'on ne pourra jamais être certain d'apprendre une hypothèse dont le risque réel soit faible ; néanmoins, on peut quantifier et limiter la probabilité d'apprendre une hypothèse qui ait un risque réel trop grand (une « mauvaise » hypothèse).

L'apprentissage PAC s'intéresse au nombre d'exemples suffisant pour apprendre probablement une hypothèse approximativement correcte.

Définition 3.1.3 (*Sample complexity* d'un ensemble d'hypothèses)**.** *La sample complexity d'un ensemble d'hypothèses $\mathcal{H}$ est le nombre minimal $m(\epsilon, \delta)$ d'exemples tel qu'il existe un algorithme d'apprentissage tel que, quelle que soit la fonction cible $f \in \mathcal{H}$, quelle que soit la distribution de probabilité p selon laquelle sont tirés les exemples de $\mathcal{E}$, quels que soient $0 < \epsilon < 1/2$, $0 < \delta < 1/2$, cet algorithme apprenne une hypothèse $h \in \mathcal{H}$ à partir d'un ensemble d'exemples non bruités $\mathcal{E}$ telle que :*

$$|\mathcal{E}| \geq m(\epsilon, \delta) \Rightarrow \Pr(e(h) \leq \epsilon) \geq 1 - \delta$$

La *sample complexity* permet de définir l'apprenabilité d'un ensemble d'hypothèses :

Définition 3.1.4 (Apprenabilité d'un ensemble d'hypothèses)**.** *Soit $m(\epsilon, \delta)$ la sample complexity d'un ensemble d'hypothèses $\mathcal{H}$. $\mathcal{H}$ est dit apprenable si $m(\epsilon, \delta)$ est polynomial en $\frac{1}{\epsilon}$, en $\frac{1}{\delta}$ et en les autres paramètres de $\mathcal{H}$.*

Nous avons fait ici deux hypothèses : la fonction cible appartient à l'ensemble d'hypothèses et les exemples ne sont pas bruités ; c'est ce qu'on appelle le cadre *gnostique* (car on *sait* que la fonction fait partie des hypothèses et que les exemples ne sont pas bruités), qui est celui initialement considéré par Valiant. En retirant ces deux hypothèses, on se place dans un cadre plus général, dit *agnostique*, introduit par [KSS94].

Heureusement, l'apprenabilité dans un cadre gnostique est équivalente à l'apprenabilité dans un cadre agnostique, comme cela a été montré par [KSS94].

Calculer la fonction $m(\epsilon, \delta)$ n'est pas toujours évident ; heureusement, il existe une relation directe entre cette fonction et la notion de dimension de Vapnik-Chervonenkis, que nous allons maintenant présenter.

Dimension de Vapnik-Chervonenkis

La dimension de Vapnik-Chervonenkis [VC15], ou plus simplement dimension VC, est un concept fondamental de l'apprentissage artificiel théorique pour quantifier l'expressivité d'un ensemble d'hypothèses.

Un ensemble $\mathcal{E}$ d'observations est dit pulvérisé par un ensemble d'hypothèse $\mathcal{H}$ si, pour toute partition $\{\mathcal{E}^+, \mathcal{E}^-\}$ de $\mathcal{E}$, il existe une hypothèse $h \in \mathcal{H}$ telle que :

- pour toute observation $\mathbf{x} \in \mathcal{E}^+$, $h(\mathbf{x}) = +$
- pour toute observation $\mathbf{x} \in \mathcal{E}^-$, $h(\mathbf{x}) = -$

Définition 3.1.5 (Dimension VC)**.** *La dimension VC d'un ensemble d'hypothèses $\mathcal{H}$, notée $\mathsf{VC}(\mathcal{H})$, est la taille du plus grand ensemble d'exemples qui puisse être pulvérisé par $\mathcal{H}$. Si un tel ensemble d'exemples n'existe pas, alors $\mathsf{VC}(\mathcal{H}) = +\infty$.*

Une propriété immédiate de cette définition est que si un ensemble d'hypothèses $\mathcal{H}$ est inclus dans un autre ensemble d'hypothèses $\mathcal{H}'$, i.e. $\mathcal{H} \subseteq \mathcal{H}'$, alors la dimension VC de $\mathcal{H}$ ne pourra pas être plus grande que la dimension VC de $\mathcal{H}'$, i.e. $\mathsf{VC}(\mathcal{H}) \leq \mathsf{VC}(\mathcal{H}')$.

Le lien entre apprenabilité et dimension VC finie est expliqué par le théorème suivant :

Proposition 3.1.1 ([BEHW89]). *Soit $\mathcal{H}$ un ensemble d'hypothèses. Alors $\mathcal{H}$ est apprenable si et seulement si sa dimension VC est finie.*

Mais il y a encore mieux ! La *sample complexity* d'un problème de classification est directement liée à sa dimension VC :

Proposition 3.1.2 ([Han16]). *Soit $\mathcal{H}$ un ensemble d'hypothèses dont la dimension VC est finie. Alors il existe une fonction $m(\epsilon, \delta)$ telle que :*

$$m(\epsilon, \delta) = O(\frac{\mathsf{VC}(\mathcal{H}) + \ln \frac{1}{\delta}}{\epsilon})$$

pour laquelle :

$$|\mathcal{E}| \geq m(\epsilon, \delta) \Rightarrow \Pr(e(h) \leq \epsilon) \geq 1 - \delta$$

Maintenant que nous avons défini l'apprenabilité d'un ensemble d'hypothèses et vu quelques unes de ses propriétés, il convient de nous demander comment choisir, lors d'un apprentissage, l'ensemble d'hypothèses le plus adapté. Ce choix est tout sauf évident et est souvent le résultat d'un compromis biais-variance.

3.1.3 Compromis biais-variance

Le biais et la variance sont deux sources d'erreurs lors de l'apprentissage d'une hypothèse au sein d'un ensemble d'hypothèses. Plus cet ensemble d'hypothèses est petit, plus le biais sera important, et plus cet ensemble sera grand, plus la variance sera importante. Pour cette raison, choisir un ensemble d'hypothèses, c'est faire un compromis biais-variance (aussi appelé biais-complexité).

Nous allons détailler ces deux notions.

Surapprentissage

Si l'ensemble d'hypothèses est très grand, il est possible que l'algorithme d'apprentissage puisse être victime de *surapprentissage*. Le surapprentissage (*overfitting* en anglais) désigne l'apprentissage d'une hypothèse qui a un risque empirique très faible sur les exemples d'apprentissage mais qui a de piètres qualités de généralisation sur des observations qu'il n'a jamais vues, ce qui se traduit par des performances basses sur de nouvelles données. Ce phénomène surgit lorsque l'ensemble d'hypothèses est important car il y a des hypothèses qui peuvent trop « coller » aux données qu'on leur donne ; autrement dit, l'algorithme d'apprentissage peut « apprendre par cœur » les exemples d'apprentissage. Le fait que dans ce cas le modèle appris est largement dépendant des exemples d'apprentissage peut être interprétée comme une variance importante.

Biais

Lorsque l'ensemble d'hypothèses est petit, le risque de surapprentissage est bien plus limité. Par contre, dans le cadre agnostique où la fonction cible ne fait pas partie de l'ensemble d'hypothèses, même la meilleure hypothèse pourra être assez éloignée de la fonction cible. On appelle biais la distance entre la fonction cible et l'ensemble d'hypothèses.

Supposons, par exemple, qu'on cherche à apprendre le concept qui regroupe l'ensemble des points du disque unitaire et dont la fonction cible est la suivante :

$$f(\mathbf{x}) = \begin{cases} + \text{ si } |\mathbf{x}| < 1 \\ - \text{ sinon} \end{cases}$$

Si l'ensemble des hypothèses est celui des séparateurs linéaires, il n'y a aucune hypothèse qui ait un risque réel faible. Il y a donc un biais non nul.

Toutes les notions générales de l'apprentissage automatique dont nous avons besoin sont à présent introduites. Nous allons pouvoir nous intéresser au cas particulier de l'apprentissage de préférences.

3.2 Problèmes d'apprentissage de préférences

Parmi les problèmes du domaine de l'apprentissage de préférences, les problèmes d'apprendre à ordonner [1] (en anglais : *learning to rank*) sont ceux qui reçoivent le plus d'attention. Dans chacun de ces problèmes, l'objectif est d'apprendre un ordre à partir d'observations étiquetées ; il s'agit donc d'apprentissage supervisé.

On distingue classiquement trois problèmes d'apprendre à ordonner. Bien que nous les présentions toutes, seul un de ces problème nous intéressera par la suite. Nous présenterons enfin un quatrième problème largement étudié : l'apprentissage de préférences par élicitation. Le lecteur intéressé pourra approfondir le sujet avec le livre [FH11].

3.2.1 Ordonner des objets à partir de comparaisons

Dans le problème d'apprendre à ordonner des objets à partir de comparaisons (aussi plus simplement appelé « classement d'objets », *object ranking* [CSS98]), on considère généralement un ensemble d'objets $\underline{\mathcal{X}}$ combinatoire, où chaque objet est représenté par un vecteur. L'algorithme d'apprentissage dispose d'un ensemble de comparaisons $\{\mathbf{x}_i \succ \mathbf{y}_i\}_i$, $\mathbf{x}_i, \mathbf{y}_i \in \underline{\mathcal{X}}$. L'objectif est d'apprendre une fonction de comparaison f qui puisse ordonner tout ensemble d'objets. Cette fonction f peut être représentée sous la forme d'un modèle dans un langage de représentation de préférences.

Ce problème est notamment celui qui est rencontré par les moteurs de recherche, qui doivent trier un nombre très important de résultats. Dans ce cas, les comparaisons peuvent être obtenues à partir des clics des utilisateurs (le lien cliqué est interprété comme préféré à tous les liens qui n'ont pas été cliqués).

C'est le problème le plus étudié en apprentissage de modèles de représentation de préférences en espace combinatoire.

> **Exemple 3.2.1.** Un restaurant met en avant deux menus du jour. À chaque fois que le client choisit l'un de ces menus, on acquiert une nouvelle information : le menu choisi est préféré au menu qui n'a pas été choisi. Si ce processus est répété plusieurs fois, on peut obtenir un ensemble d'exemples

1. La traduction française varie selon les auteurs ; on peut ainsi parfois lire « apprentissage d'ordonnancement » ou « apprendre à comparer ».

qui ressemble à :

$$\{chocolat/viande/vin\ rouge \succ chocolat/poisson/vin\ blanc,\\ chocolat/viande/vin\ blanc \succ fruit/poisson/vin\ blanc,\\ fruit/poisson/vin\ rouge \succ fruit/viande/vin\ rouge\}$$

L'objectif est alors d'apprendre une fonction qui puisse ordonner un quelconque ensemble de menus, par exemple pouvoir répondre à la question suivante : *quel est l'ordre de préférences du client pour les plats suivants :*

- *chocolat/viande/vin rouge*
- *chocolat/viande/vin blanc*
- *fruit/poisson/vin rouge*

3.2.2 Ordonner des objets à partir de notes

Le problème d'apprendre à ordonner des d'objets à partir de notes (aussi appelé « classement d'instances », *instance ranking*) consiste à apprendre à ordonner à partir d'un ensemble d'objets notés par un utilisateur. Ces notes sont cardinales ; il peut s'agir par exemple de la note d'un produit (noté de 1 à 5) ou de l'acceptation d'un papier (rejet, rejet faible, acceptation faible, acceptation).

Tout comme le cas précédent, l'objectif est d'apprendre une fonction de comparaison f qui puisse ordonner tout ensemble d'objets. Le plus souvent, cette fonction associe à chaque objet une note qu'elle utilise pour ordonner les objets.

Nous n'aborderons pas ce problème dans la suite.

Exemple 3.2.2. Un restaurant reçoit sur un site d'avis participatifs plusieurs avis de clients qui mettent une note (entre 1 et 5) sur leur repas. Pour un client donné, un ensemble d'exemples peut ressembler à :

$$\{chocolat/poisson/vin\ blanc : 2, chocolat/viande/vin\ rouge : 4\\ fruit/poisson/vin\ blanc : 5, fruit/poisson/vin\ rouge : 4\}$$

L'objectif est le même que précédemment : pouvoir ordonner un ensemble quelconque de menus.

3.2.3 Ordonner des étiquettes

Le problème d'apprendre à ordonner des étiquettes a pour objectif d'apprendre à ordonner, pour un contexte $\mathbf{x}$, un petit ensemble d'étiquettes $\{y_i\}_i$. Il peut s'agir, par exemple, de prédire les préférences d'un utilisateur sur des modèles d'aspirateurs (parmi cinq possibles, ce sont les $\{y_i\}_i$) à partir de son âge, de la surface de son foyer et de son niveau de revenus (le contexte $\mathbf{x}$).

Il s'agit encore une fois d'un problème d'apprentissage supervisé : l'algorithme d'apprentissage dispose d'un ensemble d'exemples (observation, classe), où l'observation est un contexte $\mathbf{x}$ et où la classe est un ordre total sur les $\{y_i\}_i$.

Nous n'aborderons pas ce problème dans la suite.

Exemple 3.2.3. Un restaurant a limité sa carte à trois menus possibles : *chocolat/poisson/vin blanc* (noté *menu A*), *fruit/poisson/vin blanc* (noté *menu B*) et *chocolat/viande/vin rouge* (noté *menu C*). Afin de connaître précisément les préférences d'un client, un restaurant lui demande régulièrement d'ordonner ces trois menus en fonction du contexte. Ce contexte est composé de deux attributs :

Occasion : soit formel (notée *formel*), soit détendu (noté *détendu*).

Heure : soit le déjeuner (noté *déjeuner*), soit le dîner (noté *dîner*).

Un ensemble d'apprentissage sera de la forme :

$$\{\textit{formel/déjeuner} : \textit{menu } A \succ \textit{menu } C \succ \textit{menu } B,\\ \textit{détendu/dîner} : \textit{menu } C \succ \textit{menu } B \succ \textit{menu } A\}$$

L'objectif est ici de pouvoir prédire la préférence sur les trois menus en fonction d'un contexte, par exemple pouvoir répondre à la question suivante : « *quel est l'ordre de préférences de l'utilisateur sur les trois menus pour un dîner formel ?* ».

3.2.4 Apprentissage de préférences par élicitation

Nous avons précédemment introduit la notion d'élicitation, autrement appelé apprentissage actif, qui se base sur l'interrogation de son environnement. Lorsque l'élicitation est appliquée à l'apprentissage de préférences, l'objectif est d'apprendre à partir de requêtes un modèle de préférence $\mathcal{M}$ qui soit le plus proche possible du modèle caché $\mathcal{M}'$. [Ang88] présente six types de questions, dont deux sont utilisées pour l'apprentissage de préférences :

Requête d'appartenance : demande à l'utilisateur si un modèle $\mathcal{M}$ est identique au modèle caché $\mathcal{M}'$. Si ce n'est pas le cas, l'utilisateur indique un contre-exemple $(\mathbf{o}, \mathbf{o}')$ qui montre en quoi $\mathcal{M}$ et $\mathcal{M}'$ diffèrent. Un contre-exemple est dit positif si $\mathbf{o} \succ_{\mathcal{M}'} \mathbf{o}'$ et $\mathbf{o} \not\succ_{\mathcal{M}} \mathbf{o}'$ et dit négatif si $\mathbf{o} \not\succ_{\mathcal{M}'} \mathbf{o}'$ et $\mathbf{o} \succ_{\mathcal{M}} \mathbf{o}'$.

Requête d'équivalence : demande à l'utilisateur si $\mathbf{o} \succ_{\mathcal{M}'} \mathbf{o}'$.

3.3 Apprentissage de modèles graphiques

Il existe plusieurs approches pour résoudre les trois problèmes que nous venons de voir :

- l'utilisation de techniques d'estimation locale, à la manière de l'algorithme de k plus proches voisins ;
- l'apprentissage d'une fonction d'utilité *in extenso* ;
- l'apprentissage d'un modèle graphique de représentation des préférences.

C'est sur ce dernier point que s'est particulièrement portée cette thèse ; néanmoins, nous comparerons expérimentalement nos résultats avec des méthodes basées sur les plus proches voisins dans la partie III.

Cette section a pour objectif de donner un aperçu des connaissances actuelles en matière d'apprentissage de modèles de représentation de préférences. Plus exactement, nous nous limiterons à l'apprentissage de modèles graphiques, car ce sont ces modèles que nous utiliserons dans la suite de cette thèse.

3.3.1 Apprentissage de CP-net

L'apprentissage des CP-nets a été très étudié : nous disposons d'algorithmes d'apprentissage par élicitation et à partir de comparaisons non bruitées et bruitées.

Apprentissage de CP-net acyclique par élicitation

Dans leur article [KZ10], les auteurs ont montré qu'un CP-net acyclique ne peut pas toujours être appris avec un nombre polynomial de requêtes d'équivalence, mais qu'il peut toujours l'être avec un nombre polynomial de requêtes d'équivalence et de requêtes d'appartenance (nous avons précédemment défini ces requêtes à la section 3.2.4).

[KZ10] propose d'ailleurs un algorithme d'apprentissage par élicitation utilisant les requêtes d'équivalence et d'appartenance. Rappelons que lorsqu'une requête d'appartenance est soumise à l'utilisateur, celui-ci doit fournir un contre-exemple sous la forme d'un couple $(\mathbf{o}, \mathbf{o}')$ telle que l'ordre entre $\mathbf{o}$ et $\mathbf{o}'$ est différent entre le modèle en cours d'apprentissage et le modèle caché. Les auteurs restreignent la requête d'appartenance en y ajoutant la contrainte suivante : les contre-exemples renvoyés doivent être des *swaps* ; en effet, si un contre-exemple existe, alors il existe aussi un contre-exemple qui soit un *swap*.

L'algorithme commence par interroger l'utilisateur avec une requête d'appartenance. S'il n'y a aucun contre-exemple, le CP-net est correct et l'apprentissage est fini. Dans le cas contraire, l'utilisateur renvoie un *swap* contenant un contre-exemple. L'algorithme va alors mettre à jour la table et les parents de l'attribut X *swappé* grâce à des requêtes d'équivalence. L'algorithme continue jusqu'à ce qu'il n'y ait plus aucun contre-exemple.

Un autre algorithme d'élicitation a été introduit par [GAG13] ; il n'est pas garanti de retrouver le CP-net original mais le CP-net appris sera cohérent avec toutes les requêtes effectuées. La complexité temporelle de cet algorithme est polynomiale dès que le nombre de parents est borné.

Apprentissage de CP-net acyclique à partir de comparaisons non bruitées

La difficulté de l'apprentissage de CP-net à partir d'un ensemble de comparaisons $\mathcal{E}$ a été mis en avant par [DMA09]. En effet, les auteurs montrent que vérifier qu'il existe un CP-net cohérent avec $\mathcal{E}$ est un problème NP-difficile (des résultats supplémentaires de complexité d'apprentissage sont détaillés dans [CKL$^+$10] si le lecteur est intéressé). [DMA09] propose un algorithme d'apprentissage de CP-net acyclique, dont chaque nœud a au plus k parents, à partir de comparaisons. Il fonctionne de la manière suivante. L'algorithme construit

un CP-net en y ajoutant les nœuds un par un, en commençant par un réseau vide. Pour l'expliquer, intéressons-nous à une de ses itérations, et supposons que l'algorithme a déjà construit un CP-net partiel $\mathcal{N}$ contenant les attributs $\mathbf{U} \subset \mathcal{X}$ Pour choisir le prochain nœud à ajouter, l'algorithme va tester pour chaque attribut $X \notin \mathbf{U}$ et pour chaque sous-ensemble $\mathbf{P} \subseteq \mathbf{U}$ d'au plus k attributs si X pourrait être ajouté à $\mathcal{N}$ avec pour parents $\mathbf{P}$, c'est-à-dire s'il existe une table de préférence $\mathsf{CPT}(X)$ pour X avec pour parents $\mathbf{P}$ et qui soit cohérente avec tous les exemples de $\mathcal{E}$. Si l'algorithme trouve un tel X et $\mathbf{P}$, alors X est ajouté à $\mathcal{N}$ avec pour parents $\mathbf{P}$ et comme table de préférence $\mathsf{CPT}(X)$. Si un tel X n'existe pas, alors l'algorithme détecte un échec de l'apprentissage. Cet algorithme est assuré d'apprendre un CP-net acyclique.

Cet algorithme est sain (le CP-net qu'il trouve est cohérent avec $\mathcal{E}$) mais n'est par contre pas complet (s'il existe un CP-net cohérent avec $\mathcal{E}$ dont chaque nœud ait au plus k parents, l'algorithme n'est pas assuré de le trouver).

Apprentissage de CP-net acyclique à partir de comparaisons bruitées

Dans leur article [LZM+17], les auteurs proposent un algorithme d'apprentissage de CP-net à partir de comparaisons qui soit robuste vis-à-vis des comparaisons bruitées, en utilisant une mesure provenant de la théorie de l'information, l'entropie de Shannon.

3.3.2 Apprentissage de LP-tree

Il existe des algorithmes d'apprentissage de LP-trees à partir de comparaisons bruitées et non bruitées. L'apprentissage par élicitation a par contre été peu étudié.

Apprentissage de LP-tree à partir de comparaisons non bruitées

[BCL+10] présentent un algorithme glouton d'apprentissage de LP-tree à partir d'un ensemble de comparaisons non bruitées, dont voici le fonctionnement.

Soit $\mathcal{E}$ un ensemble de comparaisons cohérent avec un LP-tree caché. Cet algorithme construit un LP-tree incrémentalement. Notons $\mathcal{L}$ le LP-tree partiel en cours d'apprentissage. Pour apprendre l'étiquette et la table de préférence d'un nœud N de $\mathcal{L}$, l'algorithme construit l'ensemble des exemples de $\mathcal{E}$ cohérents avec les instanciations des nœuds situés entre la racine et N, c'est-à-dire :

$$\mathcal{E}(N) = \{(\mathbf{o}, \mathbf{o}') \in \mathcal{E} \mid \mathbf{o}[\mathbf{Inst}(N)] = \mathbf{o}'[\mathbf{Inst}(N)] = \mathbf{inst}(N)\}$$

L'algorithme vérifie pour chaque attribut $X \in \mathcal{X} \setminus \mathbf{Anc}(N)$ s'il existe une table de préférences sur les valeurs $\underline{X}$ qui soit compatible avec tous les exemples $\mathcal{E}(N)$. Auquel cas, N est étiqueté par l'un de ces attributs X et par cette table de préférences.

Cet algorithme doit ensuite choisir entre un unique arc sortant de N ou $|\underline{X}|$ arcs ; pour cela, il vérifie s'il existe au moins un exemple de $\mathcal{E}(N)$ compatible pour chaque valeur de $\underline{X}$. Si c'est le cas, N aura $|\underline{X}|$ arcs sortants. Sinon, afin d'éviter de créer un sous-arbre qui n'aurait aucun exemple compatible, l'algorithme ajoute un unique arc non étiqueté sortant de N. Cela signifie également qu'il y aura au plus $|\mathcal{E}|$ feuilles à l'arbre appris $\mathcal{L}$.

Les articles [LT15] et [YWLd10] proposent des variations de cet algorithme ; le premier pour apprendre des LP-trees partiels, le second pour apprendre des LP-trees linéaires.

Apprentissage de k-LP-tree à partir de comparaisons bruitées

Dans leur papier introduisant les k-LP-trees, [BH12] proposent également un algorithme glouton pour les apprendre, basé sur l'algorithme précédent de [BCL$^+$10]. La principale différence est que cet algorithme peut travailler avec des comparaisons bruitées.

Pour cela, les auteurs utilisent les mesures de performances introduites par [CRBH10]. Ces mesures ont pour objectif d'évaluer une relation de préférence partielle $\succeq$ vis-à-vis d'un ensemble de comparaisons $\mathcal{E}$. La première mesure est le score CR (pour *correctness*), un nombre entre -1 et 1. Si le score est proche de -1, la relation $\succ$ est en désaccord avec les exemples $\mathcal{E}$; inversement, si le score est proche de 1, $\succ$ est en accord avec $\mathcal{E}$.

La seconde mesure est le score CP (pour *completeness*), un nombre entre 0 et 1, et indique la proportion des exemples de $\mathcal{E}$ qui ne sont pas incomparables selon $\succeq$. Si $\succeq$ est un ordre total, alors $CP(\succeq, \mathcal{E}) = 1$.

Quand il doit choisir l'étiquette d'un nœud N, l'algorithme trouve l'ensemble d'au plus k attributs qui maximise le score CR ; en cas d'égalité, c'est le score CP qui départage. De plus, afin d'éviter d'itérer sur tous les ensembles d'au plus k attributs, les auteurs utilisent une heuristique de sélection. À un nœud N, l'algorithme construit temporairement un sous-arbre 1-LP-tree de profondeur k. Une fois ce sous-arbre construit, on note V^* l'ensemble des attributs apparaissant dans ce sous-arbre (V^* contient donc au moins k et au plus $2^k - 1$ attributs différents). Ce sous-arbre est ensuite jeté. Pour trouver l'étiquette de N, l'algorithme va chercher une étiquette pour N parmi les ensembles d'au plus k attributs qui sont inclus dans V^*.

3.3.3 Apprentissage de GAI-net

Apprentissage de GAI-net par élicitation

Il n'y a pour le moment aucune méthode permettant d'apprendre par élicitation la structure d'un GAI-net. Par contre, une fois la structure connue (elle peut être renseignée par un expert, par exemple), [BB05] indique comment éliciter les fonctions d'utilité locales. Autrement dit, les ensembles $\mathbf{Z}_1, \ldots, \mathbf{Z}_k$ sont supposés connus et les fonctions d'utilité $U_i : \mathbf{Z}_i \mapsto \mathbb{R}$ sont apprises à partir de requêtes.

Apprentissage de GAI-net à partir de comparaisons non bruitées

[BFMZ12] propose une méthode d'apprentissage de décompositions GAI (qui peuvent être compilées sous la forme d'un GAI-net) à partir de comparaisons non bruitées. Cette méthode consiste en une traduction du problème en un problème de programmation linéaire en transformation l'ensemble des exemples en un système d'inégalités linéaires.

Pour cela, les auteurs limitent leur recherche aux décompositions GAI faisant intervenir au plus k attributs dans chaque sous-utilité. Notons $U_{\mathbf{Z}}$ la fonction d'utilité définie sur les attributs $\mathbf{Z}$ (qui respecte $|\mathbf{Z}| \leq k$). Soit $\sigma > 0$ une valeur

arbitraire (qui n'influence pas les décompositions GAI apprises). L'algorithme de [BFMZ12] est le suivant.

Chaque exemple est transformé en contrainte linéaire. Si l'exemple est de la forme $\mathbf{o} \succeq \mathbf{o}'$, cette contrainte est :

$$\sum_{\mathbf{Z} \subseteq \mathcal{X}, |\mathbf{Z}| \leq k} U_{\mathbf{Z}}(\mathbf{o}[\mathbf{Z}]) \geq \sum_{\mathbf{Z} \subseteq \mathcal{X}, |\mathbf{Z}| \leq k} U_{\mathbf{Z}}(\mathbf{o}'[\mathbf{Z}])$$

S'il est de la forme $\mathbf{o} \succ \mathbf{o}'$, l'inégalité est :

$$\sum_{\mathbf{Z} \subseteq \mathcal{X}, |\mathbf{Z}| \leq k} U_{\mathbf{Z}}(\mathbf{o}[\mathbf{Z}]) \geq \sigma + \sum_{\mathbf{Z} \subseteq \mathcal{X}, |\mathbf{Z}| \leq k} U_{\mathbf{Z}}(\mathbf{o}'[\mathbf{Z}])$$

Ainsi, l'algorithme transforme un ensemble d'exemples en un ensemble de contraintes linéaires dont les variables sont les valeurs $U_{\mathbf{Z}}(\mathbf{z})$ pour tout $\mathbf{Z} \subseteq \mathcal{X}, |\mathbf{Z}| \leq k$ et $\mathbf{z} \in \underline{\mathbf{Z}}$. Ce système linéaire peut être résolu avec un algorithme d'optimisation linéaire.

Cet algorithme d'apprentissage de décompositions GAI ne peut par contre pas être directement utilisé avec des comparaisons bruitées car dans ce cas le système linéaire peut ne pas avoir de solution.

3.3.4 Apprentissage de réseau bayésien

Contrairement aux autres langages évoqués précédemment qui sont dédiés à la représentation de préférences, les réseaux bayésiens sont un outil beaucoup plus répandu et sur lesquels la littérature est bien plus importante (je recommande au lecteur intéressé les livres [NWL+11], en français, et [Dar09], en anglais). La tâche d'apprentissage des réseaux bayésiens est classiquement de l'apprentissage à partir d'exemples positifs (bien que de l'apprentissage par élicitation ait été occasionnellement étudié [TK00, TK01]), où on suppose que les observations sont tirées à partir de la distribution de probabilité qu'on cherche à apprendre.

L'apprentissage d'un réseau bayésien à partir d'exemples positifs se fait en deux étapes : d'abord trouver la structure du réseau, c'est-à-dire le graphe acyclique orienté qui sous-tend le réseau bayésien, puis trouver ses paramètres, c'est-à-dire les tables de probabilité conditionnelles. Lors des deux phases, on cherche à maximiser la vraisemblance du jeu de données d'apprentissage, avec généralement une pénalité envers le nombre de paramètres du réseau.

Trouver le réseau bayésien qui maximise la vraisemblance d'un jeu de données est un problème NP-difficile [Chi95], même quand on se limite à apprendre un réseau bayésien dont chaque variable a au plus deux parents. Des méthodes heuristiques ont donc dû être trouvées. Il y a principalement deux familles d'approches dans l'apprentissage de la structure du réseau : celles basées sur l'optimisation d'un score et celles qui recherchent des indépendances conditionnelles. Cette distinction entre ces deux familles d'approches est à nuancer ; comme l'a montré [Cow01], sous certaines conditions ces deux familles d'approches peuvent apprendre les mêmes réseaux bayésiens.

Méthodes basées sur une optimisation de score

Ces premières méthodes recherchent la structure d'un réseau qui maximise un score qui mesure à quel point le réseau explique bien les données [CH91].

Ce score prend également en compte la complexité du réseau afin de limiter le surapprentissage.

On peut regrouper les scores qui s'appuient sur la vraisemblance avec un terme de pénalisation qui dépend de la taille du modèle. Ces scores tirent leur origine de la théorie de l'information.

- BIC [Sch78] ;
- MDL [Bou93] ;
- AIC [Aka98] ;
- fNML [SRKM08] ;
- qNML [SLJR18].

Il y a également les scores basés sur une approche bayésienne, qui cherche à maximiser la probabilité postérieure du réseau bayésien appris :

- BDeu [Bun91] ;
- K2 [CH92] ;
- BDe [HGC95] ;
- UPSM [KC02]
- BDs [Scu16] ;

Algorithmiquement, on procède généralement par recherche locale dans l'espace des structures possibles ; quelques méthodes utilisées : *hill-climbing* glouton, recherche tabou, algorithme génétique [CWR07, CVN18], recuit simulé [JN06]. Le lecteur intéressé pourra trouver une présentation de ces méthodes dans le livre [RN10].

Méthodes basées sur la recherche d'indépendances conditionnelles

La seconde famille d'approches se base sur des calculs d'indépendances conditionnelles. En effet, un réseau bayésien permet de trouver facilement les 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 Verma et Pearl [VP90]. L'algorithme le plus utilisé est l'algorithme PC, plus précisément PC-stable [CM14]. On peut également citer les algorithmes *Grow Shrink* [Mar03] et Inter-IAMB [YM05].

Méthodes hybrides

Enfin, des méthodes hybrides existent, comme le célèbre MMHC (*Max-Min Hill Climbing* [TBA06]), qui apprend la structure non-orientée du réseau avec une approche basée sur les contraintes (appelée MMPC, *Min-Max Parent Child*) puis oriente les arcs du graphe acyclique orienté avec une méthode basée sur un score ; un autre exemple est RSMAX2 [SHBM14].

3.4 Apprenabilité des langages de représentation de préférences

Nous finirons ce chapitre en étudiant comment adapter les concepts d'apprenabilité (définis pour la classification binaire) au problème d'apprendre à ordonner à partir de comparaisons.

3.4.1 L'apprentissage à partir de comparaisons, une classification binaire cachée

Le problème d'apprentissage à partir de comparaisons a pour données d'apprentissage un ensemble de comparaisons $\{\mathbf{x}_i \succ \mathbf{y}_i\}_i$ et cherche à apprendre un préordre cible $\succeq$. Même si, dans ce problème, il n'y a que des observations et pas de classe associée, on peut facilement interpréter ce problème comme un problème de classification binaire, où la fonction cible est :

$$f(\mathbf{x}, \mathbf{y}) = \begin{cases} + \text{ si } \mathbf{x} \succ \mathbf{y} \\ - \text{ sinon} \end{cases}$$

Auquel cas chaque exemple $(\mathbf{x} \succ \mathbf{y})$ peut être interprété en classification binaire en $((\mathbf{x}, \mathbf{y}), +)$, c'est-à-dire une observation $(\mathbf{x}, \mathbf{y})$ associée à la classe $+$.

Un préordre est donc, dans la terminologie de la classification, une hypothèse. Étant donné qu'un langage de représentation de préférences peut exprimer un ensemble de préordres, il définit un ensemble d'hypothèses. Nous pouvons donc nous intéresser à l'apprenabilité d'un langage et à sa dimension VC.

3.4.2 Dimension VC de quelques langages de représentation de préférence

Une proposition de [BCL$^+$10] va nous permettre de fixer une borne supérieure sur la dimension VC des langages de représentation de préférences :

Proposition 3.4.1 ([BCL$^+$10])**.** *Soit $\mathcal{X}$ un ensemble de n attributs binaires. La dimension VC des relations transitives et non-réflexives sur $\mathcal{X}$ est strictement inférieure à 2^n.*

Cette proposition s'applique à tous les langages ordinaux, que ce soit la famille des LP-trees ou des CP-nets, car ils représentent tous un préordre dont la partie asymétrique est une relation transitive et non-réflexive. Cette proposition démontre qu'une relation transitive et non-réflexive ne peut pulvériser aucun ensemble de 2^n couples d'objets car il existe dans chacun de ces ensembles une répartition des classes qui amènerait à un cycle.

Un tableau récapitulatif de la dimension VC de différents langages de représentation de préférences en présenté dans la table 3.1 avec certains résultats dont les preuves suivent.

Proposition 3.4.2. *Soient $\mathcal{X}$ un ensemble de n attributs binaires, $k \geq 1$. La dimension VC des k-LP-tree sur $\mathcal{X}$ est égale à $2^n - 1$.*

Démonstration. Tout LP-tree peut être représenté par un k-LP-tree (et ce pour tout $k \geq 1$ fixé), donc la dimension VC des k-LP-trees est supérieure ou égale à

Langage	Dimension VC
LP-tree linéaire	n [SM06]
LP-tree	$2^n - 1$ [BCL$^+$10]
k-LP-tree	$2^n - 1$ (prop. 3.4.2)
CP-net séparable	n [AMZ16]
CP-net acyclique	$2^n - 1$ [AMZ16]
CP-net satisfiable	$2^n - 1$ [AMZ16]
TCP-net	$2^n - 1$ (prop. 3.4.3)
Réseau bayésien	$2^n - 1$ (prop. 3.4.4)
GAI-net[a]	$O(2^k n^{k+1})$[Big15]

[a] Où k est le nombre maximal de variables par nœud du GAI-net

TABLE 3.1 – Dimension VC de langages de représentation de préférences

celle des LP-trees, qui vaut $2^n - 1$ [BCL$^+$10]. Or, la dimension VC des k-LP-trees est strictement inférieure à 2^n d'après la proposition 3.4.1.

Ainsi, la dimension VC des k-LP-trees est égale à $2^n - 1$. □

Proposition 3.4.3. *Soit $\mathcal{X}$ un ensemble de n attributs binaires. La dimension VC des TCP-net sur $\mathcal{X}$ est égale à $2^n - 1$.*

Démonstration. Le même raisonnement que précédemment s'applique aux TCP-net, qui sont une extension des CP-net acycliques dont la dimension VC est déjà maximale, i.e. $2^n - 1$ [AMZ16]. □

Proposition 3.4.4. *Soit $\mathcal{X}$ un ensemble de n attributs binaires. La dimension VC des réseaux bayésiens sur $\mathcal{X}$ est de $2^n - 1$.*

Démonstration. Numérotons les objets de $\mathcal{X}$ de 1 à 2^n dans un ordre arbitraire et notons $\mathbf{o}_i$ l'objet numéroté i. Soit $\mathcal{E}$ l'ensemble de $2^n - 1$ paires d'objets défini de la manière suivante : $\mathcal{E} = \{(\mathbf{o}_i, \mathbf{o}_{i+1})\}_{i=1}^{2^n-1}$. Soit $\mathcal{G}$ le graphe dirigé dont les nœuds sont les objets de $\mathcal{X}$ et les arcs relient les couples de $\mathcal{E}$. $\mathcal{G}$ est un arbre et, de ce fait, n'a pas de cycle ; ainsi quelque soit la manière dont on oriente ses arcs, $\mathcal{G}$ sera un graphe acyclique orienté. Cela signifie que quelque soit la classe des couples de $\mathcal{E}$, il existe un ordre total $\succ$ tel que, s'il y a un arc depuis un objet $\mathbf{o}$ vers un objet $\mathbf{o}'$, alors $\mathbf{o} \succ \mathbf{o}'$ (cet ordre total est un tri topologique des nœuds de $\mathcal{G}$).

Un réseau bayésien peut représenter toute distribution de probabilité qui ne s'annule pas et peut donc représenter n'importe quel ordre total sur ces $2^n - 1$ exemples ; en particulier, il peut représenter l'ordre total $\succ$ dont nous avons précédemment parlé. Ainsi, la dimension VC des réseaux bayésiens sur ce problème est au moins de $2^n - 1$. En appliquant la proposition 3.4.1, on peut en conclure que la dimension VC des réseaux bayésiens est égale à $2^n - 1$ □

La proposition précédente montre la dimension VC des réseaux bayésiens sur ce problème de classification particulier. Selon les problèmes, la dimension VC d'un langage peut varier.

En appliquant le théorème 3.1.2, nous pouvons en conclure à l'apprenabilité des LP-trees linéaires, des CP-nets séparables et des GAI-nets dont le nombre maximal de variables par nœud est fixé.

Deuxième partie

Contributions théoriques à l'apprentissage et à l'inférence

Chapitre 4

DRC : inférence en réseau bayésien par estimation directe

Sommaire

4.1 Méthodes d'inférences 58
4.1.1 Requêtes d'inférence 58
4.1.2 Calcul de marginale : approche naïve 58
4.1.3 Inférence exacte 59
4.1.4 Inférence stochastique 59
4.1.5 Recursive conditioning 59
4.2 Direct Recursive Conditioning 62
4.2.1 Motivation 62
4.2.2 Calcul d'indépendances conditionnelles 64
4.2.3 *Direct Recursive Conditioning* 65
4.2.4 Construction d'un arbre de décomposition ternaire . 69
4.2.5 Construction de partitions admissibles 72
4.3 Conclusion 76

Les réseaux bayésiens sont des modèles très étudiés en intelligence artificielle. Alors que le chapitre 3 s'est intéressé à l'apprentissage de tels modèles, ce chapitre se penche sur la manière d'exploiter des réseaux bayésiens élicités ou appris, et plus précisément au calcul de loi de probabilité marginale.

Dans notre contexte d'apprentissage de préférences, nous apprenons un réseau bayésien à partir d'un ensemble d'exemples positifs. Notre contribution consiste en un algorithme d'inférence qui calcule une loi marginale en exploitant la structure d'un réseau bayésien, qui représente des dépendances probabilistes, et un ensemble d'exemples positifs qui nous permet d'estimer directement des probabilités à la volée.

Ce chapitre est largement adapté de notre article [FGM18b].

4.1 Méthodes d'inférences

4.1.1 Requêtes d'inférence

Les requêtes d'inférence ont pour objectif d'inférer les valeurs les plus probables de certaines variables à partir d'une observation (souvent notée $\mathbf{e}$, pour *evidence*). Il peut par exemple s'agir de connaître le diagnostic de plus probable à partir de certains symptômes observés ou de résultats d'analyse.

On distingue classiquement deux requêtes classiques d'inférence dans les réseaux bayésiens :

MAP (Maximum A Posteriori) : pour une observation $\mathbf{e} \in \underline{E}$ avec $E \subset \mathcal{X}$ et un ensemble de variables $\mathbf{X}$ pour lequel $\mathbf{X} \cap E = \varnothing$, renvoyer $\operatorname{argmax}_{\mathbf{x} \in \underline{\mathbf{X}}} p(\mathbf{x} \mid \mathbf{e})$.

MPE (*Most Probable Explanation*, « explication la plus probable ») : pour une observation $\mathbf{e} \in \underline{E}$ avec $E \subset \mathcal{X}$, renvoyer $\operatorname{argmax}_{\mathbf{x} \in \underline{\mathcal{X} \setminus \mathbf{E}}} p(\mathbf{x} \mid \mathbf{e})$.

La requête MPE est un problème NP-complet [Shi94]. La requête MAP est un problème $\mathrm{NP}^{\mathrm{PP}}$-complet dans le cas général et NP-complet si le réseau bayésien est un poly-arbre [PD04].

La requête MPE est un cas particulier de la requête MAP, où on interroge sur toutes les variables qui n'ont pas été observées. Dans la suite, ce sera la requête MAP, la plus générale, qui nous intéressera. La façon classique de répondre à une requête MAP est de faire la loi de probabilité marginale $p(\mathbf{X} \mid \mathbf{e})$, et c'est sur ce calcul de marginal que nous allons nous pencher.

4.1.2 Calcul de marginale : approche naïve

Soit un réseau bayésien $(\mathcal{G}, \Theta)$. Il y a une manière simple de calculer la probabilité d'une instanciation complète $p(\mathbf{o})$ selon ce réseau bayésien (que nous avions déjà évoquée en présentant les réseaux bayésiens au chapitre 2) :

$$p(\mathbf{o}) = \prod_{X \in \mathcal{X}} \Theta(\mathbf{o}[X] \mid \mathbf{o}[\mathbf{Pa}(X)])$$

Voici un exemple de réseau bayésien avec lequel nous travaillerons pendant tout ce chapitre.

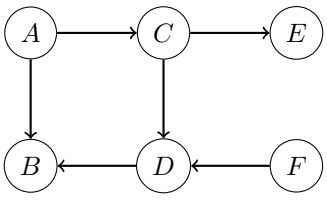


FIGURE 4.1 – Un exemple de réseau bayésien

Exemple 4.1.1. Un exemple de réseau bayésien est représenté sur la figure 4.1. La probabilité de l'instanciation *abcdef* est donnée par :

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

Lorsqu'on souhaite calculer la probabilité d'une instanciation partielle $\mathbf{q}$, une solution naïve est d'additionner les probabilités de toutes les instanciation complètes compatibles avec $\mathbf{q}$:

$$p(\mathbf{q}) = \sum_{\mathbf{o} \sim \mathbf{q}} p(\mathbf{o}) = \sum_{\mathbf{o} \sim \mathbf{q}} \prod_{X \in \mathcal{X}} \Theta(\mathbf{o}[X] \mid \mathbf{o}[\mathbf{Pa}(X)]) \tag{4.1}$$

Dans cette équation, pour une taille de $\mathbf{q}$ fixée, le nombre d'instanciations $\mathbf{o}$ 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 [Coo90, DL93]. Heureusement, des méthodes efficaces et rapides existent.

4.1.3 Inférence exacte

Des méthodes d'inférence exacte, comme *Variable Elimination* [ZP94], *Value Elimination* [BDP03], *Jointree algorithms* [LS88], *Cutset Conditioning* [Pea85], *Recursive Conditioning* [Dar01] 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.

4.1.4 Inférence stochastique

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 [KP83]) et l'autre sur l'échantillonnage stochastique (cf. [Hen86]). 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.

4.1.5 Recursive conditioning

Notre contribution est une nouvelle méthode d'inférence basée sur l'algorithme *Recursive conditioning* de [Dar01]. Avant de présenter notre méthode d'inférence, nous allons donc présenter l'algorithme *Recursive conditioning*.

L'équation 4.1 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 4.1.1 (Facteur RC)**.** *Étant donnés un réseau bayésien* $(\mathcal{G}, \Theta)$ *sur* $\mathcal{X}$*, deux ensembles de variables* $\mathbf{U}, \mathbf{V} \subseteq \mathcal{X}$ *tels que* $\mathbf{Pa}(\mathbf{V}) \subseteq \mathbf{U} \cup \mathbf{V}$ *et* $\mathbf{u} \in \underline{\mathbf{U}}$*, 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* [Dar01], 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 [Dar01]. [1]

Proposition 4.1.1. *Soient 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 :*

$$F_{\mathbf{V}}(\mathbf{u}) = \sum_{\substack{\mathbf{c} \in \underline{\mathbf{C}} \\ \mathbf{c} \sim \mathbf{u}}} F_{\mathbf{V}_1}(\mathbf{u}[\mathbf{V}_1 \cup \mathbf{Pa}(\mathbf{V}_1)]\mathbf{c}) \times F_{\mathbf{V}_2}(\mathbf{u}[\mathbf{V}_2 \cup \mathbf{Pa}(\mathbf{V}_2)]\mathbf{c}) \tag{4.2}$$

Démonstration. Notons que $\mathbf{C} \subseteq \mathbf{V}$. Soient $\mathbf{V}'_1 = \mathbf{V}_1 \smallsetminus \mathbf{C}$ et $\mathbf{V}'_2 = \mathbf{V}_2 \smallsetminus \mathbf{C}$, alors :

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

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

1. [Dar01] 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.

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

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

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. □

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.

Exemple 4.1.2. Reprenons le réseau bayésien de l'exemple 4.1.1 et cherchons à calculer $p(bf)$. En utilisant l'équation 4.1 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. [Dar01] 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 4.1.1 est dessiné sur la figure 4.2 : 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,

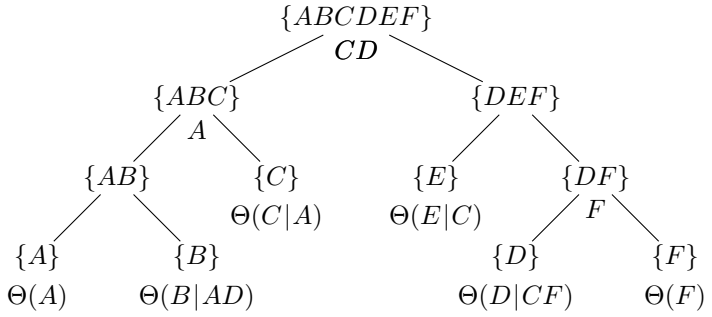


FIGURE 4.2 – Un *dtree* possible pour le réseau bayésien de l'exemple 4.1.1. Pour chaque nœud T, les variables au-dessus du nœud correspondent à $\mathbf{V}_T$. Si T est un nœud interne, les variables en-dessous correspondent au *cutset* de T. Si T est une feuille, il y a en-dessous la table de probabilités associée à la variable de T.

sous chaque nœud binaire, l'ensemble de variables $\mathbf{C}$ de la proposition 4.1.1, appelé *cutset*.

[Dar01] donne plusieurs algorithmes permettant de calculer des *dtree* équilibrés tout en essayant de minimiser la taille des ensembles $\mathbf{C}$ à chaque étape. De plus, il propose l'utilisation d'un cache, pour éviter de refaire plusieurs fois les mêmes calculs lors du calcul d'une probabilité. L'algorithme 2 implémente l'approche de [Dar01].

4.2 Direct Recursive Conditioning

4.2.1 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 ensemble d'exemples positifs sur lequel le réseau bayésien a été appris. Cela signifie que pour calculer par exemple $p(bf)$ sur l'exemple 4.1.2, le plus simple n'est pas de conditionner récursivement comme le fait RC mais plutôt d'utiliser directement l'ensemble d'exemples pour estimer la fréquence de bf dans les produits vendus. On aura une bonne précision si $|\mathcal{H}(bf)|$ 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

Algorithm 2: RC

Input: T : un nœud du *dtree*, $\mathbf{u}$ une instanciation 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
3          facteur_cache(u) ← LookUp (T, u)
4      else
5          p ← 0
6          C ← le cutset de T
7          T1, T2 ← les deux enfants de T
8          for each c ∈ C, c ∼ u do
9              p ← p + RC(T1, uc) × RC(T2, uc)
10         facteur_cache(u) ← p
11 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
3      return Θ_{X|Pa(X)}(u[X] | u[Pa(X)])
4  else return 1
```

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'ensemble d'exemples des probabilités intermédiaires, évitant ainsi de descendre dans toutes les branches du *dtree*.

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'ensemble d'exemples 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'ensemble d'exemples. 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. Nous appelons cet algorithme, qui est une variation de RC qui exploite directement les exemples, *Direct Recursive Conditioning* (DRC).

DRC est donc basé sur deux idées maîtresses :

1. utiliser directement le jeu de données pour estimer les probabilités quand c'est possible ;

2. conditionner pour simplifier les calculs, comme le fait RC, 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.

4.2.2 Calcul d'indépendances conditionnelles

La détermination des indépendances conditionnelles peut se faire grâce à plusieurs outils. Historiquement, la *d-separation* (*directed separation*) [Pea86] fut la première méthode graphique de détermination d'indépendances conditionnelles. Depuis, d'autres algorithmes ont vu le jour, comme la séparation par graphe moral [LDLL90] et l'algorithme Bayes-Ball [Sha98]. Des méthodes récentes incluent la *m-separation* [RRS12], la *i-separation* (*inaugural separation*) [BdSdSOG16], la *u-separation* [BEdSSOG16] ou la *rp-separation* (*relevant path separation*) [BdSdSOG18].

Nous utiliserons par la suite la séparation par graphe moral.

Définition 4.2.1 (Graphe moral)**.** *Le graphe moralisé [LS88] (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 4.1.1 est présenté en figure 4.3.

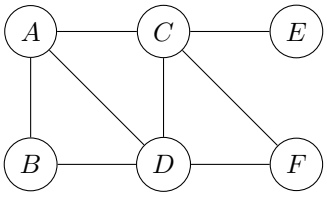


FIGURE 4.3 – Le graphe moral correspondant au réseau bayésien de l'exemple 4.1.1.

Définition 4.2.2 (Séparation)**.** *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 $\mathbf{Anc}^+(\mathbf{U}) = \mathbf{Anc}(\mathbf{U}) \cup \mathbf{U}$ l'ensemble composé des ancêtres de $\mathbf{U}$ et de $\mathbf{U}$ lui-même. On note $\mathcal{G}_\mathbf{U}$ la restriction du graphe $\mathcal{G}$ sur l'ensemble de ses nœuds $\mathbf{U}$.

La proposition suivante, prouvée par [LDLL90], nous montre l'intérêt du graphe moralisé.

Proposition 4.2.1 ([LDLL90])**.** *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 $\mathcal{G}_{\mathbf{Anc}^+(\mathbf{UVZ})}$,* $\mathbf{U}$ *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.

4.2.3 *Direct Recursive Conditioning*

L'algorithme DRC va récursivement décomposer un calcul jusqu'à atteindre des termes qu'on peut estimer à partir de $\mathcal{H}$. 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'ensemble d'exemples. La décomposition de DRC s'appuie sur la proposition suivante :

Proposition 4.2.2. *Soient 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 $\mathcal{G}_{\mathbf{Anc}^+(\mathbf{U})}$. Si $\mathbf{C}$ sépare $\mathbf{G}_1$ de $\mathbf{G}_2$, alors pour toute instanciation partielle $\mathbf{u}$:*

$$p(\mathbf{u}) = \sum_{\substack{\mathbf{c} \in \mathbf{C} \\ \mathbf{c} \sim \mathbf{u}}} \frac{p(\mathbf{u}[\mathbf{G}_1]\mathbf{c})\ p(\mathbf{u}[\mathbf{G}_2]\mathbf{c})}{p(\mathbf{c})} \tag{4.3}$$

Démonstration. D'après la proposition 4.2.1, si $\mathbf{C}$ sépare $\mathbf{G}_1$ de $\mathbf{G}_2$ dans le graphe moralisé de $\mathcal{G}_{\mathbf{Anc}^+(\mathbf{U})}$, 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) ci-dessous :

$$\begin{aligned} p(\mathbf{u}) &= \sum_{\mathbf{c} \sim \mathbf{u}} p(\mathbf{u} \mid \mathbf{c})\ p(\mathbf{c}) \\ &= \sum_{\mathbf{c} \sim \mathbf{u}} p(\mathbf{u}[\mathbf{G}_1] \mid \mathbf{c})\ p(\mathbf{u}[\mathbf{G}_2] \mid \mathbf{c})\ p(\mathbf{c}) \\ &= \sum_{\mathbf{c} \sim \mathbf{u}} \frac{p(\mathbf{u}[\mathbf{G}_1]\mathbf{c})\ p(\mathbf{u}[\mathbf{G}_2]\mathbf{c})}{p(\mathbf{c})} \end{aligned}$$

□

Nous souhaitons souligner deux différences importantes entre la décomposition de RC (équation (4.2)) et celle de DRC (équation (4.3)).

La première différence est que DRC décompose récursivement un terme en trois sous-termes ($p(\mathbf{u}[\mathbf{G}_1]\mathbf{c})$, $p(\mathbf{u}[\mathbf{G}_2]\mathbf{c})$ et $p(\mathbf{c})$), alors que RC décompose un terme en deux sous-termes ($F_{\mathbf{V}_1}(\mathbf{u}[\mathbf{V}_1 \cup \mathbf{Pa}(\mathbf{V}_1)]\mathbf{c})$ et $F_{\mathbf{V}_2}(\mathbf{u}[\mathbf{V}_2 \cup \mathbf{Pa}(\mathbf{V}_2)]\mathbf{c})$). 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 (4.3) 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 avant d'atteindre les feuilles. Chaque appel de DRC vise à calculer une probabilité jointe $p(\mathbf{u})$. Or, nous rappelons que nous disposons d'un ensemble d'exemples $\mathcal{H}$. Si $|\mathcal{H}(\mathbf{u})|$ est assez grand, on peut facilement estimer $p(\mathbf{u})$ par :

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

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 $|\mathcal{H}(\mathbf{u})| > s$, alors nous estimons $p(\mathbf{u})$ par comptage direct. Si $|\mathcal{H}(\mathbf{u})|$ est trop petit, DRC décompose le calcul comme indiqué par l'équation (4.3).

Il peut arriver que DRC ne puisse décomposer $\mathbf{U}$ et soit obligé d'estimer $p(\mathbf{u})$ avec l'ensemble d'exemples même si $|\mathcal{H}(\mathbf{u})|$ est faible. En particulier, si $|\mathcal{H}(\mathbf{u})| = 0$, 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 :

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

Plus $|\mathcal{H}(\mathbf{u})|$ 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 (cf. (4.2.2)), il s'agit d'un arbre ternaire, et pas binaire comme le *dtree* de RC.

Définition 4.2.3 (Arbre de décomposition ternaire)**.** *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 $\mathcal{G}_{\mathbf{Anc}^+(\mathbf{V}_T)}$ 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.*

La figure 4.4 montre un arbre de décomposition ternaire pour le réseau bayésien de l'exemple 4.1.1.

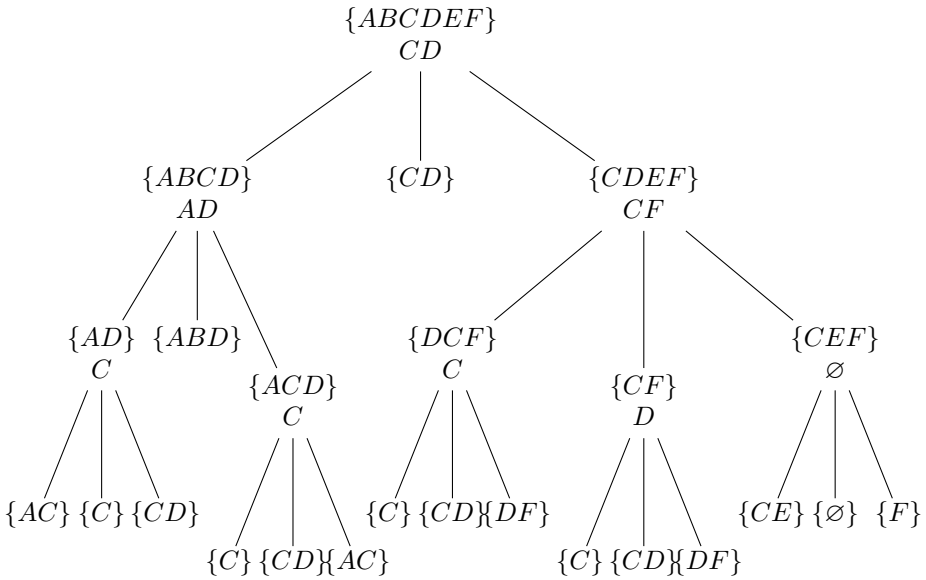


FIGURE 4.4 – Un arbre de décomposition ternaire possible pour le réseau bayésien de l'exemple 4.1.1. Pour chaque nœud T, les variables au-dessus du nœud correspondent à $\mathbf{V}_T$. Si T est un nœud interne, les variables en-dessous correspondent au séparateur $\mathbf{C}_T$ de T.

DRC est présenté dans l'algorithme 3. La ligne 6 est le cas terminal : on estime la probabilité si T est une feuille ou si $|\mathcal{H}(\mathbf{u})|$ 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 4.2.2 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*.

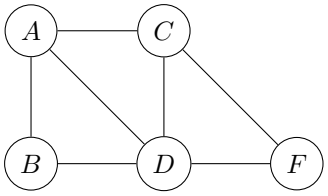
Afin de compter rapidement dans l'ensemble d'exemples (lignes 5 et 6), nous avons compilé celui-ci sous la forme d'un ADD (*Algebraic Decision Diagram* [BFG+97]). Cette compilation, qui peut représenter l'ensemble d'exemples de manière plus compacte qu'une représentation extensive, accélère les calculs par rapport à une approche plus naïve.

Exemple 4.2.1. Reprenons le réseau bayésien de l'exemple 4.1.1 (page 59) et cherchons à calculer $p(bf)$. Le graphe moral de $\mathcal{G}_{\mathbf{Anc}^+(\{B,F\})}$ est :

Algorithm 3: DRC

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

```
1  if U = ∅ then return 1
2  else
3      if proba_cache(u) n'est pas défini then
4          T1, T2, TC ← les trois enfants de T
5          if |H(u)| > s ou T est une feuille then
6              proba_cache(u) ← (|H(u)| + k/|U|)/(|H| + k)
7          else if U ⊆ V_TC then proba_cache(u) ← DRC(TC, u)
8          else if U ⊆ V_T1 then proba_cache(u) ← DRC(T1, u)
9          else if U ⊆ V_T2 then proba_cache(u) ← DRC(T2, u)
10         else
11             p ← 0
12             for each c ∈ C_T, c ∼ u do
13                 p ← p+
                   DRC(T1, u[G1,T]c) × DRC(T2, u[G2,T]c) ÷ DRC(TC, c)
14             proba_cache(u) ← p
15     return proba_cache(u)
```



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'ensemble d'exemples, 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'ensemble d'exemples 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 [Hen86]. Comme ces algorithmes DRC utilise un jeu d'affectations complètes pour estimer des probabilités. La différence majeure est que DRC utilise le jeu de données *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 dans les résultats en termes de taux d'erreur des expérimentations du chapitre 7, 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 allons voir comment en construire un.

4.2.4 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 $\mathcal{G}_{\mathbf{Anc}^+(\mathbf{V}_T)}$ 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 4.2.4 (Ensemble de variables décomposable)**.** *Un ensemble de variables* $\mathbf{V}$ *est dit* décomposable *dans* $\mathcal{G}$ *si le graphe moralisé du sous-graphe induit par* $\mathbf{V}$ *n'est pas complet.*

La construction d'un arbre de décomposition ternaire est détaillée dans l'algorithme 4. Nous supposons disposer d'une fonction *FindPartition*($\mathbf{V}$, $\mathbf{Anc}^+(\mathbf{V})$)

qui renvoie une partition $\{\mathbf{G}_1, \mathbf{G}_2, \mathbf{C}\}$ de $\mathbf{Anc}^+(\mathbf{V})$ telle que $\mathbf{C}$ sépare $\mathbf{G}_1$ et $\mathbf{G}_2$. Cette fonction sera détaillée dans la prochaine sous-section.

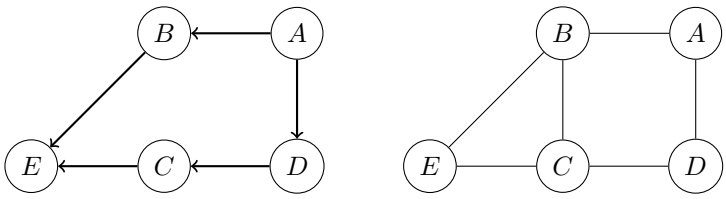
Algorithm 4: Construction d'un arbre de décomposition ternaire

Input: $\mathbf{V}$ un ensemble de variables
Output: un arbre de décomposition ternaire pour $\mathbf{V}$
Algorithme `ConstruitArbre(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 4 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. L'exemple suivant donne l'intuition sur la nécessité d'une telle notion.

Exemple 4.2.2. Soient le réseau bayésien ci-dessous et son graphe moral. Supposons qu'on cherche à construire un arbre de décomposition ternaire. Nous cherchons donc à décomposer $\mathcal{X} = \{A, B, C, D, E\}$.



Considérons les trois étapes suivantes.

1. Initialement, $\mathbf{V} = \mathcal{X}$. Choisissons comme première partition $\mathbf{G}_1 = \{E, C\}, \mathbf{G}_2 = \{A\}, \mathbf{C} = \{B, D\}$; on remarque que $\mathbf{C}$ sépare bien $\mathbf{G}_1$ et $\mathbf{G}_2$. L'algorithme 4 va donc chercher récursivement une décomposition à $\{B, D\}$, $\{B, C, D, E\}$ et $\{A, B, D\}$.

2. Intéressons-nous au cas où $\mathbf{V}' = \{B, C, D, E\}$ et décomposons l'ensemble $\mathbf{Anc}^+(\mathbf{V}') = \mathbf{Anc}^+(\{B, C, D, E\})$. Nous utilisons pour cela la partition $\mathbf{G}'_1 = \{E, B\}, \mathbf{G}'_2 = \{D\}, \mathbf{C}' = \{A, C\}$. Encore une fois, on peut vérifier que $\mathbf{C}'$ sépare $\mathbf{G}'_1$ et $\mathbf{G}'_2$. L'algorithme 4 va donc chercher récursivement une décomposition à $\{A, C\}$, $\{A, B, C, E\}$ et $\{A, C, D\}$.

3. Intéressons-nous cette fois à la décomposition de $\mathbf{Anc}^+(\{A, B, C, E\})$. Nous utilisons pour cela la partition $\mathbf{G}''_1 = \{E, C\}, \mathbf{G}''_2 = \{A\}, \mathbf{C}'' = \{B, D\}$. On peut vérifier que $\mathbf{C}''$ sépare $\mathbf{G}''_1$ et $\mathbf{G}''_2$. L'algorithme 4 va

donc chercher récursivement une décomposition à $\{B, D\}$, $\{B, C, D, E\}$ et $\{A, B, D\}$.

Or, nous voyons que nous avons la même décomposition à l'étape 3 qu'à l'étape 1 ! On pourrait continuer indéfiniment ces trois étapes et l'algorithme 4 ne finirait pas.

Avant d'introduire formellement la notion de partition admissible, 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 peut remarquer que dès que $\mathbf{V}$ est non-vide, $\mathbf{F}_{\mathcal{G}}(\mathbf{V})$ est non-vide car le graphe acyclique orienté induit sur $\mathbf{V}$ a au moins une feuille.

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 par convention $d_{\mathcal{G}_m}(X, Y) = \infty$.

Définition 4.2.5 (Hauteur d'un ensemble de variables dans un graphe)**.** *La hauteur de* $\mathbf{V}$ *dans le graphe moral* $\mathcal{G}_m$ *de* $\mathcal{G}_{\mathbf{Anc}^+(\mathbf{V})}$ *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 4.2.6 (Partition admissible)**.** *Soient* $\mathcal{G}$ *un réseau bayésien,* $\mathbf{V}$ *un ensemble de variables et* $\mathcal{G}_m$ *le graphe moralisé de* $\mathcal{G}_{\mathbf{Anc}^+(\mathbf{V})}$*. 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* $\mathbf{E} \in \{\mathbf{C} \cup (\mathbf{G}_1 \cap \mathbf{V}), \mathbf{C} \cup (\mathbf{G}_2 \cap \mathbf{V}), \mathbf{C}\}$ *:*

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

ou

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

Exemple 4.2.2 (Suite de l'exemple précédent)**.** Les deux décompositions que nous avons utilisées dans l'exemple précédent n'étaient pas admissibles. En effet :

- à l'étape 1, $\mathbf{Anc}^+(\mathbf{C} \cup (\mathbf{G}_1 \cap \mathbf{V})) = \mathbf{Anc}^+(\{B, C, D, E\}) = \mathcal{X}$;
- à l'étape 2, $\mathbf{Anc}^+(\mathbf{C}' \cup (\mathbf{G}'_1 \cap \mathbf{V}')) = \mathbf{Anc}^+(\{A, B, C, E\}) = \mathcal{X}$.

Si les décompositions utilisées avaient été admissibles, l'algorithme 4 n'aurait pas pu aboutir à une boucle infinie.

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.

4.2.5 Construction de partitions admissibles

Nous avons précédemment introduit la notion d'ensemble décomposable (définition 4.2.4). 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 $\mathcal{G}_{\mathbf{Anc}^+(\mathbf{V})}$).

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

Lemme 4.2.1. *Soient 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 $H_{\mathcal{G}_m}(\mathbf{V}) > 1$.*

Démonstration. ($\Leftarrow$) Soient 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 $\mathbf{V}$ soit 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 telle que $X \in \mathbf{V}$ et $\mathbf{V} \subseteq \{X\} \cup \mathbf{Pa}(X)$. Soit $X \in \mathbf{F}_{\mathcal{G}}(\mathbf{V})$; X existe car $\mathbf{V}$ est non vide. Soit $Y \in \mathbf{V}$ différent de X. Le graphe moralisé étant complet, X et Y sont voisins dans celui-ci. 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 un fils $Z \in \mathbf{Anc}^+(\mathbf{V})$. 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.

Donc X est le fils de toutes les autres variables de $\mathbf{V}$; autrement dit, $\mathbf{V} \setminus \{X\} \subseteq \mathbf{Pa}(X)$.

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 $\mathbf{V} = \varnothing$.

($\Rightarrow$) 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 $\mathcal{G}_{\mathbf{Anc}^+(\mathbf{V})}$. 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 dans $\mathcal{G}_m$. Or, il y a une arête dans le graphe moral $\mathcal{G}_m$ si, dans le réseau bayésien :

Cas 3.1 : X est le parent de Y ou X et Y ont a un fils $Z \in \mathbf{Anc}^+(\mathbf{V})$. Or, on a supposé que X n'est l'ancêtre d'aucune autre variable de $\mathbf{V}$, donc ce cas est impossible.

Cas 3.2 : X est le fils de Y.

Ce raisonnement tenant pour tout $Y \in \mathbf{V}$ différent de X, cela signifie 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. □

La proposition suivante va nous permettre de caractériser les ensembles de variables qui admettent une partition admissible : ce sont simplement les ensembles décomposables.

Proposition 4.2.3. *Soient $\mathbf{V}$ 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 $\mathbf{V}$.*

Démonstration. 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 $\mathbf{V}$ décomposable, de construire une partition admissible.

Montrons d'abord qu'un ensemble de variables non décomposable n'admet pas de partition admissible de $\mathcal{G}_m$.

Lemme 4.2.2. *Soient $\mathbf{V}$ 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 $\mathbf{V}$.*

Démonstration. Supposons que $\mathbf{V}$ ne soit pas décomposable dans $\mathcal{G}$ et montrons qu'il n'existe pas de partition admissible.

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, d'après le lemme 3, le sous-graphe induit sur $\mathbf{V}$ est complet, ce qui signifie que Y et Z sont voisins, et donc qu'ils ne peuvent pas être séparés par $\mathbf{C}$. Donc une telle partition n'existe pas. □

Il ne nous reste donc qu'à prouver la réciproque, c'est-à-dire que si $\mathbf{V}$ est décomposable, alors il existe une partition admissible pour $\mathbf{V}$. Nous supposons donc par la suite que $\mathbf{V}$ est décomposable dans $\mathcal{G}$, i.e. $\mathbf{V} \neq \varnothing$ et $H_{\mathcal{G}_m}(\mathbf{V}) \geq 2$ (d'après le lemme 4.2.1). Notre objectif est de construire une partition admissible de $\mathcal{G}_m$ pour $\mathbf{V}$. Nous utiliserons le lemme suivant :

Lemme 4.2.3. *Soient $\mathcal{G}$ un réseau bayésien, $\mathbf{V}$ un ensemble de variables. Alors, pour tout ensemble $\mathbf{E} \subseteq \mathbf{Anc}^+(\mathbf{V})$, s'il existe une variable $X \in \mathbf{F}_{\mathcal{G}}(\mathbf{V})$ telle que $X \notin \mathbf{E}$, alors :*

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

Démonstration. Soient $\mathcal{G}$ un réseau bayésien, $\mathbf{V}$ un ensemble de variables. Alors, pour tout ensemble $\mathbf{E} \subseteq \mathbf{Anc}^+(\mathbf{V})$, s'il existe une variable $X \in \mathbf{F}_{\mathcal{G}}(\mathbf{V})$ telle que $X \notin \mathbf{E}$, alors :

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

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

On sait donc que

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

Nous pouvons donc en conclure que $|\mathbf{Anc}^+(\mathbf{E})| < |\mathbf{Anc}^+(\mathbf{V})|$. □

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 4.2.3 implique que $|\mathbf{Anc}^+(\mathbf{E})| < |\mathbf{Anc}^+(\mathbf{V})|$.

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 $\mathbf{C} \cup (\mathbf{G}_1 \cap \mathbf{V}) \subseteq \mathbf{Anc}^+(\mathbf{V})$, $|\mathbf{Anc}^+(\mathbf{C} \cup (\mathbf{G}_1 \cap \mathbf{V}))| \leq |\mathbf{Anc}^+(\mathbf{V})|$. 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 : $|\mathbf{F}_{\mathcal{G}}(\mathbf{V})| > 1$

Soient 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 4.2.3 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. □

Nous pouvons nous servir de cette preuve comme d'un algorithme de construction de partition admissible : c'est l'algorithme 5.

Algorithm 5: 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(`$\mathcal{G}$`, V)`

```
1  | Gm ← graphe moralisé de G_{Anc+(V)}
2  | F ← {X sans descendant appartenant à V dans G}
3  | if |F| = 0 then return ∅
4  | else if |F| = 1 then
5  |     X ← l'unique variable de F
6  |     d_X ← fonction de distance à X dans Gm
7  |     l ← max_{Y∈V} d_X(Y)
8  |     if l ≤ 1 then return ∅
9  |     else
10 |         Z ← argmax_{Y∈V} d_X(Y)
11 |         P ← {Y ∈ Gm | d_X(Y) < l}
12 |         {G1, G2, C} ← VertexSeparator(P, {X}, Z)
13 |         return {G1, G2, C}
14 | else
15 |     X, Y ← deux variables de F
16 |     {G1, G2, C} ← VertexSeparator(Anc+(V), {X}, {Y})
17 |     return {G1, G2, C}
```

Nous faisons appel dans l'algorithme 5 aux lignes 12 et 16 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 [BJ92]. Dans notre implémentation, nous utilisons une réduction de VS vers HP (*Hypergraph Partitioning*) ; en effet, les deux problèmes sont duaux [ZG14]. Cette réduction a pour but d'utiliser les heuristiques très efficaces développées pour HP.

La présentation de DRC est complète. 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'ensemble d'exemples.

4.3 Conclusion

L'approche d'inférence que nous proposons, *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 ensemble d'exemples qui évolue, tant que les relations d'indépendance conditionnelle ne changent pas. Cette approche peut être utilisée pour n'importe quelle application qui se base sur l'inférence bayésienne à partir d'un jeu de données d'objets combinatoires.

L'idée à la base de notre algorithme d'inférence DRC, l'utilisation d'exemples positifs lors de l'inférence, peut être probablement transposée à d'autres algorithmes d'inférence dans les réseaux bayésiens que RC. Si DRC a une plus grande complexité temporelle que RC, c'est parce que RC ne manipule pas des probabilités mais des facteurs, qui ne peuvent pas être estimés directement à partir des exemples positifs. Nous ne connaissons pas d'autre algorithme d'inférence qui manipule des probabilités, mais un tel algorithme pourrait être facilement adapté à l'estimation directe.

Chapitre 5

Apprentissage de k-LP-trees à partir d'exemples positifs

Sommaire

5.1 Problème d'apprentissage à partir d'exemples positifs 78
5.1.1 Présentation du problème 78
5.2 Distance et score 79
5.2.1 Distances usuelles entre deux ordres 79
5.2.2 *Ranking loss* 80
5.2.3 Rang moyen empirique 82
5.2.4 La vraisemblance 84
5.3 Apprentissage heuristique de k-LP-tree 86
5.3.1 Apprentissage des tables de préférences 86
5.3.2 Heuristique d'étiquetage de nœuds 89
5.3.3 Heuristique de scission 98
5.3.4 Complexité temporelle 98
5.3.5 Élagage pour réduire le surapprentissage 99
5.4 Apprentissage optimal de LP-tree linéaire 100
5.5 *Sample complexity* des LP-trees linéaires binaires . 103
5.6 Expérimentations 106
5.6.1 Protocole 106
5.6.2 Résultats 107
5.7 Perspectives 109

Comme nous avons pu le voir dans le chapitre précédent, la littérature considère classiquement trois tâches d'apprentissage de préférences passif :

- ordonner des objets à partir de comparaison (défini section 3.2.1) ;

- ordonner des objets à partir de notes (défini section 3.2.2) ;
- ordonner des étiquettes (défini section 3.2.3).

Ces problèmes se différencient notamment par les données sur lesquelles se base l'apprentissage. Il existe néanmoins des situations dans lesquelles ni comparaisons, ni notes ne sont disponibles, comme cela peut être le cas pour les configurateurs en ligne. Par contre, les entreprises d'*e-commerce* conservent généralement une liste des produits déjà vendus. Ces objets vendus nous apportent des informations sur les goûts des utilisateurs ; on pourrait donc chercher à apprendre les préférences de ces utilisateurs à partir d'un ensemble d'exemples.

Ce problème, apprendre un modèle ordinal à partir d'un ensemble d'objets sélectionnés par les utilisateurs, n'a encore jamais été traité. Pour cette raison, nous avons décidé de chercher à apprendre des k-LP-trees à partir d'exemples positifs ; en effet, il s'agit de modèles ordinaux relativement simples tout en étant une représentation généralement pertinente des comportements humains [GG96].

5.1 Problème d'apprentissage à partir d'exemples positifs

On pourrait imaginer que l'utilisateur, dans l'éventualité où il peut choisir un objet de son choix, va toujours prendre son objet préféré. Néanmoins, ce n'est pas toujours ce qui se passe, car de nombreux événements peuvent modifier le choix de l'utilisateur :

- le produit désiré est en rupture de stock ;
- une publicité ou une promotion influence son choix ;
- l'abondance et la complexité des possibilités l'empêchent de trouver son objet préféré [HK98, Sch09].

Pourtant, on peut raisonnablement penser que plus un utilisateur préfère un objet, plus grande est la probabilité qu'il choisisse cet objet.

Plus formellement, nous faisons l'hypothèse habituelle que chaque utilisateur a un ordre total de préférences sur les produits. Notre idée est de considérer qu'il y a une distribution de probabilité décroissante p définie sur $[1, |\underline{\mathcal{X}}|]$, où $p(i)$ est la probabilité que l'utilisateur choisisse l'objet de rang i (nous faisons l'hypothèse que la probabilité qu'un objet soit choisi ne dépend que de son rang). Si on note $\mathsf{rank}(\succ^{\smile}, \mathbf{o})$ le rang de l'objet $\mathbf{o}$ selon l'ordre total $\succ^{\smile}$ qui représente les préférences de l'utilisateur, on note $p_{\succ}(\mathbf{o})$ la probabilité que l'utilisateur ayant pour préférence $\succ^{\smile}$ choisisse l'objet $\mathbf{o}$. Autrement dit, $p_{\succ}(\cdot) = p(\mathsf{rank}(\succ^{\smile}, \cdot))$. Par la suite, par abus de notation, nous utiliserons la notation p pour désigner $p_{\succ}$.

Maintenant que cette modélisation est motivée, nous pouvons définir plus formellement le problème d'apprentissage de préférences à partir d'exemples positifs.

5.1.1 Présentation du problème

Nous introduisons la tâche d'apprendre des préférences à partir d'exemples positifs de la manière suivante.

On considère un ensemble d'objets $\underline{\mathcal{X}}$ combinatoires, où chaque objet est représenté par un vecteur. On suppose que les préférences de l'utilisateur sont représentables par un ordre total $\succsim$ sur $\underline{\mathcal{X}}$. L'algorithme d'apprentissage dispose d'un multiensemble $\mathcal{H}$ d'objets de $\underline{\mathcal{X}}$ qui ont été choisis par l'utilisateur de manière indépendante et identiquement distribuée (c'est-à-dire en procédant à un tirage avec remise) selon une loi p. Cette distribution de probabilité est croissante par rapport à $\succsim$, i.e. pour tout $\mathbf{o}, \mathbf{o}' \in \underline{\mathcal{X}}$:

$$\mathbf{o} \succsim \mathbf{o}' \Rightarrow p(\mathbf{o}) \geq p(\mathbf{o}')$$

L'objectif est d'apprendre un ordre total qui puisse ordonner toute paire d'objets. Cet ordre total peut être représenté sous la forme d'un modèle dans un langage de représentation de préférences donné.

Dans ce chapitre, nous nous intéresserons uniquement à l'apprentissage d'un k-LP-tree.

Exemple 5.1.1. Un restaurant enregistre les menus qu'il vend. Pour un utilisateur, il pourrait y avoir l'ensemble de ventes suivant :

$\{$*chocolat/viande/vin rouge*, *chocolat/poisson/vin blanc*,
chocolat/viande/vin blanc, *fruit/poisson/vin blanc*,
fruit/poisson/vin rouge, *fruit/poisson/vin rouge*$\}$

L'objectif est alors d'apprendre un modèle de préférences qui puisse, par exemple, pouvoir répondre à la question suivante : « *que préfère l'utilisateur : chocolat/viande/vin rouge ou fruit/poisson/vin rouge ?* ».

On pourrait étendre ce problème en ajoutant également un multiensemble d'objets explicitement refusés par les utilisateurs, avec l'hypothèse que moins un objet est préféré, plus il a de chances d'être refusé. Néanmoins, nous nous limiterons à un ensemble d'exemples acceptés dans la suite de la thèse.

5.2 Distance et score

Afin d'évaluer la qualité d'un apprentissage, plusieurs distances permettant de mesurer la ressemblance entre deux ordres totaux $\succ$ et $\succsim$ ont été définies.

5.2.1 Distances usuelles entre deux ordres

Deux mesures couramment utilisées sont :

- La distance tau de Kendall [Ken38] est le nombre de paires $(\mathbf{o}, \mathbf{o}')$ ordonnées différemment par $\succ$ et $\succsim$.
- La distance rho de Spearman [Spe04] est un score entre -1 (ordres inversés) et 1 (ordres identiques) qui se calcule de la manière suivante :

$$1 - \frac{6 \sum_{\mathbf{o} \in \underline{\mathcal{X}}} (\mathsf{rank}(\succsim, \mathbf{o}) - \mathsf{rank}(\succ, \mathbf{o}))^2}{|\underline{\mathcal{X}}|(|\underline{\mathcal{X}}|^2 - 1)}$$

Ces distances pondèrent de la même manière les erreurs de rang sur les objets préférés et les objets moins préférés, ce qui a deux inconvénients :

- D'une part, dans un objectif de recommandation, identifier la relation de préférence sur les objets préférés est plus utile que l'identifier sur les objets les moins préférés.
- D'autre part, les historiques de vente contiennent des exemples positifs, donc qui sont préférés. Pour cette raison, apprendre les objets préférés sera plus simple qu'apprendre les objets les moins préférés.

5.2.2 *Ranking loss*

Pour cette raison, nous introduisons notre propre distance qui est adaptée à notre problème : la *ranking loss*.

Définition 5.2.1 (Ranking loss)**.** *La* ranking loss *entre un ordre cible $\breve{\succ}$ et un ordre quelconque $\succ$ est égale à la différence des rangs de chaque item de $\underline{\mathcal{X}}$, pondérée par leur probabilité p, et normalisée par le rang maximal. Formellement :*

$$\mathsf{rloss}_p(\breve{\succ}, \succ) = \frac{1}{|\underline{\mathcal{X}}|} \sum_{\mathbf{o} \in \underline{\mathcal{X}}} p(\mathbf{o})(\mathsf{rank}(\succ, \mathbf{o}) - \mathsf{rank}(\breve{\succ}, \mathbf{o}))$$

Il est important de remarquer qu'*on ne prend pas la valeur absolue de la différence des rangs*. Il peut donc y avoir des compensations possibles. Néanmoins, la *ranking loss* a des propriétés appréciables.

Proposition 5.2.1. *Soient $\breve{\succ}$ les préférences de l'utilisateur, p la distribution de probabilité de tirage des objets croissante par rapport à $\breve{\succ}$, et $\succ$ une relation d'ordre quelconque. Alors :*

- $0 \leq \mathsf{rloss}_p(\breve{\succ}, \succ) < 1$ *;*
- *Si p est strictement croissante par rapport à $\breve{\succ}$, alors* $\mathsf{rloss}_p(\breve{\succ}, \succ) = 0$ *ssi* $\breve{\succ} = \succ$.

Démonstration. Le fait que $\mathsf{rloss}_p(\breve{\succ}, \succ) < 1$ est une conséquence du fait que pour chaque objet $\mathbf{o}$, $(\mathsf{rank}(\succ, \mathbf{o}) - \mathsf{rank}(\breve{\succ}, \mathbf{o}))/|\mathcal{X}| < 1$ car tous les rangs appartiennent à $\{1, \ldots, |\mathcal{X}|\}$.

Les deux autres propriétés suivent de l'inégalité de réarrangement [HLP52, sect. 10.2], qui peut être formulée comme suit.

Soient deux séquences de nombres réels $r_1 \leq \ldots \leq r_N$ et $p_1 \geq \ldots \geq p_N$. Pour toute permutation σ de $\{1, \ldots, N\}$:

$$r_1 p_1 + \ldots r_N p_N \leq r_{\sigma(1)} p_1 + \ldots + r_{\sigma(N)} p_N$$

De plus, si ces séquences sont respectivement strictement croissante et strictement décroissante, alors l'égalité est atteinte seulement lorsque σ est l'identité.

On peut appliquer l'inégalité de réarrangement en considérant que :

- les $\{r_i\}_i$ sont les rangs $\mathsf{rank}(\breve{\succ}, \mathbf{o})$ ordonnés par $\breve{\succ}$, c'est-à-dire $r_i = i$ (c'est donc une séquence strictement croissante) ;

- les $\{p_i\}_i$ sont les probabilités correspondantes $p(\mathbf{o})$ ordonnées par $\breve{\succ}$ (c'est donc une séquence décroissante) ;
- σ est la permutation des rangs entre $\breve{\succ}$ et $\succ$, c'est-à-dire telle que pour tout $\mathbf{o}$:

$$\mathsf{rank}(\succ, \mathbf{o}) = \sigma(\mathsf{rank}(\breve{\succ}, \mathbf{o}))$$

En application l'inégalité de réarrangement, on obtient que :

$$\begin{aligned}\sum_{\mathbf{o}} p(\mathbf{o})\mathsf{rank}(\breve{\succ}, \mathbf{o}) &= r_1 p_1 + \dots r_N p_N \\ &\leq r_{\sigma(1)} p_1 + \dots + r_{\sigma(N)} p_N \\ &= \sum_{\mathbf{o}} p(\mathbf{o})\sigma(\mathsf{rank}(\breve{\succ}, \mathbf{o})) \\ &= \sum_{\mathbf{o}} p(\mathbf{o})\mathsf{rank}(\succ, \mathbf{o})\end{aligned}$$

De cette inégalité découle directement la positivité de la *ranking loss*.

De plus, si p est strictement croissante par rapport à $\breve{\succ}$, on peut appliquer le cas limite de l'inégalité de réarrangement : la *ranking loss* s'annule seulement si $\sigma = Id$, i.e. $\succ = \breve{\succ}$. □

Par la suite, par abus de notation, nous noterons $\mathsf{rloss}(\breve{\mathcal{L}}, \mathcal{L})$ la *ranking loss* entre les ordres totaux représentés par les k-LP-trees $\breve{\mathcal{L}}$ et $\mathcal{L}$.

Dans le cas des k-LP-trees, nous disposons d'une borne supérieure à la *ranking loss* de deux k-LP-trees qui ont au moins d étages en commun. Pour cela, nous introduisons la notation suivante : pour tout $\mathcal{L}$, $\mathcal{L}'$, on note $\mathcal{L} \overset{d}{=} \mathcal{L}'$ si et seulement si tous les nœuds dont la profondeur est inférieure ou égale à d sont identiques dans $\mathcal{L}$ et dans $\mathcal{L}'$ (la profondeur est définie, avec d'autres notions de la théorie des graphes, dans l'annexe A).

Proposition 5.2.2. *Soit $d \in [0, n]$, soient $\breve{\mathcal{L}}, \mathcal{L}$ deux k-LP-trees qui représentent respectivement les ordres $\breve{\succ}$ et $\succ$, dont les attributs sont binaires et tels que $\breve{\mathcal{L}} \overset{d}{=} \mathcal{L}$. Soit p une probabilité croissante par rapport à $\breve{\succ}$. Alors :*

$$|\mathsf{rloss}_p(\breve{\succ}, \succ)| < 2^{-d}$$

Démonstration. Soit $\breve{\mathcal{L}}$ le k-LP-tree cible. Étant donné que $\breve{\mathcal{L}} \overset{d}{=} \mathcal{L}$, alors pour tout objet $\mathbf{o} \in \underline{\mathcal{X}}$, $\mathbf{o}$ sera compatible avec un unique $\mathbf{inst}(N)$ où N est à la profondeur d et est donc commun aux deux arbres. Dons la différence entre le $\mathsf{rank}(\breve{\succ}, \mathbf{o})$ et $\mathsf{rank}(\succ, \mathbf{o})$ est strictement inférieur à 2^{n-d}, i.e. :

$$|\mathsf{rank}(\breve{\succ}, \mathbf{o}) - \mathsf{rank}(\succ, \mathbf{o})| < 2^{n-d}$$

Alors :

$$\begin{aligned}
|\mathsf{rloss}_p(\breve{\mathcal{L}}, \mathcal{L})| &= |\frac{1}{2^n} \sum_{\mathbf{o} \in \underline{\mathcal{X}}} p(\mathbf{o})(\mathsf{rank}(\succ, \mathbf{o}) - \mathsf{rank}(\breve{\succ}, \mathbf{o}))| \\
&\leq \frac{1}{2^n} \sum_{\mathbf{o} \in \underline{\mathcal{X}}} p(\mathbf{o})|(\mathsf{rank}(\succ, \mathbf{o}) - \mathsf{rank}(\breve{\succ}, \mathbf{o}))| \\
&< \frac{1}{2^n} \sum_{\mathbf{o} \in \underline{\mathcal{X}}} p(\mathbf{o}) 2^{n-d} \\
&= 2^{-d}
\end{aligned}$$

□

Cette proposition nous permet d'interpréter une *ranking loss* ; si elle vaut par exemple 10^{-7}, cela équivaut à $-\log_2(10^{-7}) \approx 20$ étages corrects.

On peut remarquer que cette distance peut également s'interpréter comme la différence (normalisée par $|\underline{\mathcal{X}}|$) du rang moyen espéré selon $\succ$ et $\breve{\succ}$.

$$\mathsf{rloss}_p(\breve{\succ}, \succ) = \frac{1}{|\underline{\mathcal{X}}|} \mathbb{E}_p[\mathsf{rank}(\succ, \cdot)] - \frac{1}{|\underline{\mathcal{X}}|} \mathbb{E}_p[\mathsf{rank}(\breve{\succ}, \cdot)] \tag{5.1}$$

Ceci nous montre que minimiser la *ranking loss* revient à trouver le modèle qui minimise l'espérance du rang. Étant donné que l'espérance du rang n'est pas calculable à partir d'un ensemble d'exemples (il s'agit d'un risque réel), cela nous amène naturellement à chercher à minimiser le rang moyen empirique (le risque empirique).

5.2.3 Rang moyen empirique

Soit $\mathcal{L}$ un modèle. Pour évaluer sa pertinence vis-à-vis du jeu de données $\mathcal{H}$, nous proposons de calculer le rang moyen empirique des éléments de $\mathcal{H}$ selon l'ordre représenté par $\mathcal{L}$. Formellement, nous le définissons de la manière suivante.

Définition 5.2.2 (Rang moyen empirique)**.** *Le rang moyen empirique des éléments de $\mathcal{H}$ selon le modèle $\mathcal{L}$ est :*

$$\overline{\mathsf{rank}}(\mathcal{L}, \mathcal{H}) = \frac{1}{|\mathcal{H}|} \sum_{\mathbf{o} \in \mathcal{H}} \mathsf{rank}(\mathcal{L}, \mathbf{o})$$

Le rang moyen empirique est très facile à calculer pour un k-LP-tree, pour lequel le calcul du rang d'un objet a une complexité temporelle polynomiale en le nombre de variables [LMX12].

Intuitivement, les objets de $\mathcal{H}$ sont probablement des objets appréciés, et qui ont donc un rang assez faible. Le modèle appris doit donc associer aux objets de $\mathcal{H}$ des rangs faibles ; le rang moyen empirique de $\mathcal{H}$ selon $\mathcal{L}$ devrait donc être aussi faible que possible.

La proposition suivante confirme cette intuition.

Proposition 5.2.3. *Soit L un langage fini, soient $\breve{\mathcal{L}} \in L$, p une probabilité croissante par rapport à l'ordre représenté par $\breve{\mathcal{L}}$ et $\mathcal{H}$ une suite (infinie) de tirages indépendants et identiquement distribués selon p. Pour tout k, soit $\mathcal{H}_k$*

l'ensemble composé des k premiers éléments de $\mathcal{H}$. Soit $\mathcal{L}_k^* \in L$ un modèle qui minimise le rang moyen empirique de $\mathcal{H}_k$. Alors la probabilité pour que la ranking loss entre $\mathcal{L}_k^*$ et $\breve{\mathcal{L}}$ tende vers 0 quand le nombre d'exemples tend vers l'infini est égale à 1, i.e. :

$$\Pr\left(\lim_{k\to+\infty} \mathsf{rloss}_p(\breve{\mathcal{L}}, \mathcal{L}_k^*) = 0\right) = 1$$

Autrement dit, la ranking loss de $\mathcal{L}_k^*$ tend vers zéro presque sûrement lorsque le nombre d'exemples tend vers l'infini.

Démonstration. Soit $\mathcal{L} \in L$ un modèle quelconque. D'après la loi forte des grands nombres, la moyenne empirique $\overline{\mathsf{rank}}(\mathcal{L}, \mathcal{H}_k)$ converge presque sûrement vers l'espérance $\mathbb{E}_p[\mathsf{rank}(\mathcal{L}, \cdot)]$, c'est-à-dire :

$$\Pr\left(\lim_{k\to+\infty} \overline{\mathsf{rank}}(\mathcal{L}, \mathcal{H}_k) = \mathbb{E}_p[\mathsf{rank}(\mathcal{L}, \cdot)]\right) = 1$$

Faisons l'hypothèse, par la suite, que :

$$\forall \mathcal{L} \in L, \lim_{k\to+\infty} \overline{\mathsf{rank}}(\mathcal{L}, \mathcal{H}_k) = \mathbb{E}_p[\mathsf{rank}(\mathcal{L}, \cdot)] \tag{5.2}$$

Notons $\breve{L}$ l'ensemble des modèles de L dont le rang moyen empirique convergent vers $\mathbb{E}_p[\mathsf{rank}(\breve{\mathcal{L}}, \cdot)]$. D'après l'équation (5.1), cela signifie que $\breve{L}$ est l'ensemble des k-LP-trees dont la *ranking loss* est nulle, i.e. $\breve{L} = \{\mathcal{L} : \mathsf{rloss}_p(\breve{\mathcal{L}}, \mathcal{L}) = 0\}$. En peut remarquer que $\breve{L}$ n'est pas vide car $\breve{\mathcal{L}} \in \breve{L}$.

Cas 1 : $\breve{L} = L$. Alors en particulier $\mathcal{L}_k^* \in \breve{L}$ et donc $\mathsf{rloss}_p(\breve{\mathcal{L}}, \mathcal{L}_k^*) = 0$.

Cas 2 : $L \subset \breve{L}$. Notons $\delta = \min_{\mathcal{L}\in L\setminus\breve{L}} \mathbb{E}_p[\mathsf{rank}(\mathcal{L}, \cdot)] - \mathbb{E}_p[\mathsf{rank}(\breve{\mathcal{L}}, \cdot)]$. Ce minimum existe car L est fini. δ est, à un facteur multiplicatif près, égal à une *ranking loss* d'après l'équation (5.1). La loi p étant croissante par rapport à l'ordre représenté par $\breve{\mathcal{L}}$, la proposition 5.2.1 nous indique que $\delta > 0$.

Étant donné que nous avancé supposé (5.2), il existe une valeur K telle que, pour tout $\mathcal{L} \in L$, pour tout $k > K$ (K existe car L est fini) :

$$|\overline{\mathsf{rank}}(\mathcal{L}, \mathcal{H}_k) - \mathbb{E}_p[\mathsf{rank}(\mathcal{L}, \cdot)]| < \frac{\delta}{2}$$

Soient $\mathcal{L} \in L \setminus \breve{L}$, $k > K$. Alors :

$$\begin{aligned}
\overline{\mathsf{rank}}(\mathcal{L}, \mathcal{H}_k) - \overline{\mathsf{rank}}(\breve{\mathcal{L}}, \mathcal{H}_k) &= \frac{\delta}{2} + \frac{\delta}{2} + \overline{\mathsf{rank}}(\mathcal{L}, \mathcal{H}_k) - \overline{\mathsf{rank}}(\breve{\mathcal{L}}, \mathcal{H}_k) - \delta \\
&> |\mathbb{E}_p[\mathsf{rank}(\mathcal{L}, \cdot)] - \overline{\mathsf{rank}}(\mathcal{L}, \mathcal{H}_k)| + |\mathbb{E}_p[\mathsf{rank}(\breve{\mathcal{L}}, \cdot)] - \overline{\mathsf{rank}}(\breve{\mathcal{L}}, \mathcal{H}_k)| \\
&\qquad + \overline{\mathsf{rank}}(\mathcal{L}, \mathcal{H}_k) - \overline{\mathsf{rank}}(\breve{\mathcal{L}}, \mathcal{H}_k) - \delta \\
&> \mathbb{E}_p[\mathsf{rank}(\mathcal{L}, \cdot)] - \overline{\mathsf{rank}}(\mathcal{L}, \mathcal{H}_k) - \mathbb{E}_p[\mathsf{rank}(\breve{\mathcal{L}}, \cdot)] + \overline{\mathsf{rank}}(\breve{\mathcal{L}}, \mathcal{H}_k) \\
&\qquad + \overline{\mathsf{rank}}(\mathcal{L}, \mathcal{H}_k) - \overline{\mathsf{rank}}(\breve{\mathcal{L}}, \mathcal{H}_k) - \delta \\
&> \mathbb{E}_p[\mathsf{rank}(\mathcal{L}, \cdot)] - \mathbb{E}_p[\mathsf{rank}(\breve{\mathcal{L}}, \cdot)] - \delta \\
&> \min_{\mathcal{L}\in L\setminus\breve{L}} \mathbb{E}_p[\mathsf{rank}(\mathcal{L}, \cdot)] - \mathbb{E}_p[\mathsf{rank}(\breve{\mathcal{L}}, \cdot)] - \delta \\
&> 0
\end{aligned}$$

Donc $\overline{\mathsf{rank}}(\mathcal{L}, \mathcal{H}_k) > \overline{\mathsf{rank}}(\breve{\mathcal{L}}, \mathcal{H}_k)$. Or, $\mathcal{L}_k^*$ minimise par définition le rang moyen empirique de $\mathcal{H}_k$ pour tout k : nous pouvons donc en déduire que $\mathcal{L}_k^* \in \breve{L}$. En effet, si $\mathcal{L}_k^* \notin \breve{L}$, alors $\overline{\mathsf{rank}}(\breve{\mathcal{L}}, \mathcal{H}_k) < \overline{\mathsf{rank}}(\mathcal{L}_k^*, \mathcal{H}_k)$, ce qui contredit l'hypothèse selon laquelle $\mathcal{L}_k^*$ minimise le rang moyen empirique. Étant donné que $\mathcal{L}_k^* \in \breve{L}$, $\mathsf{rloss}_p(\breve{\mathcal{L}}, \mathcal{L}_k^*) = 0$.

Nous venons donc de prouver l'implication suivante dans les deux cas :

$$\forall \mathcal{L} \in L, \lim_{k \to +\infty} \overline{\mathsf{rank}}(\mathcal{L}, \mathcal{H}_k) = \mathbb{E}_p[\mathsf{rank}(\mathcal{L}, \cdot)] \Longrightarrow \mathsf{rloss}_p(\breve{\mathcal{L}}, \mathcal{L}_k^*) = 0$$

Donc :

$$\begin{aligned}
\Pr\left(\mathsf{rloss}_p(\breve{\mathcal{L}}, \mathcal{L}_k^*) = 0\right) &\geq \Pr\left(\forall \mathcal{L} \in L, \lim_{k \to +\infty} \overline{\mathsf{rank}}(\mathcal{L}, \mathcal{H}_k) = \mathbb{E}_p[\mathsf{rank}(\mathcal{L}, \cdot)]\right) \\
&= 1 - \Pr\left(\exists \mathcal{L} \in L, \neg\left(\lim_{k \to +\infty} \overline{\mathsf{rank}}(\mathcal{L}, \mathcal{H}_k) = \mathbb{E}_p[\mathsf{rank}(\mathcal{L}, \cdot)]\right)\right) \\
&= 1 - \sum_{\mathcal{L} \in L} \Pr\left(\neg\left(\lim_{k \to +\infty} \overline{\mathsf{rank}}(\mathcal{L}, \mathcal{H}_k) = \mathbb{E}_p[\mathsf{rank}(\mathcal{L}, \cdot)]\right)\right) \\
&= 1
\end{aligned}$$

Autrement dit, $\Pr\left(\mathsf{rloss}_p(\breve{\mathcal{L}}, \mathcal{L}_k^*) = 0\right) = 1$, ce qui achève la preuve. □

5.2.4 La vraisemblance

Un score classique en apprentissage de modèle probabiliste est la vraisemblance. La vraisemblance peut se calculer uniquement pour les modèles qui représentent une loi de probabilité. Si $\mathcal{L}$ représente une distribution de probabilité $p_{\mathcal{L}}$, la vraisemblance de $\mathcal{L}$ pour un ensemble $\mathcal{H}$ d'exemples tirés de manière indépendante et identiquement distribuée est définie de la manière suivante :

$$L(\mathcal{L}, \mathcal{H}) = p_{\mathcal{L}}(\mathcal{H}) = \prod_{\mathbf{o} \in \mathcal{H}} p_{\mathcal{L}}(\mathbf{o})$$

Un score dérivé et équivalent, également très utilisé, est la log-vraisemblance :

$$LL(\mathcal{L}, \mathcal{H}) = \log L(\mathcal{L}, \mathcal{H}) = \sum_{\mathbf{o} \in \mathcal{H}} \log p_{\mathcal{L}}(\mathbf{o})$$

Ce score est défini pour les modèles qui représentent une distribution de probabilité. Mais dans notre cas, nous cherchons à apprendre un ordre $\succ$ et pas une loi de probabilité ! Pour cette raison, nous ne pouvons pas calculer la vraisemblance d'un modèle.

De plus, le modèle qui maximise la vraisemblance dépend de p (ce qui signifie que faire une hypothèse sur la forme de p conditionnerait le modèle appris), comme le montre le contre-exemple suivant.

Exemple 5.2.1. Soit $\mathcal{H}$ un ensemble d'exemples positifs (l'attribut « Dessert » a été omis pour simplifier l'exemple) :

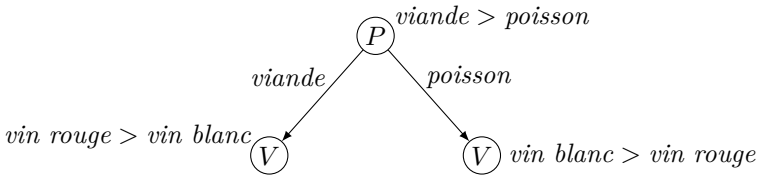
viande/vin rouge	*viande/vin blanc*
20 items	8 items

poisson/vin rouge	*poisson/vin blanc*
14 items	15 items

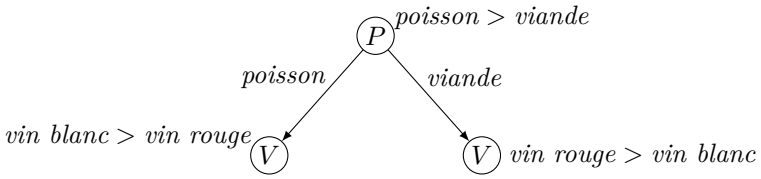
Afin de montrer que le LP-tree qui maximise la vraisemblance de $\mathcal{H}$ dépend de la probabilité p de tirage des objets, nous construisons deux distributions de probabilité p_1 et p_2 :

$$p_1(r) = \begin{cases} 1 \text{ si } r = 1 \\ 0 \text{ sinon} \end{cases} \quad \text{et } p_2(r) = \begin{cases} \frac{1}{2} \text{ si } r \in \{1, 2\} \\ 0 \text{ sinon} \end{cases}$$

Le LP-tree qui maximise la vraisemblance de $\mathcal{H}$ selon p_1 est :



Le LP-tree qui maximise la vraisemblance de $\mathcal{H}$ selon p_2 est :



L'arbre qui maximise la vraisemblance de $\mathcal{H}$ selon p_1 n'est pas le même que celui qui maximise la vraisemblance de $\mathcal{H}$ selon p_2. Donc, sans information sur la loi de probabilité des exemples, on ne peut pas espérer maximiser la vraisemblance de $\mathcal{H}$.

Nous ne pouvons donc pas utiliser ce score classique qu'est la vraisemblance sans faire d'hypothèse sur la loi p.

Le rang moyen empirique n'est toutefois pas dépourvu de lien avec la vraisemblance. En effet, si on fait l'hypothèse que p est une loi géométrique tronquée sur $[1, |\underline{\mathcal{X}}|]$, minimiser le rang moyen empirique est équivalent à maximiser la vraisemblance, comme le montre la proposition suivante.

Proposition 5.2.4. *Soit $\succcurlyeq$ un ordre total, soit p une loi géométrique tronquée de paramètre $\mu \in]0, 1[$ et strictement croissante par rapport à $\succcurlyeq$ et $\mathcal{H}$ un ensemble d'exemples positifs tirés selon p. Alors le modèle $\mathcal{L}$ qui minimise le rang moyen empirique de $\mathcal{H}$ maximise également la vraisemblance de $\mathcal{H}$; ou, plus formellement :*

$$\underset{\mathcal{L}}{\operatorname{argmax}}\, L(\mathcal{L}, \mathcal{H}) = \underset{\mathcal{L}}{\operatorname{argmin}}\, \overline{\mathsf{rank}}(\mathcal{L}, \mathcal{H})$$

Démonstration. Si p une loi géométrique tronquée de paramètre $\mu \in]0, 1[$, cela signifie que $p(\mathbf{o}) = K(1-\mu)^{\mathsf{rank}(\succcurlyeq, \mathbf{o})-1}\mu$, où K est une constante de normalisation car p est tronquée.

Le reste n'est qu'un calcul :

$$\begin{aligned}
\operatorname*{argmax}_{\mathcal{L}} L(\mathcal{L},\mathcal{H}) &= \operatorname*{argmax}_{\mathcal{L}} LL(\mathcal{L},\mathcal{H}) \\
&= \operatorname*{argmax}_{\mathcal{L}} \sum_{\mathbf{o}\in\mathcal{H}} \log p(\mathbf{o}) \\
&= \operatorname*{argmax}_{\mathcal{L}} \sum_{\mathbf{o}\in\mathcal{H}} \log(K(1-\mu)^{\mathsf{rank}(\succ,\mathbf{o})-1}\mu) \\
&= \operatorname*{argmax}_{\mathcal{L}} \sum_{\mathbf{o}\in\mathcal{H}} (\mathsf{rank}(\succ,\mathbf{o})-1)\log(1-\mu) \\
&= \operatorname*{argmin}_{\mathcal{L}} \sum_{\mathbf{o}\in\mathcal{H}} \mathsf{rank}(\succ,\mathbf{o}) \\
&= \operatorname*{argmin}_{\mathcal{L}} \overline{\mathsf{rank}}(\mathcal{L},\mathcal{H})
\end{aligned}$$

□

Cette proposition est particulièrement intéressante car l'hypothèse d'une décroissance exponentielle est régulièrement faite en apprentissage de préférences, notamment au travers du modèle de Bradley–Terry [BT52] qui permet d'estimer des comparaisons à partir d'observations.

5.3 Apprentissage heuristique de k-LP-tree

Notre première contribution dans ce chapitre est un algorithme glouton qui apprend un k-LP-tree qui minimise approximativement le rang moyen empirique. Notre approche suit le schéma générique défini par [BCL+10, BH12] pour apprendre des LP-trees à partir de comparaisons, que nous étendons à l'apprentissage à partir d'exemples positifs. Les résultats de cette section ont été précédemment publiés dans [FGM18a].

Cet algorithme (Algorithme 6), construit un arbre de manière gloutonne, depuis la racine jusqu'au feuilles et en considérant, pour chaque nœud N, les éléments du jeu d'exemples qui sont compatibles avec les instanciations $\mathbf{inst}(N)$.

Soient $\mathbf{u} \in \underline{\mathbf{U}}$, où $\mathbf{U} \subseteq \mathcal{X}$, et $\mathcal{H}$ un jeu d'exemples positifs. Nous noterons $\mathcal{H}(\mathbf{u})$ les éléments de $\mathcal{H}$ compatible avec $\mathbf{u}$:

$$\mathcal{H}(\mathbf{u}) = \{\mathbf{o} \in \mathcal{H} \mid \mathbf{o} \sim \mathbf{u}\}$$

Ainsi, à un nœud N (initialement la racine) qu'on cherche à étiqueter, la ligne 3 considère l'ensemble $\mathcal{H}(\mathbf{inst}(N))$ des éléments de $\mathcal{H}$ qui sont compatibles avec les affectations faites entre la racine et N. La fonction chooseAttributes retourne l'ensemble des attributs et la table de préférence qui vont étiqueter N. L'arbre est ensuite étendu sous N selon l'ensemble d'étiquettes renvoyé par generateLabels.

5.3.1 Apprentissage des tables de préférences

Étant donné un nœud N, la fonction chooseAttribute retourne des attributs et une CPT (table de préférences conditionnelles) de manière à expliquer aussi

Algorithm 6: Apprend un k-LP-tree à partir de $\mathcal{H}$

Input: $\mathcal{X}$, un jeu d'exemples positifs $\mathcal{H}$ sur $\mathcal{X}$
Output: $\mathcal{L}$ le k-LP-tree appris
Algorithm `LearnLPTree`$(\mathcal{X}, \mathcal{H})$

```
1  L ← racine non étiquetée
2  while L contient au moins un nœud non étiqueté N do
3      (X, table) ← ChooseAttributes(N)
4      étiqueter N avec les attributs X et la CPT table
5      L ← GenerateLabels(N, X)
6      for each l ∈ L do
7          ajouter un nouveau nœud non étiqueté à L, attaché à N par
           une arête étiquetée par l
8  return L
```

bien que possible $\mathcal{H}(\mathbf{inst}(N))$, c'est-à-dire à produire un sous-arbre attribuant aux éléments de $\mathcal{H}(\mathbf{inst}(N))$ un rang moyen aussi faible que possible.

Supposons d'abord qu'on connaisse l'ensemble d'attributs $\mathbf{X}$ qui doit étiqueter N, et qu'on cherche la CPT de N. Notre algorithme apprend une unique règle inconditionnelle.

Nous noterons par la suite $\mathbf{x}^{(i)}_{\mathbf{n},\mathcal{H}}$ la i^{e} valeur de $\mathbf{X}$ avec le plus d'occurrences dans $\mathcal{H}(\mathbf{n})$. Par exemple, $\mathbf{x}^{(1)}_{\mathbf{n},\mathcal{H}} = \text{argmax}_{\mathbf{x}\in\underline{\mathbf{X}}} |\mathcal{H}(\mathbf{xn})|$. Étant donné qu'on ne traitera que d'un seul jeu $\mathcal{H}$ à la fois, l'indice $\mathcal{H}$ dans $\mathbf{x}^{(i)}_{\mathbf{n},\mathcal{H}}$ sera omis par la suite.

Le théorème suivant montre que les valeurs de $\underline{\mathbf{X}}$ doivent être ordonnées par leur nombre d'occurrences dans $\mathcal{H}(\mathbf{inst}(N))$ afin de minimiser le rang moyen empirique de $\mathcal{H}$.

Proposition 5.3.1. *Soit $\mathcal{H}$ un ensemble quelconque d'exemples positifs. Soient $\mathcal{L}$ un k-LP-tree quelconque, et N l'un de ses nœuds étiqueté par $\mathbf{X}$. Construisons $\mathcal{L}'$ qui soit identique à $\mathcal{L}$ hormis le fait que la table de préférence de N ordonne les valeurs de $\underline{\mathbf{X}}$ par nombre d'occurrences dans $\mathcal{H}(\mathbf{inst}(N))$ décroissant. Alors $\mathcal{L}'$ a un score inférieur ou égal à $\mathcal{L}$, i.e. :*

$$\overline{\mathsf{rank}}(\mathcal{L}', \mathcal{H}) \leq \overline{\mathsf{rank}}(\mathcal{L}, \mathcal{H})$$

Démonstration. La table de préférence de N n'influence que l'ordre des objets compatibles avec $\mathbf{inst}(N)$. Supposons donc sans perdre de généralité que N est la racine de $\mathcal{L}$; si ce n'est pas le cas, on peut s'y ramener en s'intéressant au sous-arbre dont N est la racine. Notons $\mathbf{X}$ l'étiquette de N et d la cardinalité de $\mathbf{X}$, i.e. $d = |\underline{\mathbf{X}}|$.

Notons $\{\mathcal{L}'_i\}_i$ les sous-arbres de N selon les valeurs $\mathbf{x}^{(i)}_{\mathbf{n}}$; c'est-à-dire que $\mathcal{L}'_i$ est le sous-arbre de N dont la racine R_i vérifie $\mathbf{inst}(R_i) = \{\mathbf{x}^{(i)}_{\mathbf{n}}\}$ (ou $\mathbf{inst}(R_i) = \varnothing$ si N a un seul enfant et un arc non étiqueté). Notons $\bar{r}_i$ le rang moyen des éléments de $\mathcal{H}(\mathbf{x}^{(i)}_{\mathbf{n}})$ selon chaque sous-arbre de N dans $\mathcal{L}'$, i.e. :

$$\bar{r}_i = \frac{1}{|\mathcal{H}(\mathbf{x}^{(i)}_{\mathbf{n}})|} \sum_{\mathbf{o}\in\mathcal{H}(\mathbf{x}^{(i)}_{\mathbf{n}})} \mathsf{rank}(\mathcal{L}'_i, \mathbf{o})$$

Nous pouvons exprimer en fonction de ces $\bar{r}_i$ le rang moyen empirique de $\mathcal{L}'$:

$$\overline{\mathsf{rank}}(\mathcal{L}', \mathcal{H}) = \sum_{i=1}^{d} |\mathcal{H}(\mathbf{x}_{\mathbf{n}}^{(i)})|(\bar{r}_i + (i-1)\frac{|\mathcal{X}|}{d})$$

Notons à présent $\mathsf{rank}(\succ', \cdot)$ le rang des valeurs de N selon $\succ'$, la table de préférences de N dans $\mathcal{L}'$, et $\mathsf{rank}(\succ, \cdot)$ le rang des valeurs de N selon $\succ$, la table de préférences de N dans $\mathcal{L}$. Il existe une permutation σ telle que $\mathsf{rank}(\succ, \cdot) = \sigma(\mathsf{rank}(\succ', \cdot))$. Le rang moyen empirique de $\mathcal{L}$ peut alors s'exprimer de la manière suivante :

$$\overline{\mathsf{rank}}(\mathcal{L}, \mathcal{H}) = \sum_{i=1}^{d} |\mathcal{H}(\mathbf{x}_{\mathbf{n}}^{(\sigma(i))})|(\bar{r}_{\sigma(i)} + (i-1)\frac{|\mathcal{X}|}{d})$$

Ou, en procédant à un changement d'indice :

$$\overline{\mathsf{rank}}(\mathcal{L}, \mathcal{H}) = \sum_{i=1}^{d} |\mathcal{H}(\mathbf{x}_{\mathbf{n}}^{(i)})|(\bar{r}_i + (\sigma^{-1}(i)-1)\frac{|\mathcal{X}|}{d})$$

La différence entre ces deux rangs moyens empiriques est donc :

$$\begin{aligned}\overline{\mathsf{rank}}(\mathcal{L}, \mathcal{H}) - \overline{\mathsf{rank}}(\mathcal{L}', \mathcal{H}) &= \sum_{i=1}^{d} |\mathcal{H}(\mathbf{x}_{\mathbf{n}}^{(i)})|\frac{|\mathcal{X}|}{d}((\sigma^{-1}(i)-1) - (i-1)) \\ &= \frac{|\mathcal{X}|}{d}\left(\sum_{i=1}^{d} |\mathcal{H}(\mathbf{x}_{\mathbf{n}}^{(i)})|\sigma^{-1}(i) - \sum_{i=1}^{d} |\mathcal{H}(\mathbf{x}_{\mathbf{n}}^{(i)})|i\right)\end{aligned}$$

On peut appliquer une fois de plus l'inégalité de réarrangement : $\{|\mathcal{H}(\mathbf{x}_{\mathbf{n}}^{(i)})|\}_i$ est une suite décroissante, $\{i\}_i$ est une suite croissante et $\{\sigma^{-1}(i)\}_i$ est une permutation de $\{i\}_i$. L'inégalité de réarrangement nous permet d'en déduire que cette différence est positive car le premier terme est plus grand (ou égal) au second terme.

Donc le rang empirique moyen de $\mathcal{L}'$ est inférieur ou égal au rang moyen empirique de $\mathcal{L}$. □

Exemple 5.3.1. Soient deux attributs P et V avec pour domaines respectifs $\{viande, poisson, soupe\}$ et $\{vin\ rouge, vin\ blanc\}$. Supposons que l'ensemble $\mathcal{H}$ contienne 45 objets distribués de la manière suivante :

viande/vin rouge	*viande/vin blanc*	*poisson/vin rouge*
10 items	9 items	8 items

poisson/vin blanc	*soupe/vin rouge*	*soupe/vin blanc*
7 items	6 items	5 items

Supposons qu'on ait choisit P comme étiquette de la racine du LP-tree appris. La CPT de la racine doit contenir l'ordre *viande* > *poisson* >

soupe sur le domaine $\underline{P}$, car *viande* a 19 occurrences dans $\mathcal{H}$, *poisson* a 15 occurrences et *soupe*, 11 occurrences.

5.3.2 Heuristique d'étiquetage de nœuds

Afin de décider quel ensemble d'attributs devrait étiqueter N, nous devons analyser les interactions entre les différents ensembles candidats.

Rappelons que, dans un k-LP-tree, un attribut ne peut apparaître qu'une fois par branche. C'est pour cela que nous chercherons les attributs qui vont étiqueter N parmi les attributs de $\mathcal{X} \setminus \mathbf{Anc}(N)$, car les attributs de $\mathbf{Anc}(N)$ sont ceux qui apparaissent déjà dans la branche.

Choix entre deux ensembles disjoints

Pour illustrer notre heuristique d'étiquetage, considérons deux ensembles disjoints d'attributs $\mathbf{X}, \mathbf{Y}$ tels que $\mathbf{X} \cap \mathbf{Anc}(N) = \varnothing$ et $\mathbf{Y} \cap \mathbf{Anc}(N) = \varnothing$ et supposons que $\mathbf{X}$ soit plus important que $\mathbf{Y}$ conditionnellement à $\mathbf{inst}(N)$. Par souci de concision, notons $\mathbf{n} = \mathbf{inst}(N)$.

Pour chaque tuple de valeurs $\mathbf{x}' \in \underline{\mathbf{X}}$, $\mathbf{x}' \neq \mathbf{x}_\mathbf{n}^{(1)}$, et $\mathbf{y}' \in \underline{\mathbf{Y}}$, $\mathbf{y}' \neq \mathbf{y}_\mathbf{n}^{(1)}$, chaque alternative qui étend $\mathbf{n}\mathbf{x}_\mathbf{n}^{(1)}\mathbf{y}'$ est préféré à chaque alternative qui étend $\mathbf{n}\mathbf{x}'\mathbf{y}_\mathbf{n}^{(1)}$. Autrement dit, pour tout $\mathbf{u} \in \underline{\mathcal{X} \setminus \mathbf{X} \cup \mathbf{Y} \cup \mathbf{Anc}(N)}$, $\mathbf{u}\mathbf{n}\mathbf{x}_\mathbf{n}^{(1)}\mathbf{y}' \succ \mathbf{u}\mathbf{n}\mathbf{x}'\mathbf{y}_\mathbf{n}^{(1)}$. Donc, si p est strictement croissante par rapport à $\succ$, $p(\mathbf{u}\mathbf{n}\mathbf{x}_\mathbf{n}^{(1)}\mathbf{y}') > p(\mathbf{u}\mathbf{n}\mathbf{x}'\mathbf{y}_\mathbf{n}^{(1)})$.

Comme cela est vrai pour tout $\mathbf{u}$, on peut en déduire que :

$$\begin{aligned} p(\mathbf{n}\mathbf{x}_\mathbf{n}^{(1)}\mathbf{y}') &= \sum_{\mathbf{u}} p(\mathbf{u}\mathbf{n}\mathbf{x}_\mathbf{n}^{(1)}\mathbf{y}') \\ &> \sum_{\mathbf{u}} p(\mathbf{u}\mathbf{n}\mathbf{x}'\mathbf{y}_\mathbf{n}^{(1)}) \\ &= p(\mathbf{n}\mathbf{x}'\mathbf{y}_\mathbf{n}^{(1)}) \end{aligned}$$

De plus, si on prend la moyenne de ces probabilités sur les valeurs $\mathbf{x}' \in \underline{\mathbf{X}}$, $\mathbf{x}' \neq \mathbf{x}_\mathbf{n}^{(1)}$ et $\mathbf{y}' \in \underline{\mathbf{Y}}$, $\mathbf{y}' \neq \mathbf{y}_\mathbf{n}^{(1)}$, nous obtenons à la proposition suivante :

Proposition 5.3.2. *Soient $\mathbf{X}, \mathbf{Y}$ deux sous-ensembles disjoints de $\mathcal{X}$, $\mathbf{I} \subseteq \mathcal{X} \setminus (X \cup Y)$ et $\mathbf{n} \in \underline{\mathbf{I}}$ tels que $\mathbf{X}$ soit plus important que $\mathbf{Y}$ conditionnellement à $\mathbf{n}$ dans $\succ$. Soit p une distribution de probabilité strictement croissante par rapport à $\succ$. Alors :*

$$\frac{\sum\limits_{\mathbf{y}' \in \underline{\mathbf{Y}} \setminus \{\mathbf{y}_\mathbf{n}^{(1)}\}} p(\mathbf{n}\mathbf{x}_\mathbf{n}^{(1)}\mathbf{y}')}{|\underline{\mathbf{Y}}| - 1} > \frac{\sum\limits_{\mathbf{x}' \in \underline{\mathbf{X}} \setminus \{\mathbf{x}_\mathbf{n}^{(1)}\}} p(\mathbf{n}\mathbf{x}'\mathbf{y}_\mathbf{n}^{(1)})}{|\underline{\mathbf{X}}| - 1}$$

Démonstration. Cf l'explication dans le texte précédent la proposition. □

Comme nous n'avons pas accès à la loi de probabilité p, nous estimons la probabilité $p(\mathbf{u})$ (pour un tuple u quelconque) à partir de la proportion d'occurrences de $\mathbf{u}$ dans $\mathcal{H}$, i.e. $\frac{|\mathcal{H}(\mathbf{u})|}{|\mathcal{H}|}$. Si $\mathcal{H}$ est assez représentatif de la loi de probabilité p, alors on peut réécrire la proposition précédente de la manière

suivante. Si $\mathbf{X}$ et $\mathbf{Y}$ sont deux ensembles disjoints de $\mathcal{X}$ tels que :

$$\frac{\sum_{\mathbf{y}' \in \underline{\mathbf{Y}} \setminus \{\mathbf{y}_\mathbf{n}^{(1)}\}} |\mathcal{H}(\mathbf{n}\mathbf{x}_\mathbf{n}^{(1)}\mathbf{y}')|}{|\mathcal{H}|(|\underline{\mathbf{Y}}| - 1)} > \frac{\sum_{\mathbf{x}' \in \underline{\mathbf{X}} \setminus \{\mathbf{x}_\mathbf{n}^{(1)}\}} |\mathcal{H}(\mathbf{n}\mathbf{x}'\mathbf{y}_\mathbf{n}^{(1)})|}{|\mathcal{H}|(|\underline{\mathbf{X}}| - 1)}$$

alors $\mathbf{Y}$ ne doit pas étiqueter N. C'est pour cela que cette comparaison nous servira d'heuristique.

Exemple 5.3.2. Considérons une fois de plus l'ensemble $\mathcal{H}$ de l'exemple 5.3.1 et cherchons l'étiquette de la racine.

$$\begin{aligned}\frac{|\mathcal{H}(\textit{viande/vin blanc})|}{|\underline{V}| - 1} &= \frac{9}{1} \\ &> \frac{8+6}{2} \\ &= \frac{|\mathcal{H}(\textit{poisson/vin rouge})| + |\mathcal{H}(\textit{soupe/vin rouge})|}{|\underline{P}| - 1}\end{aligned}$$

Ainsi, A va être choisi pour étiqueter la racine. Le LP-tree résultant va ordonner correctement les objets en fonction de leur nombre d'occurrences dans $\mathcal{H}$.

Choix entre deux ensembles non inclus l'un dans l'autre

Cette approche ne nous permet que de comparer deux ensembles d'attributs disjoints. Bien que cela nous permettre d'étiqueter les nœuds de 1-LP-trees (où les ensembles qui peuvent étiqueter les nœuds ne comportent qu'un seul élément, et sont donc nécessairement disjoints deux à deux), il nous faudra gérer le cas où deux ensembles d'attributs candidats ne sont pas disjoints pour apprendre des k-LP-trees avec $k > 1$.

Considérons à présent deux ensembles d'attributs $\mathbf{X}$ et $\mathbf{Y}$ qui ne contiennent aucun attribut de $\mathbf{Anc}(N)$ et qui ne sont pas inclus l'un dans l'autre (ils ne sont pas nécessairement disjoints). Supposons que $\mathbf{X}$ soit plus important que chacune des autres variables de $\mathcal{X} \setminus \mathbf{Anc}(N)$ conditionnellement à $\mathbf{n} = \mathbf{inst}(N)$.

Soit $\mathbf{U} = \mathbf{X} \setminus \mathbf{Y}$ l'ensemble des attributs qui sont dans $\mathbf{X}$ mais pas dans $\mathbf{Y}$ et soit $\mathbf{V} = \mathbf{Y} \setminus \mathbf{X}$, l'inverse. Pour chaque $\mathbf{x}' \in \underline{\mathbf{X}} \setminus \{\mathbf{x}_\mathbf{n}^{(1)}\}$, $\mathbf{v}', \mathbf{v}'' \in \underline{\mathbf{V}}$, chaque alternative qui étend $\mathbf{n}\mathbf{x}_\mathbf{n}^{(1)}\mathbf{v}'$ sera préférée à toute alternative qui étend $\mathbf{n}\mathbf{x}'\mathbf{v}''$. Choisissons $\mathbf{v}'' = \mathbf{y}_\mathbf{n}^{(1)}[\mathbf{V}]$. Alors, pour chaque instanciation $\mathbf{u}'$ de $\mathbf{U}$ qui n'est pas compatible avec $\mathbf{x}_\mathbf{n}^{(1)}$, si on note $\mathbf{x}' = \mathbf{u}'\mathbf{y}_\mathbf{n}^{(1)}[\mathbf{X}]$, alors les alternatives qui étendent $\mathbf{n}\mathbf{x}_\mathbf{n}^{(1)}\mathbf{v}'$ sont préférées à celles qui étendent $\mathbf{n}\mathbf{x}'\mathbf{v}'' = \mathbf{n}\mathbf{u}'\mathbf{y}_\mathbf{n}^{(1)}$.

En appliquant le même raisonnement que précédemment, nous obtenons la proposition suivante :

Proposition 5.3.3. *Soient $\mathbf{X}, \mathbf{Y}$ deux sous-ensembles de $\mathcal{X}$, $\mathbf{A} \subseteq \mathcal{X} \setminus (X \cup Y)$, $\mathbf{I} \subseteq \mathbf{A}$ et $\mathbf{n} \in \underline{\mathbf{I}}$ tels que $\mathbf{X}$ soit plus important que $\mathcal{X} \setminus \mathbf{A}$ conditionnellement à $\mathbf{n}$ dans $\succ$. Soit p une distribution de probabilité strictement croissante par rapport à $\succ$. Alors :*

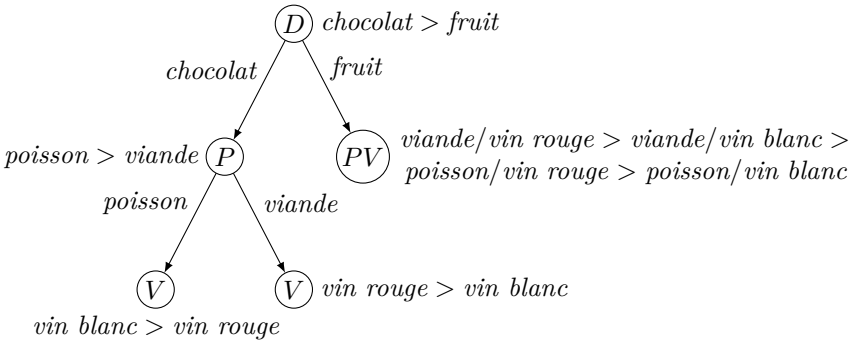
$$\frac{\sum\limits_{\mathbf{v}' \in \underline{\mathbf{V}} \setminus \{y_{\mathbf{n}}^{(1)}[\mathbf{V}]\}} p(\mathbf{n}\mathbf{x}_{\mathbf{n}}^{(1)}\mathbf{v}')}{|\underline{\mathbf{V}}| - 1} > \frac{\sum\limits_{\mathbf{u}' \in \underline{\mathbf{U}} \setminus \{x_{\mathbf{n}}^{(1)}[\mathbf{U}]\}} p(\mathbf{n}\mathbf{u}'\mathbf{y}_{\mathbf{n}}^{(1)})}{|\underline{\mathbf{U}}| - 1} \tag{5.3}$$

Démonstration. Cf l'explication dans le texte précédent la proposition. □

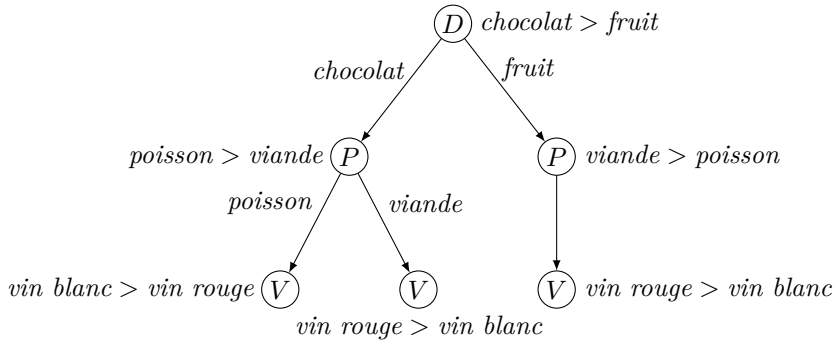
Choix entre deux ensembles inclus

Nous avons vu comment choisir entre soit deux ensembles disjoints, soit deux ensembles dont l'intersection et la différence sont non vides. Il reste encore la troisième possibilité : choisir entre deux ensembles dont l'un est strictement inclus dans l'autre. Néanmoins, contrairement aux cas précédents, il est possible que les deux ensembles soient des étiquettes valides. En effet, étant donné que ce sont des ensembles d'attributs qui étiquettent les nœuds des k-LP-trees, il existe des k-LP-trees différents qui représentent la même relation, comme le montre l'exemple suivante.

Exemple 5.3.3. Prenons le 2-LP-tree suivant.



Son nœud étiqueté par $\{P, V\}$ peut-être remplacé de la manière suivante :



Ce second LP-tree représente la même relation de préférence que précédemment.

Nous ne souhaitons pas apprendre de k-LP-trees dont les étiquettes des nœuds soient plus grandes que nécessaire. Nous proposons une manière de reconnaître ce cas lorsqu'il se présente.

Dans l'exemple précédent, nous avons vu que l'étiquette $\{P, V\}$ pourrait être remplacée par l'étiquette P. Pour formaliser cette approche, nous introduisons la notion de *décomposabilité* : dans cet exemple, P décompose $\{P, V\}$ car il existe un k-LP-tree étiqueté par P représentant la même relation de préférence que celui étiqueté par $\{P, V\}$.

Définition 5.3.1 (Décomposabilité d'un ensemble d'attributs)**.** *Soient* $\mathbf{X} \supset \mathbf{Y}$ *deux sous-ensembles de* $\mathcal{X}$*. Soit* $>_{\mathbf{X}}$ *une relation de préférence sur* $\underline{\mathbf{X}}$*. On dit que* $\mathbf{Y}$ *décompose* $\mathbf{X}$ *pour* $>_{\mathbf{X}}$ *si et seulement s'il existe un ordre* $>_{\mathbf{Y}}$ *sur* $\underline{\mathbf{Y}}$ *tel que la fonction* $\mathbf{x} \mapsto \mathbf{x}[\mathbf{Y}]$ *soit croissante par rapport à* $>_{\mathbf{X}}$ *(pour le domaine) et* $>_{\mathbf{Y}}$ *(pour le codomaine), i.e. :*

$$\mathbf{x} = \mathbf{x}' \text{ ou } \mathbf{x} \succ_{\mathbf{X}} \mathbf{x}' \Rightarrow \mathbf{x}[\mathbf{Y}] = \mathbf{x}'[\mathbf{Y}] \text{ ou } \mathbf{x}[\mathbf{Y}] \succ_{\mathbf{Y}} \mathbf{x}'[\mathbf{Y}]$$

Un algorithme qui permet de vérifier la décomposabilité d'un ensemble par un autre est présenté à l'algorithme 7 ; il est une application directe de la définition de la décomposabilité.

Algorithm 7: Détermine si un ensemble en décompose un autre

Input: $\mathbf{Y}$ et $\mathbf{X}$ deux ensembles d'attributs tels que $\mathbf{Y} \subset \mathbf{X}$, $\succ_{\mathbf{X}}$ une relation d'ordre sur $\underline{\mathbf{X}}$

Output: un ordre $\succ_{\mathbf{Y}}$ selon lequel $\mathbf{Y}$ décompose $\mathbf{X}$ pour $\succ_{\mathbf{X}}$, nil si un tel ordre n'existe pas

Algorithm `IsDecomposable`$(\mathbf{Y}, \mathbf{X}, \succ_{\mathbf{X}})$

```
1  listeVus ← ∅
2  values ← liste des valeurs de X ordonnées selon ≻_X
3  last ← nil
4  for each x ∈ values do
5      if x[Y] ∈ listeVus et x[Y] ≠ last then return nil
6      if x[Y] ∉ listeVus then listeVus ← listeVus ∪ {x[Y]}
7      last ← x[Y]
8  return l'ordre ≻_Y des objets dans listeVus
```

Exemple 5.3.4. Considérons une fois de plus les attributs de l'exemple 5.3.1 et supposons que $>$ ordonne $\underline{\{P, V\}}$ de la manière suivante :

$$\begin{aligned} &\textit{vin rouge/viande} > \textit{vin rouge/soupe} > \textit{vin rouge/poisson} \\ &> \textit{vin blanc/poisson} > \textit{vin blanc/soupe} > \textit{vin blanc/viande} \end{aligned}$$

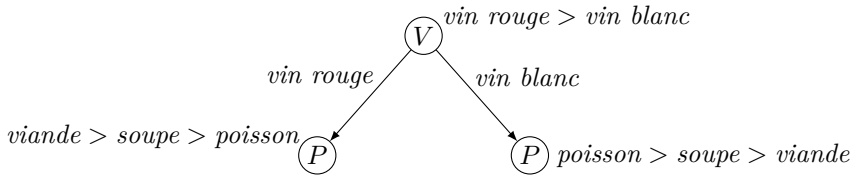
Un 2-LP-tree trivial qui représente cette relation est :

$$\begin{aligned} &\textit{vin rouge/viande} > \textit{vin rouge/soupe} \\ &> \textit{vin rouge/poisson} > \textit{vin blanc/poisson} \\ (PV) \quad &> \textit{vin blanc/soupe} > \textit{vin blanc/viande} \end{aligned}$$

Néanmoins, cette relation de préférence pourrait être représentée avec

un 1-LP-tree : en effet, on peut remarquer les plats avec vin rouge sont toujours préférés aux plats avec vin blanc.

Pour mettre cela en évidence, la définition précédente propose de regarder la projection de cet ordre total sur V, qui nous donne ici la succession suivante de valeurs : *vin rouge*, *vin rouge*, *vin rouge*, *vin blanc*, *vin blanc*, *vin blanc*. C'est une suite croissante si on considère la relation *vin rouge* $>_V$ *vin blanc*. Cela signifie qu'on peut réécrire le 2-LP-tree de la figure précédente en un 1-LP-tree (présenté à la figure suivante), dont la racine est étiquetée par l'attribut V et dont la CPT a pour règle *vin rouge* $>$ *vin blanc*, c'est-à-dire l'ordre trouvé précédemment.



Cette notion de décomposabilité est directement reliée à la notion d'importance, comme le montre la propriété suivante :

Proposition 5.3.4. *Soient* $\mathbf{X}, \mathbf{Y}$ *deux sous-ensembles disjoints de* $\mathcal{X}$*. Soit* $>_{\mathbf{XY}}$ *une relation de préférence sur* $\underline{\mathbf{X} \cup \mathbf{Y}}$*.*

Alors $\mathbf{X}$ *décompose* $\mathbf{X} \cup \mathbf{Y}$ *pour* $>_{\mathbf{XY}}$ *si et seulement si* $\mathbf{X}$ *est plus important que* $\mathbf{Y}$ *pour* $>_{\mathbf{X}}$*.*

Démonstration. D'une part, $\mathbf{X}$ est plus important que $\mathbf{Y}$ si et seulement s'il existe une relation d'ordre totale $\succ_{\mathbf{X}}$ définie sur $\underline{\mathbf{X}}$ telle que pour tout $\mathbf{x}, \mathbf{x}' \in \underline{\mathbf{X}}$, $\mathbf{y}, \mathbf{y}' \in \underline{\mathbf{Y}}$:

$$\mathbf{x} \succ_{\mathbf{X}} \mathbf{x}' \Rightarrow \mathbf{xy} \succ \mathbf{x}'\mathbf{y}'$$

D'autre $\mathbf{X}$ un ensemble d'attributs qui décompose $\mathbf{X} \cup \mathbf{Y}$ si et seulement si la projection de $\succ$ sur $\mathbf{X}$ est croissante par rapport à $\succ$, i.e.

$$\mathbf{xy} \succ \mathbf{x}'\mathbf{y}' \text{ ou } \mathbf{xy} = \mathbf{x}'\mathbf{y}' \Rightarrow \mathbf{x} = \mathbf{x}' \text{ ou } \mathbf{x} \succ_{\mathbf{X}} \mathbf{x}'$$

Or, étant donné que pour un ordre total $\succ$, $a \not\succ b$ est équivalent à $b \succ a$ ou $a = b$, ces deux propriétés sont la contraposée l'une de l'autre, et sont donc équivalentes. □

Nous pouvons à présent introduire les notions de nœud et de k-LP-tree décomposables.

Définition 5.3.2 (Nœud décomposable d'un k-LP-tree)**.** *Soit* N *un nœud d'un* k*-LP-tree et notons* $\mathbf{X}$ *son étiquette. Alors* N *est dit décomposable s'il existe un ensemble* $\mathbf{Y} \subset \mathbf{X}$ *tel que, pour tout règle* $\mathbf{u} :\succ_X$ *de* CPT_N*,* $\mathbf{Y}$ *décompose* $\mathbf{X}$ *pour* $\succ_X$*.*

Définition 5.3.3 (k-LP-tree décomposable)**.** *Un* k*-LP-tree est dit décomposable s'il possède au moins un nœud décomposable.*

Proposition 5.3.5. *Soit un* k*-LP-tree représentant une relation d'ordre* $\succ$*. Alors il existe un* k*-LP-tree non-décomposable qui représente la même relation d'ordre* $\succ$*.*

Démonstration. Il s'agit d'une preuve constructive : l'algorithme 8 permet de transformer un k-LP-tree décomposable en un LP-tree non-décomposable qui représente la même relation d'ordre. Pour savoir si un nœud est décomposable, on peut itérer sur tous les sous-ensembles de son étiquette et appliquer l'algorithme 7. Si un nœud étiqueté par $\mathbf{X}$ est décomposable par un ensemble $\mathbf{Y}$, on remplace ce nœud par un nœud étiqueté par $\mathbf{X}$ et on ajoute un ou plusieurs enfants étiquetés par $\mathbf{X} \setminus \mathbf{Y}$. Les règles de ces nœuds sont complétées de manière à représenter un ordre identique à celui du nœud décomposé. □

Algorithm 8: Transforme un k-LP-tree en un k-LP-tree non-décomposable qui représente la même relation de préférence.

Input: $\mathcal{L}$ un k-LP-tree
Output: Un k-LP-tree non-décomposable qui représente le même ordre total que $\mathcal{L}$
Algorithm `DecomposeLPTree(`$\mathcal{L}$`)`

```
1  L' un k-LP-tree identique à L
2  N ← l'ensemble des nœuds décomposables de L
3  for each N ∈ N do
4      X ← l'étiquette de N
5      Y ← un ensemble qui décompose X
6      N' ← un nouveau nœud vide étiqueté par Y
7      for each u :≻_X ∈ CPT_N do
8          ≻_Y ← l'ordre selon lequel Y décompose X selon ≻_X
9          ajoute à CPT_N' la règle u :≻_Y
10     if N' est décomposable then N ← N ∪ {N'}
11     Z ← X \ Y
12     if N a plusieurs enfants then
13         for each y ∈ Y do
14             N_y ← un nouveau nœud étiqueté par Z
15             ajoute un arc depuis N' vers N_y étiqueté par y
16             for each u :≻_X ∈ CPT_N do
17                 ≻_Z défini tel que ∀z, z' ∈ Z, z ≻_Z z' ssi zy ≻_X z'y
18                 ajoute à CPT_N_y la règle uy :≻_Z
19             if N_y est décomposable then N ← N ∪ {N_y}
20         remplace dans L' le nœud N par les nœuds N' et N_y
21     else
22         N'' ← un nouveau nœud étiqueté par Z
23         ajoute un arc non étiqueté depuis N' vers N''
24         for each y ∈ Y do
25             for each u :≻_X ∈ CPT_N do
26                 ≻_Z défini tel que ∀z, z' ∈ Z, z ≻_Z z' ssi zy ≻_X z'y
27                 ajoute à CPT_N'' la règle uy :≻_Z
28         if N'' est décomposable then N ← N ∪ {N''}
29         remplace dans L' le nœud N par les nœuds N' et N''
30 return L'
```

Nous cherchons à apprendre un k-LP-tree non-décomposable. De ce fait, pour choisir entre deux ensembles $\mathbf{Y}$ et $\mathbf{X}$ tels que $\mathbf{Y} \subset \mathbf{X}$ et que $\mathbf{Y}$ décompose $\mathbf{X}$, on rejettera l'ensemble $\mathbf{X}$ pour le remplacer par $\mathbf{Y}$ (on ne cherche que des ensembles non-décomposables).

De la même manière, si $\mathbf{Y} \supset \mathbf{X}$ et que $\mathbf{X}$ *ne décompose pas* $\mathbf{Y}$, alors $\mathbf{X}$ est rejeté en faveur de $\mathbf{Y}$. En effet, si $\mathbf{X}$ était l'étiquette de N, alors les autres attributs $\mathbf{Y} \setminus \mathbf{X}$ devraient être moins important que $\mathbf{X}$ et donc $\mathbf{X}$ devrait décomposer $\mathbf{Y}$.

En résumé : l'heuristique d'étiquetage

Nous avons étudié toutes les possibilités pour reconnaître l'étiquette d'un nœud. Nous pouvons à présent tout récapituler et présenter notre heuristique d'étiquetage pour apprendre un k-LP-tree.

L'algorithme 9 énumère l'ensemble de tous les sous-ensembles de $\mathcal{X} \setminus \mathbf{Anc}(N)$ de cardinalité inférieure ou égale à k pour k fixé (k est le nombre maximal d'attributs qui peuvent étiqueter un nœud). L'ordre dans lequel ces ensembles sont énumérés est quelconque. À chaque étape, l'algorithme a de côté le meilleur candidat courant. Dès que l'algorithme rencontre un ensemble $\mathbf{Y}$ qui invalide le meilleur candidat courant, $\mathbf{Y}$ prend la place de ce candidat (autrement dit, s'il s'agit d'un algorithme de recherche de maximum).

Un ensemble $\mathbf{Y}$ invalide un ensemble $\mathbf{X}$ si :

- $\mathbf{Y} \supset \mathbf{X}$ et $\mathbf{X}$ ne décompose pas $\mathbf{Y}$; ou
- $\mathbf{Y} \subset \mathbf{X}$ et $\mathbf{Y}$ décompose $\mathbf{X}$; ou
- aucune de ces deux inclusions n'est vérifiée, et l'inégalité 5.3 n'est pas non plus vérifiée.

Enfin, la table de préférence conditionnelle d'un nœud N est construite de manière à ce qu'une valeur $\mathbf{x}$ soit préférée à une valeur $\mathbf{x}'$ si $\mathbf{x}$ est plus souvent observé que $\mathbf{x}'$ dans $\mathcal{H}(\mathbf{n})$, comme le préconise la proposition 5.3.1. En cas d'égalité dans le nombre d'occurrences de deux valeurs, l'ordre entre ces deux valeurs est choisi arbitrairement.

Proposition 5.3.6. *Soit un k-LP-tree $\breve{\mathcal{L}}$. Soit $\mathcal{H}$ un ensemble d'objets tel que la distribution empirique $p_{\mathcal{H}}$, définie par $p_{\mathcal{H}}(\mathbf{o}) = |\mathcal{H}(\mathbf{o})|/|\mathcal{H}|$, soit strictement croissante par rapport à $\succ_{\breve{\mathcal{L}}}$. Si l'algorithme 6 utilise l'algorithme 9 pour* chooseAttributes*, et si* generateLabels *renvoie toujours $\underline{\mathbf{X}}$ quand elle est appelée pour l'étiquette $\mathbf{X}$, alors le k-LP-tree appris représentera exactement $\succ_{\breve{\mathcal{L}}}$.*

Démonstration. Nous allons nous intéresser à la recherche de l'étiquette de la racine ; le même raisonnement s'applique aux autres nœuds.

Premièrement, on peut remarquer que parmi les ensembles d'attributs de taille identique, il y a uniquement un ensemble $\mathbf{Y}$ qui invalide tous les autres ensembles. En effet, le seul cas où la proposition 5.3.3 ne peut pas comparer deux ensembles d'attributs $\mathbf{U}$ et $\mathbf{V}$ est quand $\mathbf{U} \subseteq \mathbf{V}$ ou $\mathbf{V} \subseteq \mathbf{U}$. Or, parmi les ensembles d'attributs de même taille, aucun ensemble n'est strictement inclus dans un autre.

Soit $\mathbf{X}$ l'ensemble des attributs qui étiquettent la racine de $\breve{\mathcal{L}}$. Supposons sans perdre de généralité (comme cela fut prouvé à la proposition 5.3.5) que le

Algorithm 9: Choisit un ensemble d'attributs et sa table de préférence

Input: $\mathcal{X}$, un nœud N, un ensemble d'objets $\mathcal{H}$ sur $\mathcal{X}$, k le nombre maximal d'attributs par étiquette
Output: Un ensemble d'attributs $\mathbf{X}$ et une table de préférence T
Algorithm `ChooseAttributes`$(\mathcal{X}, N, \mathcal{H}, k)$

$\mathcal{G}_k \leftarrow$ l'ensemble des ensembles d'attributs de cardinalité au plus k
$\mathbf{n} \leftarrow \mathbf{inst}(N)$
$\mathbf{X} \leftarrow$ n'importe quel ensemble de $\mathcal{G}_k$
for each $\mathbf{Y} \in \mathcal{G}_k \setminus \{\mathbf{X}\}$ **do**
  if $\mathbf{Y}$ *invalide* $\mathbf{X}$ **then**
    $\mathbf{X} \leftarrow \mathbf{Y}$
$> \leftarrow$ ordonne les valeurs de $\underline{\mathbf{X}}$ par nombre d'occurrences dans $\mathcal{H}(\mathbf{n})$ décroissant
return $\mathbf{X}, >$

k-LP-tree caché soit non-décomposable ; de ce fait, $\mathbf{X}$ n'est pas décomposable. Pour chaque $i \in [1, k]$, notons $\mathbf{Y}_i$ l'ensemble de i attributs qui invalide tous les autres ensembles de i attributs. D'après la proposition 5.3.3, l'étiquette de la racine est l'un de ces $\mathbf{Y}_i$, i.e. $\mathbf{X} \in \{\mathbf{Y}_i\}_i$. Soient $i, j \in [1, k], i < j$ et vérifions de $\mathbf{Y}_i$ et $\mathbf{Y}_j$ quel ensemble invalide l'autre. Il y a deux possibilités.

Dans le premier cas, supposons que $\mathbf{Y}_i \not\subset \mathbf{Y}_j$. On peut dans ce cas utiliser l'inégalité de la proposition 5.3.3 pour déterminer quel ensemble n'étiquette pas la racine.

Second cas, supposons que $\mathbf{Y}_i \subset \mathbf{Y}_j$. On ne peut pas comparer $\mathbf{Y}_i$ et $\mathbf{Y}_j$ avec la proposition 5.3.3. On vérifier alors si $\mathbf{Y}_i$ décompose $\mathbf{Y}_j$ ou non. Si $\mathbf{Y}_i$ décompose $\mathbf{Y}_j$, alors $\mathbf{Y}_j$ est invalidé car on souhaite apprendre le plus petit ensemble décomposable. Sinon, $\mathbf{Y}_i$ est invalidé car d'après la proposition 8, ce ne peut pas être la racine. □

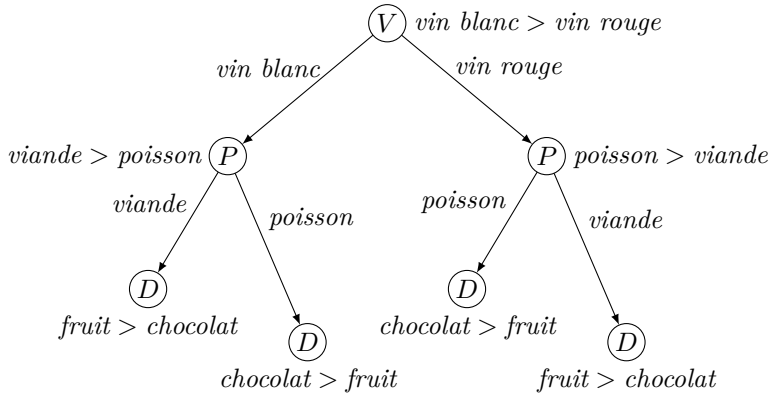
Non-optimalité de l'heuristique

Cette heuristique d'étiquetage, bien qu'elle nous assure la convergence vers l'ordre total cible, ne minimise pas forcément le rang moyen empirique, ce que montre l'exemple suivante.

Exemple 5.3.5. Soit $\mathcal{H}$ le jeu d'exemples positifs suivant.

viande/vin rouge/fruit	*viande/vin rouge/chocolat*
5 items	1 item
viande/vin blanc/fruit	*viande/vin blanc/chocolat*
11 items	1 item
poisson/vin rouge/fruit	*poisson/vin rouge/chocolat*
1 item	6 items
poisson/vin blanc/fruit	*poisson/vin blanc/chocolat*
1 item	6 items

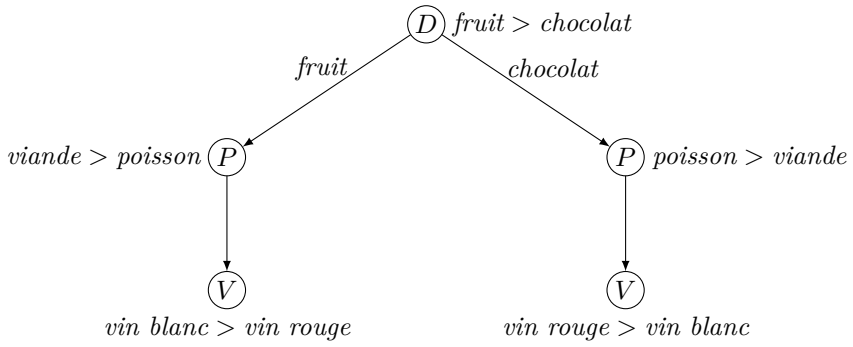
L'algorithme 6 apprend le LP-tree présenté à la figure suivante.



L'ordre total qu'il représente est :

$$viande/vin\ blanc/fruit \succ viande/vin\ blanc/chocolat \succ poisson/vin\ blanc/chocolat \succ poisson/vin\ blanc/fruit \succ poisson/vin\ rouge/chocolat \succ poisson/vin\ rouge/fruit \succ viande/vin\ rouge/fruit \succ viande/vin\ rouge/chocolat$$

Le rang moyen empirique de $\mathcal{H}$ selon ce LP-tree est 3.5625.
Néanmoins, si on considère le LP-tree présenté à la figure suivante :



L'ordre total qu'il représente est :

$$viande/vin\ blanc/fruit \succ viande/vin\ rouge/fruit \succ poisson/vin\ blanc/fruit \succ poisson/vin\ rouge/fruit \succ poisson/vin\ rouge/chocolat \succ poisson/vin\ blanc/chocolat \succ viande/vin\ rouge/chocolat \succ viande/vin\ blanc/chocolat$$

Le rang moyen empirique de $\mathcal{H}$ selon ce second LP-tree est 3.40625, ce qui est inférieur au rang moyen empirique selon le premier LP-tree.

Ce qui prouve que le LP-tree appris par l'algorithme 6 ne minimise pas nécessairement le rang moyen empirique.

Nous étudierons l'apprentissage optimal d'arbre de préférence lexicographique linéaire dans la section 5.4.

Résumé des résultats de convergence

Nous avons jusque-là présenté deux preuves de convergence (propositions 5.2.3 et 5.3.6) ainsi que le contre-exemple 5.3.5. Il peut donc être intéressant de faire un point sur ces différents résultats.

- La proposition 5.2.3 est générale : elle s'applique à tout langage fini. Elle montre qu'apprendre le modèle qui minimise le rang moyen empirique est suffisant pour que la *ranking loss* converge vers zéro quand le nombre d'exemples tend vers l'infini.
- Le contre-exemple 5.3.5 montre que notre algorithme d'apprentissage de k-LP-trees ne minimise pas le rang moyen empirique, ce qui fait que la proposition précédente ne peut pas être utilisée pour notre algorithme d'apprentissage.
- La proposition 5.3.6 montre en revanche que même si notre algorithme d'apprentissage ne minimise pas le rang moyen empirique, sa *ranking loss* converge bien vers zéro.

5.3.3 Heuristique de scission

Rappelons qu'un nœud d'un k-LP-tree étiqueté par $\mathbf{X}$ peut soit être une feuille, soit avoir $|\underline{\mathbf{X}}|$ arêtes étiquetées chacune par une valeur de $\mathbf{X}$, auquel cas les préférences de chaque sous-arbre peut dépendre de la valeur de $\mathbf{X}$, soit avoir une seule arête non-étiquetée, auquel cas les préférences de chaque sous-arbre ne dépendent pas de la valeur de $\mathbf{X}$.

Un k-LP-tree dont chaque nœud interne a plusieurs arêtes sortantes a une taille exponentielle en le nombre d'attributs. Apprendre un tel k-LP-tree n'est pas souhaitable pour plusieurs raisons ; notamment parce qu'avec un nombre restreint d'exemples ($|\mathcal{H}| \ll |\underline{\mathcal{X}}|$), il n'est pas raisonnable de chercher à apprendre un modèle avec un nombre exponentiel de paramètres. Pour cette raison, nous allons choisir pour chaque nœud s'il aura une seule arête sortante ou plusieurs en fonction du nombre d'exemples qu'aurait chaque sous-arbre, ce qui nous permettra de limiter la taille de l'arbre appris.

Plus exactement, notre démarche est d'imposer à toute branche du k-LP-tree d'être cohérent avec au moins τ exemples, τ étant une constante définie empiriquement. Cela signifie qu'à l'apprentissage d'un nœud N étiqueté par $\mathbf{X}$, N se verra attribué plusieurs arêtes sortantes si et seulement chacune de ces arêtes est associés à au moins τ exemples ; ou, plus formellement, si, pour chaque valeur $\mathbf{x} \in \underline{\mathbf{X}}$, $|\mathcal{H}(\mathbf{inst}(N), \mathbf{x})| \geq \tau$.

La fonction GenerateLabels, présentée à l'algorithme 10, implémente cette approche.

5.3.4 Complexité temporelle

Lorsqu'on utilise un seuil τ, le nombre de branches du k-LP-tree est limité car il doit y avoir au moins τ exemples qui correspondent à chaque branche. Il y a donc au plus $\lfloor \frac{|\mathcal{H}|}{\tau} \rfloor$ branches et $n \lfloor \frac{|\mathcal{H}|}{\tau} \rfloor$ nœuds.

Algorithm 10: Génère les étiquettes des arêtes sortantes d'un nœud

Input: $\mathcal{X}$, un nœud N, son étiquette $\mathbf{X}$, un ensemble d'alternatives $\mathcal{H}$
Output: un ensemble d'étiquettes
Algorithm `GenerateLabels(`$\mathcal{X}, N, \mathbf{X}, \mathcal{H}$`)`

```
    // Si X est une feuille
1   if Anc(N) ∪ X = 𝒳 then
2       return ∅
    // S'il y a assez d'exemples pour chaque sous-arbre
3   else if for each x ∈ X̲, |ℋ(x, inst(N))| ≥ τ then
4       return X̲
    // S'il n'y a pas assez d'exemples
5   else
6       return {étiquette vide}
```

Dans l'algorithme 9, les occurrences de chaque valeur de tous les ensembles d'au plus k attributs sont comptées et triées, ce qui peut être fait avec une complexité de $O(\binom{n}{k} k \log(d) d^k |\mathcal{H}|)$ (où d est la taille maximale des domaines des attributs de $\mathcal{X}$, autrement dit $d = \max_{X \in \mathcal{X}} |\underline{X}|$).

La vérification de la décomposabilité peut être vérifiée en $O(k \log(d) d^k |\mathcal{H}|)$. De plus, l'algorithme 9 est appelé pour chaque nœud. La complexité temporelle totale est donc $O(n\binom{n}{k} k^2 \log^2(d) d^{2k} |\mathcal{H}|^2 / \tau)$.

5.3.5 Élagage pour réduire le surapprentissage

L'algorithme 6 renvoie un arbre $\mathcal{L}$ qui minimise approximativement le rang moyen empirique de $\mathcal{H}$ – au sein des arbres acceptables, définis par le seuil choisi. Cet apprentissage peut amener à un surapprentissage, c'est-à-dire que le modèle appris s'ajuste trop aux données et a des capacités de généralisation réduites. Les modèles surappris ont souvent une taille trop grande ; c'est pour cela qu'une méthode standard pour modérer ce phénomène est d'introduire dans le score un terme qui va pénaliser les modèles qui ont une grande taille. On définit :

$$S_\phi(\mathcal{H}, \mathcal{L}) = -|\mathcal{H}| \cdot \overline{\mathsf{rank}}(\succ_{\mathcal{L}}, \mathcal{H}) - |\mathcal{L}| \cdot \phi(|\mathcal{H}|)$$

où $|\mathcal{L}|$ désigne la taille du modèle, ce qui sera dans le cas des LP-trees le nombre de ses nœuds, et ϕ est une fonction de pénalité (par exemple une fonction constante).

Une fois qu'un LP-tree est appris, il peut être élagué de manière à améliorer son score S_ϕ. Cette procédure est exécutée de manière *bottom-up*, c'est-à-dire depuis les feuilles vers la racine. Pour chaque nœud interne N de $\mathcal{L}$, on vérifie si ce nœud a plusieurs arêtes sortantes. Si c'est le cas, on redirige toutes ces arêtes vers l'enfant de N associée à la valeur préférée selon la CPT de N et on détruit les sous-arbres issus des autres enfants de N. Si le score de l'arbre résultant est amélioré, alors cet arbre est élagué. Sinon, on restaure les arêtes sortantes.

L'élagage est présenté à l'algorithme 11.

Algorithm 11: Élague un LP-tree

Input: $\mathcal{L}$ un LP-tree, $\mathcal{H}$ un ensemble d'exemples, ϕ une fonction de pénalité
Output: Le LP-tree élagué
Algorithm `Pruning`$(\mathcal{L}, \mathcal{H}, \phi)$

```
1  listeElagage ← les nœuds de L ordonnés selon le parcours postfixe
     de L (ordre bottom-up)
2  while listeElagage n'est pas vide do
3      retire le nœud N au début de listeElagage
4      if N a plusieurs arêtes sortantes then
5          L' ← L
6          redirige dans L les arêtes sortantes de N vers son fils préférée
7          if S_φ(H, L') > S_φ(H, L) then L ← L'
8  return L
```

5.4 Apprentissage optimal de LP-tree linéaire

L'algorithme d'apprentissage des k-LP-trees peut facilement être adapté à l'apprentissage de LP-tree linéaire ; en effet, il suffit pour cela que GenerateLabels retourne toujours une étiquette vide, auquel cas alors chaque nœud de l'arbre construit par l'algorithme 6 aura un seul enfant.

Lorsqu'on limite la recherche aux LP-trees linéaires, notre algorithme heuristique trouve le LP-tree linéaire qui minimise le rang moyen empirique si $\mathcal{X}$ est un ensemble de variables binaires.

Dans cette partie, nous proposons une autre fonction ChooseAttributes qui, dans le cadre de l'apprentissage d'arbre linéaire, nous permettra d'avoir un algorithme glouton qui apprend toujours le LP-tree linéaire qui minimise le rang moyen empirique, y compris lorsque $\mathcal{X}$ n'est pas un ensemble de variables binaires. Cette fonction ChooseAttributes s'appuie sur la proposition suivante.

Proposition 5.4.1. *Soit $\mathcal{H}$ un ensemble d'exemples positifs de $\underline{\mathcal{X}}$ et soit $\mathcal{L}^*$ le LP-tree linéaire qui minimise le rang moyen empirique de $\mathcal{H}$.*

Notons $S(X, \mathcal{H})$ le score de X pour $\mathcal{H}$ défini de la manière suivante :

$$S(X, \mathcal{H}) = \frac{1}{|\underline{X}| - 1} \sum_{l=1}^{|\underline{X}|-1} l|\mathcal{H}(x_X^{(l+1)})|$$

Alors, pour tout couple d'attributs (X, Y), si la profondeur de X dans $\mathcal{L}^*$ est inférieure à celle de Y, alors :

$$S(X, \mathcal{H}) \leq S(Y, \mathcal{H})$$

Démonstration. D'après le théorème 5.3.1, les tables de préférences du LP-tree linéaire qui minimise le rang moyen empirique de $\mathcal{H}$ ordonnent les valeurs d'une variable X par le nombre d'occurrences de ses valeurs $\underline{X}$ dans $\mathcal{H}$. Dans la suite de cette preuve, tous les LP-trees que nous manipulerons auront de telles tables.

Nous utiliserons la notation $[a = b]$ définie de la manière suivante :

$$[a = b] = \begin{cases} 1 \text{ si } a = b \\ 0 \text{ sinon} \end{cases}$$

En notant $X(\mathcal{L}, i)$ l'attribut à la profondeur i dans un LP-tree linéaire $\mathcal{L}$, le rang d'un élément $\mathbf{o}$ dans $\mathcal{L}$ peut s'écrire de la manière suivante :

$$\mathsf{rank}(\mathcal{L}, \mathbf{o}) = 1 + \sum_{i=1}^{n} \left(\left(\sum_{l=0}^{|\underline{X}(\mathcal{L},i)|-1} l \times [\mathbf{o}[X(\mathcal{L}, i)] = x_{X(\mathcal{L},i)}^{(l+1)}] \right) \prod_{j=i+1}^{n} |\underline{X}(\mathcal{L}, j)| \right)$$

Nous allons utiliser cette relation dans notre recherche du LP-tree linéaire optimal.

$$\begin{aligned}
&\operatorname*{argmin}_{\mathcal{L}} \overline{\mathsf{rank}}(\mathcal{L}, \mathcal{H}) = \operatorname*{argmin}_{\mathcal{L}} \sum_{\mathbf{o} \in \mathcal{H}} \mathsf{rank}(\mathcal{L}, \mathbf{o}) \\
&= \operatorname*{argmin}_{\mathcal{L}} \sum_{\mathbf{o} \in \mathcal{H}} 1 + \sum_{i=1}^{n} \left(\left(\sum_{l=0}^{|\underline{X}(\mathcal{L},i)|-1} l \times [\mathbf{o}[X(\mathcal{L}, i)] = x_{X(\mathcal{L},i)}^{(l+1)}] \right) \prod_{j=i+1}^{n} |\underline{X}(\mathcal{L}, j)| \right) \\
&= \operatorname*{argmin}_{\mathcal{L}} \sum_{i=1}^{n} \sum_{\mathbf{o} \in \mathcal{H}} \left(\left(\sum_{l=0}^{|\underline{X}(\mathcal{L},i)|-1} l \times [\mathbf{o}[X(\mathcal{L}, i)] = x_{X(\mathcal{L},i)}^{(l+1)}] \right) \prod_{j=i+1}^{n} |\underline{X}(\mathcal{L}, j)| \right) \\
&= \operatorname*{argmin}_{\mathcal{L}} \sum_{i=1}^{n} \left(\left(\sum_{l=0}^{|\underline{X}(\mathcal{L},i)|-1} l \times \sum_{\mathbf{o} \in \mathcal{H}} [\mathbf{o}[X(\mathcal{L}, i)] = x_{X(\mathcal{L},i)}^{(l+1)}] \right) \prod_{j=i+1}^{n} |\underline{X}(\mathcal{L}, j)| \right) \\
&= \operatorname*{argmin}_{\mathcal{L}} \sum_{i=1}^{n} \left(\left(\sum_{l=0}^{|\underline{X}(\mathcal{L},i)|-1} l \times |\mathcal{H}(x_{X(\mathcal{L},i)}^{(l+1)})| \right) \prod_{j=i+1}^{n} |\underline{X}(\mathcal{L}, j)| \right) \\
&= \operatorname*{argmin}_{\mathcal{L}} \sum_{i=1}^{n} \frac{\sum_{l=0}^{|\underline{X}(\mathcal{L},i)|-1} l \times |\mathcal{H}(x_{X(\mathcal{L},i)}^{(l+1)})|}{\prod_{j=1}^{i} |\underline{X}(\mathcal{L}, j)|} |\underline{\mathcal{X}}| \\
&= \operatorname*{argmin}_{\mathcal{L}} \sum_{i=1}^{n} \frac{\sum_{l=0}^{|\underline{X}(\mathcal{L},i)|-1} l \times |\mathcal{H}(x_{X(\mathcal{L},i)}^{(l+1)})|}{\prod_{j=1}^{i} |\underline{X}(\mathcal{L}, j)|}
\end{aligned}$$

Afin d'alléger la notation de la suite de la preuve, nous introduisons la notation suivante :

$$K(X) = \sum_{l=0}^{|\underline{X}|-1} l \times |\mathcal{H}(x_X^{(l+1)})|$$

Nous nous retrouvons donc avec l'égalité suivante :

$$\operatorname*{argmin}_{\mathcal{L}} \overline{\mathsf{rank}}(\mathcal{L}, \mathcal{H}) = \operatorname*{argmin}_{\mathcal{L}} \sum_{i=1}^{n} \frac{K(X(\mathcal{L}, i))}{\prod_{j=1}^{i-1} |\underline{X}(\mathcal{L}, j)|}$$

Soit $\mathcal{L}^*$ le LP-tree linéaire qui minimise le rang moyen empirique de $\mathcal{H}$. Soit $\mathcal{L}_k^*$ un LP-tree linéaire identique à $\mathcal{L}^*$ à l'exception des attributs aux profondeurs k et $k+1$ qui sont inversés. On peut donc remarquer que, par construction :

$$\begin{cases} X(\mathcal{L}^*, k) = X(\mathcal{L}^*_k, k+1) \\ X(\mathcal{L}^*, k+1) = X(\mathcal{L}^*_k, k) \\ X(\mathcal{L}^*, i) = X(\mathcal{L}^*_k, i) \text{ si } i \neq k \text{ et } i \neq k+1 \end{cases}$$

Étant donné que $\mathcal{L}^*$ minimise le rang empirique moyen :

$$\begin{aligned}
&\sum_{i=1}^{n} \frac{K(X(\mathcal{L}^*, i))}{\prod_{j=1}^{i} |\underline{X}(\mathcal{L}^*, j)|} \leq \sum_{i=1}^{n} \frac{K(X(\mathcal{L}^*_k, i))}{\prod_{j=1}^{i} |\underline{X}(\mathcal{L}^*_k, j)|} \\
&\Leftrightarrow \sum_{i=1}^{n} \left(\frac{K(X(\mathcal{L}^*, i))}{\prod_{j=1}^{i} |\underline{X}(\mathcal{L}^*, j)|} - \frac{K(X(\mathcal{L}^*_k, i))}{\prod_{j=1}^{i} |\underline{X}(\mathcal{L}^*_k, j)|} \right) \leq 0 \\
&\Leftrightarrow \sum_{i=k}^{k+1} \left(\frac{K(X(\mathcal{L}^*, i))}{\prod_{j=1}^{i} |\underline{X}(\mathcal{L}^*, j)|} - \frac{K(X(\mathcal{L}^*_k, i))}{\prod_{j=1}^{i} |\underline{X}(\mathcal{L}^*_k, j)|} \right) \leq 0 \\
&\Leftrightarrow \frac{K(X(\mathcal{L}^*, k))}{\prod_{j=1}^{k} |\underline{X}(\mathcal{L}^*, j)|} + \frac{K(X(\mathcal{L}^*, k+1))}{\prod_{j=1}^{k+1} |\underline{X}(\mathcal{L}^*, j)|} \\
&\qquad - \frac{K(X(\mathcal{L}^*_k, k))}{\prod_{j=1}^{k} |\underline{X}(\mathcal{L}^*_k, j)|} - \frac{K(X(\mathcal{L}^*_k, k+1))}{\prod_{j=1}^{k+1} |\underline{X}(\mathcal{L}^*_k, j)|} \leq 0 \\
&\Leftrightarrow \frac{K(X(\mathcal{L}^*, k))}{|\underline{X}(\mathcal{L}^*, k)|} + \frac{K(X(\mathcal{L}^*, k+1))}{|\underline{X}(\mathcal{L}^*, k)| \times |\underline{X}(\mathcal{L}^*, k+1)|} \\
&\qquad - \frac{K(X(\mathcal{L}^*_k, k))}{|\underline{X}(\mathcal{L}^*_k, k)|} - \frac{K(X(\mathcal{L}^*_k, k+1))}{|\underline{X}(\mathcal{L}^*_k, k)| \times |\underline{X}(\mathcal{L}^*_k, k+1)|} \leq 0 \\
&\Leftrightarrow \frac{K(X(\mathcal{L}^*, k))}{|\underline{X}(\mathcal{L}^*, k)|} + \frac{K(X(\mathcal{L}^*, k+1))}{|\underline{X}(\mathcal{L}^*, k)| \times |\underline{X}(\mathcal{L}^*, k+1)|} \\
&\qquad - \frac{K(X(\mathcal{L}^*, k+1))}{|\underline{X}(\mathcal{L}^*, k+1)|} - \frac{K(X(\mathcal{L}^*, k))}{|\underline{X}(\mathcal{L}^*, k+1)| \times |\underline{X}(\mathcal{L}^*, k)|} \leq 0 \\
&\Leftrightarrow K(X(\mathcal{L}^*, k))|\underline{X}(\mathcal{L}^*, k+1)| + K(X(\mathcal{L}^*, k+1)) \\
&\qquad - K(X(\mathcal{L}^*, k+1))|\underline{X}(\mathcal{L}^*, k)| - K(X(\mathcal{L}^*, k)) \leq 0 \\
&\Leftrightarrow K(X(\mathcal{L}^*, k))(|\underline{X}(\mathcal{L}^*, k+1)| - 1) \leq K(X(\mathcal{L}^*, k+1))(|\underline{X}(\mathcal{L}^*, k)| - 1) \\
&\Leftrightarrow \frac{K(X(\mathcal{L}^*, k))}{(|\underline{X}(\mathcal{L}^*, k)| - 1)} \leq \frac{K(X(\mathcal{L}^*, k+1))}{(|\underline{X}(\mathcal{L}^*, k+1)| - 1)}
\end{aligned}$$

Si on note $S(X, \mathcal{H}) = \frac{K(X)}{|\underline{X}|-1}$, alors on nous obtenons l'inégalité suivante :

$$S(X(\mathcal{L}^*, k), \mathcal{H}) \leq S(X(\mathcal{L}^*, k+1), \mathcal{H})$$

Comme cela est vrai pour tout k, par transitivité cette inégalité est vraie pour tout i, j tel que i soit plus important que j (i.e. $i < j$) dans $\mathcal{L}^*$:

$$i < j \Rightarrow S(X(\mathcal{L}^*, i), \mathcal{H}) \leq S(X(\mathcal{L}^*, j), \mathcal{H})$$

Ce qui achève la preuve. □

Cette proposition nous présente un score à partir duquel apprendre un LP-tree linéaire : il suffit d'ordonner les attributs par score croissant. Auquel cas, on sera assuré de trouver le LP-tree linéaire qui minimise le rang empirique moyen.

Algorithm 12: Apprend le LP-tree linéaire qui minimise le rang moyen empirique de $\mathcal{H}$

Input: $\mathcal{X}$, un ensemble d'objets $\mathcal{H}$ sur $\mathcal{X}$
Output: Le LP-tree linéaire qui minimise le rang moyen empirique de $\mathcal{H}$
Algorithm `LearnExactLinearLPTree`$(\mathcal{X}, N, \mathcal{H}, k)$

```
for each X ∈ 𝒳 do
    score(X) ← S(X, ℋ)
listeAttr ← les attributs de 𝒳 triés par score décroissant
ℒ ← un LP-tree linéaire vide
for each X ∈ listeAttr do
    N ← un nouveau nœud non étiqueté
    étiquette N par X
    CPT_N ← liste des valeurs de X ordonnées selon ≻_X
    if ℒ est vide then
        ℒ ← N
    else
        ajoute un arc non étiqueté depuis la feuille de ℒ vers N
return ℒ
```

Un algorithme implémentant cette proposition est présenté dans l'algorithme 12. La complexité temporelle de cet algorithme est $O(n|\mathcal{H}|d\log d)$, où n est le nombre de variable et d la plus grande cardinalité des attributs de $\mathcal{X}$. En effet, pour toutes les n variables, il faut compter les occurrences de ses d valeurs parmi $|\mathcal{H}|$ exemples et trier ces d valeurs.

5.5 *Sample complexity* des LP-trees linéaires binaires

L'apprenabilité et l'apprentissage PAC, que nous avons présentés dans le chapitre 3, ont été définis dans le cadre de la classification. Ces notions sont fondés sur le risque réel d'un modèle appris à partir d'un ensemble d'exemples. L'équivalent du risque réel est, dans notre cas, la mesure de la distance entre le modèle qu'on cherche à apprendre et le modèle appris : la *ranking loss*. La *ranking loss*, tout comme le risque réel, est comprise entre 0 et 1.

Proposition 5.5.1. *Soient $\mathcal{X}$ un ensemble d'attributs binaires et L l'ensemble des LP-trees linéaires sur $\mathcal{X}$, soient $\breve{\mathcal{L}} \in L$ le LP-tree linéaire cible, p une distribution de probabilité croissante par rapport à $\breve{\mathcal{L}}$ selon laquelle sont tirés les exemples de $\mathcal{H}$, ϵ et δ deux réels tels que $0 < \epsilon < 1/2$, $0 < \delta < 1/2$. L'algorithme 12 apprend un LP-tree linéaire $\mathcal{L} \in L$ à partir d'un ensemble d'exemples $\mathcal{H}$ qui vérifie la propriété suivante :*

$$|\mathcal{H}| \geq \frac{2}{\epsilon^2}\left(\log\frac{1}{\delta} + \log 2n\right) \Rightarrow \Pr(\mathsf{rloss}_p(\breve{\mathcal{L}}, \mathcal{L}) \leq \epsilon) \geq 1 - \delta$$

Démonstration. Soient $\breve{\mathcal{L}}$ le LP-tree caché qu'on souhaite apprendre, p une distribution de probabilité croissante par rapport à $\breve{\mathcal{L}}$, $\mathcal{H}$ un ensemble d'exemples

tiré selon p et $\mathcal{L}^*$ le LP-tree qui minimise le rang moyen empirique de $\mathcal{H}$ (apprenable par l'algorithme 12). Notons $\hat{p}$ la distribution de probabilité empirique estimée à partir de $\mathcal{H}$.

Soit $\mathcal{L}$ un LP-tree linéaire. Notons $X(\mathcal{L}, d)$ l'attribut à la profondeur d dans $\mathcal{L}$ et notons $x^*(\mathcal{L}, d)$ sa valeur préférée selon la table de préférence. Dans le cas où tous les attributs sont binaires, le rang d'un objet $\mathbf{o}$ est :

$$\mathsf{rank}(\mathcal{L}, \mathbf{o}) = 2^n - \sum_{d=1}^{n} 2^{n-d} \mathbf{1}_{\mathbf{o}[X(\mathcal{L},d)]=x^*(\mathcal{L},d)}$$

Le rang moyen espéré d'un LP-tree linéaire $\mathcal{L}$ est :

$$\begin{aligned}
\mathbb{E}_p[\mathsf{rank}(\mathcal{L}, \cdot)] &= \sum_{\mathbf{o} \in \underline{\mathcal{X}}} p(\mathbf{o}) \mathsf{rank}(\mathcal{L}, \mathbf{o}) \\
&= \sum_{\mathbf{o} \in \underline{\mathcal{X}}} p(\mathbf{o}) \left(2^n - \sum_{d=1}^{n} 2^{n-d} \mathbf{1}_{\mathbf{o}[X(\mathcal{L},d)]=x^*(\mathcal{L},d)} \right) \\
&= 2^n - \sum_{d=1}^{n} 2^{n-d} \sum_{\mathbf{o} \in \underline{\mathcal{X}}} p(\mathbf{o}) \mathbf{1}_{\mathbf{o}[X(\mathcal{L},d)]=x^*(\mathcal{L},d)} \\
&= 2^n - \sum_{d=1}^{n} 2^{n-d} \sum_{\mathbf{o}[X(\mathcal{L},d)]=x^*(\mathcal{L},d)} p(\mathbf{o}) \\
&= 2^n - \sum_{d=1}^{n} 2^{n-d} p(x^*(\mathcal{L}, d))
\end{aligned}$$

Nous pouvons exprimer la *ranking loss* entre $\breve{\mathcal{L}}$ et $\mathcal{L}^*$:

$$\mathsf{rloss}_p(\breve{\mathcal{L}}, \mathcal{L}^*) = \sum_{d=1}^{n} 2^{-d} \left(p(x^*(\breve{\mathcal{L}}, d)) - p(x^*(\mathcal{L}^*, d)) \right)$$

Afin de majorer la *ranking loss*, nous allons chercher à majorer $|p(x^*(\breve{\mathcal{L}}, d)) - p(x^*(\mathcal{L}^*, d))|$. C'est ce que fait le lemme suivant.

Lemme 5.5.1. *Soient $\mathcal{X}$ un ensemble d'attributs binaires, p et $\hat{p}$ deux distributions de probabilité sur $\underline{\mathcal{X}}$ et $\gamma > 0$. Supposons que, pour chaque attribut $X \in \mathcal{X}$ et chaque valeur $x \in \underline{X}$, $|p(x) - \hat{p}(x)| < \gamma$.*

Notons $x_1, x_2, \ldots, x_{2n}$ les valeurs $\bigcup_{X \in \mathcal{X}} \underline{X}$ triées selon p, i.e. $p(x_1) \geq p(x_2) \geq \ldots \geq p(x_{2n})$. Notons de la même manière $y_1, y_2, \ldots, y_{2n}$ triées selon $\hat{p}$.

Alors, pour tout $i \in [1, n]$, $|p(x_i) - p(y_i)| < 2\gamma$.

Démonstration. Soit $i \in [1, n]$. Montrons tout d'abord que $|p(x_i) - \hat{p}(y_i)| < \gamma$.

Cas 1 : $\hat{p}(x_i) > \hat{p}(y_i)$

Nous allons décomposer ce cas en deux sous-cas, selon l'ordre entre $p(x_i)$ et $\hat{p}(y_i)$.

Cas 1.1 : $p(x_i) \geq \hat{p}(y_i)$

Étant donné que $\hat{p}(x_i) > \hat{p}(y_i)$, $x_i \in \{y_j\}_{j<i}$. Or, $x_i \notin \{x_j\}_{j<i}$. Ces deux ensembles, $\{y_j\}_{j<i}$ et $\{x_j\}_{j<i}$, ont la même taille ; il existe donc une valeur $z \in \{x_j\}_{j<i} \setminus \{y_j\}_{j<i}$. Autrement dit, $p(z) > p(x_i)$ et $\hat{p}(y_i) > \hat{p}(z)$. On obtient donc les comparaisons suivantes :

$$p(z) > p(x_i) \geq \hat{p}(y_i) > \hat{p}(z)$$

Étant donné que $|p(z) - \hat{p}(z)| < \gamma$, on en déduit que $|p(x_i) - \hat{p}(y_i)| < \gamma$.

Cas 1.2 : $p(x_i) < \hat{p}(y_i)$

Dans ce cas, on a $\hat{p}(x_i) > \hat{p}(y_i) > p(x_i)$. Étant donné que $|p(x_i) - \hat{p}(x_i)| < \gamma$, on en déduit que $|p(x_i) - \hat{p}(y_i)| < \gamma$.

Cas 2 : $p(y_i) > p(x_i)$

Ce cas est similaire au cas 1.

Cas 3 : $\hat{p}(y_i) \geq \hat{p}(x_i)$ **et** $p(x_i) \geq p(y_i)$

Dans ce cas, $\hat{p}(y_i) \geq \hat{p}(x_i) > p(x_i) - \gamma$. De plus, $\hat{p}(y_i) < p(y_i) + \gamma \leq p(x_i) + \gamma$. Ce qui prouve que $|p(x_i) - \hat{p}(y_i)| < \gamma$.

Nous avons montré que, dans tous les cas, $|p(x_i) - \hat{p}(y_i)| < \gamma$. On peut conclure en exploitant l'inégalité triangulaire :

$$\begin{aligned} |p(x_i) - p(y_i)| &= |p(x_i) - \hat{p}(y_i) + \hat{p}(y_i) - p(y_i)| \\ &\leq |p(x_i) - \hat{p}(y_i)| + |\hat{p}(y_i) - p(y_i)| \\ &< 2\gamma \end{aligned}$$

Ce qui achève la preuve du lemme. □

Grâce à ce lemme, nous pouvons montrer que :

$$\begin{aligned} \mathsf{rloss}_p(\breve{\mathcal{L}}, \mathcal{L}^*) &= \sum_{d=1}^{n} 2^{-d} \left(p(x^*(\breve{\mathcal{L}}, d)) - p(x^*(\mathcal{L}^*, d)) \right) \\ &\leq \sum_{d=1}^{n} 2^{-d} \times 2\gamma \\ &= 2\gamma(1 - 2^{-n}) \\ &< 2\gamma \end{aligned}$$

Si nous posons $\gamma = \frac{\epsilon}{2}$, alors $\mathsf{rloss}_p(\breve{\mathcal{L}}, \mathcal{L}^*) < \epsilon$.

Ce qui signifie que, si pour chaque attribut, la distance entre $p(x)$ et $\hat{p}(x)$ est inférieure à $\epsilon/2$, alors la *ranking loss* est inférieur à ϵ. Afin de connaître le nombre d'exemples nécessaires à une estimation correcte de p, nous utilisons

l'inégalité de Boole puis l'inégalité d'Hoeffding. Notons par la suite N le nombre d'exemples d'apprentissage.

$$\begin{aligned}\Pr(\mathsf{rloss}_p(\breve{\mathcal{L}}, \mathcal{L}^*) \geq \epsilon) &\leq \Pr(\bigcup_{X \in \mathcal{X}} |p(x) - \hat{p}(x)| \geq \frac{\epsilon}{2}) \\ &\leq n \max_{X \in \mathcal{X}} \Pr(|p(x) - \hat{p}(x)| \geq \frac{\epsilon}{2}) \\ &\leq 2ne^{-N\epsilon^2/2}\end{aligned}$$

Si on souhaite que $2ne^{-N\epsilon^2/2} \leq \delta$, alors :

$$\begin{aligned}&2ne^{-N\epsilon^2/2} \leq \delta \\ \Leftrightarrow &\log 2n - N\frac{\epsilon^2}{2} \leq \log \delta \\ \Leftrightarrow &N\frac{\epsilon^2}{2} \geq -\log \delta + \log 2n \\ \Leftrightarrow &N \geq \frac{2}{\epsilon^2}(\log \frac{1}{\delta} + \log 2n)\end{aligned}$$

Ainsi, si $N \geq \frac{2}{\epsilon^2}(\log \frac{1}{\delta} + \log 2n)$, alors $\Pr(\mathsf{rloss}_p(\breve{\mathcal{L}}, \mathcal{L}^*) \geq \epsilon) \leq \delta$, ce qui termine la preuve. □

5.6 Expérimentations

Ces expérimentations ont pour objectif de mesurer empiriquement l'efficacité de notre algorithme d'apprentissage glouton. Pour cela, nous allons évaluer l'efficacité de notre algorithme à réapprendre un arbre caché. Plus précisément, le protocole se résume en trois étapes :

- un LP-tree $\breve{\mathcal{L}}$ est généré aléatoirement ;
- un ensemble d'exemples est tiré à partir d'une loi de probabilité p et de $\breve{\mathcal{L}}$;
- un LP-tree est appris à partir de ces exemples.

La *ranking loss* nous indiquera la distance entre l'arbre caché et l'arbre appris.

Des expérimentations supplémentaires concernant l'utilisation des LP-trees dans le problème de la recommandation de valeur en configuration interactive sont présentées au chapitre 7.

5.6.1 Protocole

Nous générons des LP-trees ($k = 1$) avec 10 attributs, chacun possédant entre 2 et 6 valeurs (ce nombre est tiré avec une loi uniforme). Chaque arbre est généré de manière *top-down*, c'est-à-dire depuis la racine jusqu'à feuilles. À chaque nœud interne N, un attribut X absent de $\mathbf{Anc}(N)$ est tiré aléatoirement ainsi qu'un ordre sur ses valeurs. Avec une probabilité σ (appelé coefficient de scission), N aura autant d'enfants que X a de valeurs ; avec une probabilité

$1-\sigma$, N aura un unique enfant. Plus σ est proche de 1, plus l'arbre généré aura de nœuds.

Pour un LP-tree généré aléatoirement $\breve{\mathcal{L}}$, on tire un nombre fixé d'exemples $\mathcal{H}$ à partir d'une distribution géométrique tronquée : la probabilité de tirer un objet $\mathbf{o}$ de rang r selon $\breve{\mathcal{L}}$ est de $p_\mu(r) = K\mu(1-\mu)^{r-1}$, où $1/\mu$ est proche de la moyenne de p_μ (ce n'est pas exactement la moyenne de p_μ car la distribution est tronquée).

Dans nos expériences, nous avons fixé le coefficient de scission $\sigma = 0.2$ et μ est choisi de manière à ce que le rang moyen de p_μ soit proche de $\underline{\mathcal{X}}/4$. La valeur de μ, qui dépend donc de la taille du domaine combinatoire, assure que, si l'attribut de la racine a deux valeurs, alors 88 % des objets seront tirés dans le sous-arbre préféré et 12 % dans le sous-arbre moins préféré.

5.6.2 Résultats

Nous avons comparé trois variantes de notre algorithme d'apprentissage (LP-tree linéaire avec la méthode heuristique de la section 5.3, LP-tree linéaire avec la méthode exacte de la section 5.4, LP-tree avec élagage, LP-tree sans élagage) en termes de :

- *ranking loss* entre l'arbre caché et l'arbre appris ;
- taille de l'arbre appris ;
- temps CPU nécessaire à l'apprentissage (incluant l'élagage quand il est utilisé).

Le seuil de scission est de $\tau = 20$ et la fonction de pénalité de l'élagage est fixée à $\phi = 1$.

Les figures 5.1, 5.2 et 5.3 présentent les différents résultats. Nous avons estimé la *ranking loss* avec une méthode de Monte-Carlo avec 100 000 échantillons car la cardinalité de $\underline{\mathcal{X}}$ peut être gigantesque. Chaque point est une valeur moyenne pour quelques 1000 LP-trees cachés.

La figure 5.1 montre que la *ranking loss* est très faible pour les quatre variantes. On peut remarquer que la *ranking loss* des LP-trees linéaires semblent atteindre un seuil : étant donné que les LP-tree cachés ne sont pas linéaires, les relations qu'ils représentent sont difficilement capturables par une structure linéaire. D'autre part, l'écart de *ranking loss* entre le LP-tree linéaire appris avec la méthode heuristique et celui appris avec la méthode exacte est négligeable : la méthode heuristique est très proche de la solution exacte.

Comme l'indique la figure 5.2, la taille des arbres appris croît linéairement avec le nombre d'exemples lorsqu'aucun élagage n'est fait. Avec l'élagage, la croissance de la taille des arbres est fortement limité. Ceci conjugué au fait que la *ranking loss* des LP-trees élagués est toujours meilleure que celle des LP-tree non-élagués suggère que notre algorithme fait du surapprentissage.

L'élagage réduit la taille du LP-tree appris et accroît sa précision au coût d'un temps d'apprentissage nettement plus long, comme le montrent les figures 5.2 et 5.3.

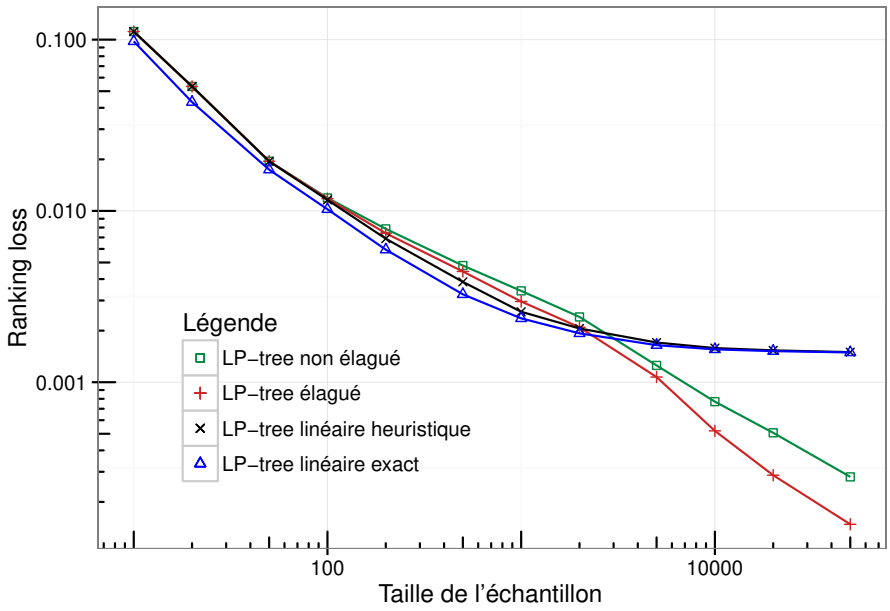


FIGURE 5.1 – *Ranking loss* en fonction du nombre d'exemples.

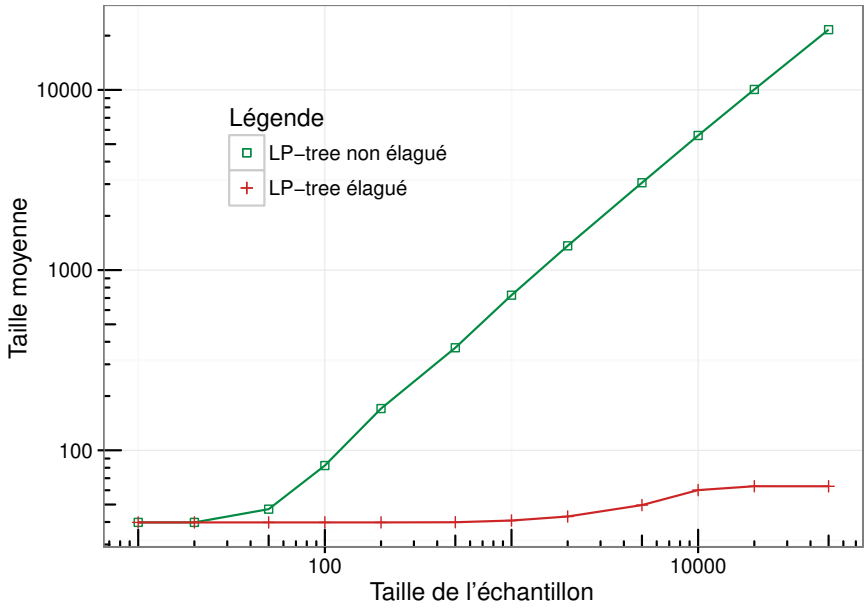


FIGURE 5.2 – Nombre de nœuds de l'arbre appris en fonction du nombre d'exemples avec et sans élagage.

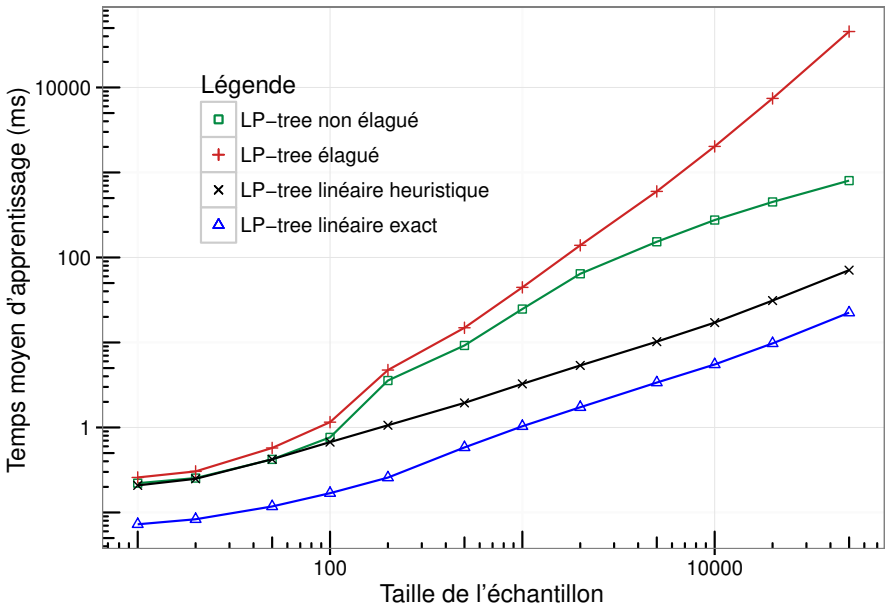


FIGURE 5.3 – Temps d'apprentissage (en ms) en fonction du nombre d'exemples.

5.7 Perspectives

Ceci constitue la première approche au problème général d'apprentissage de préférences ordinales à partir d'exemples positifs. Nous avons présenté un algorithme d'apprentissage de k-LP-trees à partir d'exemples positifs et nos expériences témoignent de l'efficacité de cette méthode pour réapprendre un modèle caché. Les prochaines idées que nous comptons explorer sont :

- adapter nos résultats au cas où il y a également un ensemble d'objets explicitement refusés ;
- analyser la complexité d'un apprentissage optimal de k-LP-trees (i.e. qui minimise le rang moyen empirique), et notamment savoir s'il s'agit d'un problème NP-complet (notre intuition penche vers la NP-complétude) ;
- étudier la *sample complexity* des k-LP-trees, c'est-à-dire le nombre d'exemples suffisants pour apprendre probablement un modèle assez bon (une étude expérimentale avec des données réelles est réalisée dans la partie III) ;
- adapter l'idée d'apprendre à partir d'exemples positifs à d'autres langages, notamment les CP-nets. Plus généralement, notre approche, qui suppose que le modèle qu'on cherche à apprendre représente un ordre total, pourrait être généralisée aux ordres partiels.

Troisième partie

Application à la recommandation en configuration interactive

Chapitre 6

Recommandation en configuration interactive

Sommaire

6.1 Les familles de systèmes de recommandation 114
6.1.1 Recommandation par filtrage collaboratif 114
6.1.2 Recommandation basée sur le contenu 114
6.1.3 Recommandation basée sur la connaissance 115
6.2 Configuration interactive 115
6.2.1 Prise en compte des contraintes de faisabilité 116
6.2.2 Recommandation en configuration interactive 117
6.2.3 Données disponibles dans les problèmes Renault . . 118
6.3 Méthodes de recommandation 118
6.3.1 Réseau bayésien naïf 118
6.3.2 k plus proches voisins 119
6.3.3 Recommandation à partir de l'extension préférée . . 122
6.3.4 Recommandation la plus probable à partir d'un réseau bayésien 123
6.4 Résumé 123

Les systèmes de recommandation ont pour objectif de guider les utilisateurs dans un grand ensemble d'alternatives. Ils sont spécialisés dans un domaine (généralement le catalogue d'un site) et s'appuient sur les préférences des utilisateurs pour personnaliser leurs résultats.

Nous nous intéressons dans cette troisième partie à l'application de l'apprentissage automatique de préférences à la recommandation en configuration interactive. Ce chapitre a pour objectif de présenter les outils actuels tandis que le chapitre suivant est consacré à nos expérimentations.

Nous commencerons en exposant brièvement les trois grandes familles de systèmes de recommandation. Nous introduirons ensuite notre problème, la recommandation en configuration interactive, et étudierons la pertinence des

systèmes de recommandation largement utilisés dans ce contexte. Enfin, nous présenterons des méthodes de recommandation qui sont applicables à la recommandation en configuration interactive.

6.1 Les familles de systèmes de recommandation

Afin de présenter les différentes familles de systèmes de recommandation, mettons-nous dans la situation où on cherche à recommander un film parmi un catalogue important. Le lecteur intéressé pourra approfondir sa lecture avec le livre [JZFF10].

6.1.1 Recommandation par filtrage collaboratif

L'idée fondamentale de la recommandation par filtrage collaboratif, dont les premiers travaux remontent à [GNOT92], est que si deux utilisateurs ont des goûts similaires sur les produits qu'ils ont tous les deux achetés, alors ils ont probablement des goûts similaires sur les autres produits. Avec notre exemple des films, cela signifie que si deux utilisateurs ont visionnés beaucoup de films en commun, et que l'un d'eux a particulièrement aimé *Iron Man*, alors l'autre va probablement l'aimer aussi ; on pourra alors lui recommander ce film.

Il existe également une variante à cette idée, introduite dans [SKKR01], qui consiste à transposer la recherche d'utilisateurs qui apprécient les mêmes films aux films appréciés par les mêmes utilisateurs. Par exemple, si l'ensemble des personnes qui ont visionné *Iron Man* recoupe fortement l'ensemble des personnes qui ont aimé le film *H2G2*, et si un utilisateur a vu l'un et pas vu l'autre, alors on pourra lui recommander celui qu'il n'a pas vu.

D'une manière plus générale, l'idée de la recommandation par filtrage collaboratif est d'exploiter les similarités entre utilisateurs pour effectuer sa recommandation. C'est une recommandation très répandue (utilisée par Amazon [LSY03], Netflix [GUH16], LinkedIn, Spotify...), mais qui rencontre néanmoins des limitations :

- il faut un nombre importants d'utilisateurs pour que la recommandation soit efficace ;
- quand un nouveau film sort et que personne ne l'a encore vu, il ne pourra pas être recommandé.

Intuitivement, on pourrait se dire que ce dernier point pourrait être résolu à condition d'avoir des informations supplémentaires sur le film lui-même : par exemple, si ce nouveau film est un film de science-fiction, on pourrait le recommander aux amateurs de films de science-fiction. C'est l'idée de la recommandation basée sur le contenu.

6.1.2 Recommandation basée sur le contenu

La recommandation basée sur le contenu utilise des informations sur les produits eux-même ; pour un film, les attributs intéressants pourraient être le genre, l'année de sortie, la nationalité, le nom du réalisateur, le nombre d'Oscars

qu'il a gagné, etc. À partir du profil d'un utilisateur, obtenu par élicitation ou appris à partir de ses précédents visionnages, le système de recommandation va chercher les films les plus pertinents. Il est difficile d'établir un point de naissance de la recommandation basée sur le contenu car plusieurs de ses méthodes ont été conçues pour d'autres domaines, notamment la recherche d'information [JZFF10].

Les limitations de cette méthode proviennent du fait que :

- l'information sur les objets n'est pas toujours facile à obtenir s'il n'existe pas déjà une base de données. L'annotation automatique (grâce au traitement automatique de la langue) peut aider à construire cette base de données.
- les informations disponibles ne sont pas forcément suffisantes pour avoir des recommandations adaptées ; un utilisateur peut par exemple aimer les films contemplatifs, un genre relativement flou et mal défini.

6.1.3 Recommandation basée sur la connaissance

La dernière catégorie des systèmes de recommandation est dite « basée sur la connaissance » (*knowledge-based*). Il s'agit d'une catégorie qui regroupe une multitude de techniques différentes, très souvent utilisées en complément des méthodes que nous avons vues précédemment.

La recommandation basée sur la connaissance s'applique essentiellement aux produits pour lesquels les deux méthodes précédentes se retrouvent limitées, par exemple les produits qui sont achetés peu souvent (une voiture, par exemple) ; en effet, dans ce cas :

- la recommandation par filtrage collaboratif est limitée car l'historique d'achats d'un utilisateur est très réduit ;
- en cas d'achats anonymes, la recommandation basée sur le contenu est limitée car le profil des utilisateurs est très peu renseigné.

La recommandation basée sur la connaissance s'appuie sur d'autres informations, comme des questions posées directement à l'utilisateur ou des connaissances métier sur les produits, comme par exemple des contraintes de faisabilité sur des produits en espace combinatoire. La délimitation entre recommandation basée sur la connaissance et la recommandation basée sur le contenu est parfois assez floue.

6.2 Configuration interactive

Le premier exemple de configuration de produit remonte à [McD82]. Selon le modèle de [MF89], la configuration de produit est un processus de conception où le produit conçu est un assemblage de composants prédéfinis (ces composants forment donc un espace combinatoire). Ces produits peuvent être des produits matériels, comme des véhicules, ou encore des logiciels. Une configuration est alors définie comme l'ensemble des composants qui constituent un produit et la manière dont ceux-ci sont assemblés. Plus précisément, l'assemblage de composants suit le plus souvent des règles qui limitent les combinaisons possibles.

À partir d'une description du produit recherché ou d'un critère d'évaluation des différents produits, la configuration de produit a pour objectif de trouver une configuration qui respecte les règles d'assemblage et qui corresponde à la description désirée ou qui maximise le critère d'évaluation.

La configuration de produits basée sur les contraintes cherche à construire un objet satisfaisant un certain nombre de contraintes liées au domaine ; contrairement au cas précédent, la problématique de l'assemblage est écartée. Il peut par exemple s'agir de la configuration d'une voiture soumise à des contraintes telles que « une voiture décapotable ne peut pas avoir de toit ouvrant ».

Pour choisir un objet en espace combinatoire, il est souvent impossible de parcourir l'intégralité du catalogue. Afin de pouvoir recherche efficacement l'objet qui plaît de plus à l'utilisateur, des méthodes d'exploration de l'espace combinatoire ont été mises en place. Parmi les différentes méthodes de configuration basée sur les contraintes, la configuration interactive permet à l'utilisateur de créer sur mesure la configuration qui lui convient de manière incrémentale.

Plus exactement, la configuration interactive est effectuée dans le cadre d'une session de configuration dont l'organisation est la suivante. Au début de la session, aucun attribut n'est affecté. Puis, l'utilisateur choisit un attribut et lui affecte l'une des valeurs possibles. L'utilisateur peut à tout moment choisir de revenir sur un choix antérieur et de désaffecter un attribut. Lorsque tous les attributs ont une valeur, le produit est complet.

La configuration interactive s'effectue typiquement avec un outil en ligne nommé *configurateur*. Il existe des configurateurs de voitures (Renault), d'ordinateurs (PC Part Picker), de cuisines (Cuisinella)...

Par la suite, on nommera « configuration partielle » l'objet partiellement affecté $\mathbf{u}$ d'une session de configuration. De plus, on notera **Assigned** les attributs sur lesquels $\mathbf{u}$ a une valeur affectée. Enfin, on notera Next le prochain attribut choisi par l'utilisateur.

6.2.1 Prise en compte des contraintes de faisabilité

La plupart des objets complexes configurables sont soumis à des contraintes de faisabilité. Ces contraintes peuvent avoir plusieurs origines, notamment :

- technique (pas de toit ouvrant sur une voiture décapotable) ;
- marketing (pas de volant en cuir sur une voiture bas de gamme) ;
- de gestion ou logistique (produit qui n'est plus en stock ou indisponible) ;
- légal (les réfrigérateurs au CFC ne sont plus autorisés à la vente).

Ces contraintes doivent être prises en compte lors de la configuration ; par exemple, si l'utilisateur a choisi un toit ouvrant, il ne faut pas que l'option « décapotable » soit disponible. Plus exactement, pour un produit partiellement configuré $\mathbf{u}$ et un attribut Next, le domaine de Next présenté à l'utilisateur doit être réduit aux seules valeurs pour lesquelles il existe un objet complet compatible avec $\mathbf{u}$ et qui ait pour l'attribut Next l'une de ces valeurs. La prise en compte de ces contraintes peut se faire grâce à différentes techniques telles que la propagation de contraintes [Wal72, FFH+98, BFL13] ou la compilation [Par02, AFM02, HWA07].

Dans la suite, une valeur $x \in \underline{\mathsf{Next}}$ sera dite *admissible* pour $\mathbf{u}$ s'il existe un objet complet $\mathbf{o}$ satisfaisant les contraintes tel que $\mathbf{o}[\mathbf{Assigned}] = \mathbf{u}$ et $\mathbf{o}[\mathsf{Next}] = x$.

6.2.2 Recommandation en configuration interactive

La configuration interactive permet de construire l'objet de son choix. Bien qu'il s'agisse d'une méthode pratique pour créer l'objet que l'on cherche, la configuration interactive peut s'avérer longue et laborieuse pour des objets définis sur des dizaines d'attributs. De plus, il peut y avoir des attributs pour lesquels l'utilisateur n'a que peu ou aucun avis, comme par exemple la technologie des phares d'une voiture ; l'utilisateur ne sait alors pas quelle valeur choisir car il ne connaît pas forcément les différences qui distinguent les valeurs proposées.

La recommandation a pour objectif de faciliter la recherche de l'utilisateur, mais la recommandation en configuration reste encore peu étudiée [1]. Dans le contexte de la configuration interactive, nous pouvons imaginer au moins trois types de recommandation possibles.

Recommandation d'une complétion de la configuration courante

La recommandation d'une complétion de la configuration courante cherche à recommander un objet complet $\mathbf{o}$ compatible avec la configuration partielle courante $\mathbf{u}$. La complétion automatique de l'objet partiel courant $\mathbf{u}$ a par exemple été étudiée par [JBGS10] qui propose, dans le cas binaire, d'appliquer le *shopping principle* selon lequel un produit partiellement configuré devrait être complété en affectant tous les attributs non assignés à *faux* ; l'idée derrière cette complétion est que tout ce que l'utilisateur n'a pas spécifiquement choisi devrait absent de la configuration finale. Dans un cas multivalué, on pourrait par exemple compléter la configuration courante de manière à minimiser son coût final.

Nous ne traiterons pas de ce problème dans la suite.

Recommandation du prochain attribut

La recommandation du prochain attribut à affecter cherche à recommander un attribut Next en fonction de la configuration partielle courante. Cette recommandation, bien que potentiellement utile pour l'utilisateur, se heurte dans notre cas à l'absence de données réelles sur l'ordre des attributs choisi par les utilisateurs.

Nous ne traiterons donc pas de ce problème dans la suite.

Recommandation d'une valeur pour un attribut

La recommandation d'une valeur pour un attribut Next fixé cherche à recommander une valeur admissible parmi $\underline{\mathsf{Next}}$ en fonction de la configuration partielle courante.

C'est à cette recommandation que nous nous sommes intéressés.

1. [RRS11] écrit à ce sujet : « *recommendation of configurable products pushes the limits of current recommender technologies* »

6.2.3 Données disponibles dans les problèmes Renault

Notre problème de recommandation de valeurs en configuration interactive anonyme s'appuie sur des données du constructeur automobile Renault dans lequel les produits sont des voitures. Dans ce problème, le système de recommandation dispose comme données d'un ensemble d'objets précédemment vendus et d'aucune autre information. Ces informations limitées vont restreindre l'ensemble des méthodes de recommandation que nous pourrons utiliser.

En particulier, nous ne disposons d'aucune information sur le profil de l'utilisateur, ce qui limite grandement les méthodes de recommandation basées sur le contenu. De plus, nous ne disposons pas d'un historique par utilisateur (les achats étant anonymes), ce qui va limiter la recommandation par filtrage collaboratif.

La section suivante va détailler les méthodes utilisables avec ces données.

6.3 Méthodes de recommandation

Voici des méthodes de recommandation que nous pouvons utiliser pour répondre à notre problématique de recommandation de valeur en configuration interactive anonyme. Nous présenterons d'abord les réseaux bayésiens naïfs qui serviront de base pour exposer ensuite trois méthodes adaptées des k plus proches voisins introduites par [CGO+02]. Nous proposerons enfin deux nouvelles méthodes de recommandation de valeur en configuration interactive.

6.3.1 Réseau bayésien naïf

Les réseaux bayésiens naïfs sont des classifieurs. Rappelons qu'une tâche de classification consiste à prédire une classe pour une observation. Par la suite, nous noterons $\mathbf{X}$ l'ensemble de m attributs sur lesquels sont définies les observations, et nous noterons C l'attribut de classe dont nous cherchons à prédire une valeur.

Le classifieur

Les réseaux bayésiens naïfs sont un sous-ensemble des réseaux bayésiens dont les nœuds sont étiquetés par $\mathbf{X} \cup \{C\}$ et où les attributs $X_i \in \mathbf{X}$ sont indépendants deux à deux sachant C. Structurellement, cela signifie qu'il y a $|\mathbf{X}|$ arcs, chaque arc sortant de C et arrivant dans un nœud $X_i \in \mathbf{X}$ différent ; autrement dit, C a pour enfant tous les nœuds $\mathbf{X}$ et les nœuds $\mathbf{X}$ sont des feuilles. Un réseau bayésien naïf est présenté à la figure 6.1.

Pour apprendre un réseau bayésien naïf à partir d'un ensemble d'exemples $\mathcal{E}$, il suffit d'en apprendre les tables, étant donné que la structure est connue. Pour cela, on estime les probabilités conditionnelles à partir des exemples, avec le plus souvent un lissage de Laplace :

$$\begin{cases} P(c) = \frac{\#(c)}{|\mathcal{E}|} \text{ pour chaque } c \in \underline{C} \\ P(x_i \mid c) = \frac{\#(x_i,c)+1}{\#(c)+|\underline{X_i}|} \text{ pour chaque } x_i \in \underline{X_i}, c \in \underline{C} \end{cases}$$

où $\#(\mathbf{u})$ est le nombre d'exemples de $\mathcal{E}$ compatibles avec $\mathbf{u}$, i.e. $\#(\mathbf{u}) = |\{\mathbf{v} \in \mathcal{E} \mid \mathbf{u} \sim \mathbf{v}\}|$.

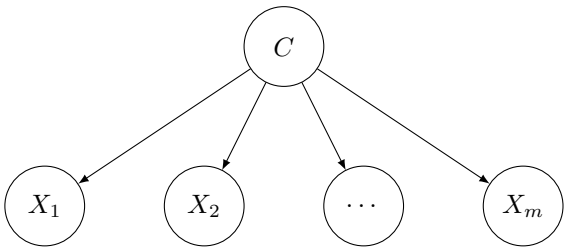


FIGURE 6.1 – Un réseau bayésien naïf

La structure particulière des réseaux bayésiens naïfs et les hypothèses d'indépendances conditionnelles permettent une inférence simplifiée :

$$\begin{aligned}\operatorname*{argmax}_{c \in \underline{C}} P(c \mid \mathbf{x}) &= \operatorname*{argmax}_{c \in \underline{C}} \frac{P(c, \mathbf{x})}{P(\mathbf{x})} \\ &= \operatorname*{argmax}_{c \in \underline{C}} P(c, \mathbf{x}) \\ &= \operatorname*{argmax}_{c \in \underline{C}} \left(P(c) \prod_{X \in \mathbf{X}} P(\mathbf{x}[X] \mid c) \right)\end{aligned}$$

La classe prédite est la valeur la plus probable, c'est-à-dire la valeur c de C qui maximise $P(c) \prod_{X \in \mathbf{X}} P(\mathbf{x}[X] \mid c)$.

Adaptation à la recommandation

L'adaptation des réseaux bayésiens naïfs à la recommandation de valeur en configuration interactive se base sur l'idée suivante : on considère que l'attribut Next est l'attribut de classe (que nous avons notée C) et que les attributs $\mathbf{U}$ sont les attributs qui décrivent une observation (que nous avons notés $\mathbf{X}$). La valeur recommandée est la valeur la plus probable, i.e. :

$$\operatorname*{argmax}_{r \in \underline{\mathsf{Next}}} \left(P(r) \prod_{X \in \mathbf{U}} P(\mathbf{u}[X] \mid r) \right)$$

6.3.2 k plus proches voisins

Le classifieur

La méthode des k plus proches voisins est une méthode utilisée en classification. Pour prédire la classe d'une observation $\mathbf{x}$ à partir d'un ensemble d'apprentissage $\mathcal{E} = \{(\mathbf{x}_i, c_i)\}_i$, la méthode des k plus proches voisins procède en deux phases :

- Une phase de recherche des voisins : on recherche dans $\mathcal{E}$ les k observations les plus « proches » de $\mathbf{x}$ (selon une certaine distance), dont les classes sont $\{c_i\}_{i=1}^{k}$.
- Une phase d'agrégation des classes des voisins : la classe prédite est la classe la plus fréquente parmi $\{c_i\}_{i=1}^{k}$.

La valeur k est généralement estimée expérimentalement.

La distance la plus couramment utilisée lorsque les attributs sont ordinaux est la distance de Hamming, où la distance $d(\mathbf{a}, \mathbf{b})$ entre deux vecteurs $\mathbf{a}$ et $\mathbf{b}$ de $\underline{\mathbf{X}}$ est le nombre d'attributs de $\mathbf{X}$ dont les valeurs dans $\mathbf{a}$ et $\mathbf{b}$ diffèrent, i.e. :

$$d(\mathbf{a}, \mathbf{b}) = |\{X \in \mathbf{X} \mid \mathbf{a}[X] \neq \mathbf{b}[X]\}|$$

Adaptation à la recommandation

L'adaptation des k plus proches voisins à la recommandation de valeurs est similaire à ce que nous avons vu pour les réseaux bayésiens naïfs : on considère l'attribut Next comme étant l'attribut de classe (que nous avons notée C) et les attributs $\mathbf{U}$ comme étant les attributs qui décrivent une observation (que nous avons notés $\mathbf{X}$).

La principale différence est que, dans notre problème de recommandation, il y a également des attributs non affectés. Ces attributs seront simplement ignorés lors de la recherche des voisins de $\mathbf{u}$; pour cela, on restreint la distance de Hamming aux attributs de $\mathbf{U}$:

$$d_{\mathbf{U}}(\mathbf{a}, \mathbf{b}) = |\{X \in \mathbf{U} \mid \mathbf{a}[X] \neq \mathbf{b}[X]\}| \tag{6.1}$$

[CGO+02] propose trois algorithmes de recommandation basés sur les k plus proches voisins, que nous allons tout de suite présenter.

Par la suite, nous noterons $N(k, \mathbf{u})$ l'ensemble des k plus proches voisins de $\mathbf{u}$ au sens de la distance $d_{\mathbf{U}}$.

Votant majoritaire pondéré

Le plus simple des algorithmes d'agrégation est le votant majoritaire pondéré (*Weighted Majority Voter*). Il recommande une valeur pour Next sur la base d'un vote pondéré entre les k plus proches voisins. Le poids d'un voisin $\mathbf{v} \in N(k, \mathbf{u})$ est le nombre de valeurs identiques avec $\mathbf{u}$, ou, plus formellement :

$$poids(\mathbf{u}, \mathbf{v}) = |\{X \in \mathbf{U} \mid \mathbf{u}[X] = \mathbf{v}[X]\}|$$

Ou, de manière équivalente :

$$poids(\mathbf{u}, \mathbf{v}) = |\mathbf{U}| - d_{\mathbf{U}}(\mathbf{u}, \mathbf{v})$$

La valeur recommandée pour Next est alors :

$$\underset{r \in \underline{\mathsf{Next}}}{\operatorname{argmax}} \sum_{\mathbf{v} \in N(k, \mathbf{u})} poids(\mathbf{u}, \mathbf{v}) \mathbf{1}_{\mathbf{v}[\mathsf{Next}]=r}$$

Votant bayésien naïf

Le votant bayésien naïf, comme son nom l'indique, est assez proche du réseau bayésien naïf que nous avons vu précédemment. La différence est que le réseau bayésien naïf utilise tous les exemples pour apprendre ses tables là où le votant bayésien naïf estime ses tables à partir des voisins de $\mathbf{u}$ uniquement, de la manière suivante :

$$\begin{cases} P(r) = \frac{\#_k(r)}{k} \text{ pour chaque } r \in \underline{\mathsf{Next}} \\ P(\mathbf{u}[X_i] \mid r) = \frac{\#_k(\mathbf{u}[X_i], r)+1}{\#_k(r)+|\underline{X_i}|} \text{ pour chaque } X_i \in \mathbf{U}, r \in \underline{\mathsf{Next}} \end{cases}$$

où $\#_k(\mathbf{a})$ est le nombre d'exemples de $N(k, \mathbf{u})$ compatibles avec $\mathbf{a}$, i.e. $\#_k(\mathbf{a}) = |\{\mathbf{v} \in N(k, \mathbf{u}) \mid \mathbf{v} \sim \mathbf{a}\}|$.

Nous avons modifié légèrement la formule par rapport à celle de [CGO+02], qui est :

$$\begin{cases} P(r) = \frac{\#_k(r)}{k} \text{ pour chaque } r \in \underline{\mathsf{Next}} \\ P(\mathbf{u}[X_i] \mid r) = \frac{\#_k(\mathbf{u}[X_i], r)+1}{\#_k(r)+k} \text{ pour chaque } X_i \in \mathbf{U}, r \in \underline{\mathsf{Next}} \end{cases}$$

Nous avons remplacé le terme k au dénominateur de $P(\mathbf{u}[X_i] \mid r)$ par $|\underline{X_i}|$, car sinon la somme des probabilités conditionnelles ne vaut pas 1.

La valeur recommandée est la valeur la plus probable, i.e. :

$$\operatorname*{argmax}_{r \in \underline{\mathsf{Next}}} P(r) \prod_{X \in \mathbf{U}} P(\mathbf{u}[X] \mid r) = \operatorname*{argmax}_{r \in \underline{\mathsf{Next}}} \frac{\#_k(r)}{k} \prod_{X \in \mathbf{U}} \frac{\#_k(\mathbf{u}[X], r) + 1}{\#_k(r) + |\underline{X}|}$$

Choix le plus populaire

L'algorithme du choix le plus populaire (*Most Popular Choice*) recommande la valeur de Next du voisin de $\mathbf{u}$ le plus populaire. Pour estimer la probabilité de chacun des voisins, [CGO+02] utilisent un réseau bayésien inspiré du réseau bayésien naïf.

Dans le réseau bayésien naïf utilisé par le votant bayésien naïf, l'attribut Next est parent de tous les attributs $\mathbf{U}$. L'algorithme du choix le plus populaire utilise un réseau bayésien dont chaque attribut de $\mathbf{V} = \mathcal{X} \setminus \mathbf{U}$ non affecté dans $\mathbf{u}$ est un parent de tous les attributs de $\mathbf{U}$, comme cela est représenté à la figure 6.2.

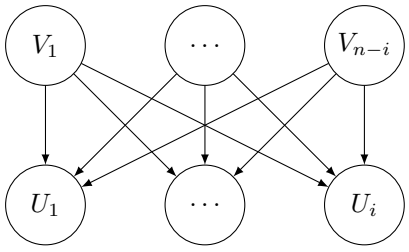


FIGURE 6.2 – Le réseau bayésien implicitement utilisé par l'algorithme du choix le plus populaire

Soit $\mathbf{x} \in \mathcal{X}$. Alors $P(\mathbf{x})$ peut se calculer de la manière suivante :

$$P(\mathbf{x}) = \prod_{A \in \mathbf{V}} p(\mathbf{x}[A]) \prod_{B \in \mathbf{U}} p(\mathbf{x}[B] \mid \mathbf{x}[\mathbf{V}])$$

Les tables sont apprises de la manière suivante :

$$\begin{cases} P(x) = \frac{\#_k(x)}{k} \text{ pour chaque } x \in \underline{V}, V \in \mathbf{V} \\ P(u \mid \mathbf{v}) = \frac{\#_k(u,\mathbf{v})+1}{\#_k(\mathbf{v})+|\underline{U_i}|} \text{ pour chaque } U_i \in \mathbf{U}, u \in \underline{U_i}, \mathbf{v} \in \underline{\mathbf{V}} \end{cases}$$

Ce réseau bayésien permet d'estimer la probabilité de chaque voisin de $\mathbf{u}$. Néanmoins, l'objectif n'est pas de trouver le voisin le plus probable mais un voisin à la fois similaire à $\mathbf{u}$ et assez probable. Pour cela, les auteurs ne vont pas simplement rechercher le voisin $\mathbf{v}$ qui maximise $P(\mathbf{v})$. Leur idée est la suivante : si la valeur de l'attribut $U \in \mathbf{U}$ est différente pour $\mathbf{v}$ et $\mathbf{u}$, i.e. $\mathbf{v}[U] \neq \mathbf{u}[U]$, alors il faut pénaliser ce voisin. De cette manière, plus un voisin a de valeurs en commun avec $\mathbf{u}$, plus il aura une haute probabilité.

Pour y parvenir, on transforme un voisin $\mathbf{x}$ en l'objet $\phi(\mathbf{x})$ de la manière suivante :

$$\begin{cases} \phi(\mathbf{x})[A] = \mathbf{x}[A] \text{ pour tout } A \in \mathbf{V} \\ \phi(\mathbf{x})[B] = \mathbf{x}[B] \text{ si } \mathbf{x}[B] = \mathbf{u}[B] \text{ et } B \in \mathbf{U} \\ \phi(\mathbf{x})[B] = \diamond \text{ si } \mathbf{x}[B] \neq \mathbf{u}[B] \text{ et } B \in \mathbf{U} \end{cases}$$

où $\diamond$ est une valeur fictive, i.e. par définition $\#_k(\diamond) = 0$.

Le voisin le plus populaire est le voisin $\mathbf{x}^* \in N(k, \mathbf{u})$ dont $\phi(\mathbf{x}^*)$ maximise la probabilité, i.e. :

$$\begin{aligned} \mathbf{x}^* &= \operatorname*{argmax}_{\mathbf{x} \in N(k,\mathbf{u})} P(\phi(\mathbf{x}^*)) \\ &= \operatorname*{argmax}_{\mathbf{x} \in N(k,\mathbf{u})} \prod_{A \in \mathbf{V}} \frac{\#_k(\mathbf{x}[A])}{k} \prod_{B \in \mathbf{U}} \frac{\mathbf{1}_{\mathbf{x}[U]=\mathbf{u}[U]} \times \#_k(\mathbf{x}[B], \mathbf{x}[\mathbf{V}]) + 1}{\#_k(\mathbf{x}[\mathbf{V}]) + |\underline{U}|} \end{aligned}$$

La valeur recommandée est finalement la valeur de Next pour $\mathbf{x}^*$, i.e. $\mathbf{x}^*[\mathsf{Next}]$.

Les travaux présentés dans les deux chapitres précédents suggèrent deux autres méthodes : la recommandation à partir de l'extension préférée et la recommandation la plus probable à partir d'un réseau bayésien.

6.3.3 Recommandation à partir de l'extension préférée

Supposons qu'on ait établi ou appris un modèle de préférence qui représente un ordre dont nous notons la partie asymétrique $\succ$. Nous cherchons à produire une recommandation pour un attribut Next connaissant une instanciation partielle $\mathbf{u}$.

Une méthode de recommandation consiste à chercher l'extension préférée de $\mathbf{u}$, c'est-à-dire l'objet $\mathbf{o} \in \underline{\mathcal{X}}$ tel que $\mathbf{o}$ soit compatible avec $\mathbf{u}$ et qu'il n'existe aucun $\mathbf{o}' \in \underline{\mathcal{X}}$ compatible avec $\mathbf{u}$ tel que $\mathbf{o}' \succ \mathbf{o}$, et à recommander la valeur de Next dans cette extension $\mathbf{o}$, c'est-à-dire à recommander la valeur $\mathbf{o}[\mathsf{Next}]$.

Comme nous l'avons vu dans la section 2.3, la requête de meilleure extension est une requête classique, dont nous rappelons à la table 6.1 la complexité selon différents langages.

Nous utiliserons cette idée avec les k-LP-trees, un langage dont nous savons apprendre un modèle à partir d'exemples positifs.

k-LP-tree	P
CP-net	NP-easy
CP-net acyclique	P
Réseau bayésien	NP-hard
VCSP	NP-hard
GAI-tree	NP-hard

TABLE 6.1 – Complexité de la requête de meilleure extension

6.3.4 Recommandation la plus probable à partir d'un réseau bayésien

Dans le cas des réseaux bayésiens, la requête qui permet de trouver l'extension la plus probable est la requête MPE, que nous avons évoquée au chapitre 4. Étant donné qu'un réseau bayésien représente une loi de probabilité d'acceptation sur l'ensemble des objets combinatoires, nous pouvons rechercher la valeur de Next *la plus probable* sachant **u**. Et c'est ce que fait, par définition, la requête MAP.

La différence entre une requête MPE et une requête MAP est présentée dans l'exemple suivant.

Exemple 6.3.1. Soit la loi de probabilité suivante (par souci de brièveté, nous avons retiré l'attribut du plat principal) :

vin rouge/fruit	0.35
vin rouge/chocolat	0.05
vin blanc/fruit	0.33
vin blanc/chocolat	0.27

Cherchons une recommandation pour le vin (sans aucun autre attribut affecté). Le plat préféré est *vin rouge/fruit*, donc la requête MPE va renvoyer ce plat. Le vin recommandé est donc, *vin rouge/fruit*$[V]$, soit *vin rouge*. Néanmoins, la probabilité que *vin rouge* soit choisi est seulement de 0.4, et la probabilité que *vin blanc* soit choisi est de 0.6.

En utilisant la requête MAP, par contre, la valeur recommandée sera bien *vin blanc*, la valeur la plus probable.

6.4 Résumé

Le problème recommandation en configuration interactive de Renault ne nous permet pas d'utiliser les méthodes les plus courantes en recommandation car nous avons très peu d'information sur l'utilisateur ou sur son environnement social. Heureusement, plusieurs méthodes de recommandation sont applicables :

- les langages de préférences dont des modèles peuvent être appris à partir d'une liste d'objets vendus : les différentes variantes de LP-trees (en utilisant l'extension préférée) et les réseaux bayésiens (avec la requête MAP) ;

- des méthodes de classification adaptées à la recommandation, comme les réseaux bayésiens naïfs ;
- des méthodes spécifiquement développées pour ce problème, comme les trois algorithmes basés sur les k plus proches voisins proposés par [CGO+02].

Chapitre 7

Expérimentations sur la recommandation de valeurs en configuration interactive

Sommaire

7.1 Étude de cas : Renault 126
7.2 Protocole 127
7.2.1 Oracle 128
7.3 Expériences avec les algorithmes de la littérature . 129
7.3.1 Influence de la taille du voisinage 130
7.3.2 Vaut-il mieux apprendre avec les exemples infaisables ? 131
7.3.3 Évolution du taux d'erreur 133
7.3.4 Temps de recommandation en fonction du nombre d'attributs assignés 135
7.3.5 Vaut-il mieux utiliser des clusters ? 136
7.3.6 Influence de la taille du jeu d'apprentissage 139
7.3.7 Influence des contraintes 140
7.3.8 Conclusion 142
7.4 Expériences relatives à DRC 142
7.4.1 Taux de succès avec mise à jour de l'historique . . . 144
7.5 Expériences relatives aux LP-trees 146
7.6 Conclusion 149
7.6.1 Évaluation des algorithmes de la littérature 150
7.6.2 L'avantage de DRC 150
7.6.3 Efficacité réelle des LP-trees 150

Ce chapitre détaille la contribution expérimentale de cette thèse, et a pour objectif d'apporter une vue d'ensemble sur les performances des différents algorithmes de recommandation en configuration interactive

sur des jeux de données réels et sur les paramètres qui influencent leur temps de recommandation et leur taux de succès.

Ce chapitre sera aussi l'occasion d'évaluer nos contributions théoriques dans le contexte de la recommandation en configuration interactive afin de les confronter à un problème réel.

7.1 Étude de cas : Renault

Notre étude expérimentale se base sur une étude de cas fournie par Renault, le constructeur automobile, dans le cadre du projet ANR nommé BR4CP (*Business Recommendation for Configurable Products*). Les données sont disponibles à l'adresse suivante : `http://www.irit.fr/~Helene.Fargier/BR4CP/benches.html`.

Nous disposons des données suivantes pour trois modèles d'automobile :

- un historique de ventes de voitures vendues en concession, avec le mois de chaque vente ;
- un fichier qui contient les contraintes de faisabilité des voitures.

Nous utiliserons ces deux sources d'informations : l'historique de ventes, qui contient des informations sur les préférences des utilisateurs, et le fichier des contraintes, afin de s'assurer que la voiture configurée est toujours faisable.

Les historiques de ventes contiennent également des voitures qui ne satisfont pas les contraintes. En effet, les contraintes évoluent continuellement ce qui fait que le catalogue est très variable. Les trois modèles d'automobiles qui composent l'étude de cas sont :

- le jeu *Renault-44*, qui a 44 attributs pour $3.5 \cdot 10^{22}$ véhicules potentiels, a un historique de 14786 exemples, dont 8252 sont cohérents avec les contraintes ;
- le jeu *Renault-48*, qui a 48 attributs pour $2.1 \cdot 10^{19}$ véhicules potentiels, a un historique de 27088 exemples, dont 710 sont cohérents avec les contraintes ;
- le jeu *Renault-87*, qui a 87 attributs pour $8.5 \cdot 10^{41}$ véhicules potentiels, a un historique de 17715 exemples, dont 8335 sont cohérents avec les contraintes.

En particulier, nous ne disposons pas des informations suivantes :

- de l'ordre dans lequel les attributs ont été affectés (en effet, ce sont des voitures achetées en concession) ;
- du profil des acheteurs ;
- d'autres informations contextuelles (pays d'achat, par exemple).

L'objectif de nos expériences est d'évaluer le taux de succès et le temps de calcul de la recommandation de valeur lors de la configuration de ces véhicules. En effet, le taux de succès nous permet d'estimer à quel point notre approche aurait pu fournir une recommandation utile. Le temps de recommandation est également un facteur important car la configuration est effectuée en ligne et qu'on ne peut pas se permettre de faire patienter l'utilisateur pendant plusieurs secondes à chaque recommandation.

7.2 Protocole

Afin d'évaluer les différentes méthodes de recommandation, nous effectuons une validation croisée en dix plis : chaque jeu d'exemples est découpé en dix parties. L'algorithme apprend avec neuf dixièmes du jeu d'exemple et sera évalué avec le dernier dixième, appelé jeu de test. Cette procédure est répétée dix fois, de manière à évaluer l'algorithme sur l'entièreté du jeu de données. Pour les expérimentations qui prennent en compte les contraintes, les algorithmes seront évalués uniquement sur les exemples qui satisfont les contraintes ; en effet, nous supposons que toutes les futures ventes devraient satisfaire les contraintes.

Le protocole expérimental est décrit dans l'algorithme 13. Chaque test est une simulation d'une session de configuration, c'est-à-dire une succession aléatoire (tirée uniformément) des attributs de $\mathcal{X}$. Un ordre des attributs a été utilisé par l'utilisateur pour construire cet objet, et cet ordre varie généralement selon les utilisateurs. Malheureusement, les jeux de données de Renault ne contiennent aucune information sur cet ordre car seuls les objets finaux, entièrement configurés, sont fournis. Nous allons donc générer aléatoirement un ordre d'attributs pour chaque produit $\mathbf{o}$ (ligne 5) de l'ensemble de test et simuler une session de configuration qui aurait conduit à la construction de $\mathbf{o}$.

Nous évaluons les algorithmes de recommandations sur les deux critères suivants :

- le temps nécessaire au calcul de la recommandation ;
- le taux de succès moyen, c'est-à-dire la proportion de recommandations acceptées.

Nous simulons la session de configuration en itérant sur tous les attributs de $\mathcal{X}$ dans l'ordre précédemment tiré. À chaque étape de cette session, notons Next l'attribut dont on va rechercher une recommandation et $\mathbf{u}$ le produit partiellement configuré. Si l'expérience prend en compte les contraintes, il est possible qu'à un moment donné l'ensemble des valeurs admissibles (c'est-à-dire qui mène à un véhicule satisfaisant les contraintes, notion définie en section 6.2.1) soit réduit à une seule valeur, auquel cas la recommandation est triviale (on est certain que ce sera le choix de l'utilisateur). Remarquons qu'il y a au moins une valeur admissible car la configuration partielle amène à au moins un objet faisable, $\mathbf{o}$. Ces valeurs admissibles sont calculées à la ligne 8. S'il y a une seule valeur admissible, il s'agit d'une recommandation triviale (ligne 9) qui ne sera pas prise en compte dans le calcul du taux de succès. S'il y a plusieurs valeurs admissibles, l'algorithme de recommandation doit fournir une recommandation r pour Next à partir de la configuration partielle $\mathbf{u}$ (ligne 11). Il y a deux possibilités : soit r est effectivement la valeur de Next dans le produit $\mathbf{o}$, i.e. $r = \mathbf{o}[\mathsf{Next}]$, auquel cas on considère que c'est un succès (ligne 12), soit la recommandation est différente, i.e. $r \neq \mathbf{o}[\mathsf{Next}]$, auquel cas c'est un échec (ligne 13).

Afin d'augmenter le nombre de recommandations sur lesquelles évaluer l'algorithme de recommandation, nous utiliserons dix ordres d'attributs pour chaque véhicule de l'ensemble de test et générons donc, à partir de chaque véhicule du jeu de test, dix sessions de configuration.

Algorithm 13: Protocole d'évaluation d'algorithme de recommandation en configuration interactive

Input: Un algorithme de recommandation $\mathcal{A}$ et le jeu de test *Htest*
Output: Le taux de succès
main :

```
1  succes ← 0
2  erreur ← 0
3  trivial ← 0
4  for each o ∈ Htest do
5      session ← une séquence d'attribut dans un ordre aléatoire
6      u ← tuple vide
7      for each Next ∈ session do
8          admissibles ← l'ensemble des valeurs admissibles pour Next
             sachant Assigned = u
9          if |admissibles| = 1 then incrémente trivial
10         else
11             r ← la valeur recommandée (parmi admissibles) par A pour
                 Next sachant Assigned = u
12             if r = o[Next] then incrémente succes
13             else incrémente erreur
14         u[Next] ← o[Next]
15 return succes/(succes + erreur)
```

7.2.1 Oracle

Afin d'interpréter facilement les résultats de la validation croisée, nous souhaitons estimer le plus haut taux de succès atteignable. Notre idée est la suivante : si nous pouvions utiliser un algorithme qui connaît déjà le jeu de test, il pourrait faire de meilleures recommandation que les autres méthodes car il aurait accès à l'ensemble des véhicules sur lequel il serait évalué. Il recommanderait alors pour l'attribut Next, sachant les valeurs déjà assignées $\mathbf{u}$, la valeur la plus probable dans le sous-ensemble du jeu de test qui vérifie $\mathbf{u}$. Plus précisément, pour tout $x \in \underline{\mathsf{Next}}$, cet algorithme approcherait $p(x \mid \mathbf{u})$ par $\#(x, \mathbf{u})/\#(\mathbf{u})$, où $\#(\mathbf{u})$ désigne le nombre d'objets compatibles avec $\mathbf{u}$ dans le jeu de test. On remarque que $\#(\mathbf{u})$ 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 14. Son taux de succès est maximal mais n'est pas égal à 100 % étant donné qu'il y a une certaine variabilité entre les utilisateurs.

Proposition 7.2.1. *La probabilité de succès de l'algorithme Oracle est supérieure ou égale à celle de tout algorithme.*

Démonstration. 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 protocole), 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.

Algorithm 14: Oracle

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

```
succes ← 0
erreur ← 0
for each o in H_test do
    session ← une séquence d'affectations attribut-valeur de o dans un
     ordre aléatoire
    Assigned ← ∅
    u ← tuple vide
    for each (Next, x) ∈ session do
        admissibles ← l'ensemble des valeurs admissibles pour Next
         sachant Assigned = u
        r ← argmax_{r∈admissibles} #(ru)
        if |admissible| = 1 then incrémente trivial
        else if r = x then incrémente succes
        else incrémente erreur
        Assigned ← Assigned ∪ {Next}
        u[Next] ← x
return succes / (succes + erreur)
```

Or, c'est exactement ce que fait l'Oracle. Donc l'Oracle est la méthode qui maximise la probabilité de succès. □

Nous avons calculé pour chaque jeu de test un taux de réussite maximal grâce à l'Oracle. 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.

7.3 Expériences avec les algorithmes de la littérature

Nos premières expériences ont pour objectif d'évaluer les méthodes que nous avons détaillées dans le chapitre 6 : le réseau bayésien, le réseau bayésien naïf, le votant majoritaire pondéré, le votant bayésien naïf et le choix le plus populaire.

Toutes les expériences sont réalisées 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. Le code source est disponible à l'adresse suivante : `https://github.com/PFGimenez/PhD`. L'apprentissage de réseau bayésien est effectué avec la librairie *bnlearn* [Scu10]. Nous utilisons un compilateur de contraintes nommé SaLaDD [Sch15a]. SaLaDD compile les contraintes sous la forme d'un SLDD et sa rapidité le rend compatible avec une utilisation en ligne qui impose des temps de réponse faibles.

Les questions que nous allons aborder sont :

1. pour les méthodes basées sur les k plus proches voisins, quelles sont les valeurs de k qui maximisent le taux de réussite ?
2. lors de l'apprentissage en vue d'une évaluation qui prend en compte les contraintes, est-il préférable d'apprendre avec les éléments qui ne satisfont pas les contraintes ou vaut-il mieux les ignorer ?
3. comment évolue le taux d'erreur en fonction du nombre d'attributs affectés ?
4. comment évolue le temps de recommandation en fonction du nombre d'attributs affectés ?
5. partitionner l'ensemble d'apprentissage en clusters permet-il d'augmenter le taux de succès ?
6. comment la taille du jeu d'apprentissage influence-t-elle le taux de succès de la recommandation ?
7. à quel point l'ajout de contraintes modifie-t-il le taux de succès ?

7.3.1 Influence de la taille du voisinage

Les algorithmes basés sur la sélection des k plus proches voisins dépendent du paramètre k. La question de choisir une bonne valeur pour k n'est pas abordée par [CGO+02] alors que cela peut avoir une influence directe sur les performances des algorithmes de recommandation.

En effet, le temps de recommandation d'un algorithme basé sur voisinage est la somme du temps nécessaire pour trouver les k plus proches voisins et la durée de l'agrégation (qui diffère selon l'algorithme). Notre implémentation utilise un algorithme qui trouve les k plus proches voisins dans un jeu d'apprentissage de $|\mathcal{H}|$ exemples en $O(k|\mathcal{H}|)$.

Les algorithmes basés sur les k plus proches voisins sont très utilisés en classification et des méthodes efficaces de recherche ont été créées en effectuant un prétraitement sur l'ensemble des exemples en amont de la requête de classification. Malheureusement, ces méthodes ne sont pas applicables dans le cas de la recommandation en configuration interactive car la distance utilisée pour trouver les plus proches voisins est $d_{\mathbf{Assigned}}$ et dépend donc de l'ensemble **Assigned** (cf. l'équation (6.1)). Or, nous ne pouvons pas appliquer ce prétraitement pour chaque distance $d_{\mathbf{Assigned}}$ car pour n attributs, il y a 2^n possibilités pour **Assigned**.

Afin d'étudier l'influence de k sur le temps et le taux de succès de la recommandation, nous avons mené une étude empirique sans contrainte : les figures 7.1 et 7.2 présentent respectivement le taux d'erreur moyen et le temps de recommandation du Votant Bayésien Naïf sur *Renault-44* et *Renault-48* en fonction de k, le nombre de voisins. Les résultats pour les autres algorithmes basés sur les voisins (Votant majoritaire pondéré et Choix le plus populaire) sont similaires.

Nous pouvons clairement observer une courbe en forme de cloche : trop peu de voisins, et le Votant bayésien naïf n'a pas assez d'exemples pour effectuer une prédiction correcte. S'il y a trop de voisins, le voisinage commence à contenir des exemples qui peuvent être trop éloignés de la configuration courante, ce qui dégrade la qualité de la recommandation. Au final, le taux d'erreur minimum est obtenu pour k entre 20 et 50.

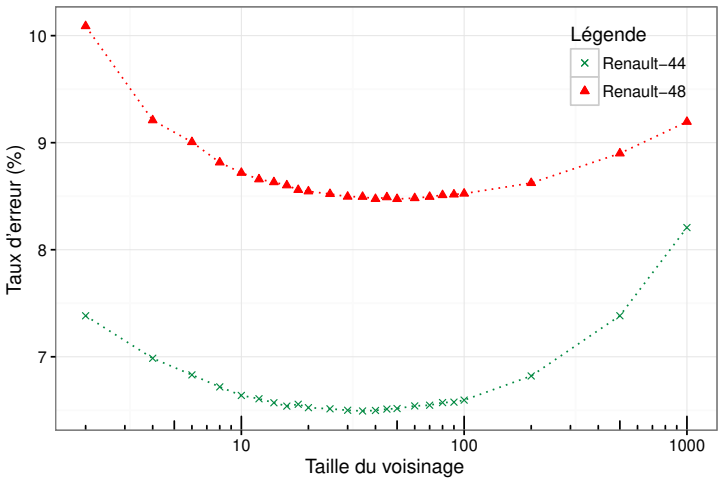


FIGURE 7.1 – Expérience de la section 7.3.1. Taux d'erreur moyen du Votant Bayésien Naïf sur *Renault-44* et *Renault-48* en fonction de k, le nombre de voisins.

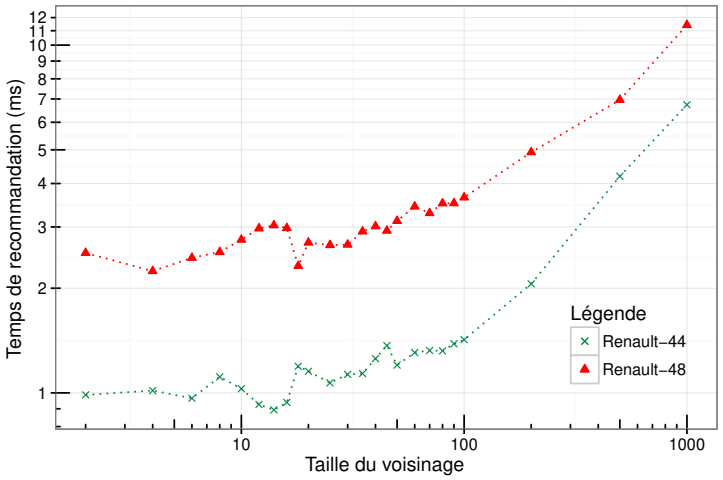


FIGURE 7.2 – Expérience de la section 7.3.1. Temps de recommandation du Votant Bayésien Naïf sur *Renault-44* et *Renault-48* en fonction de k, le nombre de voisins.

Le temps de recommandation est irrégulier jusqu'à $k = 100$ puis devient clairement plus important pour des valeurs de k plus grandes.

Pour la suite des expériences, nous fixerons $k = 20$, une valeur adaptée aux différents jeux de données.

7.3.2 Vaut-il mieux apprendre avec les exemples infaisables ?

Quand bien même les futurs produits qui seront vendus satisferaient les contraintes, les jeux de données peuvent contenir des exemples qui ne les sa-

tisfont pas. La question de savoir s'il est préférable ou non de les utiliser lors de l'apprentissage n'est pas triviale. D'un côté, ces exemples, bien qu'ils ne satisfassent pas les contraintes, contiennent des informations sur les préférences des utilisateurs. D'un autre côté, ils peuvent nuire au taux de succès de la recommandation car ils ne sont pas représentatifs des produits qui seront configurés. C'est pour cela que nous avons effectué une vérification expérimentale de la pertinence de la conservation des exemples qui ne satisfont pas les contraintes. Lors de cette expérience, toutes les recommandations porteront sur des objets faisables.

Nous avons conduit ces expériences sur les trois jeux de données, *Renault-44*, *Renault-48* et *Renault-87*. Les résultats sont présentés respectivement aux tables 7.1, 7.2 et 7.3.

Taux d'erreur	Tous exemples	Exemples faisables
Votant bayésien naïf	19.90 %	18.13 %
Votant majoritaire pondéré	20.14 %	19.24 %
Choix le plus populaire	20.39 %	19.12 %
Réseau bayésien	19.14 %	18.28 %

TABLE 7.1 – Le taux d'erreur de quelques algorithmes de recommandation sur *Renault-44* en apprenant avec tous les exemples ou seulement les exemples faisables.

Taux d'erreur	Tous exemples	Exemples faisables
Votant bayésien naïf	23.71 %	20.29 %
Votant majoritaire pondéré	25.36 %	24.38 %
Choix le plus populaire	26.36 %	23.59 %
Réseau bayésien	24.47 %	20.91 %

TABLE 7.2 – Le taux d'erreur de quelques algorithmes de recommandation sur *Renault-48* en apprenant avec tous les exemples ou seulement les exemples faisables.

Taux d'erreur	Tous exemples	Exemples faisables
Votant bayésien naïf	7.97 %	9.11 %
Votant majoritaire pondéré	7.99 %	9.03 %
Choix le plus populaire	8.40 %	9.52 %
Réseau bayésien	12.51 %	13.17 %

TABLE 7.3 – Le taux d'erreur de quelques algorithmes de recommandation sur *Renault-87* en apprenant avec tous les exemples ou seulement les exemples faisables.

Alors que le taux d'erreur diminue lorsque seuls les exemples faisables sont utilisés pour *Renault-44* et *Renault-48*, il augmente pour *Renault-87*. Nous

pouvons également observer que le changement de taux de succès (que ce soit un gain ou une dégradation) ne semble dépendre que du jeu de données et pas de l'algorithme utilisé. Étant donné que les différences de taux de succès sont significatives, nous ne pouvons pas conclure dans le cas général.

Par la suite, dans les expériences avec contraintes, nous utiliserons toujours tous les exemples, faisables et infaisables.

7.3.3 Évolution du taux d'erreur

Les figures 7.3, 7.4 et 7.5 présentent le taux d'erreur en l'absence de contraintes des approches basées sur les réseaux bayésiens (réseau bayésien et réseau bayésien naïf), des méthodes basées sur les plus proches voisins et ainsi que de l'Oracle.

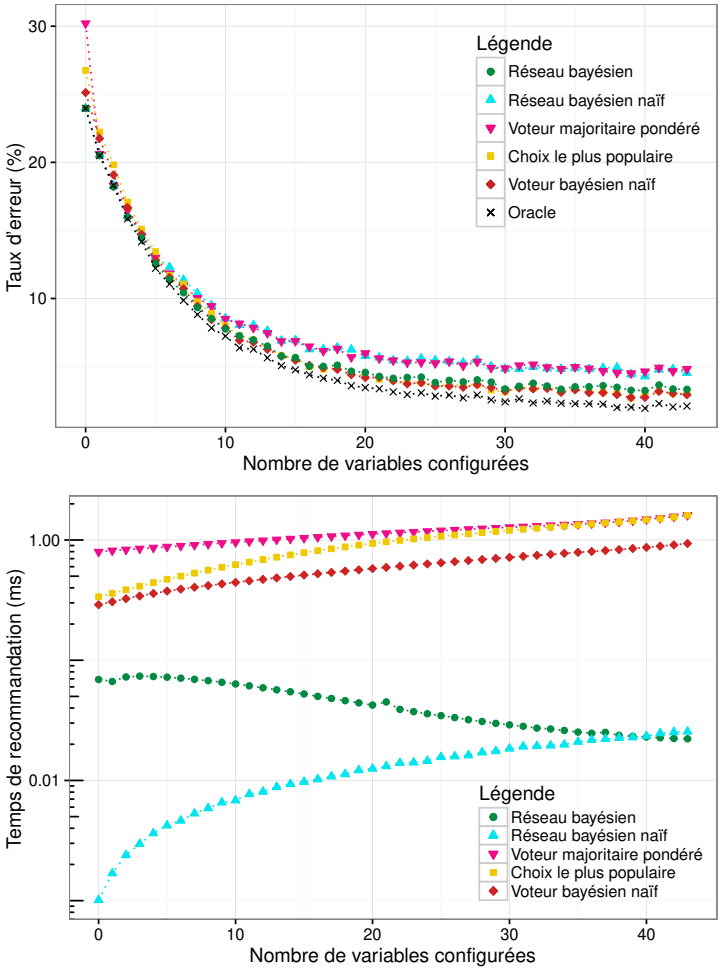


FIGURE 7.3 – Expérience de la section 7.3.3. Taux d'erreur moyen et temps de recommandation sur *Renault-44*

Nous pouvons constater que le réseau bayésien naïf, qui fait de fortes hypothèses d'indépendance, a le plus faible taux de succès. Ce n'est pas étonnant,

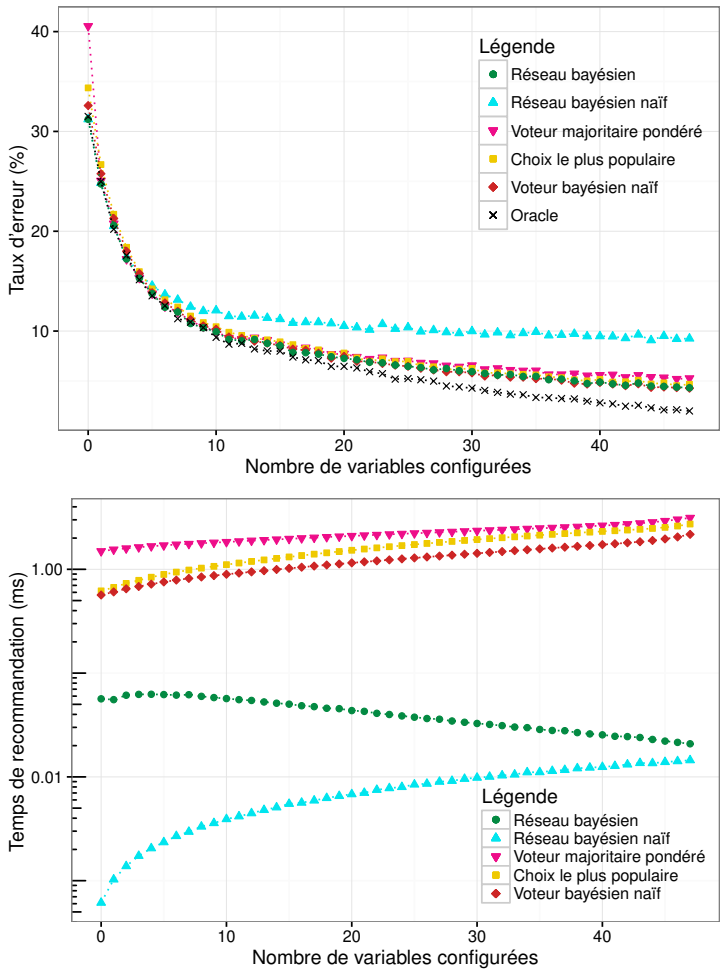


FIGURE 7.4 – Expérience de la section 7.3.3. Taux d'erreur moyen et temps de recommandation sur *Renault-48*

car les hypothèses d'indépendance entre attributs ne sont pas vérifiées. On peut par contre remarquer que lorsque le réseau bayésien naïf est appris à partir d'un voisinage (Votant bayésien naïf et Choix le plus populaire), le taux de succès est bien moins mauvais ; en effet, la recherche de voisins permet déjà de capturer certaines dépendances en apprenant uniquement à partir d'objets similaires à celui en cours de configuration.

Trois algorithmes de recommandation sont très précis, à seulement quelques points du taux de succès de l'Oracle : le réseau bayésien, le Votant bayésien naïf et le Choix le plus populaire. L'écart avec le taux de succès de l'Oracle augmente avec le nombre d'attributs assignés car les performances de l'Oracle sont de moins en moins atteignables. En effet, la prédiction de l'Oracle se base sur le jeu de test, qui inclut le produit à partir duquel la session de configuration est simulée. Quand peu d'attributs sont affectés dans la configuration courante, l'Oracle fait sa prédiction à partir de l'ensemble relativement grand des objets

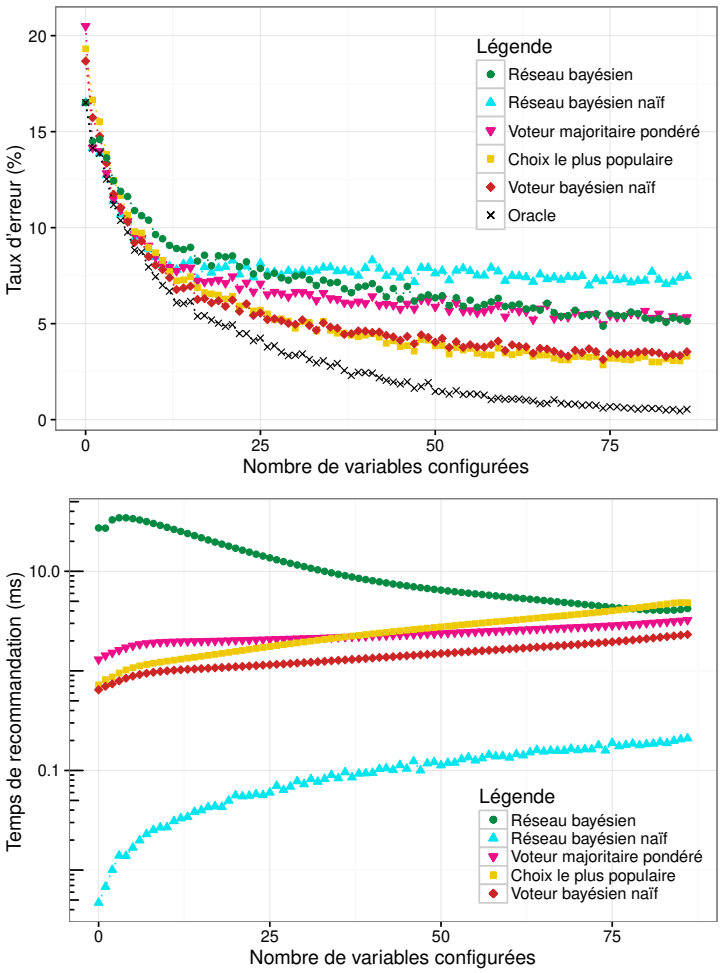


FIGURE 7.5 – Expérience de la section 7.3.3. Taux d'erreur moyen et temps de recommandation sur *Renault-87*

compatibles avec la configuration courante. Par contre, quand un grand nombre d'attributs sont affectés, l'ensemble des objets compatibles avec la configuration partielle est bien plus petit, voire même réduit à l'objet à partir duquel la session est simulée (auquel cas l'Oracle aura un taux de succès maximal). On peut donc en conclure que le surapprentissage de l'Oracle est particulièrement visible sur le jeu de données *Renault-87* qui contient bien plus d'attributs que les jeux de données *Renault-44* et *Renault-48*.

7.3.4 Temps de recommandation en fonction du nombre d'attributs assignés

Le temps CPU en l'absence de contraintes, présenté aux figures 7.3, 7.4 et 7.5, répartit clairement les algorithmes de recommandation en deux groupes : ceux qui font un apprentissage hors ligne (réseau bayésien, réseau bayésien naïf)

et ceux qui calculent le voisinage en ligne (Votant bayésien naïf, Choix le plus populaire, Votant majoritaire pondéré). Ce premier groupe est plus rapide d'un ordre de magnitude que le second groupe sur les jeux de données *Renault-44* et *Renault-48* (voire parfois deux ordres de grandeur). Cette différence significative peut s'expliquer par le temps d'extraction des k plus proches voisins. Nous pouvons remarquer que le temps d'extraction des k plus proches voisins est peu sensible au nombre d'attributs du problème ; il reste faible sur le jeu de données *Renault-87*.

On peut remarquer que sur ces trois jeux de données réels, tous les algorithmes de recommandation ont un temps de recommandation compatible avec une utilisation en ligne avec jamais plus de 50 ms. Sans surprise, le réseau bayésien naïf est le plus rapide (moins de 0.05 ms). Le temps nécessaire au réseau bayésien est du même ordre de grandeur : moins de 0.1 ms pour *Renault-44* et *Renault-48* et moins de 0.4 ms pour *Renault-87*.

7.3.5 Vaut-il mieux utiliser des clusters ?

Une technique bien connue pour augmenter le taux de succès en apprentissage automatique est le clustering, qui regroupe les objets en groupes homogènes et apprend les préférences sur chaque cluster. Le clustering a été utilisé avec succès dans des méthodes de recommandation par filtrage collaboratif [UF98]. C'est pour cela que nous avons décidé de tester empiriquement si le clustering peut améliorer le taux de succès des algorithmes de recommandation que nous évaluons dans le contexte de la configuration interactive.

Lors de l'apprentissage de préférences à partir d'exemples positifs, la littérature fait souvent l'hypothèse que l'ensemble des exemples provient d'un unique utilisateur avec une certaine relation de préférences. Néanmoins, dans le cas des données Renault, chaque client achète trop rarement une voiture pour avoir un historique de ventes assez important pour nous renseigner sur ses préférences. À la place, nous allons regrouper les utilisateurs en groupes homogènes (clusters) et apprendre les préférences de chaque groupe.

Construction des clusters

La construction des clusters est une tâche classique. Nous avons utilisé la célèbre méthode des k moyennes qui peut être expliquée de la manière suivante. Cette méthode cherche k objets qui deviendront les centres des clusters, et établit une partition des éléments de l'ensemble des exemples en k ensembles de manière à minimiser la distance intra-cluster, i.e. la distance moyenne entre les objets d'un cluster et le centre de ce cluster.

Nous avons implémenté l'algorithme de Lloyd qui est une approche itérative initialisée avec k centres tirés aléatoirement parmi l'ensemble d'apprentissage et qui améliore progressivement ces centres. Il s'agit d'un algorithme heuristique qui converge vers un optimum local. C'est pour cette raison que nous relançons cet algorithme 500 fois en ne gardant que le meilleur ensemble de clusters.

Nous allons ensuite apprendre un modèle de préférences par cluster ; c'est-à-dire que s'il y a k clusters, il y aura k modèles de préférences correspondants.

Sélection du cluster lors d'une recommandation

Pour recommander une valeur de Next en connaissant la configuration partielle $\mathbf{u}$, on utilise le modèle de préférence associé au cluster le plus proche de $\mathbf{u}$.

Dans le cas des méthodes basées sur les plus proches voisins, nous utiliserons une distance de Hamming pour trouver le cluster le plus adapté. Étant donné que $\mathbf{u}$ est une configuration partielle, on utilisera une distance de Hamming restreint aux attributs **Assigned**. Autrement dit, si on note $\mathbf{c}_1, \ldots, \mathbf{c}_k$ les k centres des clusters, le numéro du cluster utilisé est :

$$\underset{i\in[1,k]}{\operatorname{argmin}}\, d_{\mathbf{Assigned}}(\mathbf{u}, \mathbf{c}_i) = \underset{i\in[1,k]}{\operatorname{argmin}} \,|\, \{X \in \mathbf{Assigned} \mid \mathbf{c}_i[X] \neq \mathbf{u}[X]\} \,|$$

Dans le cas des réseaux bayésiens, nous n'allons pas utiliser la distance de Hamming mais la vraisemblance de $\mathbf{u}$ afin d'utiliser le réseau bayésien le plus vraisemblable. Autrement dit, si on note $p_1, \ldots, p_k$ les lois de probabilité représentées par les k réseaux bayésiens, le numéro du réseau bayésien utilisé pour la recommandation est :

$$\underset{i\in[1,k]}{\operatorname{argmax}}\, p_i(\mathbf{u})$$

Le choix du cluster utilisé est effectué à chaque recommandation. Ceci signifie que pour une session de configuration qui contient plusieurs recommandations (une par attribut de $\mathcal{X}$), des modèles de préférences issus de clusters différents pourront être utilisés alternativement.

Nous avons essayé plusieurs nombres de clusters, de un à trois. Le taux d'erreur de la recommandation en fonction du nombre de clusters, en l'absence de contraintes, est présenté à la figure 7.6. Nous pouvons observer une dégradation du taux de succès lorsqu'on augmente le nombre de clusters utilisés, et ce pour tous les algorithmes de recommandation. Par exemple, la précision du réseau bayésien naïf varie :

- de 8.55 % de taux d'erreur pour un cluster à 10.06 % pour trois clusters sur *Renault-44* ;
- de 6.52 % de taux d'erreur pour un cluster à 9.74 % pour trois clusters sur *Renault-48* ;
- de 5.13 % de taux d'erreur pour un cluster à 6.31 % pour trois clusters sur *Renault-87*.

La figure 7.7 montre le taux d'erreur du réseau bayésien et du Votant bayésien naïf avec un ou trois clusters en fonction du nombre d'attributs affectés. Le comportement du taux de succès est le même pour le réseau bayésien et le Votant bayésien naïf, ce qui suggère que cette perte provient du clustering lui-même et pas des algorithmes de recommandation.

La chute du taux de succès des algorithmes basés sur le voisinage due au clustering peut être expliquée de la manière suivante. Ces algorithmes font une recommandation à partir des k plus proches voisins de $\mathbf{u}$: si la recommandation avec clustering est différente de la recommandation sans clustering, cela signifie que les voisins trouvés sans clustering sont différents des voisins trouvés avec

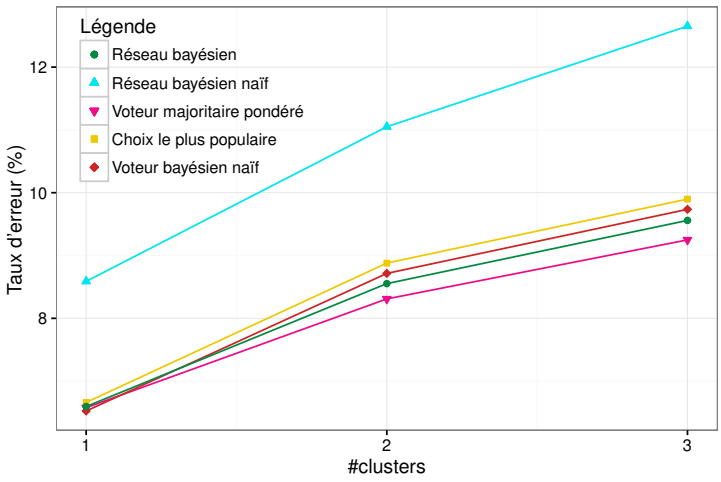


FIGURE 7.6 – Expérience de la section 7.3.5. Taux d'erreur moyen sur toute la session sur *Renault-44* sans contraintes en fonction du nombre de clusters.

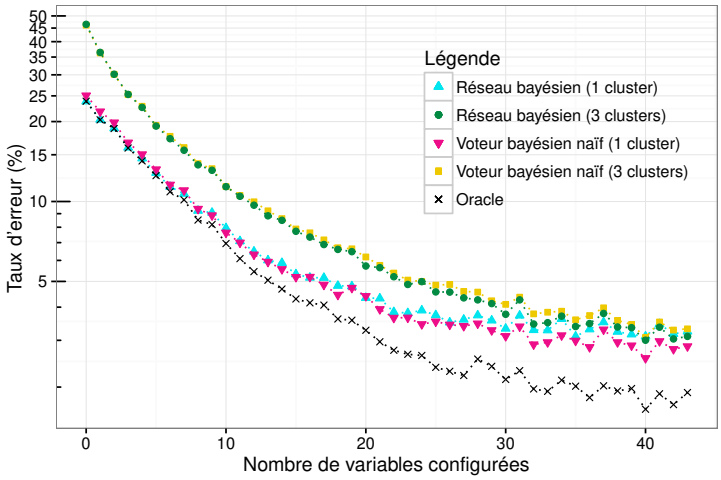


FIGURE 7.7 – Expérience de la section 7.3.5. Taux d'erreur moyen sur *Renault-44* pour les réseaux bayésiens et le Votant bayésien naïf avec un et trois clusters. L'échelle du taux d'erreur est logarithmique.

clustering ; ou, autrement dit, les plus proches voisins $\mathbf{u}$ ne sont pas tous dans le cluster le plus proche de $\mathbf{u}$. Pour expliquer l'origine de ce phénomène, il est important de rappeler que ces algorithmes de recommandation recherchent les voisins les plus proches par rapport à la distance $d_{\mathbf{Assigned}}$ alors que le clustering regroupent les objets selon la distance $d_{\mathcal{X}}$. C'est pour cela que les voisins les plus proches de $\mathbf{u}$ ne sont pas nécessairement dans le cluster calculé lors du clustering. Ce problème affecte également les performances du réseau bayésien, car les exemples du cluster de $\mathbf{u}$ ne sont pas forcément les plus proches de $\mathbf{u}$ au sens de $d_{\mathbf{Assigned}}$. C'est pour cela que, dans les résultats expérimentaux présentés à la

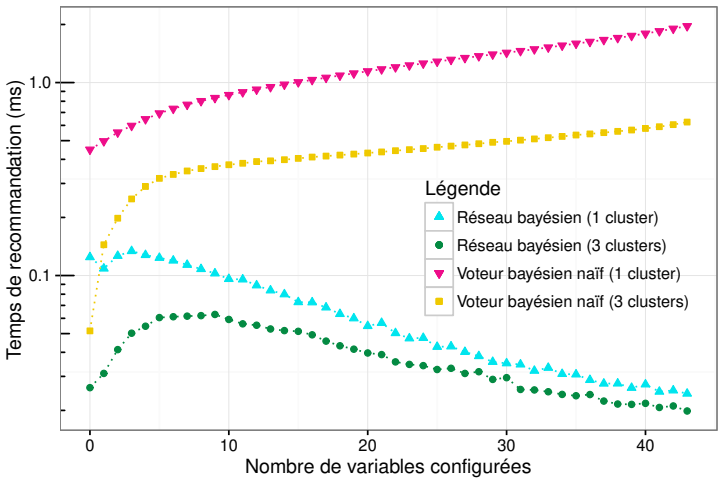


FIGURE 7.8 – Expérience de la section 7.3.5. Temps de recommandation sur *Renault-44* pour les réseaux bayésiens et le Votant bayésien naïf avec un et trois clusters.

figure 7.7, plus le nombre d'attributs assignés est grand, plus l'écart dans le taux d'erreur entre la version avec clustering et la version sans clustering est faible : en effet, la distance $d_{\mathbf{Assigned}}$ converge vers $d_{\mathcal{X}}$ quand **Assigned** converge vers $\mathcal{X}$.

Le clustering a néanmoins des avantages : la figure 7.8 montre que pour les algorithmes basés sur le voisinage et le réseau bayésien, l'utilisation du clustering réduit le temps de recommandation. Pour les algorithmes basés sur le voisinage, cela provient simplement du fait que les voisins sont recherchés dans un cluster, i.e. un sous-ensemble du jeu d'apprentissage. Pour le réseau bayésien, nous pensons que la taille réduite du cluster favorise des modèles plus simples, ce qui réduit le temps de l'inférence. La partie suivante présente des expériences complémentaires sur la taille du jeu d'apprentissage pour les algorithmes basés sur le voisinage.

Par la suite, nous n'utiliserons aucun clustering.

7.3.6 Influence de la taille du jeu d'apprentissage

L'inconvénient des méthodes basées sur le voisinage est que leur performance semble dépendre de la taille du jeu d'exemples : plus le jeu d'exemples est grand, plus le taux de succès peut être grand mais plus le temps de recommandation pour y arriver est important. Pour confirmer ceci, nous avons conduit une expérience en faisant varier la taille du jeu d'apprentissage, depuis un jeu complet à un centième de celui-ci. Nous n'avons pas inclus les contraintes dans cette expérimentation. Les résultats sont présentés aux figures 7.9 et 7.10.

Le réseau bayésien tout comme le Votant bayésien naïf souffrent d'une dégradation du taux de succès quand la taille du jeu d'exemples est réduit. Néanmoins, nous pouvons remarquer la dégradation du réseau bayésien est modérée (le taux d'erreur augmente de 17.20 % à 18.47 %) alors qu'elle est bien plus importante pour le Votant bayésien naïf (le taux d'erreur augmente de

16.76 % à 19.76 %). Visiblement, le taux de succès du réseau bayésien dépend bien moins de la taille du jeu d'apprentissage que le taux de succès du Votant bayésien naïf.

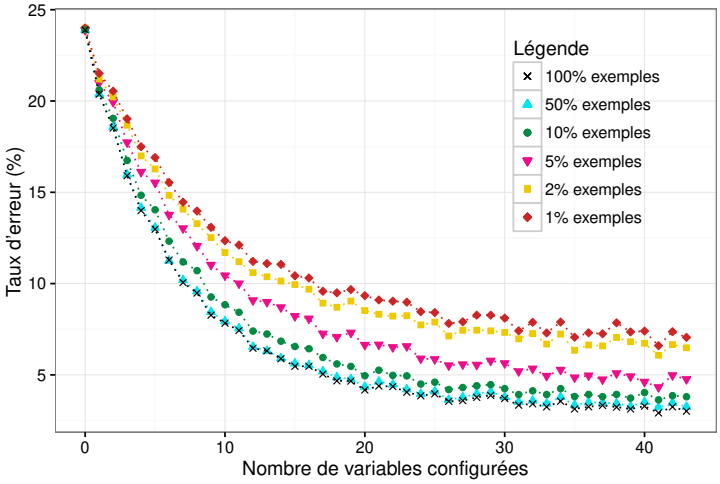


FIGURE 7.9 – Expérience de la section 7.3.6. Taux d'erreur moyen du réseau bayésien sur le jeu de données *Renault-44* en fonction de la taille du jeu d'apprentissage.

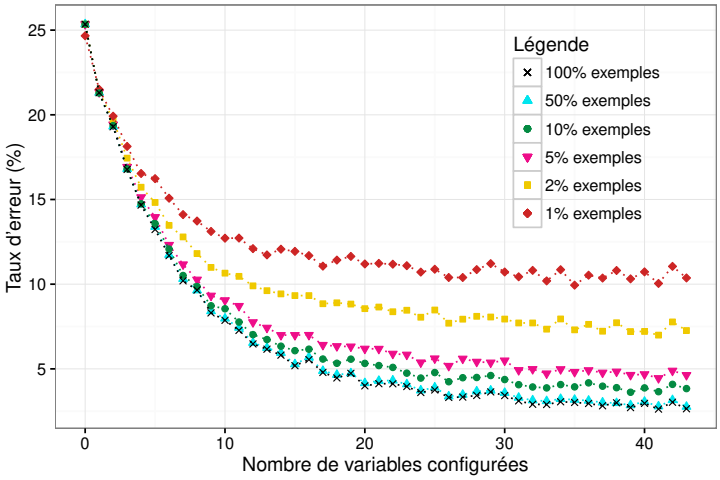


FIGURE 7.10 – Expérience de la section 7.3.6. Taux d'erreur moyen du Votant bayésien naïf sur le jeu de données *Renault-44* en fonction de la taille du jeu d'apprentissage.

7.3.7 Influence des contraintes

Afin d'étudier l'influence des contraintes sur le taux de succès de la recommandation, nous avons créé deux versions allégées du CSP des contraintes de

Renault-48 : un CSP avec environ 30 % des contraintes et un CSP avec environ 60 % des contraintes.

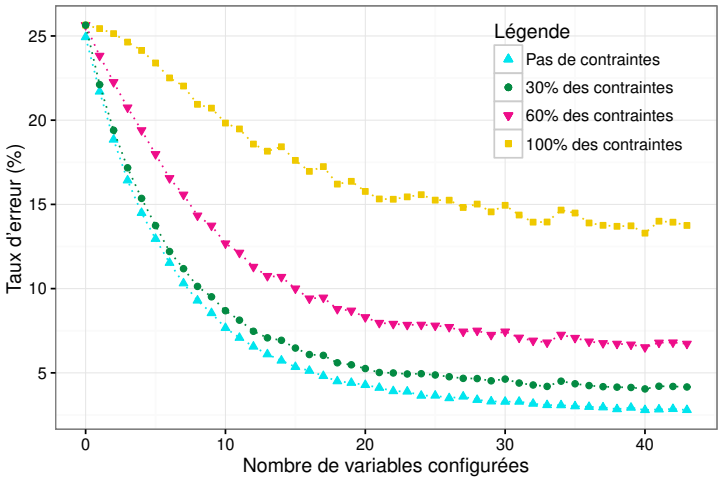


FIGURE 7.11 – Expérience de la section 7.3.7. Taux d'erreur moyen du Votant bayésien naïf avec différents CSP sur *Renault-48*.

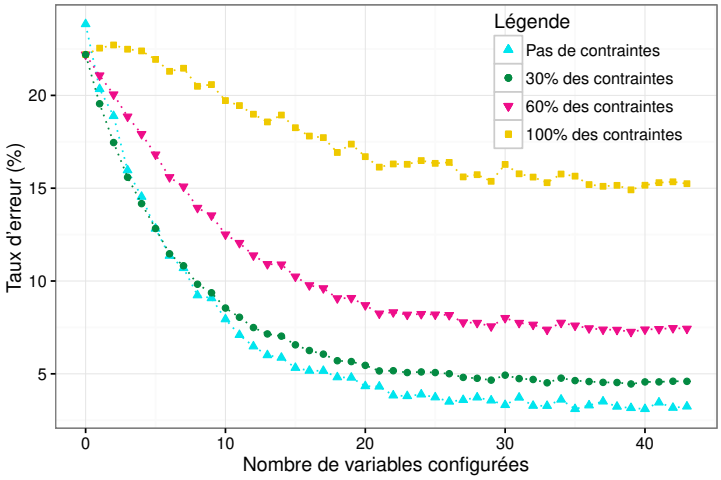


FIGURE 7.12 – Expérience de la section 7.3.7. Taux d'erreur moyen du réseau bayésien avec différents CSP sur *Renault-48*.

Le taux d'erreur en fonction du pourcentage de contraintes est représenté aux figures 7.11 et 7.12 : on peut voir une dégradation claire du taux de succès dans les problèmes les plus contraints, et ce pour les deux algorithmes de recommandations. Leur taux d'erreur global est résumé dans la table 7.4. La différence de taux d'erreurs des deux algorithmes est négligeable : ils sont similairement affectés par les contraintes.

Ratio de contraintes	Votant bayésien naïf	Réseau bayésien
0 %	6.49 %	6.60 %
30 %	8.40 %	8.14 %
60 %	12.30 %	11.97 %
100 %	19.55 %	19.14 %

TABLE 7.4 – Taux d'erreur du Votant bayésien naïf et du réseau bayésien en fonction du ratio de contraintes utilisées sur *Renault-48*

7.3.8 Conclusion

Nos expériences montrent que les réseaux bayésiens sont compatibles avec une inférence en ligne en dépit du fait que calculer la requête MAP soit un problème NP-difficile. De plus, leur taux de succès est proche du meilleur possible. Les réseaux bayésien naïf ont un taux de succès médiocre mais sont bien plus rapides. Enfin, les méthodes basées sur le voisinage (Votant bayésien naïf, Votant majoritaire pondéré, Choix le plus populaire) ont un taux de succès proche de celui des réseaux bayésiens et un temps de recommandation indépendant du nombre d'attributs du problème – il dépend par contre fortement de la taille de l'ensemble d'apprentissage.

7.4 Expériences relatives à DRC

L'objectif de cette section est d'évaluer l'algorithme DRC dans le contexte de la recommandation en configuration interactive. Nous avons utilisé la méthode *hc* du *package* R *bnlearn* [Scu10] 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 [Kut13] et notre propre implémentation de RC. Dans les expériences suivantes, RC utilise un *dtree* qui a été généré par la méthode décrite par [DH01] 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 [KK98]. Le nombre minimum d'exemples nécessaires pour estimer une probabilité à partir de l'historique est fixé à 50.

Les figures 7.13 et 7.14 indiquent le taux d'erreur de la recommandation par inférence bayésienne classique (Jointree et RC qui ont le même taux d'erreur) et par DRC en fonction du nombre d'attributs configurés. L'Oracle est indiqué comme une ligne idéale. Les figures 7.15 et 7.16 indiquent le temps de recommandation moyen de Jointree, RC et DRC en fonction du nombre d'attributs configurés.

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 méthodes ont des taux de succès proches de la courbe idéale de l'Oracle.

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 d'attributs configurés, il peut y avoir assez de configurations dans l'historique de ventes pour faire une estimation directe. Lorsqu'il y a plus d'attributs, le

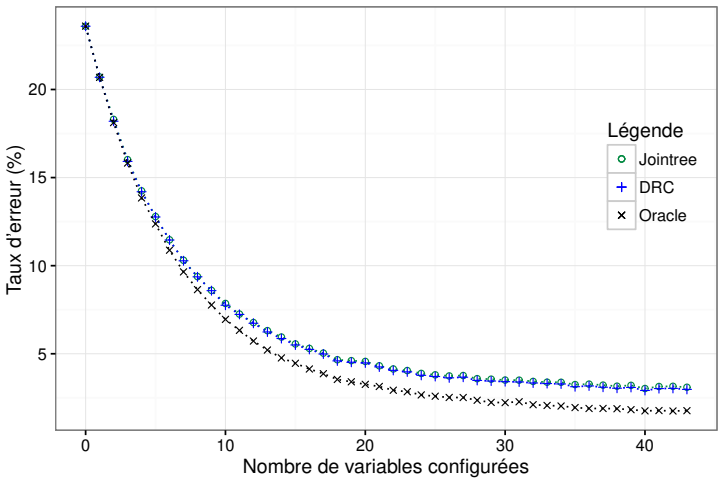


FIGURE 7.13 – Expérience de la section 7.4. Taux d'erreur moyen sur les jeux de données *Renault-44*. Les courbes de Jointree et de DRC sont superposées.

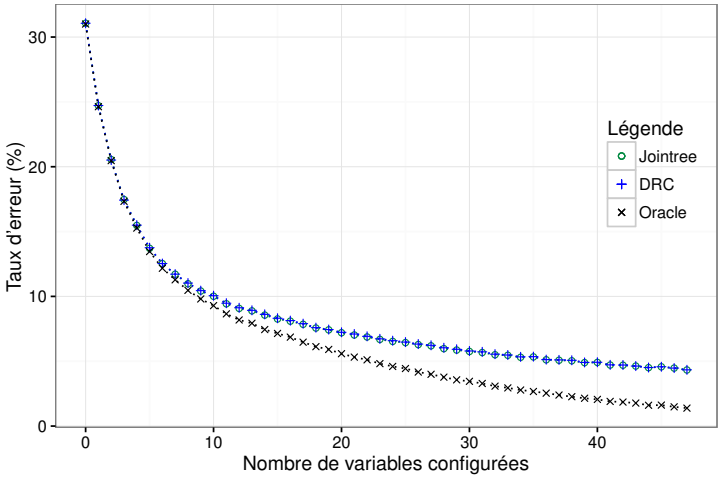


FIGURE 7.14 – Expérience de la section 7.4. Taux d'erreur moyen sur les jeux de données *Renault-48*. Les courbes de Jointree et de DRC sont superposées.

temps de calcul augmente rapidement car DRC descend plus bas dans l'arbre de décomposition ternaire tout en conditionnant à chaque fois des attributs. À la fin de la recommandation, étant donné que la majorité des attributs sont affectés, DRC a besoin de conditionner moins de attributs, 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 compatible une utilisation en ligne : moins de 10 ms pour *Renault-48* et moins de 30 ms pour *Renault-44* (voir figures 7.15 et 7.16).

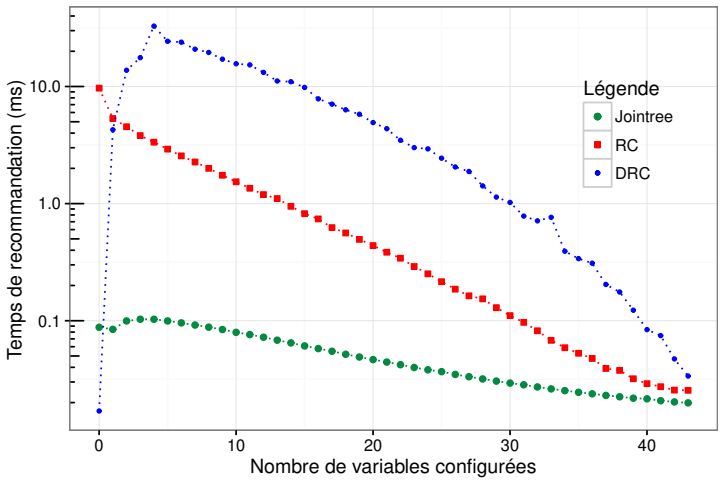


FIGURE 7.15 – Expérience de la section 7.4. Temps moyen de recommandation sur les jeux de données *Renault-44*

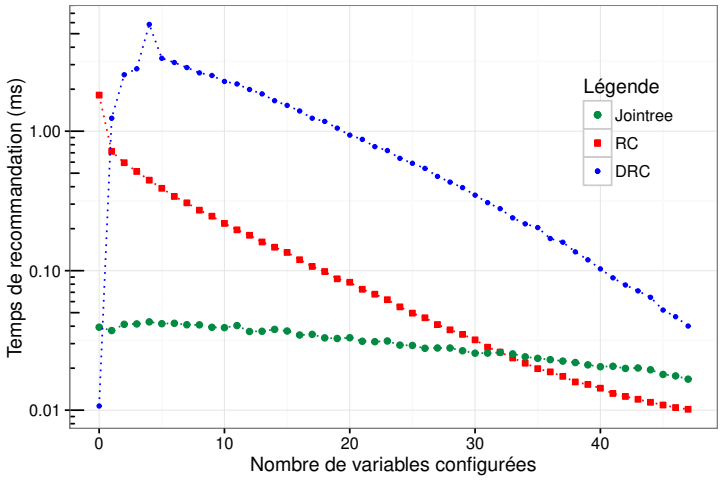


FIGURE 7.16 – Expérience de la section 7.4. Temps moyen de recommandation sur les jeux de données *Renault-48*

7.4.1 Taux de succès avec mise à jour de l'historique

Les sections précédentes concluent sur l'intérêt des méthodes d'inférence bayésienne 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 *Renault-48* sont datées, ce qui nous permet de les classer en quatre périodes, notés P1, P2,

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

TABLE 7.5 – Taux d'erreur de DRC et de RB avec historique incrémental

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

[a] Sur le problème *Renault-48*, aucun produit vendu dans les périodes P1 et P2 ne satisfaisant les contraintes, les plis ne fournissent aucun scenario de test

TABLE 7.6 – Taux d'erreur de DRC et de RB avec historique incrémental avec contraintes

P3 et P4. Nous avons appris les dépendances entre attributs avec les ventes de la première période et nous avons mesuré le taux de succès des recommandations de RB et de DRC sur les périodes suivantes.

RB apprend le réseau bayésien (structure et tables) de la 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 10^{e} pli de P2, il utilise P1 et les neufs autres plis de P2, pour recommander sur le 10^{e} 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 10^{e}.

Les résultats de cette expérience sont présentés dans le tableau 7.5 pour la variante sans contraintes et dans le tableau 7.6 pour la variante avec contraintes. Nous donnons ici le taux d'erreur moyen, indépendamment du nombre de attributs déjà assignés.

Nous pouvons constater une baisse du taux de succès global 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 des acheteurs 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 celui 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.

7.5 Expériences relatives aux LP-trees

L'objectif de cette section est d'évaluer l'utilisation des k-LP-trees pour la recommandation en configuration interactive. Ces expériences ont été réalisées sur le jeu de données *Renault-48*. Nous rappelons que des expériences sur des données simulées ont été précédemment présentées dans le chapitre 5.

Tout d'abord, nous pouvons remarquer que les LP-trees linéaires ont un intérêt très limité dans le cas de la recommandation en configuration interactive ; en effet, étant donné que les règles de préférences d'un LP-tree linéaire sont toutes inconditionnelles (elles ne dépendent pas des valeurs des autres attributs), un LP-tree linéaire n'exploitera pas le produit partiellement configurée pour recommander une valeur. Or, comme nous l'avons vu à la section 7.3.3, la connaissance d'attributs déjà assignés permet d'améliorer significativement le taux de succès de la recommandation. C'est pour cette raison que nous allons nous intéresser aux k-LP-trees uniquement.

Le nombre minimal d'exemples dans une branche du k-LP-trees, que nous avons précédemment noté τ, est fixé à 20. Nous avons expérimenté avec élagage (avec la fonction de pénalité $\phi = 1$) et sans élagage. L'écart médian du taux d'erreur entre les deux méthodes est de 0.71 % : apprendre avec élagage a amené à un taux de succès plus bas que sans élagage dans plus de la moitié des cas. Les résultats présentés par la suite sont ceux obtenus sans élagage.

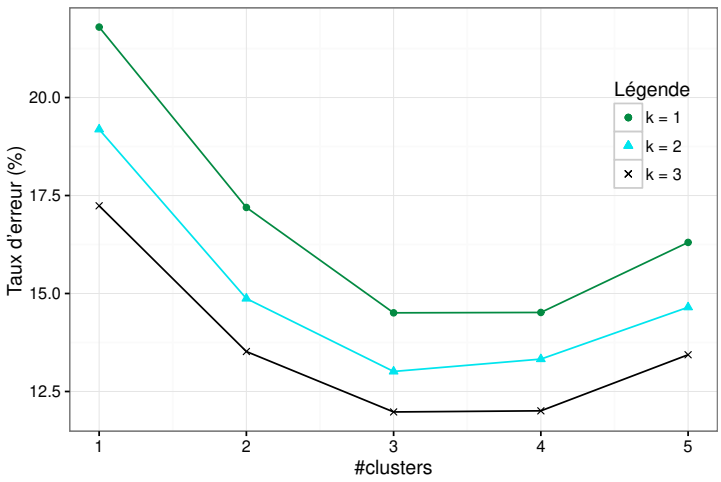


FIGURE 7.17 – Expérience de la section 7.5. Taux d'erreur sur *Renault-48* en fonction du nombre de clusters et du nombre maximum d'attributs par nœud, k.

Tout d'abord, nous pouvons remarquer à la figure 7.18 une corrélation positive entre le taux de succès et le rang moyen normalisé, ce qui suggère que notre score empirique, le rang empirique, est en effet corrélé avec un haut taux de succès expérimental, ce qui justifie notre démarche de minimiser le rang empirique moyen.

La taille du modèle appris (qui est la somme des nœuds des k-LP-trees appris pour chaque cluster), présentée à la figure 7.19, semble globalement indépendante du nombre de clusters, ce qui n'est pas étonnant car la taille des k-LP-trees est

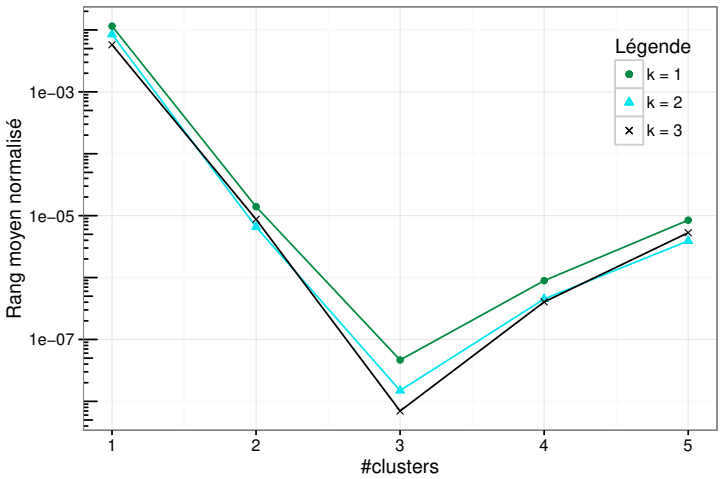


FIGURE 7.18 – Expérience de la section 7.5. Rang moyen normalisé sur *Renault-48* en fonction du nombre de clusters et du nombre maximum d'attributs par nœud, k.

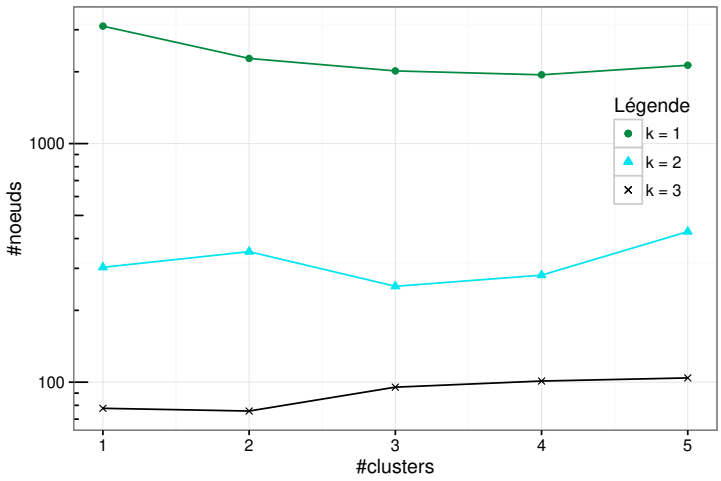


FIGURE 7.19 – Expérience de la section 7.5. Nombre de nœuds du LP-tree (non élagué) en fonction du nombre de clusters et du nombre maximum d'attributs par nœud, k, sur *Renault-48*

limitée par le nombre d'exemples à disposition. L'effet du paramètre k est par contre saisissant : on passe de 2300 nœuds pour $k = 1$ à 90 nœuds pour $k = 3$.

Nous avons également utilisé des clusters en suivant le protocole précédemment décrit dans 7.3.5 : l'ensemble d'apprentissage est réparti en plusieurs clusters et un LP-tree est appris pour chaque cluster. Pour effectuer une recommandation connaissant une configuration partielle $\mathbf{u}$, le LP-tree utilisé est celui qui correspond au cluster dont le centre est le plus proche de $\mathbf{u}$. Les taux d'erreur

sont présentés à la figure 7.17. Le taux d'erreur varie de 21.8 % (pour $k = 1$ et 1 cluster) à 12.0 % (pour $k = 3$ et 3 clusters) : ce faible taux d'erreur confirme l'intérêt pratique de cette approche. Sur ce même jeu de données, le meilleur résultat que nous ayons obtenu avec les autres méthodes de recommandation est un taux d'erreur de 8.5 % avec le réseau bayésien. Si nous avons arrêté notre expérience à $k = 3$, c'est que la complexité temporelle de l'apprentissage avec un terme en $\binom{n}{k}$ rend impraticables de plus grandes valeurs de k. Nous avons toutefois fait l'expérience avec $k = 4$ et $k = 5$, avec 3 clusters. Nous avons obtenu un taux d'erreur de 11.8 % pour $k = 4$ et 11.5 % pour $k = 5$, un gain faible par rapport à $k = 3$ (12.0 %) ; il semble donc que des valeurs de k plus grandes que 3 n'apportent pas grand chose dans ce problème.

La baisse du taux d'erreur avec l'ajout de clusters peut s'expliquer par le fait qu'il existe probablement des profils d'utilisateurs distincts ; étant donné que les k-LP-trees ont une expressivité limitée, ils ne peuvent pas représenter ces profils au sein d'un même modèle, ce qui nuit à leur précision. D'une manière générale, un ensemble de k-LP-trees a une expressivité plus importante qu'un seul k-LP-tree.

La figure 7.20 permet de comparer le comportement du taux d'erreur des k-LP-trees par rapport à deux autres méthodes, le réseau bayésien et le réseau bayésien naïf (les autres méthodes ont été omises par souci de lisibilité). En utilisant un seul cluster, nous pouvons remarquer que le taux d'erreur décroit bien moins rapidement que le taux d'erreur du réseau bayésien ou du réseau bayésien naïf. Ce taux d'erreur plus important que celui des deux autres méthodes peut s'expliquer de la manière suivante. Pour recommander une valeur de Next pour une configuration partielle $\mathbf{u}$, on parcourt le LP-tree depuis sa racine jusqu'à trouver un nœud N avec l'attribut Next ; quand ce nœud est trouvé, on utilise sa table de préférences conditionnelles pour recommander une valeur. Ainsi, seules les valeurs $\mathbf{u}[\mathbf{Inst}(N)]$ ont été utilisées lors de la recommandation, c'est-à-dire que seules sont prises en compte les valeurs de la configuration partielle dont les attributs sont des ancêtres de N et dont le nœud associé a plusieurs enfants. Une certaine partie de l'information provenant de la configuration partielle est donc ignorée car elle concerne des attributs moins importants que Next. On peut remarquer que dans le cas des LP-trees linéaires, $\mathbf{Inst}(N) = \varnothing$ pour tout nœud N : c'est pour cela que les LP-trees linéaires n'utilise pas la configuration partielle pour effectuer une recommandation.

La valeur du rang moyen normalisé est en elle-même intéressante car un k-LP-tree aléatoire aurait un rang moyen normalisé d'environ 0.5 ; les k-LP-trees que nous avons appris ont un rang moyen normalisé compris entre 1.2 % (pour $k = 1$ et 1 cluster) et $7 \cdot 10^{-7}$ % (pour $k = 3$ et 3 clusters).

Le taux d'erreur a une courbe en cloche en fonction du nombre de clusters ; il est minimal pour 3 et 4 clusters sur le jeu de données *Renault-48*. Alors qu'avoir plusieurs clusters permet d'augmenter le nombre de modèles utilisés et donc l'expressivité de l'algorithme de recommandation, chaque cluster contient moins d'exemples, ce qui rend l'apprentissage difficile, d'où une augmentation du taux d'erreur avec 5 clusters. Avec plus d'attributs pour étiqueter les nœuds (le paramètre k), l'expressivité des k-LP-trees est augmentée, ce qui explique ce recul du taux d'erreur.

De plus, la recommandation est très rapide ; l'algorithme que nous utilisons pour calculer l'extension préférée a pour complexité temporelle $O(dn)$, où n est le nombre d'attributs et d la taille maximale des domaines des attributs. Quel

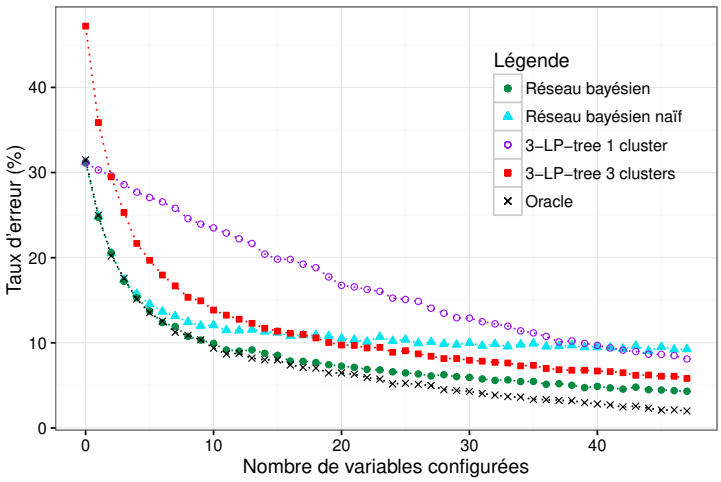


FIGURE 7.20 – Expérience de la section 7.5. Taux d'erreur de 3-LP-trees et d'autres méthodes sur *Renault-48*.

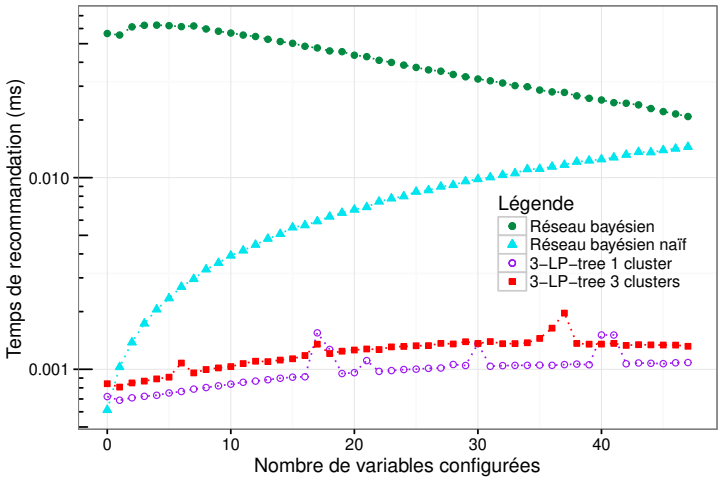


FIGURE 7.21 – Expérience de la section 7.5. Temps de recommandation de 3-LP-trees et d'autres méthodes sur *Renault-48*.

que soit le nombre de clusters ou la valeur de k, le temps d'une recommandation est de l'ordre de grandeur d'une microseconde, comme le montre la figure 7.21.

7.6 Conclusion

Ce chapitre a servi un triple objectif.

7.6.1 Évaluation des algorithmes de la littérature

Nous avons étudié les performances de différents algorithmes décrits par la littérature et nous avons étudié les paramètres d'apprentissage qui peuvent modifier leur qualité de recommandation. En résumé :

- le temps de recommandation est, pour tous les jeux de données et tous les algorithmes, inférieur à 50 ms, ce qui montre que ces méthodes sont toutes compatibles avec une utilisation en ligne. Les deux algorithmes les plus rapides sont les réseaux bayésiens naïfs et les LP-trees, ces derniers étant bien plus précis ;
- le taux d'erreur décroit en fonction du nombre d'attributs affectés. Les taux d'erreur des différentes méthodes sont similaires et sont tous proches du maximum possible (indiqué par l'Oracle) ;
- lorsqu'il y a peu d'exemples d'apprentissage, les réseaux bayésiens ont plus haut taux de succès que les méthodes basées sur les voisins ;
- le taux d'erreur en fonction du nombre de voisins suit une courbe en forme de cloche, avec un minimum vers environ 100 voisins ;
- l'utilisation de clusters dégrade la précision des méthodes basées sur le voisinage et du réseau bayésien, mais améliore significativement la précision des LP-trees
- la précision de la recommandation est significativement dégradée sur les problèmes les plus contraints.

7.6.2 L'avantage de DRC

DRC est aussi précis que les autres méthodes d'inférence exacte mais est plus lent. Néanmoins, DRC se révèle utile lorsqu'on s'intéresse au problème de mise à jour de l'historique. DRC peut dans ce cas tirer profit du nouvel historique et être plus précis que les autres méthodes d'inférence en réseau bayésien.

7.6.3 Efficacité réelle des LP-trees

L'utilisation de k-LP-trees avec des données réelles a permis de vérifier à quel point ce langage, qui fait des hypothèses sur la relation d'ordre à apprendre, est adapté à notre problème : avec de bons paramètres, nous arrivons à obtenir une excellente précision, tout en ayant un temps de recommandation extrêmement faible (la complexité temporelle de l'algorithme de recommandation est linéaire).

De plus, nos expériences montrent que le score que nous utilisons lors de l'apprentissage, le rang moyen empirique, est corrélé à la précision de la recommandation. Cela montre une forte adéquation entre notre étude théorique et un problème industriel et cela justifie notre approche théorique consistant à minimiser ce score.

Chapitre 8

Conclusion

Cette thèse a développé deux axes de recherche : l'inférence en réseau bayésien exploitant des exemples positifs et l'apprentissage d'arbres de préférences lexicographiques à partir d'exemples positifs.

Contributions

En ce qui concerne les réseaux bayésiens, notre contribution est l'algorithme DRC [FGM18b], un algorithme d'inférence dans les réseaux bayésiens. Alors que les autres méthodes d'inférence utilisent le réseau bayésien comme unique source d'information, DRC exploite le fait qu'un réseau bayésien est souvent appris à partir d'exemples positifs. Ces exemples positifs sont une source d'information supplémentaire qui peut être utilisée lors de l'inférence. L'algorithme DRC, de ce fait, n'utilise que la structure du réseau bayésien appris, qui capture des indépendances conditionnelles entre attributs et estime les probabilités conditionnelles directement à partir du jeu de données. DRC est particulièrement adapté à une utilisation dans un contexte où les lois de probabilité évoluent mais où les indépendances conditionnelles ne changent pas.

De plus, nous nous sommes intéressés à l'étude de l'apprentissage de LP-trees à partir d'exemples positifs [FGM18a]. L'apprentissage de modèles de préférences à partir d'exemples positifs a été très peu étudié par le passé ; et pourtant, les jeux d'exemples positifs sont très communs dans l'*e-commerce*, sous la forme d'historiques de vente. Nous avons introduit une distance ainsi qu'un score adaptés à ce problème. Nous avons présenté des algorithmes d'apprentissage pour les k-LP-trees et les LP-trees linéaires ; ces algorithmes convergent vers le modèle caché lorsque le nombre d'exemples tend vers l'infini et, dans le cas de l'apprentissage le LP-trees linéaires, le modèle appris est celui qui minimise le score. Nous avons aussi montré que la *sample complexity* des LP-trees linéaires est faible : logarithmique en le nombre d'attributs.

Enfin, notre étude expérimentale [FGM16] a permis d'évaluer l'efficacité de plusieurs méthodes (les réseaux bayésiens, les LP-trees et des méthodes basées sur les k plus proches voisins) sur un problème réel de Renault : la recommandation de valeurs en configuration interactive à partir d'historiques de vente. Nous pouvons conclure que la précision de la recommandation de ces méthodes est

proche de la précision optimale, car nous avons pu estimer celle-ci avec notre méthode appelée Oracle.

Perspectives

L'idée à la base de DRC, l'utilisation d'exemples positifs lors de l'inférence, peut être probablement transposée à d'autres algorithmes d'inférence dans les réseaux bayésiens que RC. Si DRC a une plus grande complexité temporelle que RC, c'est parce que RC ne manipule pas des probabilités mais des facteurs, qui ne peuvent pas être estimés directement à partir des exemples positifs. Nous ne connaissons pas d'autre algorithme d'inférence qui manipule des probabilités, mais un tel algorithme pourrait être facilement adapté à l'estimation directe. Plus généralement, cette idée d'utiliser les données d'apprentissage lors de l'inférence remet en cause la séparation hermétique qui est souvent faite entre l'apprentissage d'un modèle et son exploitation : les données utilisées lors de l'apprentissage peuvent également être utiles lors de l'exploitation du modèle appris.

Notre étude de l'apprentissage de LP-trees à partir d'exemples positifs démontre sa faisabilité et les performances obtenues en recommandation de valeurs nous semblent prometteuses. Parmi les questions en suspens, celle de la complexité de la minimisation du rang moyen empirique dans le cas des k-LP-trees nous paraît un axe de recherche important. Si ce problème est NP-difficile (ce vers quoi penche notre intuition), notre heuristique d'apprentissage pourrait servir de base à d'autres heuristiques plus sophistiquées. L'existence d'heuristiques plus sophistiquées est soutenue par le fait que notre algorithme d'apprentissage de k-LP-trees ne correspond pas, quand on le restreint à l'apprentissage de LP-trees linéaires, à notre algorithme d'apprentissage de LP-trees linéaires qui minimisent le rang moyen empirique.

Les langages de représentation de préférences ont des avantages indéniables sur des langages moins spécialisés, comme les réseaux bayésiens ; par exemple, les LP-trees et les CP-nets sont des langages graphiques permettant de donner une interprétation simple à une comparaison, ce qui les rend appréciables pour faire de l'extraction de connaissances et de l'explication. Et pourtant, ils trouvent un écho plus faible dans la littérature et les applications industrielles que, par exemple, les réseaux bayésiens. Pourquoi ? Les réseaux bayésiens sont largement utilisés, notamment dans le diagnostic (avec un réseau bayésien construit à la main), la classification, mais aussi pour l'extraction de connaissances, où ils sont appris à partir d'exemples pour obtenir une représentation des interactions entre différentes variables (par exemple pour apprendre un réseau écologique à partir de données d'abondance d'espèce). La plupart de ces applications sont possibles car l'apprentissage de réseaux bayésiens à partir d'exemples positifs a été longuement étudié.

A contrario, les langages de représentation de préférences sont limités dans leur application car on ne sait en apprendre un modèle qu'à partir d'un ensemble de comparaisons (acquis activement, par élicitation, ou passivement). On remarque que, malheureusement, la plupart des articles sur le sujet n'ont pas d'expérimentations sur données réelles ; lorsqu'il y en a, il s'agit en général d'un jeu de données composé de *ratings* [LZM+17] ou d'observations étiquetées de classification [BH12, BHKG17] qui a été transformé en ensemble de comparaisons. Cette relative absence d'expérimentations avec données réelles suggère, à notre

avis, une inadéquation de certaines méthodes d'apprentissage aux besoins des problèmes industriels. Cela en limite l'utilité pratique et, par ricochet, l'attention portée par la communauté à ce sujet.

Nos travaux ouvrent la voie à un nouvel axe de recherche : l'apprentissage de modèles de préférences à partir d'exemples positifs appliqué à d'autres langages, comme les forêts de LP-trees partiels, les CP-nets, les TCP-nets ou les GAI-nets. L'apprentissage à partir d'exemples positifs pourrait également être étendu aux exemples positifs et négatifs, ce qui permettrait, par exemple, d'apprendre des préférences à partir de *ratings* en les répartissant en deux catégories : note favorable et note défavorable. De quoi, nous l'espérons, contribuer au développement et à la popularisation de ces langages qui méritent une plus grande attention.

Bibliographie

[AFM02] Jérôme Amilhastre, Hélène Fargier, and Pierre Marquis. Consistency restoration and explanations in dynamic CSPs application to configuration. *Artificial Intelligence*, 135(1-2) :199–234, 2002.

[Aka98] Hirotogu Akaike. *Information Theory and an Extension of the Maximum Likelihood Principle*, pages 199–213. Springer, 1998.

[AMZ16] Eisa Alanazi, Malek Mouhoub, and Sandra Zilles. The complexity of learning acyclic CP-nets. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016*, pages 1361–1367, 2016.

[Ang88] Dana Angluin. Queries and concept learning. *Machine Learning*, 2(4) :319–342, 1988.

[BADGP14] Nahla Ben Amor, Didier Dubois, Héla Gouider, and Henri Prade. Possibilistic networks : A new setting for modeling preferences. In *Proceedings of the 8th International Conference on Scalable Uncertainty Management, SUM 2014*, pages 1–7, 2014.

[BB05] Darius Braziunas and Craig Boutilier. Local utility elicitation in GAI models. In *Proceedings of the Twenty-First Conference on Uncertainty in Artificial Intelligence, UAI 2005*, pages 42–49, 2005.

[BBB01] Craig Boutilier, Fahiem Bacchus, and Ronen I Brafman. UCP-networks : A directed graphical representation of conditional utilities. In *Proceedings of the Seventeenth conference on Uncertainty in Artificial Intelligence, UAI 2001*, pages 56–64, 2001.

[BBD+04] Craig Boutilier, Ronen I. Brafman, Carmel Domshlak, Holger H. Hoos, and David Poole. CP-nets : A tool for representing and reasoning with conditional ceteris paribus preference statements. *Journal of Artificial Intelligence Research*, 21 :135–191, 2004.

[BCD+93] Salem Benferhat, Claudette Cayrol, Didier Dubois, Jérôme Lang, and Henri Prade. Inconsistency management and prioritized syntax-based entailment. In *Proceedings of the 13th International*

Joint Conference on Artificial Intelligence, IJCAI 1993, pages 640–647, 1993.

[BCL+10] Richard Booth, Yann Chevaleyre, Jérôme Lang, Jérôme Mengin, and Chattrakul Sombattheera. Learning conditionally lexicographic preference relations. In *Proceedings of the 19th European Conference on Artificial Intelligence, ECAI 2010*, pages 269–274, 2010.

[BD02] Ronen I. Brafman and Carmel Domshlak. Introducing variable importance tradeoffs into CP-nets. In *Proceedings of the Eighteenth conference on Uncertainty in Artificial Intelligence, UAI 2002*, pages 69–76, 2002.

[BD03] Ronen I. Brafman and Yannis Dimopoulos. A new look at the semantics and optimization methods of CP-networks. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, IJCAI 2003*, pages 1033–1038, 2003.

[BDP03] Fahiem Bacchus, Shannon Dalmao, and Toniann Pitassi. Value elimination : Bayesian inference via backtracking search. In *Proceedings of the 19th Conference in Uncertainty in Artificial Intelligence, UAI 2003*, pages 20–28, 2003.

[BDS06] Ronen I. Brafman, Carmel Domshlak, and Solomon Eyal Shimony. On graphical modeling of preference and importance. *Journal of Artificial Intelligence Research*, 25 :389–424, 2006.

[BdSdSOG16] Cory J. Butz, André E. dos Santos, Jhonatan de S. Oliveira, and Christophe Gonzales. Testing independencies in Bayesian networks with i-separation. In *Proceedings of the Twenty-Ninth International Florida Artificial Intelligence Research Society Conference, FLAIRS 2016*, pages 644–649, 2016.

[BdSdSOG18] Cory J. Butz, André E. dos Santos, Jhonatan de S. Oliveira, and Christophe Gonzales. An empirical study of testing independencies in Bayesian networks using rp-separation. *International Journal of Approximate Reasoning*, 92 :270–278, 2018.

[BEdSSOG16] Cory Butz, André E. dos Santos, Jhonatan S. Oliveira, and Christophe Gonzales. A simple method for testing independencies in Bayesian networks. In *Proceedings of the 29th Canadian Conference on Artificial Intelligence, AI 2016*, pages 213–223, 05 2016.

[BEHW89] Anselm Blumer, Andrzej Ehrenfeucht, David Haussler, and Manfred K Warmuth. Learnability and the Vapnik-Chervonenkis dimension. *Journal of the ACM*, 36(4) :929–965, 1989.

[BFG+97] 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. *Formal Methods in System Design*, 10(2/3) :171–206, 1997.

[BFL13] Christian Bessiere, Hélène Fargier, and Christophe Lecoutre. Global inverse consistency for interactive constraint satisfaction. In *Proceedings of the 19th International Conference on Principles and Practice of Constraint Programming, CP 2013*, pages 159–174, 2013.

[BFMZ12] Damien Bigot, Hélene Fargier, Jérôme Mengin, and Bruno Zanuttini. Using and learning GAI-decompositions for representing ordinal rankings. In *Proceedings of Preference Learning ECAI 2012 Workshop, PL 2012*, pages 5–10. Fürnkranz Johannes Hüllermeier Eyke, 2012.

[BH12] Michael Bräuning and Eyke Hüllermeyer. Learning conditional lexicographic preference trees. In Johannes Fürnkranz and Eyke Hüllermeyer, editors, *Proceedings of Preference Learning ECAI 2012 Workshop, PL 2012*, 2012.

[BHKG17] Michael Bräuning, Eyke Hüllermeier, Tobias Keller, and Martin Glaum. Lexicographic preferences for predictive modeling of human decision making : A new machine learning method with an application in accounting. *European Journal of Operational Research*, 258(1) :295–306, 2017.

[Big15] Damien Bigot. *Représentation et apprentissage de préférences*. PhD thesis, 2015. Université Toulouse 3.

[BJ92] Thang Nguyen Bui and Curt Jones. Finding good approximate vertex and edge partitions is NP-hard. *Information Processing Letters*, 42(3) :153–159, 1992.

[BMR+99] Stefano Bistarelli, Ugo Montanari, Francesca Rossi, Thomas Schiex, Gérard Verfaillie, and Hélene Fargier. Semiring-based CSPs and valued CSPs : Frameworks, properties, and comparison. *Constraints*, 4(3) :199–240, 1999.

[Bou93] Remco R Bouckaert. Probabilistic network construction using the minimum description length principle. In *Proceedings of the European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty, ECSQARU 1993*, pages 41–48. Springer, 1993.

[Bou94] Craig Boutilier. Toward a logic for qualitative decision theory. In *Proceedings of the 4th International Conference on Principles of Knowledge Representation and Reasoning, KR 1994*, pages 75–86, 1994.

[Bre89] Gerhard Brewka. Preferred subtheories : An extended logical framework for default reasoning. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence, IJCAI 1989*, pages 1043–1048, 1989.

[BT52] Ralph Allan Bradley and Milton E Terry. Rank analysis of incomplete block designs : I. the method of paired comparisons. *Biometrika*, 39(3/4) :324–345, 1952.

[Bun91] Wray L. Buntine. Theory refinement on Bayesian networks. In *Proceedings of the Seventh Annual Conference on Uncertainty in Artificial Intelligence, UAI 1991*, pages 52–60, 1991.

[BV03] Denis Bouyssou and Philippe Vincke. Relations binaires et modélisation des préférences. *Concepts et méthodes pour l'aide à la décision*, 1, 2003.

[BZFM13] Damien Bigot, Bruno Zanuttini, Hélène Fargier, and Jérôme Mengin. Probabilistic conditional preference networks. In *Proceedings of the Twenty-Ninth Conference on Uncertainty in Artificial Intelligence, UAI 2013*, 2013.

[CGO+02] Rickard Coster, Andreas Gustavsson, Tomas Olsson, Åsa Rudström, and Asa Rudström. Enhancing web-based configuration with recommendations and cluster-based help. In *Proceedings of the AH'2002 Workshop on Recommendation and Personalization in eCommerce*, pages 30–40, 2002.

[CH91] Gregory F. Cooper and Edward Herskovits. A Bayesian method for constructing Bayesian belief networks from databases. In *Proceedings of the Seventh Annual Conference on Uncertainty in Artificial Intelligence, UAI 1991*, pages 86–94, 1991.

[CH92] Gregory F. Cooper and Edward Herskovits. A Bayesian method for the induction of probabilistic networks from data. *Machine learning*, 9(4) :309–347, 1992.

[Chi95] David Maxwell Chickering. Learning Bayesian networks is NP-complete. In *Proceedings of the Fifth International Workshop on Artificial Intelligence and Statistics, AISTATS 1995*, pages 121–130, 1995.

[CKL+10] Yann Chevaleyre, Frédéric Koriche, Jérôme Lang, Jérôme Mengin, and Bruno Zanuttini. Learning ordinal preferences on multiattribute domains : The case of CP-nets. In *Preference learning*, pages 273–296. Springer, 2010.

[CLLM04] Sylvie Coste-Marquis, Jérôme Lang, Paolo Liberatore, and Pierre Marquis. Expressive power and succinctness of propositional languages for preference representation. In *Proceedings of the Ninth International Conference on Principles of Knowledge Representation and Reasoning, KR 2004*, pages 203–212, 2004.

[CM11] Antoine Cornuéjols and Laurent Miclet. *Apprentissage artificiel : concepts et algorithmes*. Editions Eyrolles, 2011.

[CM14] Diego Colombo and Marloes H Maathuis. Order-independent constraint-based causal structure learning. *The Journal of Machine Learning Research*, 15(1) :3741–3782, 2014.

[Con85] Nicolas de Condorcet. *Essai sur l'application de l'analyse à la probabilité des décisions rendues à la pluralité des voix*. Imprimerie royale, 1785.

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

[Cor12] Cristina Cornelio. Dynamic and probabilistic CP-nets. Master's thesis, University of Padua, 2012.

[Cow01] Robert G Cowell. Conditions under which conditional independence and scoring methods lead to identical selection of Bayesian network models. In *Proceedings of the Seventeenth conference on Uncertainty in Artificial İntelligence, UAI 2001*, pages 91–97, 2001.

[CRBH10] Weiwei Cheng, Michaël Rademaker, Bernard De Baets, and Eyke Hüllermeier. Predicting partial orders : Ranking with abstention. In *Proceedings of the Conference on Machine Learning and Knowledge Discovery in Databases, European Conference, ECML PKDD 2010*, pages 215–230, 2010.

[CSS98] William W. Cohen, Robert E. Schapire, and Yoram Singer. Learning to order things. In *Advances in Neural Information Processing Systems*, pages 451–457, 1998.

[CVN18] Carlo Contaldi, Fatemeh Vafaee, and Peter C Nelson. Bayesian network hybrid learning using an elite-guided genetic algorithm. *Artificial Intelligence Review*, pages 1–28, 2018.

[CWR07] Fei Chen, Xiufeng Wang, and Yimei Rao. Learning Bayesian networks using genetic algorithm. *Journal of Systems Engineering and Electronics*, 18(1) :142–147, 2007.

[Dar01] Adnan Darwiche. Recursive conditioning. *Artificial Intelligence*, 126(1-2) :5–41, 2001.

[Dar09] Adnan Darwiche. *Modeling and reasoning with Bayesian networks*. Cambridge University Press, 2009.

[DBS01] Carmel Domshlak, Ronen I. Brafman, and Solomon Eyal Shimony. Preference-based configuration of web page content. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence, IJCAI 2001*, pages 1451–1456, 2001.

[Deb64] Gerard Debreu. Continuity properties of paretian utility. *International Economic Review*, 5(3) :285–293, 1964.

[DFP93] Didier Dubois, Hélene Fargier, and Henri Prade. The calculus of fuzzy restrictions as a basis for flexible constraint satisfaction. In *Proceedings of the Second IEEE International Conference on Fuzzy Systems, FUZZ 1993*, pages 1131–1136. IEEE, 1993.

[DH01] Adnan Darwiche and Mark Hopkins. Using recursive decomposition to construct elimination orders, jointrees, and dtrees. In *Proceedings of the 6th European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty, ECSQARU 2001*, pages 180–191, 2001.

[DL93] Paul Dagum and Michael Luby. Approximating probabilistic inference in Bayesian belief networks is NP-hard. *Artificial Intelligence*, 60(1) :141–153, 1993.

[DMA09] Yannis Dimopoulos, Loizos Michael, and Fani Athienitou. Ceteris paribus preference elicitation with predictive guarantees. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence, IJCAI 2009*, pages 1890–1895, 2009.

[FFH+98] G. Fleishanderl, Gerhard E. Friedrich, Alois Haselbock, Herwig Schreiner, and Markus Stumptner. Configuring large systems using generative constraint satisfaction. *IEEE Intelligent Systems and their applications*, 13(4) :59–68, 1998.

[FGM16] Hélène Fargier, Pierre-François Gimenez, and Jérôme Mengin. Recommendation for product configuration : an experimental evaluation. In *Proceedings of the 18th International Configuration Workshop, CWS 2016*, 2016.

[FGM18a] Hélène Fargier, Pierre-François Gimenez, and Jérôme Mengin. Learning lexicographic preference trees from positive examples. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, AAAI 2018*, 2018.

[FGM18b] Hélène Fargier, Pierre-François Gimenez, and Jérôme Mengin. Recommandation par inférence bayésienne. application à la configuration de produit. *Revue d'Intelligence Artificielle*, 32(1) :39–74, 2018.

[FH11] Johannes Fürnkranz and Eyke Hüllermeier. Preference learning. In *Encyclopedia of Machine Learning*, pages 789–795. Springer, 2011.

[Fis70] Peter C Fishburn. Utility theory for decision making. Technical report, Research Analysis Corp Mclean VA, 1970.

[Fis91] Peter C Fishburn. Nontransitive preferences in decision theory. *Journal of risk and uncertainty*, 4(2) :113–134, 1991.

[FL93] Hélène Fargier and Jérôme Lang. Uncertainty in constraint satisfaction problems : a probalistic approach. In *Proceedings of the European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty, ECSQARU 1993*, pages 97–104, 1993.

[FLS93] Hélène Fargier, Jérôme Lang, and Thomas Schiex. Selecting preferred solutions in Fuzzy Constraint Satisfaction Problems. In *Proceedings of the 1st European Congress on Fuzzy and Intelligent Technologies, EUFIT 1993*, 1993.

[FW92] Eugene C. Freuder and Richard J Wallace. Partial constraint satisfaction. *Artificial Intelligence*, 58(1-3) :21–70, 1992.

[GAG13] Joshua T. Guerin, Thomas E. Allen, and Judy Goldsmith. Learning CP-net preferences online from user queries. In *Proceedings of the Third International Conference on Algorithmic Decision Theory, ADT 2013*, pages 208–220, 2013.

[GG96] G. Gigerenzer and D. Goldstein. Reasoning the fast and frugal way : Models of bounded rationality. *Psychological Review*, 103(4) :650–669, 1996.

[GLTW08] Judy Goldsmith, Jérôme Lang, Miroslaw Truszczynski, and Nic Wilson. The computational complexity of dominance and consistency in CP-nets. *Journal of Artificial Intelligence Research*, 33 :403–432, 2008.

[GNOT92] David Goldberg, David A. Nichols, Brian M. Oki, and Douglas B. Terry. Using collaborative filtering to weave an information tapestry. *Communications of the ACM*, 35(12) :61–70, 1992.

[GP04] Christophe Gonzales and Patrice Perny. GAI networks for utility elicitation. In *Proceedings of the Ninth International Conference on Principles of Knowledge Representation and Reasoning, KR 2004*, pages 224–234, 2004.

[GPQ08] Christophe Gonzales, Patrice Perny, and Sergio Queiroz. GAI-networks : Optimization, ranking and collective choice in combinatorial domains. *Foundations of computing and decision sciences*, 33(1) :3–24, 2008.

[GUH16] Carlos A. Gomez-Uribe and Neil Hunt. The netflix recommender system : Algorithms, business value, and innovation. *ACM Transactions on Management Information Systems*, 6(4) :13, 2016.

[Han92] Steve Hanks. Representations for decision-theoretic planning : Utility functions for deadline goals. In *Proceedings of the Third International Conference of Principles of Knowledge Representation and Reasoning, KR 1992*, page 71, 1992.

[Han16] Steve Hanneke. The optimal sample complexity of pac learning. *The Journal of Machine Learning Research*, 17(1) :1319–1333, 2016.

[Hen86] Max Henrion. Propagating uncertainty in Bayesian networks by probabilistic logic sampling. In *Proceedings of the Second Annual Conference on Uncertainty in Artificial Intelligence, UAI 1986*, pages 149–164, 1986.

[HGC95] David Heckerman, Dan Geiger, and David M Chickering. Learning Bayesian networks : The combination of knowledge and statistical data. *Machine learning*, 20(3) :197–243, 1995.

[HK98] Cynthia Huffman and Barbara E Kahn. Variety for sale : Mass customization or mass confusion ? *Journal of retailing*, 74(4) :491–513, 1998.

[HLP52] Godfrey Harold Hardy, John Edensor Littlewood, and George Pólya. *Inequalities*. Cambridge university press, 2nd edition, 1952.

[Hug17] Nicolas Hug. Surprise, a Python library for recommender systems. `http://surpriselib.com`, 2017.

[HWA07] Tarik Hadzic, Andrzej Wasowski, and Henrik Reif Andersen. Techniques for efficient interactive configuration of distribution networks. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence, IJCAI 2007*, pages 100–105, 2007.

[JBGS10] Mikolás Janota, Goetz Botterweck, Radu Grigore, and João P. Marques Silva. How to complete an interactive configuration process ? In *Proceedings of the 36th Conference on Current Trends in Theory and Practice of Computer Science, SOFSEM 2010*, pages 528–539, 2010.

[JN06] Martin Janžura and Jan Nielsen. A simulated annealing-based method for learning Bayesian networks from statistical data. *International Journal of Intelligent Systems*, 21(3) :335–348, 2006.

[JZFF10] Dietmar Jannach, Markus Zanker, Alexander Felfernig, and Gerhard Friedrich. *Recommender systems : an introduction*. Cambridge University Press, 2010.

[KC02] Mehmet Kayaalp and Gregory F Cooper. A Bayesian network scoring metric that is based on globally uniform parameter priors. In *Proceedings of the Eighteenth conference on Uncertainty in Artificial Intelligence, UAI 2002*, pages 251–258, 2002.

[KdBCD14] Doga Kisa, Guy Van den Broeck, Arthur Choi, and Adnan Darwiche. Probabilistic sentential decision diagrams. In *Proceedings of the Fourteenth International Conference on Principles of Knowledge Representation and Reasoning, KR 2014*, 2014.

[Ken38] Maurice G Kendall. A new measure of rank correlation. *Biometrika*, 30(1/2) :81–93, 1938.

[Kjæ90] Uffe Kjærulff. Triangulation of graphs–algorithms giving small total state space. Technical report, Technical Report R-90-09, Dept. of Mathematics and Computer Science, Aalborg University, 1990.

[KK98] George Karypis and Vipin Kumar. Multilevelk-way partitioning scheme for irregular graphs. *Journal of Parallel and Distributed computing*, 48(1) :96–129, 1998.

[KP83] Jin H. Kim and Judea Pearl. A computational model for causal and diagnostic reasoning in inference systems. In *Proceedings of the 8th International Joint Conference on Artificial Intelligence, IJCAI 1983*, pages 190–193, 1983.

[KSS94] Michael J. Kearns, Robert E. Schapire, and Linda M Sellie. Toward efficient agnostic learning. *Machine Learning*, 17(2-3) :115–141, 1994.

[Kut13] Michael Kutschke. Jayes - Bayesian network library under Eclipse Public License. 2013.

[KZ10] Frédéric Koriche and Bruno Zanuttini. Learning conditional preference networks. *Artificial Intelligence*, 174(11) :685–703, 2010.

[Lan91] Jérôme Lang. Possibilistic logic as a logical framework for min-max discrete optimisation problems and prioritized constraints. In *Proceedings of the Fundamentals of Artificial Intelligence Research, International Workshop, FAIR 1991*, pages 112–126, 1991.

[LDLL90] Steffen L. Lauritzen, A. Philip Dawid, B. N. Larsen, and Hanns-Georg Leimer. Independence properties of directed markov fields. *Networks*, 20(5) :491–505, 1990.

[Liu16] Xudong Liu. *Modeling, learning and reasoning about preference trees over combinatorial domains*. University of Kentucky, 2016.

[LL01] Celine Lafage and Jérôme Lang. Propositional distances and preference representation. In *Proceedings of the 6th European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty, ECSQARU 2001*, pages 48–59, 2001.

[LM09] Jérôme Lang and Jérôme Mengin. The complexity of learning separable ceteris paribus preferences. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence, IJCAI 2009*, pages 848–853, 2009.

[LMX12] Jérôme Lang, Jérôme Mengin, and Lirong Xia. Aggregating conditionally lexicographic preferences on multi-issue domains. In *Proceedings of the 18th International on Conference Principles and Practice of Constraint Programming, CP 2012*, pages 973–987, 2012.

[LS88] Steffen L. Lauritzen and David J Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society. Series B. (Methodological)*, pages 157–224, 1988.

[LSY03] Greg Linden, Brent Smith, and Jeremy York. Amazon.com recommendations : Item-to-item collaborative filtering. *IEEE Internet Computing*, 7(1) :76–80, 2003.

[LT15] Xudong Liu and Miroslaw Truszczynski. Learning partial lexicographic preference trees over combinatorial domains. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, AAAI 2015*, pages 1539–1545, 2015.

[LZM+17] Fabien Labernia, Bruno Zanuttini, Brice Mayag, Florian Yger, and Jamal Atif. Online learning of acyclic conditional preference networks from noisy data. In *Proceedings of the 17th IEEE International Conference on Data Mining, ICDM 2017*, 2017.

[Mar03] Dimitris Margaritis. Learning Bayesian network model structure from data. Technical report, School of Computer Science, Carnegie-Mellon University, Pittsburgh, PA, 2003.

[MC07] Denis Mindolin and Jan Chomicki. Hierarchical CP-networks. In *3rd Multidisciplinary Workshop on Advances in Preference Handling, M-PREF 2007*, 2007.

[McD82] John P. McDermott. R1 : A rule-based configurer of computer systems. *Artificial Intelligence*, 19(1) :39–88, 1982.

[MF89] Sanjay Mittal and Felix Frayman. Towards a generic model of configuraton tasks. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence, IJCAI 1989*, pages 1395–1401, 1989.

[MRS06] Pedro Meseguer, Francesca Rossi, and Thomas Schiex. Soft constraints. In *Foundations of Artificial Intelligence*, volume 2, pages 281–328. Elsevier, 2006.

[NWL+11] Patrick Naïm, Pierre-Henri Wuillemin, Philippe Leray, Olivier Pourret, and Anna Becker. *Réseaux bayésiens*. Editions Eyrolles, 2011.

[Par02] Bernard Pargamin. Vehicle sales configuration : the cluster tree approach. In *Proceedings of Configuration ECAI 2002 Workshop*, 2002.

[PD04] James D. Park and Adnan Darwiche. Complexity results and approximation strategies for map explanations. *Journal of Artificial Intelligence Research*, 21 :101–133, 2004.

[Pea85] Judea Pearl. A constraint-propagation approach to probabilistic reasoning. In *Proceedings of the First Annual Conference on Uncertainty in Artificial Intelligence, UAI 1985*, pages 357–370, 1985.

[Pea86] Judea Pearl. Fusion, propagation, and structuring in belief networks. *Artificial Intelligence*, 29(3) :241–288, 1986.

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

[RN10] Stuart J. Russell and Peter Norvig. *Artificial Intelligence - A Modern Approach*. Pearson Education, 2010.

[RRS11] Francesco Ricci, Lior Rokach, and Bracha Shapira. Introduction to recommender systems handbook. In *Recommender systems handbook*, pages 1–35. Springer, 2011.

[RRS12] Thomas S. Richardson, James M. Robins, and Ilya Shpitser. Nested markov properties for acyclic directed mixed graphs. In *Proceedings of the Twenty-Eighth Conference on Uncertainty in Artificial Intelligence, UAI 2012*, page 13, 2012.

[RVBW06] Francesca Rossi, Peter Van Beek, and Toby Walsh. *Handbook of constraint programming*. Elsevier, 2006.

[RVW] Francesca Rossi, Kristen Brent Venable, and Toby Walsh. mCP nets : Representing and reasoning with preferences of multiple agents. In *Proceedings of the Nineteenth National Conference on Artificial Intelligence, Sixteenth Conference on Innovative Applications of Artificial Intelligence, AAAI 2004*.

[Sch78] Gideon Schwarz. Estimating the dimension of a model. *The annals of statistics*, 6(2) :461–464, 1978.

[Sch09] Barry Schwartz. *The paradox of choice*. HarperCollins, 2009.

[Sch15a] Nicolas Schmidt. *Compilation de préférences – application à la configuration de produits*. PhD thesis, Université d'Artois, 2015.

[Sch15b] Nicolas Schmidt. SALADD, le compilateur SLDD. `https://github.com/SchmidtNicolas/SALADD`, 2015.

[Scu10] Marco Scutari. Learning Bayesian networks with the bnlearn R package. *Journal of Statistical Software*, 35(3) :1–22, 2010.

[Scu16] Marco Scutari. An empirical-Bayes score for discrete Bayesian networks. In *Proceedings of the Eighth International Conference on Probabilistic Graphical Models, PGM 2016*, pages 438–448, 2016.

[SFV95] Thomas Schiex, Hélène Fargier, and Gérard Verfaillie. Valued constraint satisfaction problems : Hard and easy problems. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, IJCAI 95*, pages 631–639, 1995.

[SG94] B. Seroussi and Jean-Louis Golmard. An algorithm directly finding the K most probable configurations in Bayesian networks. *International Journal of Approximate Reasoning*, 11(3) :205–233, 1994.

[Sha98] Ross D. Shachter. Bayes-ball : The rational pastime (for determining irrelevance and requisite information in belief networks and influence diagrams). In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence, UAI 1998*, pages 480–487, 1998.

[SHBM14] Marco Scutari, Phil Howell, David J. Balding, and Ian Mackay. Multiple quantitative trait analysis using Bayesian networks. *Genetics*, 198(1) :129–137, 2014.

[Shi94] Solomon Eyal Shimony. Finding MAPs for belief networks is NP-hard. *Artificial Intelligence*, 68(2) :399–410, 1994.

[SKKR01] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th international conference on World Wide Web, WWW 2001*, pages 285–295. ACM, 2001.

[SLJR18] Tomi Silander, Janne Leppä-aho, Elias Jääsaari, and Teemu Roos. Quotient normalized maximum likelihood criterion for learning Bayesian network structures. In *Proceedings of the International Conference on Artificial Intelligence and Statistics, AISTATS 2018*, pages 948–957, 2018.

[SM06] Michael Schmitt and Laura Martignon. On the complexity of learning lexicographic strategies. *Journal of Machine Learning Research*, 7(Jan) :55–83, 2006.

[Spe04] Charles Spearman. The proof and measurement of association between two things. *The American journal of psychology*, 15(1) :72–101, 1904.

[SRKM08] Tomi Silander, Teemu Roos, Petri Kontkanen, and Petri Myllymäki. Factorized normalized maximum likelihood criterion for learning Bayesian network structures. In *Proceedings of the 4th European Workshop on Probabilistic Graphical Models, PGM 2008*, 2008.

[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.

[TK00] Simon Tong and Daphne Koller. Active learning for parameter estimation in Bayesian networks. In *Advances in Neural Information Processing Systems 13, Papers from Neural Information Processing Systems 2000*, pages 647–653, 2000.

[TK01] Simon Tong and Daphne Koller. Active learning for structure in Bayesian networks. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence, IJCAI 2001*, pages 863–869, 2001.

[UF98] Lyle H. Ungar and Dean P. Foster. Clustering methods for collaborative filtering. In *Proceedings of the Workshop on Recommendation System at the 15th National Conference on Artificial Intelligence, AAAI Workshop on Recommendation Systems 1998*, volume 1, pages 114–129, 1998.

[Val84] Leslie G. Valiant. A theory of the learnable. In *Proceedings of the 16th Annual ACM Symposium on Theory of Computing, STOC 1984*, pages 436–445, 1984.

[VC15] Vladimir N. Vapnik and A. Ya Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. In *Measures of complexity*, pages 11–30. Springer, 2015.

[VP90] Thomas Verma and Judea Pearl. Equivalence and synthesis of causal models. In *Proceedings of the Sixth Annual Conference on Uncertainty in Artificial Intelligence, UAI 1990*, pages 255–270, 1990.

[VW63] Georg Henrik Von Wright. *The logic of preference*. Edinburgh University Press, 1963.

[Wal72] David L Waltz. *Generating semantic descriptions from drawings of scenes with shadows*. PhD thesis, 1972.

[WD91] Michael P. Wellman and Jon Doyle. Preferential semantics for goals. In *Proceedings of the 9th National Conference on Artificial Intelligence, AAAI 1991*, pages 698–703, 1991.

[Wil11] Nic Wilson. Computational techniques for a simple theory of conditional preferences. *Artificial Intelligence*, 175(7-8) :1053–1091, 2011.

[WZS+12] Hongbing Wang, Jie Zhang, Wenlong Sun, Hongye Song, Guibing Guo, and Xiang Zhou. WCP-nets : A weighted extension to CP-nets for web service selection. In *Proceedings of the 10th International Conference on Service-Oriented Computing, ICSOC 2012*, pages 298–312, 2012.

[YM05] Sandeep Yaramakala and Dimitris Margaritis. Speculative Markov blanket discovery for optimal feature selection. In *Proceedings of the fifth IEEE international conference on Data mining, ICDM 2005*, pages 4–pp. IEEE, 2005.

[YWLd10] Fusun Yaman, Thomas J. Walsh, Michael L. Littman, and Marie desJardins. Learning lexicographic preference models. In *Preference learning*, pages 251–272. Springer, 2010.

[ZG14] Enli Zhang and Lin Gao. A vertex separator-based algorithm for hypergraph bipartitioning. *Journal of Computers*, 9(8) :1886–1896, 2014.

[ZP94] Nevin L. Zhang and David Poole. A simple approach to Bayesian network computations. In *Proceedings of the Tenth Canadian Conference on Artificial Intelligence, AI 1994*, 1994.

Annexe A

Définitions de la théorie des graphes

Cette annexe a pour objectif de rappeler quelques définitions de la théorie des graphes. Il existe deux grandes familles de graphes : les graphes orientés et les graphes non-orientés.

Définition A.1 (Graphe non-orienté)**.** *Un graphe non-orienté est un couple (E, V) où V est un ensemble de paires de E. Les éléments de E sont appelés des nœuds (ou sommets) et ceux de V, des arêtes.*

Définition A.2 (Graphe orienté)**.** *Un graphe est un couple (E, V) où V est un ensemble de paires ordonnées de E. Les éléments de E sont appelés des nœuds (ou sommets) et ceux de V, des arcs.*

Par la suite, nous ne manipulerons que des arbres finis, c'est-à-dire dont l'ensemble de sommets est fini.

Nous aurons également besoin de la notion de graphe induit sur un ensemble de nœuds.

Définition A.3 (Graphe induit)**.** *Soit $G = (E, V)$ un graphe (orienté ou non-orienté) et $E' \subset E$ un sous-ensemble de ses nœuds. Alors le graphe induit sur E' est le sous-graphe G' de G restreint aux sommets E'. Plus formellement, $G' = (E', V')$ où V' est l'ensemble des arêtes/arcs de V dont les deux sommets appartiennent à E'.*

La notion de chemin va nous permettre de définir d'autres structures intéressantes.

Définition A.4 (Chemin)**.** *Un chemin dans un graphe orienté est une suite d'arcs successifs.*

Plus formellement, une suite d'arcs $((A_1, B_1), (A_2, B_2), \ldots, (A_N, B_N))$ est un chemin si, pour tout $i \in [1, N-1]$, $B_i = A_{i+1}$.

La notion correspondante dans un graphe non-orienté est la *chaîne*.

Définition A.5 (Chaîne)**.** *Une chaîne dans un graphe non-orienté est une suite d'arêtes successives reliant deux nœuds.*

Un type de graphe que nous manipulerons beaucoup est le graphe acyclique orienté.

Définition A.6 (Graphe acyclique orienté)**.** *Un graphe acyclique orienté (ou DAG pour* Directed Acyclic Graph*) est un graphe orienté (E, V) tel que, pour tout sommet $A \in E$, il n'existe aucun chemin de A vers A.*

Dans un graphe acyclique orienté, les notions de parent, enfant, racine et ancêtre sont largement utilisées.

Définition A.7 (Parent et enfant d'un nœud)**.** *Soit $G = (E, V)$ un graphe orienté acyclique. On dit que A est un parent de B ssi $(A, B) \in V$. Réciproquement, on dit que A est un enfant de B ssi B est un parent de A.*

Définition A.8 (Racine)**.** *Soit (E, V) un graphe orienté acyclique. On dit que $A \in E$ est une racine ssi A n'a aucun parent.*

Définition A.9 (Ancêtre d'un nœud)**.** *On dit qu'un nœud A est un ancêtre d'un nœud B ssi il existe un chemin depuis A vers B.*

Définition A.10 (Feuille, nœud interne et externe)**.** *On dit qu'un nœud N d'un graphe acyclique orienté est une feuille (aussi appelé nœud externe) ssi N n'a aucun enfant. Si N n'est pas une feuille, on dit que N est un nœud interne.*

Tout graphe acyclique orienté fini non-vide possède au moins une racine et une feuille.

Enfin, deux types particulier de graphes acycliques orientés, les arbres enracinés et les poly-arbres, nous seront utiles par la suite.

Définition A.11 (Arbre enraciné)**.** *Un arbre enraciné est un graphe acyclique orienté qui possède une unique racine et dont tous les autres nœuds ont exactement un parent.*

Définition A.12 (Profondeur d'un nœud)**.** *Soit (E, V) un arbre enraciné. La profondeur d'un nœud est la longueur du plus court chemin entre la racine et ce nœud, plus un.*

Les conventions varient concernant la profondeur de la racine ; dans cette thèse, nous adaptons la convention selon laquelle la profondeur de la racine est égale à 1.

Définition A.13 (Poly-arbre)**.** *Un poly-arbre est un graphe acyclique orienté tel qu'il existe une unique chaîne (non-orientée) entre toute paire de sommets.*

Apprentissage de préférences en espace combinatoire et application à la recommandation en configuration interactive

L'analyse et l'exploitation des préférences interviennent dans de nombreux domaines, comme l'économie, les sciences sociales ou encore la psychologie. Depuis quelques années, c'est l'*e-commerce* qui s'intéresse au sujet dans un contexte de personnalisation toujours plus poussée. Notre étude s'est portée sur la représentation et l'apprentissage de préférences sur des objets décrits par un ensemble d'attributs. Ces espaces combinatoires sont immenses, ce qui rend impossible en pratique la représentation *in extenso* d'un ordre de préférences sur leurs objets. C'est pour cette raison que furent construits des langages permettant de représenter de manière compacte des préférences sur ces espaces combinatoires. Notre objectif a été d'étudier plusieurs langages de représentation de préférences et l'apprentissage de préférences.

Nous avons développé deux axes de recherche. Le premier axe est l'algorithme DRC, un algorithme d'inférence dans les réseaux bayésiens. Alors que les autres méthodes d'inférence utilisent le réseau bayésien comme unique source d'information, DRC exploite le fait qu'un réseau bayésien est souvent appris à partir d'un ensemble d'objets qui ont été choisis ou observés. Ces exemples sont une source d'information supplémentaire qui peut être utilisée lors de l'inférence. L'algorithme DRC, de ce fait, n'utilise que la structure du réseau bayésien, qui capture des indépendances conditionnelles entre attributs et estime les probabilités conditionnelles directement à partir du jeu de données. DRC est particulièrement adapté à une utilisation dans un contexte où les lois de probabilité évoluent mais où les indépendances conditionnelles ne changent pas.

Le second axe de recherche est l'apprentissage de k-LP-trees à partir d'exemples d'objets vendus. Nous avons défini formellement ce problème et introduit un score et une distance adaptés. Nous avons obtenu des résultats théoriques intéressants, notamment un algorithme d'apprentissage de k-LP-trees qui converge avec assez d'exemples vers le modèle cible, un algorithme d'apprentissage de LP-tree linéaire optimal au sens où il minimise notre score, ainsi qu'un résultat sur le nombre d'exemples suffisants pour apprendre un « bon » LP-tree linéaire : il suffit d'avoir un nombre d'exemples qui dépend logarithmiquement du nombre d'attributs du problème.

Enfin, une contribution expérimentale évalue différents langages dont nous apprenons des modèles à partir d'historiques de voitures vendues. Les modèles appris sont utilisés pour la recommandation de valeur en configuration interactive de voitures Renault. La configuration interactive est un processus de construction de produit où l'utilisateur choisit successivement une valeur pour chaque attribut. Nous évaluons la précision de la recommandation, c'est-à-dire la proportion des recommandations qui auraient été acceptées, et le temps de recommandation ; de plus, nous examinons les différents paramètres qui peuvent influer sur la qualité de la recommandation. Nos résultats sont concluants : les méthodes que nous avons évaluées, qu'elles proviennent de la littérature ou de nos contributions théoriques, sont bien assez rapides pour être utilisées en ligne et ont une précision très élevée, proche du maximum théorique.