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 *25/02/2015* **par :**

DAMIEN BIGOT

Représentation et apprentissage de préférences

JURY

MARTIN COOPER	Professeur d'Université	Président du Jury
HÉLÈNE FARGIER	Directeur de Recherche	Directeur de thèse
FRÉDÉRIC KORICHE	Professeur d'Université	Rapporteur
JÉRÔME LANG	Directeur de Recherche	Membre du Jury
JÉRÔME MENGIN	Maître de Conférences	Co-directeur de thèse
PATRICE PERNY	Professeur d'Université	Rapporteur
BRUNO ZANUTTINI	Maître de Conférences	Co-directeur de thèse

École doctorale et spécialité :
MITT : Domaine STIC : Réseaux, Télécoms, Systèmes et Architecture

Unité de Recherche :
Institut de Recherche en Informatique de Toulouse(UMR 5505)

Directeur(s) de Thèse :
Hélène Fargier, *Jérôme Mengin* et *Bruno Zanuttini*

Rapporteurs :
Patrice Perny et *Frédéric Koriche*

À mon fils Noé.

Résumé

La modélisation des préférences par le biais de formalismes de représentation compacte fait l'objet de travaux soutenus en intelligence artificielle depuis plus d'une quinzaine d'années. Ces formalismes permettent l'expression de modèles suffisamment flexibles et riches pour décrire des comportements de décision complexes. Pour être intéressants en pratique, ces formalismes doivent de plus permettre l'élicitation des préférences de l'utilisateur, et ce en restant à un niveau admissible d'interaction. La configuration de produits combinatoires dans sa version *business to customer* et la recherche à base de préférences constituent de bons exemples de ce type de problème de décision où les préférences de l'utilisateur ne sont pas connues *a priori*.

Dans un premier temps, nous nous sommes penchés sur l'apprentissage de GAI-décompositions. Nous verrons qu'il est possible d'apprendre une telle représentation en temps polynomial en passant par un système d'inéquations linéaires. Dans un second temps, nous proposerons une version probabiliste des CP-nets permettant la représentation de préférences multi-utilisateurs afin de réduire le temps nécessaire à l'apprentissage des préférences d'un utilisateur. Nous étudierons les différentes requêtes que l'on peut utiliser avec une telle représentation, puis nous nous pencherons sur la complexité de ces requêtes. Enfin, nous verrons comment apprendre ce nouveau formalisme, soit grâce à un apprentissage hors ligne à partir d'un ensemble d'objets optimaux, soit grâce à un apprentissage en ligne à partir d'un ensemble de questions posées à l'utilisateur.

Mots clefs : Représentation de préférences, apprentissage, multi-utilisateurs, CP-net probabiliste, GAI-décomposition

Table des matières

Table des figures

Chapitre 1

Introduction

Sommaire

1.1 Contexte

Le développement de systèmes interactifs d'aide à la décision et de systèmes de recommandation a mis en évidence la nécessité de modèles capables d'utiliser les préférences d'un utilisateur pour l'orienter dans ses choix. La modélisation des préférences par le biais de formalismes de représentation compacte fait l'objet de travaux soutenus en intelligence artificielle depuis plus d'une quinzaine d'années notamment par Boutilier *et al.* [13, 14], Gonzales et Perny [39], Schiex *et al.* [62]. Ces formalismes permettent l'expression de modèles suffisamment flexibles et riches pour décrire des comportements de décisions complexes. Pour être intéressants en pratique, ces formalismes doivent de plus permettre l'élicitation des préférences de l'utilisateur, et ce en restant à un niveau admissible d'interaction. La configuration de produits combinatoires dans sa version *business to customer* [53] et la recherche à base de préférences (preference-based search ; voir par exemple [69]) constituent de bons exemples de ce type de problème de décision où les préférences de l'utilisateur ne sont pas connues *a priori*. Dans ce type d'application, l'interaction avec l'utilisateur doit rester en dessous de 0,25 s en

instantané, sans que la session totale (la phase d'élicitation totale) ne dépasse la vingtaine de minutes, et ce, alors que l'objet à recommander à l'utilisateur est à rechercher dans un ensemble combinatoire de possibilités.

Lorsque l'on souhaite représenter les préférences d'un utilisateur de façon qualitative et structurée avec des interactions « simples » entre attributs (notamment lorsqu'ils sont séparables), on peut faire appel à des représentations du type CP-nets ([12, 13, 14]) ou lexicographiques (Flach et Matsubara [30], Fraser [32]) ; l'un des intérêts de ces modèles est d'apprendre facilement et rapidement, par le biais de quelques questions, les préférences de l'utilisateur, (voir notamment Booth *et al.* [10], Dimopoulos *et al.* [26], Koriche et Zanuttini [49]).

L'inconvénient de ce type de structures est l'impossibilité de représenter certaines relations de préférence simples et intuitivement satisfaisantes entre objets combinatoires. Cette restriction vient notamment de la sémantique Ceteris Paribus (toute chose étant égale par ailleurs) des CP-nets empêchant ainsi de représenter certains ordres. Par exemple, si on considère deux variables binaires X, Y, il n'existe aucun CP-net permettant de représenter l'ordre de préférence suivant : $xy \succ \bar{x}\bar{y} \succ x\bar{y} \succ \bar{x}y$. Pour pallier cet inconvénient, on peut représenter ces relations de préférence par un modèle quantitatif tel que les GAI-décompositions, ou les réseaux GAI [3, 39]. Ils représentent une alternative aux modèles qualitatifs, en proposant de représenter la relation de préférence d'un utilisateur par une fonction d'utilité, dont on peut exploiter les propriétés de décomposabilité. Ces travaux utilisent la notion d'indépendance additive généralisée (d'où le terme GAI, pour *Generalized Addditive Independence*), introduite par Fishburn [28], pour décomposer la fonction d'utilité sur l'ensemble des caractéristiques des objets en une somme de petites fonctions d'utilités ne portant que sur certains groupes d'attributs. Ces GAI-décompositions permettent donc d'exprimer des interactions générales entre les attributs, tout en préservant une certaine décomposabilité du modèle. La question de l'élicitation des GAI-décompositions est au centre des travaux portant sur ce formalisme [17, 39] ; l'approche suivie jusqu'à présent suppose que la structure de la décomposition est donnée en entrée, ce qui est une hypothèse forte, que ne font pas, par exemple, les approches d'élicitation des préférences fondées sur des CP-nets.

Le principal problème de ces modèles de représentation, est qu'ils ne s'intéressent qu'aux relations de préférence d'un seul agent. Ce problème, dans un cadre de recommandation ou de configuration, nous oblige à inclure une phase d'élicitation des préférences de l'utilisateur. Même s'il existe des méthodes pour résoudre cette étape, elles sont coûteuses en temps de calcul. Ce temps ajouté

à celui de la recommandation ou la configuration peut décourager l'utilisateur à aller jusqu'au bout du processus. Il est donc intéressant de trouver une autre méthode pour connaître à l'avance ses préférences. Dans la pratique, il est plus facile de connaître les préférences d'un groupe d'agents notamment grâce aux historiques de ventes et de les utiliser lors de la recommandation d'un produit pour un agent[57]. Plusieurs modèles peuvent être utilisés pour représenter ce type d'information (réseaux bayésiens, MCP-nets, CP-nets possibilistes).

1.2 Problématique

Cette thèse a pour but d'approfondir le sujet sur les représentations de préférences dans les domaines combinatoires ainsi que les méthodes permettant d'apprendre de telles représentations.

La première partie de ces travaux s'est concentrée sur les représentations existantes afin de trouver de nouvelles méthodes d'apprentissage. Ils se sont naturellement orientés vers les GAI-décompositions étant donné que les algorithmes d'apprentissage permettaient d'éliciter des fonctions d'utilité lorsque la structure était connue mais ne permettaient pas d'apprendre cette structure. Nous nous sommes donc intéressé à ce formalisme afin de proposer une solution qui permet d'apprendre à la fois la structure et la fonction d'utilité d'une GAI décomposition.

Suite à ces travaux, la seconde partie de cette thèse s'est intéressée à une autre problématique liée aux représentations de préférence et aux méthodes d'apprentissage associées. En effet, dans des situations réelles telles que la configuration de produits, le système de configuration n'a pas forcément toutes les connaissances sur les préférences d'un utilisateur ni le temps pour les récolter afin d'apprendre une représentation compacte et utilisable pour le système. Afin de pallier ce problème, nous avons proposé un formalisme permettant de représenter les relations de préférence d'un ensemble d'utilisateurs. Pour cela, nous avons introduit les PCP-nets qui sont une extension probabiliste des CP-nets. Cette extension a été introduite indépendamment par Cornelio *et al.* [22].

Enfin, la dernière partie de cette thèse traite des problématiques d'apprentissage de cette nouvelle représentation. Pour cela, on s'est intéressé aux façons d'apprendre de manière efficace les préférences d'un groupe d'utilisateurs à partir de données facilement exploitables. Dans des situations réelles, les historiques de ventes sont un bon moyen d'obtenir des préférences sur un ensemble d'utilisateurs.

1.3 Guide de lecture

Cette thèse est composée de 3 grandes parties,et de 9 chapitres.

La **première partie** rappelle les notions importantes utilisées tout au long de cette thèse ainsi qu'un état de l'art sur les travaux existants. Elle est constituée des chapitres 2 et 3.

Le chapitre 2 présente toutes les notions nécessaires à la bonne compréhension de ce manuscrit. Il commence par une présentation des relations de préférence. Suite à cela, nous parlerons des fonctions d'utilités utilisées notamment dans le chapitre 4. Une fois ces notions définies, nous introduirons les notations sur les variables ainsi que la définition d'un domaine combinatoire. Étant donné que tous les formalismes étudiés utilisent des représentations graphiques, nous ferons un rappel sur la théorie des graphes. Enfin, nous présenterons les différentes méthodes d'apprentissage existantes.

Le chapitre 3 présente un ensemble de formalismes existants ainsi que les travaux liés à ces formalismes. Nous commencerons par présenter les arbres de préférences lexicographiques. Après s'être penché sur ce formalisme, nous étudierons deux travaux effectués sur l'apprentissage de ces arbres. Nous continuerons avec le formalisme des CP-nets et nous proposerons plusieurs approches de leur apprentissage. Puis nous parlerons de plusieurs extensions des CP-nets, en présentant les formalismes associés. Finalement, nous introduirons les réseaux bayésiens ainsi que les méthodes permettant de les apprendre.

La **deuxième partie** est constituée des chapitres 4 et 5. Elle traite de notre première contribution, à savoir l'apprentissage de GAI-décompositions.

Le chapitre 4 est un état de l'art des GAI. Il présente ce formalisme ainsi que les différentes requêtes qu'il peut utiliser. À la suite de cette présentation, nous étudierons deux algorithmes permettant d'éliciter les fonctions d'utilités d'un GAI-net.

Le chapitre 5 est consacré à notre première contribution : l'apprentissage des GAI-décompositions. Nous commencerons à montrer comment toute préférence élémentaire peut être représentée par une équation linéaire, dont les variables sont les éléments des tables d'une GAI-décomposition de degré k au maximum, l'idée étant de minimiser le degré de la décomposition apprise. Ensuite, nous nous intéresserons au problème de l'apprentissage d'une GAI-décomposition du point de vue théorique et algorithmique. Nous finirons par une présentation des résultats sur l'apprentissage minimal de GAI-décomposition.

La **troisième partie** est constituée des chapitres 6 et 7. Dans cette partie, nous sommes retournés sur les modèles qualitatifs, à savoir les CP-nets, et la

façon de représenter, non plus les préférences d'un utilisateur, mais d'un groupe d'utilisateurs.

Le chapitre 6 propose une extension des CP-nets : les CP-nets probabilistes. Dans un premier temps, nous présenterons cette nouvelle extension en donnant la sémantique ainsi que les requêtes qui peuvent être posées dans ce contexte. Dans un second temps, nous nous intéresserons à la requête de dominance pour les PCP-nets. Nous étudierons la complexité de cette requête dans le cas général, puis nous donnerons plusieurs résultats pour le cas des PCP-nets arborescents. Enfin, nous parlerons de la complexité de l'optimisation.

Le chapitre 7 est consacré à l'apprentissage des PCP-nets. Cette partie est composée de trois sous-parties. La première est consacrée à l'apprentissage des PCP-nets. Dans cette partie, nous montrerons comment apprendre un PCP-net de façon offline à partir d'un historique de vente. Puis, nous proposerons une méthode online qui met à jour un PCP-net à chaque nouvelle interaction avec un utilisateur. La seconde sous partie est consacrée à l'élicitation de l'objet optimal d'un utilisateur à partir d'un PCP-net. Enfin, la dernière partie parlera des résultats expérimentaux des deux méthodes.

Le chapitre 8 conclut notre travail. Nous y rappellerons l'ensemble de nos apports ainsi que les perspectives que nous voyons à ce travail.

Enfin, le chapitre 9.1 est un récapitulatif des notations utilisées dans cette thèse.

Notons pour terminer cette introduction que l'ensemble de ces travaux a été publié. En voici la liste :

- L'apprentissage de GAI-décompositions est présenté dans Bigot *et al.* [7]
- Les CP-nets probabilistes sont présentés dans Bigot *et al.* [6]
- L'apprentissage de CP-net probabiliste est présenté dans Bigot *et al.* [8]

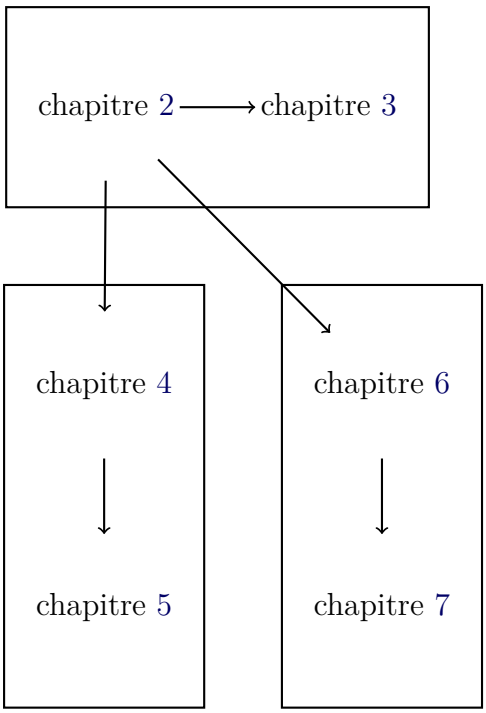


Figure 1.1 – *Guide de lecture des chapitres principaux. Les flèches indiquent les relations de dépendance entre ces chapitres.*

Première partie

Notions importantes, représentation et apprentissage de préférences

Introduction

Dans cette **première partie**, le chapitre 9.1 présentera toutes les notions nécessaires pour la bonne compréhension de ce manuscrit. Nous commencerons par des définitions des relations de préférence puis nous introduirons la notion de fonction d'utilité qui peut être utilisée dans les formalismes de représentation de préférences quantitatives. Nous poursuivrons en définissant ce qu'est une variable et un domaine combinatoire. Puis nous continuerons en effectuant un rappel sur la théorie des graphes afin d'être complètement à l'aise lors de la présentation des formalismes de représentation compacte de préférences. Enfin, nous terminerons ce chapitre par une présentation de l'apprentissage de relations de préférence dans le cas général.

Dans le second chapitre de cette partie, nous ferons l'état de l'art des différents formalismes permettant de représenter des préférences dans les domaines combinatoires. Pour chacun d'eux, nous parlerons de leurs avantages et de leurs inconvénients à travers différentes requêtes qu'il est possible d'effectuer. Pour chaque formalisme présenté, nous donnerons quand ils existent différents algorithmes permettant d'apprendre ce formalisme.

Dans ce chapitre de thèse, nous présenterons les formalismes de représentations de préférences suivants :

1. Les arbres de préférences lexicographiques présentés par Fraser [32]
2. Les réseaux de préférences conditionnelles (CP-nets) introduits par Boutilier *et al.* [14]
3. Les MCP-nets : une extension multi-agents des CP-nets proposées par Rossi *et al.* [61]
4. Les TCP-nets : une extension lexicographique des CP-nets proposées par Brafman et Domshlak [16]
5. Les UCP-nets : une extension quantitative des CP-nets introduite par Boutilier *et al.* [12]
6. Le dernier formalisme que nous verrons dans ce chapitre est celui du réseau bayésien. Ce formalisme ne permet pas de représenter des relations de préférence. Cependant, nous en parlerons car plusieurs notions utilisées dans les réseaux bayésiens seront utilisés dans la suite de cette thèse.

Nous avons décidé de parler de ces formalismes de représentation de préférences car ce sont des formalismes connus ayant fait l'objet de nombreuses recherches ces 15 dernières années notamment dans le cadre de l'apprentissage. Il

existe bien entendu d'autres formalismes permettant de représenter des relations de préférence qui ne seront pas présentés dans cette thèse tels que les BDI [60], les diagrammes d'influence [56], les VCSP [63] ou encore les langages de préférences comparatives et expressives [71].

Un autre formalisme important, le réseau généralisé additive indépendant (GAI) [3] permettant de représenter des relations de préférence, sera présenté dans le chapitre 4.

Chapitre 2

Notions Importantes

Sommaire

2.1 Relations de préférence

Soit un ensemble d'éléments $E = \{\ldots, x, y, z\}$, on appelle une relation binaire sur E un sous-ensemble $R \subseteq E \times E$. On écrit xRy pour $(x, y) \in R$ et $\neg xRy$ pour $(x, y) \notin R$

2.1.1 Relation binaire

Une relation binaire R peut être :

- réflexive $\Leftrightarrow\ \forall x \in E\ ,\ xRx$
- irréflexive $\Leftrightarrow\ \forall x \in E\ ,\ \neg(xRx)$
- symétrique $\Leftrightarrow\ \forall x, y \in E\ ,\ xRy \Rightarrow yRx$
- anti-symétrique $\Leftrightarrow\ \forall x, y \in E\ ,\ xRy \wedge yRx \Rightarrow x = y$
- transitive $\Leftrightarrow\ \forall x, y, z \in E\ ,\ xRy \wedge yRz \Rightarrow xRz$
- complète $\Leftrightarrow\ \forall x, y \in E\ ,\ xRy$ *ou* yRx
- faiblement complète $\Leftrightarrow\ \forall x, y \in E\ ,\ x = y$ *ou* xRy *ou* yRx
- acyclique $\Leftrightarrow\ \forall n \geqslant 2,\ \forall x_i \in E, i = 1, 2, \ldots, n, x_1Rx_2$ *et* x_2Rx_3 *et* $\ldots$ et $x_{n-1}Rx_n \Rightarrow x_1 \neq x_n$

2.1.2 Notion d'ordre

Définition 2.1 (préordre)**.** *Une relation binaire R est un préordre si et seulement si elle est transitive et réflexive. Un préordre est dit total si et seulement si il est complet. Un ordre est un préordre antisymétrique et de plus, il est total si le préordre est faiblement complet.*

Définition 2.2 (ordre strict)**.** *Une relation binaire R est un ordre strict si et seulement si elle est transitive, irréflexive et anti-symétrique. De plus, l'ordre strict est total si la relation binaire est complète.*

Exemple 2.1. *Dans* $\mathbb{R}$ *la relation* $\geqslant$ *est un ordre total, et* $>$ *est un ordre total strict.*

2.1.3 Relations de préférence.

Soit une relation binaire $\succeq$ sur E telle que $x \succeq y$ signifie que "x est au moins aussi préféré que y".

Á partir de cette relation, on peut définir deux nouvelles relations :

- La relation $\sim$ est définie par $x \sim y$ si et seulement si $x \succeq y$ *et* $y \succeq x$, et est lue "x et y sont indifférents".
- La relation $\succ$ est définie par $x \succ y$ si et seulement si $x \succeq y$ *et* $y \not\succeq x$, et est lue "x est strictement préféré à y".

Donc $\succ$ est la partie asymétrique de $\succeq x$ et $\sim$ est sa partie symétrique. On dira que la relation $\succeq x$ est une relation de préférence large, $\succ$ une relation de préférence stricte et $\sim$ une relation d'indifférence.

Enfin on notera les relations opposés $\prec$ et $\preceq$ des relations de préférence $\succ$ et $\succeq$ tels que :

$$y \prec x \Leftrightarrow x \succ y \quad et \quad y \preceq x \Rightarrow x \succeq y$$

Proposition . *Lorsqu'une relation de préférence $\succeq$ sur E est un préordre alors :*

1. *la relation $\sim$ est une relation d'équivalence, c'est à dire réflexive, symétrique et transitive.*
2. *la préférence $\succ$ est un ordre strict.*
3. *$\forall x, y \in E, x \succeq y \Rightarrow x \succ y$ ou $x \sim y$*
4. *$\forall x, y \in E$ si le préordre est complet, alors au moins l'une de ces affirmations est vraie : $x \succ y, y \succ x, x \sim y$*

2.2 Fonction d'utilité

Pour représenter des préordres totaux, on utilise souvent une fonction d'utilité notamment grâce à ses propriétés extrêmement intéressantes introduites par Fishburn [29].

Théorème 2.1. *Lorsque A est un ensemble fini ou dénombrable, une relation $\succeq$ sur A est un préordre complet si et seulement s'il existe une fonction $u : A \to \mathbb{R}$ tel que $\forall a, b \in A$ on a :*

$$a \succ b \Leftrightarrow u(a) > u(b)$$

L'avantage d'une fonction d'utilité u est qu'elle est unique à une transformation strictement croissante près. On dit que u définit une échelle ordinale, c'est à dire qu'une utilité u n'a pas de signification absolue, seul l'ordre qu'elle définit est significatif.

2.3 Variables et domaine combinatoire

On considère un ensemble χ de n variables, appelées aussi attributs. Chaque variable $X \in \chi$ possède un domaine fini noté $\underline{X}$.

X est une variable binaire si son domaine contient exactement deux valeurs. Par convention, on les notera x et $\bar{x}$. Si $U \subseteq \chi$ est un sous-ensemble de variables, alors le produit cartésien des domaines des variables de U est noté $\underline{U}$.

On notera o une instanciation complète de χ, c'est-à-dire une fonction $o : \chi \to \underline{\chi}$ qui associe à toutes variables une des valeurs de son domaine. On dit aussi que o est un objet. De plus, on notera o[X] (resp. o[U]) la projection de o sur la variable X (resp. la projection de o sur l'ensemble U).

Enfin on note $\mathcal{O}$ l'ensemble d'objets représenté par le produit cartésien de χ.

2.4 Théorie des graphes.

Au fil des chapitres de cette thèse, nous utiliserons des notions de graphes ainsi que plusieurs propriétés qui les caractérisent. Afin d'éviter toute confusion future, nous donnons ici toutes les notions sur les graphes nécessaires à la bonne compréhension de cette thèse.

Définition 2.3 (graphe orienté)**.** *Un graphe orienté $G = (\mathcal{V}, \mathcal{E})$ est un couple composé d'un ensemble de sommets $\mathcal{V}$ et d'un ensemble d'arcs $\mathcal{E}$ reliant des sommets entre eux, où un arc est un couple de sommets.*

Définition 2.4 (graphe non orienté)**.** *Un graphe non orientéG $= (\mathcal{V}, \mathcal{E})$ est composé d'un ensemble de sommets $\mathcal{V}$ et d'un ensemble d'arêtes $\mathcal{E}$ reliant des sommets entre eux, où une arête est une paire d'objets. Un graphe non-orienté est un cas particulier des graphes orienté où pour chaque arc allant de S à S', il existe un arc allant de S' à S.*

On appelle chemin C de G une suite de sommets $(X_0, X_1, \ldots, X_{n-1}, X_n)$ où $X_i \in \mathcal{V}$ telle que deux sommets consécutifs quelconques X_i et X_{i+1} sont reliés par un arc/arête de G. On appelle circuit un chemin qui possède un sommet X_i qui est présent au moins deux fois dans la suite de sommets. De plus dans le cas des graphes orientés, on appelle les parents d'un sommet S, noté Pa(S), l'ensemble des sommets S' tel qu'il existe un arc partant de S' et allant sur S ; et fils d'un sommet S l'ensemble des sommets S'' tel qu'il existe un arc partant de S et allant sur S''. Enfin, on appelle un sommet une feuille si et seulement si il possède au moins un parent et aucun fils.

Définition 2.5 (acyclicité)**.** *Un graphe orienté G est dit acyclique lorsqu'il ne possède pas de circuit. Il est dit cyclique lorsqu'il possède au moins un circuit*

Définition 2.6 (racine)**.** *On appelle un sommet S une racine d'un graphe si :*

- *il existe au moins un arc sortant et aucun arc entrant vers S dans le cas des graphes orientés.*

Un graphe acyclique fini possède forcément au moins une racine.

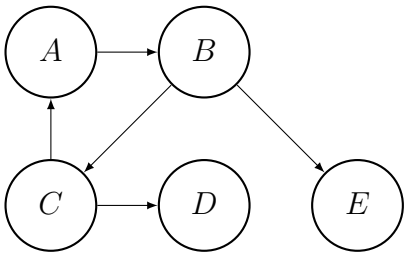


Figure 2.1 – *Exemple d'un graphe cyclique*

Exemple 2.2. *Le graphe de la Figure 2.1 possède un circuit sur les sommets ABC. Ils est donc cyclique. Si l'arc (CA) était inversé, il serait alors acyclique.*

Définition 2.7 (clique)**.** *Soit un graphe non orienté G. Un ensemble $\mathcal{V}' \subseteq \mathcal{V}$ est une clique si et seulement si $\forall x, y \in \mathcal{V}'$ il existe une arête dans G les reliant.*

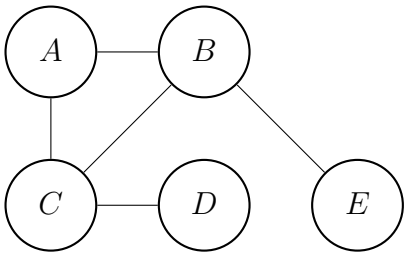


Figure 2.2 – *Exemple d'un graphe non orienté avec la clique $\{A, B, C\}$*

Exemple 2.3. *Selon le graphe de la Figure 2.2, $\{A, B, C\}$ est une clique. Par simplicité, on notera par la suite XYZ l'ensemble $\{X, Y, Z\}$ définissant une clique.*

Définition 2.8 (graphe complet)**.** *Un graphe non orienté G est dit complet lorsque toutes les paires de sommets sont reliées.*

Définition 2.9 (biparti)**.** *Un graphe G est dit biparti s'il est possible de partitionner l'ensemble de ses sommets en deux classes de telle sorte que deux sommets d'une même classe ne possèdent pas d'arc / d'arête les reliant.*

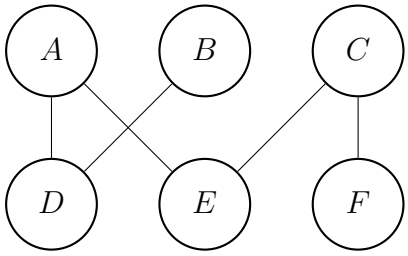


Figure 2.3 – *Exemple d'un graphe non orienté biparti*

Définition 2.10 (Connexe)**.** *Un graphe G est dit connexe si pour n'importe quelle paire de sommets de G, il existe un chemin allant d'un sommet à l'autre.*

Exemple 2.4. *Le graphe biparti de la Figure 2.3 est un graphe connexe car il existe un chemin pour n'importe quelle paire de sommets. Au contraire, le graphe cyclique de la Figure 2.1 n'est pas connexe puisqu'il est impossible d'aller du sommet D au sommet E et vice versa.*

Définition 2.11 (arbre)**.** *Un graphe connexe G est appelé arbre si pour n'importe quelle paire de sommets il existe exactement un seul chemin possible les reliant.*

Définition 2.12 (hauteur)**.** *On définit la hauteur pour une racine donnée comme étant le nombre de sommets composant le chemin le plus long reliant la racine à la feuille la plus éloigné de la racine.*

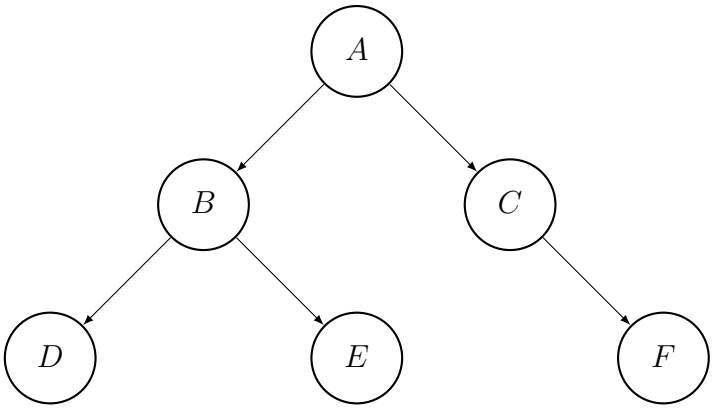


Figure 2.4 – *Exemple d'un arbre*

Exemple 2.5. *La Figure 2.4 nous montre un arbre de hauteur 3 avec le nœud A comme racine.*

On appel polyarbre, un graphe acyclique où il existe au plus un chemin entre deux nœuds.

2.5 Apprentissage de préférence

L'apprentissage de préférence consiste à déterminer et représenter de manière compact un ordre sur un ensemble d'objets qu'un utilisateur peut choisir.Plus précisément, on suppose qu'un utilisateur possède un ordre total sur un ensemble d'objets $\mathcal{O}$ (par exemple une liste de voitures qu'il peut acheter) et on cherche par le biais d'un apprentissage à apprendre cette ordre. La difficulté de cet apprentissage est que l'on cherche à apprendre le plus rapidement possible une représentation la plus précise possible avec le moins d'exemples possible.

2.5.1 Exemple

Un exemple est une information donnée par un utilisateur reflétant sa relation de préférence. Le plus souvent, un exemple est représenté par triplet (o, R, o'), o et o' étant des objets.

Définition 2.13 (exemple de comparaison de la forme (o, R, o'))**.** *Soit $\succeq$ une relation de préférence sur $\underline{\chi}$. Un* exemple *e de $\succeq$ est un triplet de la forme (o, R, o'), où $o, o' \in \underline{\chi}$ sont des objets et R est une relation dans $\{\succ, \succeq, \sim, \precsim, \prec\}$.*

Pour un ensemble *d'exemples E, on note O_E l'ensemble des objets qui interviennent dans au moins un exemple de E. Par simplicité, nous utiliserons seulement les relations $\succ$ et $\succeq$ puisqu'elles permettent de représenter toutes les autres relations.*

Dans cette thèse, nous utiliserons dans le Chapitre 7 un autre type d'exemple défini par un objet o seul qui signifie que o est l'objet qui est préféré à tous les autres.

Définition 2.14 (exemple d'objet optimal de la forme o)**.** *Soit $\succeq$ une relation de préférence sur $\underline{\chi}$. Un exemple e est un singleton o, où $o \in \underline{\chi}$ de telle sorte que $\nexists o' \in \chi$ tel que $o' \succeq o$*

2.5.2 Apprentissage supervisé / non supervisé

Un apprentissage peut être non supervisé, c'est-à-dire que l'apprentissage divise un ensemble d'éléments en sous-groupes tels que les éléments étant dans un sous-groupe sont considérés comme ayant des caractéristiques communes. Un exemple connu d'apprentissage non supervisé est le partitionnement de données [54].

Il peut aussi être supervisé, c'est-à-dire que l'apprenant essaye d'apprendre automatiquement un ensemble de règles à partir d'une base de données d'apprentissage contenant des exemples. Une base de données d'apprentissage est un ensemble de couples (x_i, u_i) tels que x_i est une entré prise dans un espace d'entrées $\underline{\chi}$ et u_i est une étiquette prise dans un espace de sortie $\mathcal{Y}$. On définit un concept comme une étiquette qui est restreint à deux valeurs(par exemple $\{0, 1\}$ ou encore $\{\succ, \prec\}$). On parle alors d'apprentissage supervisé de concept. Enfin si l'espace de sortie est fini, on parle de classification. Dans cette thèse, tous les algorithmes étudiés sont des algorithmes d'apprentissage supervisé.

On différencie deux cadres d'apprentissage permettant d'apprendre une relation de préférence à partir d'un ensemble d'exemple.

Apprentissage passif

L'apprentissage passif consiste à observer les choix d'un utilisateur sans interagir avec celui-ci. Cette observation se traduit par la récolte d'un ensemble d'exemples. L'avantage principal de cette méthode est qu'elle permet de faire cet apprentissage sans avoir à poser de question à l'utilisateur [52, 68]. Le principal problème est qu'elle nécessite la plupart du temps un nombre important d'exemples pour apprendre les relations de préférence que l'utilisateur a sur $\underline{\chi}$. Comme nous le verrons par la suite lors de la présentation de résultat d'apprentissage passif dans cette thèse, le nombre important d'exemples nécessaires se traduit par un temps d'apprentissage élevé pour apprendre correctement les relations de préférence de l'utilisateur.

Apprentissage actif

Le second type d'apprentissage est l'apprentissage actif. Comme pour l'apprentissage passif, il consiste à observer l'utilisateur grâce à un ensemble d'exemples [2]. Mais contrairement à l'apprentissage passif, il se fait en ligne et surtout le système, appelé apprenant, se permet de poser des questions à l'utilisateur afin

d'améliorer la qualité et la rapidité de l'apprentissage. En fonction des besoins, il existe différents types de questions que l'on peux poser à un utilisateur.

- La première requête est la requête d'équivalence(EQ). Elle consiste à proposer la représentation de préférences apprise à l'utilisateur, qui est censé être cohérente avec sa relation de préférence. Si ce modèle n'est pas correcte, l'utilisateur donne un contre exemple à l'apprenant.
- La seconde requête est la requête d'appartenance(MQ). Elle consiste à proposer un exemple à l'utilisateur afin de voir s'il est cohérent avec sa relation de préférence.

Le but de ces requêtes est d'apprendre avec le moins de ressources possible une représentation de préférences (c'est-à-dire avec le moins de requêtes possible et le moins de temps de calcul possible).

2.5.3 Compatibilité

En apprentissage, lorsqu'une représentation de préférences satisfait un ensemble d'exemples de comparaison, on dit que la représentation de préférences est cohérente avec les exemples. Dans le cas d'une représentation d'ordre partiel, cette cohérence est restrictive. En effet, si un utilisateur possède un ordre totale, il est possible qu'il donne comme exemple $o \succ o'$ alors que pour la représentation de préférences, les deux objets sont incomparables. Pour cela, Lang et Mengin [50] ont introduit deux autres notions de cohérence : la compatibilité faible et la compatibilité forte.

Compatibilité faible

On dit qu'une représentation de préférences $\mathcal{F}$ est faiblement compatible avec un ensemble d'exemples E lorsque pour chaque exemple $(o_1, \succ, o_2) \in E$ on a $F \not\models o_2 \succ o_1$, c'est à dire que $\mathcal{F}$ ne satisfait pas $o_2 \succ o_1 \in E$.

Compatibilité forte

E est fortement compatible avec $\mathcal{F}$ si et seulement si $E \cup \succ_{\mathcal{F}}$ est acyclique.

2.5.4 Apprenabilité

L'une des difficultés en apprentissage est de savoir si une représentation est apprennable ou non. Que ce soit pour l'apprentissage passif, ou l'apprentissage actif, plusieurs outils existent afin de dire si un problème est apprenable, c'est-à-dire s'il est possible d'apprendre une représentation compacte de la relation de préférence d'un utilisateur ou encore combien d'exemples sont nécessaires pour apprendre exactement cette représentation. Dans cette thèse, nous utiliserons la notion de VC-dimension ainsi que la notion d'Apprentissage PAC.

2.5.4.1 Apprentissage PAC

L'*Apprentissage PAC* (« probablement approximativement correct ») introduit par Valiant [68] permet de formaliser si un concept est apprenable ou non en temps polynomiale.

Si on se place dans un cadre d'apprentissage passif : l'apprenant reçoit un nombre m d'exemples du type (o, R, o') d'un concept cible C^* (Plus tard dans cette thèse, on utilisera $\succeq^*$ comme concept cible pour l'apprentissage de préférences), puis doit calculer un concept C. Le nombre m d'exemples demandés est choisi par l'apprenant en fonction du nombre de variables n et de deux paramètres $\varepsilon, \delta \in]0, 1[$. Chaque exemple est tiré aléatoirement, indépendamment et avec remise, selon une distribution de probabilités p sur $\underline{\chi} \times \underline{\chi}$ qui est fixe mais inconnue de l'apprenant. Pour o, o' ainsi tirés, l'exemple (o, R, o') est communiqué à l'apprenant, où R est déterminé sans bruit par le concept cible C^*. Un PAC-apprenant est un algorithme qui possède les propriétés suivantes :

- avec probabilité au moins $1 - \delta$, il retourne un concept C qui est ε-correct, c'est-à-dire que la masse, selon la distribution de probabilité p utilisée pour générer les exemples, des couples (o, o') incorrectement classés par C est au plus ε ; formellement, $\sum\{p(o, o') \mid oCo' \text{ mais } \neg(o\ C^*\ o')\} \leq \varepsilon$,
- le nombre m d'exemples demandés par l'algorithme est polynomial en $n, 1/\varepsilon, 1/\delta$,
- l'algorithme est polynomial en $n, 1/\varepsilon, 1/\delta$, en comptant pour une unité la demande et la réception d'un exemple.

Un problème est dit *PAC-apprenable* s'il existe un PAC-apprenant pour lui.

Intuitivement, ce cadre formalise les situations dans lesquelles l'apprenant observe pendant un certain temps des objets de son environnement (ceux qui se présentent, sans qu'il ne puisse les choisir), et est aidé par un « enseignant », qui lui fournit pour chaque objet rencontré l'étiquette correcte. La distribution de

probabilité p permet de formaliser des situations dans lesquelles tous les objets ne sont pas aussi courants les uns que les autres dans l'environnement. Ainsi, un objet rare aura peu de chances d'être rencontré pendant la phase d'apprentissage (probabilité faible selon p), mais en contrepartie, l'apprenant sera peu pénalisé pour une erreur sur le classement d'un tel objet. De façon plus générale, le résultat générique de Blumer *et al.* [9] est qu'un nombre d'exemples m en $O(\max(\frac{1}{\varepsilon}\log\frac{1}{\delta}, \frac{VC-dimension}{\varepsilon}\log\frac{1}{\varepsilon}))$ suffit pour apprendre un concepte C telle que l'ordre associé $\succeq_C$ soit ε-correct

2.5.4.2 VC-dimension

La *dimension de Vapnik-Chervonenkis* (VC-dimension) d'une classe de concepts $\mathcal{C}$ mesure en un certain sens l'expressivité de $\mathcal{C}$ et la difficulté d'apprendre des concepts de $\mathcal{C}$ à partir d'exemples. Intuitivement, la VC-dimension de $\mathcal{C}$ mesure le plus grand nombre d'exemples « indépendants » les uns des autres, en ce sens que leurs étiquettes ne dépendent d'aucune manière les unes des autres. Plus formellement, nous utiliserons les deux définitions suivantes, spécialisées pour notre contexte (pour rendre les définitions concrètes, voir par exemple l'appartenance de (o, o') au concept C, c'est-à-dire $c(o, o') = 1$, comme la validité de l'assertion $o \succeq o'$).

Définition 2.15 (éclaté)**.** *Soit $\mathcal{C}$ un ensemble de concepts binaires $C : \underline{\chi} \times \underline{\chi} \to \{0, 1\}$. Un sous-ensemble de couples d'objets $O \subseteq \underline{\chi} \times \underline{\chi}$ est dit* éclaté *par $\mathcal{C}$ si pour toute partition $\{O^1, O^0\}$ de O, il existe un concept $C \in \mathcal{C}$ vérifiant $\forall (o, o') \in O^1, C(o, o') = 1$ et $\forall (o, o') \in O^0, C(o, o') = 0$.*

Définition 2.16 (VC-dimension)**.** *Soit $\mathcal{C}$ un ensemble de concepts binaires. La* dimension de Vapnik-Chervonenkis *(VC-dimension) de $\mathcal{C}$ est la taille du plus grand ensemble d'objets $O \subseteq \underline{\chi} \times \underline{\chi}$ qui est éclaté par $\mathcal{C}$.*

Même s'il existe des résultats plus généraux, les résultats d'apprenabilité (ou de non-apprenabilité) en fonction de la VC-dimension concernent la VC-dimension de concepts binaires.

On voit donc une relation de préférence $\succeq$ comme deux relations, notées $\succ$ et $\prec$. Formellement, la relation $\succ$ associée à une relation de préférence $\succeq$ est le concept binaire $C_\succ \subseteq \chi \times \chi$ défini par $C_\succ(o, o') = 1$ pour $o \succeq o'$ et $o' \not\succeq o$, et $C_\succ(o, o') = 0$ sinon (c'est-à-dire si l'on a $o \preceq o'$), et le concept $C_\prec$ est défini de manière duale.

La VC-dimension est utilisée pour déterminer le nombre d'exemples nécessaires pour apprendre un concept. Elle est donc importante pour déterminer si

un problème est apprenable ou non. En effet, selon la définition de l'apprentissage PAC, un concept est apprenable si le nombre d'exemple est polynomiale. Donc si la VC-dimension n'est pas polynomiale pour un concept donné, on saura qu'il n'est pas apprenable.

Chapitre 3

Étude bibliographique de représentations et d'apprentissage de représentation de préférence

Sommaire

3.1 Préférence Lexicographique

3.1.1 Arbre de préférence lexicographique

Une comparaison lexicographique [30, 32] consiste à ordonner une paire d'objets (o, o') en regardant la séquence de variables qui définit les objets jusqu'à ce qu'une variable X soit atteinte de telle sorte que $o[X] \neq o'[X]$. Les deux objets sont ensuite classés en fonction d'une relation locale de préférence sur les valeurs de cette variable. Une telle comparaison utilise donc une relation d'importance entre les variables ainsi qu'une relation de préférence locale sur le domaine de chaque variable. Ces préférences locales peuvent être conditionnées par les relations d'importance des variables.On suppose que les préférences sur les valeurs d'une variable X ne dépendront que des variables qui sont plus importantes qu'elle.

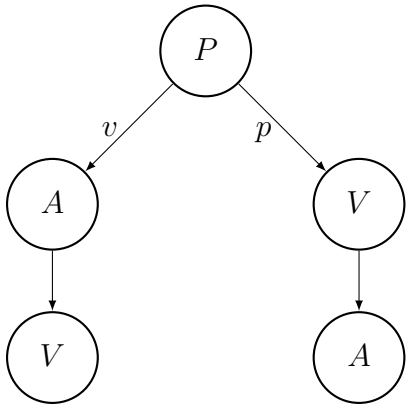
De plus, la relation d'importance peut elle aussi être conditionnelle.

La relation d'importance entre les variables peut être représentée par un arbre d'importance(AI) [32].

Définition 3.1 (arbre d'importance)**.** *Un arbre d'importance AI sur un ensemble de variables χ est un arbre dont les nœuds sont étiquetés par des variables de χ de telle sorte que chaque variable apparaisse au plus une fois dans chaque branche, et de telle sorte que les arcs entre un nœud non feuille marqué par la variable X et ses enfants sont étiquetés avec des ensembles disjoints de $\underline{X}$. Un tel arbre est dit complet si chaque variable de χ apparaît dans toutes les branches de l'arbre et si pour chaque nœud, on a des arcs sortant pour toute les valeurs possibles de la variable.*

On note Anc(n) l'ensemble des ancêtres du nœud n, c'est-à-dire les nœuds sur le chemin de la racine à n, n étant exclu. On utilise aussi Anc(n) comme étant l'ensemble des variables qui étiquettent les nœuds ancêtres de n. Enfin, on note $\underline{Anc(n)}$ le produit cartésien des étiquettes des arcs sur le chemin de la racine à n.

Lorsque l'on souhaite comparer deux objets en utilisant un ordre lexicographique, on descend dans l'arbre jusqu'à atteindre le premier attribut X qui différencie les deux objets. Une fois cette variable identifiée, il reste à voir quelle valeur de X est préférée à l'autre. C'est là qu'intervient la représentation des préférences locales.



(a) *Arbre d'importance lexicographique*

P	$v > p$
V	$p : b > r$
V	$vf : r > b$
V	$vz : b > r$
A	$p : z > f$
A	$v : f > z$

(b) *Préférence lexicographique locale*

Figure 3.1 – *Structure lexicographique*

Définition 3.2 (Préférence locales)**.** *Une règle de préférence locale (pour une variable X d'un ensemble de variables χ) est une expression de la forme $X\ u : x > \bar{x}$ où $u \in \underline{U}$, avec $U \subset \chi$, $X \in \chi - U$ et $>$ un ordre total strict sur $\underline{X}$. On appelle table de préférence locale de X l'ensemble des préférences locales sur X.*

Afin d'avoir une relation de préférence lexicographique, il suffit d'associer les préférences locales avec l'Arbre d'Importance.

Définition 3.3. *Soit T un AI, n un nœud de T. Une règle $X, u :>$ est dite applicable au nœud n si*

1. *n porte le label X.*
2. *$u \in \underline{Anc(n)}$.*

Une table de préférence locale sur une variable X, notée P, est un ensemble de règles de la forme $X\ u : x > \bar{x}$.

P est dite complète pour T si pour chaque nœud X de T, et chaque instanciation u , il y a exactement une seule règle applicable pour X connaissant u.

Définition 3.4. *Une Structure de Préférence Lexicographique(SPL) est une paire (T, P) où T est un arbre d'importance et P un ensemble de tables de préférences locales. La structure est dite complète si et seulement si T et P sont complets.*

Exemple 3.1. *Considérons 3 attributs composant un repas. Le Plat (P) est composé de 2 valeurs possibles le poisson (p) et la viande (v). Le Vin (V) peut être*

du rouge(r) ou du blanc(b). Enfin l'accompagnement peut être du riz(z) ou des frites (f). La SPL de la Figure 3.1 est une représentation de préférences lexicographiques pour ce repas. On y voit que le choix du plat est le plus importante et que la viande est préférée au poisson. Si on choisit du poisson, on préférera choisir d'abord le vin tel que le vin blanc est préféré au vin rouge, puis l'accompagnement et pour l'accompagnement, le riz est préféré aux frites. Alors que si l'on choisit la viande, on préférera choisir l'accompagnement avant le vin.

Ainsi, une SPL permet de définir un ordre partiel sur un ensemble d'objets de la façon suivante.

Définition 3.5. *Une SPL G=(T,P) définit un ordre partiel strict $\succ_G$ sur un ensemble d'objets de la façon suivante : soit une paire d'objet (o, o'). On descend l'arbre à partir de la racine en suivant les arcs des affectations de o et o' jusqu'à atteindre le premier nœud n marqué par X tel que $o[X] \neq o'[X]$. En effet, s'il existe une règle $(X, u :>)$ dans la table applicable de n, étant donné $u = Anc_o(X) = Anc'_o(X)$, alors on a $o \succ_G o'$ (resp. $o' \succ_G o$) si et seulement si $o[X] > o'[X]$ (resp. $o'[X] > o[X]$).On dit alors que n ordonne o et o'. Si une telle règle n'existe pas ou si aucun nœud ne permet d'ordonner (o, o'), on dit alors que les deux objets sont incomparables.*

Exemple 3.2. *En reprenant la SPL de la figure 3.1, l'objet optimal est vfr. En effet quelque soit $o' \in \underline{P} \times \underline{V} \times \underline{A}$ et $o' \neq vfr$ on a $vfr \succ_G o'$. De plus, $\forall x, x' \in \underline{A}$ et $\forall y, y' \in \underline{V}$ on a toujours $vxy > px'y'$ puisque le plat est l'attribut le plus important selon l'ordre lexicographique et que la préférence locale sur P dit que $v > p$.*

Enfin, toute relation de préférence définie par un SPL est transitive et irréflexive. Elle est complète si et seulement si la structure est complète.

3.1.2 Apprentissage d'un arbre lexicographique

En ce qui concerne l'apprentissage de préférence lexicographique, plusieurs travaux ressortent [10, 27, 32, 65, 73] . Nous présenterons deux travaux parlant d'arbres lexicographiques. Á savoir, les travaux de Booth *et al.* [10] et ceux de Bräuning et Hüllermeyer [18].

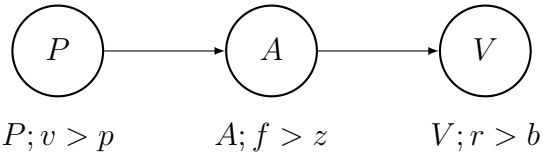


Figure 3.2 – *Structure lexicographique inconditionnelle*

3.1.2.1 Apprentissage de différente classes de préférences lexicographiques

Booth *et al.* [10] propose d'apprendre, à partir d'un ensemble d'exemples E du type $(o \succ o')$, un SPL G est dit cohérent avec l'ensemble d'exemples, si pour tous les exemples $(o \succ o')$ de E, on a $o \succ_G o'$.

Pour cela, ils s'intéressent entre autres à deux classes de SPL :

- les SPL avec des préférences conditionnelles (SPLC)
- les SPL avec des préférences inconditionnelles (SPLIC)

Définition 3.6. *Un SPLC est un arbre lexicographique où chaque arc de l'arbre, partant du nœud n vers le nœud m, est étiqueté avec une valeur appartenant au domaine de l'attribut du nœud n. De plus, pour chaque nœud n qui correspond à un assignement partiel u, la table de préférence locale possède une règle du type* $(X\ u : x > \bar{x})$ *où X est l'attribut qui marque n.*

Définition 3.7. *Un SPLIC est un arbre lexicographique possédant une structure linéaire où chaque arc est étiqueté par l'ensemble des valeurs du domaine du nœud parent correspondant et dont la table de préférences locales contient une règle inconditionnelle de la forme* $(X; \emptyset :>)$ *pour chaque attribut X apparaissant dans l'arbre.*

La classe des SPLC est la classe de tout les arbres, elle englobe toute les autres classes définies dans [10]. Les SPLIC est donc un cas particulier des SPLC où l'arbre est une chaîne et où les préférences sont inconditionnelle.

De la même façon un SPLIC est un arbre lexicographique composé de préférences locales inconditionnelles et où l'importance des attributs est inconditionnelle.

Exemple 3.3. *L'exemple de la Figure 3.2 représente un SPL inconditionnelle alors que la la Figure 3.1 nous montre un SPL conditionnel.*

Algorithm 1: apprentissage d'un arbre lexicographique

Input: χ : un ensemble de variables,
E : un ensemble d'exemples défini sur χ,
P : un ensemble de règles de préférences locales (initialement vide)
Output: Un SPL qui satisfait $\mathcal{E}$ ou échec

```
1  T ← une racine non étiquetée ;
2  while T possède un nœud non étiqueté do
3      Choisir un noeud n non étiqueté de T ;
4      (X, nouvellerégle) ← ChoisirAttribut(E(n),Anc(n),P) ;
5      if X=Echec then
6          return Echec ;
7      étiqueter n avec X ;
8      P ← P ∪ nouvelleRegle ;
9      L ← genereLabel(E(n), X) ;
10     foreach l ∈ L do
11         ajouter un nouveau nœud non étiqueté à T, attaché à n avec le
           label l sur l'arc. ;
12 return T,P
```

Le principe de l'algorithme 1 consiste à apprendre l'arbre de façon récursive en partant de la racine et en allant jusqu'aux feuilles.

A un instant donné, lorsque l'on a un nœud non étiqueté n , l'étape 4 consiste à prendre l'ensemble d'exemples $E(n) = \{(o, \succ, o') \in E | (o[Anc(n)] = o'[Anc(n)]) \in Anc(n)\}$ qui correspond à une partie des préférences qui ne sont pas encore représenté à cette étape. On cherche donc à trouver un attribut X connaissant Anc(n) qui peut être utilisé pour être compatible avec l'ensemble d'exemple $E(n)$. Ainsi que de trouver un ensemble de règles de préférences locales $X : u >$ qui est compatible avec l'ensemble d'exemples. On ajoute alors l'attribut X comme l'étiquette de n ainsi que l'ensemble de règles dans P. Ensuite, la ligne 9 regarde les valeurs de X qui ne sont pas encore traitées afin de préparer les arcs qui seront ajoutés du nœud n vers ses fils.

Apprendre un SPLC de façon exacte se fait en $O(2^n)$ [10] alors qu'apprendre un SPLIC de façon exacte se fait en $O(nlog(n))$ [27]. On voit donc qu'il est difficile d'apprendre un arbre lexicographique lorsque celui-ci n'est pas un inconditionnel.

3.1.2.2 Apprentissage d'arbre lexicographique de préférences conditionnelles

Nous verrons que dans leur articles, Bräuning et Hüllermeyer [18] proposent une généralisation des ordres lexicographiques. Nous parlerons ensuite de leur algorithme qui, comme celui de Booth *et al.* [10], utilise des comparaisons entre des objets de la forme $(o, \succ, o')$ afin d'apprendre leur représentation.

Bräuning et al proposent une extension des arbres lexicographiques présentés plus haut afin de les rendre plus expressifs. Pour cela, on suppose qu'un sous-ensemble de variables peut avoir la même importance. Il est alors possible d'avoir non pas une variable par nœud mais un ensemble de variables.

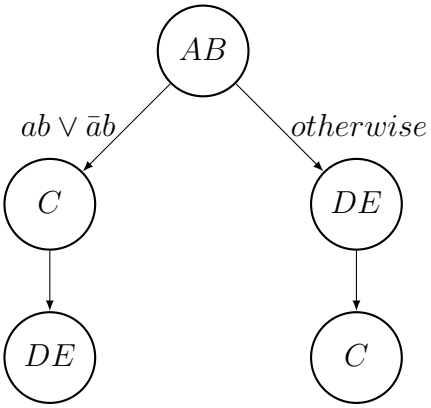
Afin d'évaluer la performance d'un arbre lexicographique T de préférences conditionnelles vis-à-vis d'un ensemble d'exemples E, on peut utiliser deux mesures de performance :

- la mesure de correction définie par :

$$CR(\succeq_T, E) = \frac{|C| - |D|}{|C| + |D|}$$

- la mesure d'exhaustivité définie par :

$$CP(\succeq, E) = \frac{|C| + |D|}{|E|}$$



(a) *Arbre d'importance lexicographique*

AB	$ab > a\bar{b} > \bar{a}b > \bar{a}\bar{b}$
C	$ab : c > \bar{c}$
C	$\bar{a}b : c > \bar{c}$
C	$\bar{a}\bar{b}de : c > \bar{c}$
C	$otherwise : \bar{c} > c$
DE	$a\bar{b} : de > d\bar{e} > \bar{d}e > \bar{d}\bar{e}$
DE	$\bar{a}bc : d\bar{e} > \bar{d}\bar{e} > \bar{d}e > de$
DE	$\bar{a}b\bar{c} : de > \bar{d}\bar{e} > d\bar{e} > \bar{d}e$
DE	$otherwise : \bar{d}e > d\bar{e} > \bar{d}\bar{e} > de$

(b) *Préférence lexicographique locale*

Figure 3.3 – *Exemple d'arbre lexicographique conditionnel selon Bräuning et Hüllermeyer [18]*

avec :

$$C = \{(o', \succ, o) \in E | o' >_T o\}$$

$$D = \{(o', \succ, o) \in E | o >_T o'\}$$

La mesure de correction CR est comprise entre -1 (désaccord complet) et +1 (complètement d'accord) ; et la mesure d'exhaustivité, CP est comprise entre 0 (pas de comparaison) et +1 (comparaison complète).

En ce qui concerne l'apprentissage, on utilise un algorithme glouton qui apprend de façon itérative la structure de l'arbre en partant de la racine jusqu'aux feuilles. Il est composé de 4 parties :

Création d'un nœud

Pour construire un nœud, on cherche à trouver un ensemble de variables $\mathcal{X}' \subseteq \mathcal{X}$ cohérent avec notre ensemble d'exemples $\mathcal{E}$. Pour cela, on prend les candidats possibles et on garde le meilleur. Ce choix est fait en maximisant la mesure de correction et d'exhaustivité sur les $\mathcal{X}'$.

Limitation de la taille des candidats

Sans limitation du nombre de variables que l'on peut attacher à un nœud, il serait possible de toutes les mettre ce qui rendrait la taille des tables de préfé-

rences locales exponentielle. Pour éviter cela, on limite la taille des candidats à une cardinalité $gmax$. Dans leur expérimentation, M. Bräuning et E. Hullermeier se sont arrêtés à un gmax de 2.

Récursion

Une fois qu'un ensemble de variables $\mathcal{X}'$ pour un nœud est trouvé, on retire tous les exemples $(o', \succ, o)$ tels que $o'[\mathcal{X}'] \neq o[\mathcal{X}']$. Puis on ajoute un arc étiqueté u pour chaque $u \in \mathcal{X}'$.

3.1.3 Conclusion

On a présenté dans cette section les arbres lexicographiques et on a étudié deux méthodes pour apprendre de telles représentations. Dans les deux cas, on a vu que l'apprentissage peut se faire par le biais d'un apprentissage passif mais nécessite un nombre d'exemples important. De plus, ce genre de représentation est peu expressive notamment à cause de l'importance des variables ainsi que la présentation arborescente de la structure. Nous allons maintenant nous intéresser à un autre type de représentation de préférences qualitatives : les réseaux de préférences conditionnelles.

3.2 Réseaux de préférences conditionnelles

3.2.1 Représentation et formalisme

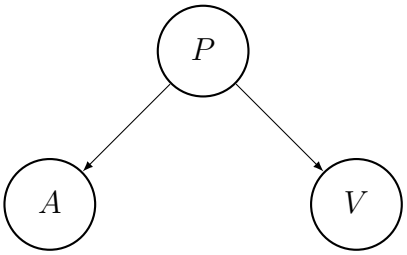
Un CP-net ("Conditional Preference network") est une représentation graphique et qualitative des préférences d'un utilisateur proposée par Boutilier *et al.* [14]. Elle définit un ordre partiel sur des objets de $\underline{\chi}$. De plus, elle est très compacte et intuitive. En effet, un CP-net est un graphe orienté $G = \{\chi, \mathcal{A}\}$ tel que :

- Un arc $\mathcal{A}$ d'un nœud X_i à un nœud X_j signifie que les préférences sur la variable X_j dépendent de la valeur de la variable X_i.
- À chaque nœud X_i est associé une table de préférences conditionnelles, notée $CPT(X_i)$ qui exprime les préférences sur les valeurs possibles de X_i grâce à un ensemble de préférences locales de la forme (X, u :>).

Cette représentation permet de représenter un ordre partiel selon le principe CETERIS PARIBUS, c'est-à-dire toutes choses étant égales par ailleurs.

Définition 3.8 (Préférence CETERIS PARIBUS)**.** *Soient $\mathcal{X}, \mathcal{Y}$et $\mathcal{Z}$ trois partitions de χ et $x_1, x_2 \in \underline{\mathcal{X}}$. On dit que x_1 est préféré à x_2 ceteris paribus (toutes choses étant égales par ailleurs) sachant $y \in \mathcal{Y}$ si et seulement si :*

$$\forall z_1, z_2 \in \underline{\mathcal{Z}}\ \forall y \in \underline{\mathcal{Y}}, x_1 y z_1 \succ x_2 y z_1 \iff x_1 y z_2 \succ x_2 y z_2$$



(a) *Cp-net pour un repas*

P	$p > v$
V	$p : b > r$
V	$v : r > b$
A	$p : z > f$
A	$v : f > z$

(b) *Table de préférence conditionnelle*

Figure 3.4 – *Exemple d'un CP-net pour le choix d'un repas*

Exemple 3.4. *La Figure 3.4 nous montre le CP-net d'un utilisateur représentant la composition d'un plat. Comme pour l'exemple 3.1 page 38, le cp-net est composé de 3 variables booléennes. Le Plat(P), le Vin(V) et l'Accompagnement(A). Le plat a comme valeurs possibles la viande (v) ou le poisson (p). Le vin peut être du rouge (r) ou du blanc (b). Enfin l'accompagnement peut être des frites (f) ou du riz(z). En regardant le CP-net, on voit que le poisson est préféré à la viande. De la même façon, on voit que si on a du poisson, on préfère du riz aux frites. L'ordre partiel représenté par ce cp-net est présenté par la Figure 3.5. On voit grâce à cet ordre que l'objet qui est préféré à tous les autres est pbz et que l'objet pbf est préféré à l'objet prf. Enfin, on voit que pbf et prz sont incomparables.*

Afin de pouvoir travailler sans avoir à développer l'ordre partiel (il y a 2^n objets à classer), plusieurs requêtes sont possibles. La première requête consiste à déterminer si un objet o domine un objet o', s'ils sont incomparables ou si o' domine o. Ce test, appelé test de dominance, est difficile à calculer.

On définit un flip comme étant un changement de valeur pour exactement une variable qui transforme un objet o en un objet o'.

Définition 3.9 (Relation de préférence définie par un CP-net)**.** *Soit un CP-net N. Soient deux objets o et o'. On note $o \succ o'$, et on dit que o domine o' si et seulement si on peut dégrader o en o', c'est-à-dire qu'on peut trouver une séquence de flips dégradants telle que $o \succ o'$.*

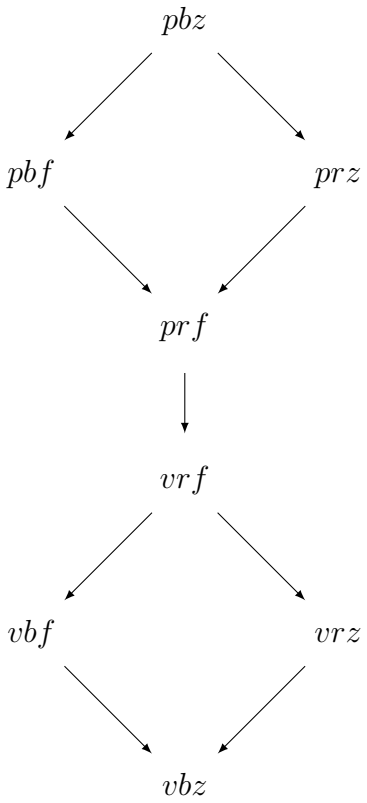


Figure 3.5 – *L'ordre partiel représenté par le cp-net de la figure 3.4*

Exemple 3.5. *Soit le CP-net de la Figure 3.4, et deux objets* pbf *et* vrz*. Il existe une séquence de règles qui permet de dégrader* pbf *en* vrz*. La règle* $(p : b > r)$ *permet de dégrader* pbf *en* prf*, puis la règle* $(p > v)$ *permet de dégrader* prf *en* vrf*, enfin la règle* $(v : f > r)$ *permet de dégrader* vrf *en* vrz*. Il existe donc une chaîne qui va de* pbf *à* vrz*, on peut donc dire que* $pbf \succ vrz$*.*

La complexité du test de dominance pour les CP-nets binaires dépend de leur classe [14]. Le tableau 3.6 donne la complexité du test de dominance pour différentes classes de CP-net. Dans le cas des polyarbres, la complexité dépend du paramètre k qui est le nombre maximum de parents qu'a une variable dans le CP-net.

La seconde requête consiste à demander au CP-net quel est l'objet optimal, un objet qui est préféré à tous les autres, ou encore quel objet n'est dominé

$CP-net\ arborescent$	$O(n)$ [6]
$CP-net\ polyarbres$	$O(2^{2^k}n^{2k+3})$ [14]
$CP-net\ acyclique$	$NP-difficile$ [14]
$CP-net\ cyclique$	$Pspace-complet$ [14]

Figure 3.6 – *Complexité du test de dominance en fonction de la classe de CP-net booléen*

par aucun autre. Cet objet est unique pour la classe des CP-nets acycliques et l'algorithme qui permet de le trouver est en O(n) [14].

Afin de trouver cet objet, il suffit de prendre pour chaque attribut en partant de la racine la valeur de l'attribut qui est optimale selon la table de préférences. Si on reprend l'exemple 3.4, pour la variable P la valeur optimale est p. Sachant que p est optimal, la valeur optimale pour V (resp. A) est b (resp. z). Donc l'objet optimal est *pbz*.

Enfin, il existe une dernière requête pour les CP-nets : le test d'ordonnancement. Elle permet de donner un ordre entre deux objets o et o' qui ne contredit pas le CP-net. Cette requête est extrêmement simple, puisqu'il suffit de trouver une variable où la valeur diffère pour les deux objets et de voir quelle valeur est préférée à l'autre. Il faut cependant faire attention car le test d'ordonnancement n'est pas équivalent au test de dominance : il nous dit que soit $o \succ o'$ soit les deux objets sont incomparables.

Exemple 3.6. *Soit le CP-net de la Figure 3.4, et deux objets* $o = pbz$ *et* $o' = prf$. *La requête d'ordonnancement nous permet de dire que* $prf \not\succ pbz$ *car la première variable qui diffère entre o et o' est A et que le CP-net nous dit que* $p : b > r$ *(l'algorithme aurai pu choisir la variable V ce qui nous donne ici le même résultat).*

Nous allons maintenant nous intéresser à l'apprentissage de CP-net. Nous commencerons par présenter les travaux sur l'apprentissage passif des CP-net séparables proposés par Lang et Mengin [50]. Puis nous verrons deux algorithmes d'apprentissage actif, l'un proposé par Koriche et Zanuttini [49] et l'autre par Guerin *et al.* [42].

3.2.2 Apprentissage de CP-net

3.2.2.1 Apprentissage de CP-net séparable

Lang et Mengin [50] se sont intéressés à l'apprentissage passif de CP-nets séparables. Il s'agit de CP-nets dont les variables ne possèdent aucune dépendance conditionnelle, c'est-à-dire que le graphe ne possède aucun arc.

Proposition . *Soit N un CP-net séparable sur un ensemble de variables binaires et $o \neq o'$. Alors $o \succ_N o'$ si et seulement si $x_i > \bar{x}_i$ pour chaque x_i différent dans o et o' et $\bar{x}_i > x_i$ sinon.*

Exemple 3.7. *$N = \{a > \bar{a}, b > \bar{b}, c > \bar{c}\}$ est un CP-net séparable à trois variables.*

Lang et Mengin [50] se sont intéressés aux trois notions de compatibilités définies 2.5.3 page 31.

La compatibilité implicative

Savoir s'il existe un CP-net séparable implicativement compatible avec un ensemble d'exemples est un problème polynomial.

La compatibilité implicative est la compatibilité la plus restrictive. En effet, un CP-net représente un ordre partiel alors qu'on suppose qu'un utilisateur possède un ordre total. Il est donc possible qu'un utilisateur donne comme exemple $o \succ o'$ alors que pour un CP-net les deux objets sont incomparables.

La compatibilité faible

Il est possible de prouver que décider si un CP-net séparable est faiblement compatible avec un ensemble d'exemples E est un problème NP-complet en effectuant une réduction polynomiale du problème 3SAT vers ce problème [50].

Exemple 3.8. *Soit N le CP-net séparable de l'exemple 3.7 et l'ensemble d'exemples $E = \{abc \succ \bar{a}bc, \bar{a}\bar{b}c \succ ab\bar{c}, \bar{a}b\bar{c} \succ a\bar{b}c, \bar{a}bc \succ \bar{a}\bar{b}\bar{c}\}$. Pour chaque exemple $e \in E$, $N \not\models \neg e$, N est donc faiblement compatible avec E*

La compatibilité forte

Exemple 3.9. *E n'est pas fortement compatible avec N car $E \cup \succ_N$ donne le cycle suivant :*

$$a\bar{b}c \succ \bar{a}\bar{b}c \succ ab\bar{c} \succ \bar{a}b\bar{c} \succ a\bar{b}c$$

Pour la compatibilité forte comme pour la compatibilité faible, décider s'il existe un CP-net séparable qui est fortement compatible avec un ensemble d'exemples est un problème NP-complet.

3.2.2.2 Elicitation de CP-net selon Koriche et Zanuttini [49]

F. Koriche et B. Zanuttini se sont intéressés à l'élicitation de CP-net booléen acyclique et arborescent [49]. Le principe de la méthode est le suivant :

L'apprenant propose à l'utilisateur un CP-net N qu'il pense être optimal pour l'utilisateur tel que $o \succ_N o'$. Si ce n'est pas le cas, l'utilisateur donne à l'apprenant un objet o' qui ne diffère que d'une seule variable de l'objet o tel que o' domine o. Pour plus de compréhension dans la suite de cette partie, on notera X_i l'attribut qui diffère entre o et o', $\bar{x}_i$ la valeur de X_i pour o et x_i la valeur de X_i pour o'. L'objet o' donné par l'utilisateur permet donc d'apprendre la règle $(X_i, u : x_i > \bar{x}_i)$ où u est l'instanciation des parents de X_i déjà connus. A partir de là, il y a deux possibilités.

- La première possibilité est qu'il n'y a aucune contradiction avec le CP-net que l'apprenant a déjà appris. Dans ce cas, il suffit d'ajouter la nouvelle règle au CP-net de l'apprenant.
- La seconde possibilité est que cette nouvelle règle est contradictoire avec le CP-net de l'apprenant, car l'apprenant avait déjà appris $(X_i, u : \bar{x}_i > x_i)$. Dans ce cas, partant de l'hypothèse que l'utilisateur possède un CP-net acyclique cohérent avec les exemples, il existe forcément un parent de X_i que l'apprenant ne connaît pas. Afin de trouver ce parent X_j, on pose à l'utilisateur des requêtes d'appartenance.

L'algorithme 2 commence à partir du CP-net vide $N = \emptyset$, et met à jour le CP-net N, de manière itérative, grâce à des requêtes d'équivalences (EQ) voir 2.5.2 page 30. En voyant un contre-exemple (o, o'), l'apprenant vérifie d'abord si N contient une règle qui couvre (o,o'). Si tel est le cas, alors on dit que (o,o') est un contre-exemple négatif, et le corps de cette règle est raffinés grâce à l'Algorithme 3 en utilisant des requêtes d'appartenance (MQ) voir 2.5.2 page 30. Dans tous les autres cas, (o, o') est un exemple positive, et une nouvelle règle est ajoutée à la table des X_i afin de couvrir cet exemple. Si, en plus, le flip de (o, o') "viole" une certaine règle r dans N, c'est-à-dire que (o', o) est un modèle de r, on cherche un nouveau parent de X_i en utilisant des requêtes d'appartenance.

Algorithm 2: Apprentissage d'un CP-net acyclique de taille polynomiale

Input: $\mathcal{V}$: un ensemble de variables
Output: N : un CP-net acyclique de taille polynomiale

Initialisation des CP-tables ;
$N \leftarrow \emptyset$;
while $EQ(N) \neq vrai$ **do**
    soit (o,o') un contre-exemple retourné par $EQ(N)$;
    Soit X_i une variable telle que $o[X_i] = x_i$ et $o'[X_i] = \bar{x}_i$;
    if *(o,o') est un modèle d'une règle* (r, o_r) *de* N **then**
        "c'est un contre-exemple négatif" ;
        $X_j \leftarrow ExtrairеParent(X_i, x_i, o, o_r, O, N)$;
        $Pa(X_i) \leftarrow Pa(X_i) \cup X_j$;
        mettre à jours chaque règle (r', o'_r) pour prendre en compte x_j
    else
        "c'est un contre-exemple positif" ;
        ajouter la règle $(X_i, t : x_i > x_j)$ où $t = o[\underline{Pa(X)}]$;
        if *(o',o) est un modèle d'une règle* (r, o_r) *de* N **then**
            $X_j \leftarrow ExtrairеParent(X_i, x_i, o', o_r, O, N)$;
            $Pa(X_i) \leftarrow Pa(X_i) \cup X_j$;
            mettre à jours chaque règle (r', o'_r) pour prendre en compte x_j
return N

L'algorithme 3 ExtraireParent, consiste à trouver un nouveau parent pour une règle r qui à été "violée", en utilisant seulement un nombre logarithmique de requêtes d'appartenance. En utilisant l'objet o_r qui a permis de trouver r et le dernier contre-exemple (o, o') donné par l'utilisateur, on opère une recherche binaire sur la séquence $(o_1, \ldots, o_n)$ où o_j est formé par les premiers littéraux j apparaissant dans o_r et les derniers $n-j$ littéraux apparaissant dans o. L'invariant maintenu par l'algorithme est que $o_a[X_i] \not\succ \overline{o_a[X_i]}$ mais $o_b[X_i] \succ \overline{o_b[X_i]}$ dans le concept cible. Ainsi, lorsque la recherche a été réduite à $a = b-1$, si on change la valeur de X_b dans o_a on change la préférence, ce qui prouve que X_b est un parent de la variable X_i

Algorithm 3: Extraire Parent

Input: X_i : une variable,
x_i une instanciation de X_i,
o,o' deux objets,
a,b deux entier
Output: pa : un parent de X tel que $pa \notin Pa(X)$

```
o_c ← o ;
if a = b − 1 then
    return x_b ;
j = (a+b)/2 ;
o_j = o[t_j] où t_j est l'instanciation des j premier littéraux présent dans
l'objet o' ;
if MQ(o_j[x_i], o_j[x̄_i]) = oui then
    return ExtraireParent(X_i, x_i, o, o', a, j) ;
else
    return ExtraireParent(X_i, x_i, o, o', j, a) ;
```

Afin d'apprendre le CP-net acyclique booléen exact de l'utilisateur, il faut poser $O(|N|)$ requêtes d'équivalence et $O(log(n))$ requêtes d'appartenance, où a est le nombre d'arcs dans la représentation minimal de N.

3.2.2.3 Élicitation de CP-net selon Guerin *et al.* [42]

J.T Guerin et al. proposent un nouvel algorithme d'élicitation de CP-net [42] qui contrairement à celui de Koriche et Zanuttini [49] ne se base pas sur des flips

d'objets mais sur des objets quelconques. Cette algorithme fonctionne en deux phases.

- La première phase consiste à apprendre un CP-net séparable avec des CP-tables par défaut.
- La seconde phase cherche à rendre le CP-net consistant grâce aux réponses à des requêtes posées à l'utilisateur.

Pour la seconde phase, on affine le CP-net en cherchant des dépendances conditionnelles entre des variables en demandant la préférence de l'utilisateur sur une paire d'objets.

Apprendre un CP-net avec cette méthode se fait en $O(n^k)$ où k est le nombre maximal de parents pour une variable dans le CP-net.

L'algorithme 4 commence en construisant le CP-net séparable de l'utilisateur en lui demandant simplement pour chaque variable quelle est son instanciation préférée (ligne 5 à 8). Ensuite une fois ce CP-net séparable construit (ligne 10 à 16), l'algorithme cherche pour chaque variable ses parents. Si elle en a, on modifie la CP-table de cette variable dans le CP-net en prenant en compte les nouveaux parents. L'algorithme 5 permet de trouver l'ensemble des parents pour une variable donnée. Pour cela, il teste toutes les combinaisons de parents possibles selon le nombre supposé de parents. Afin de connaître le bon ensemble de parents, il pose à l'utilisateur un ensemble de requêtes d'appartenance jusqu'à ce que le taux de confiance dépasse un certain seuil de confiance.

3.2.3 Conclusion

Dans cette section, on a présenté une autre représentation de préférences qualitatives : les CP-nets. Cette représentation permet de représenter des préférences conditionnelles dans un cadre CETERIS PARIBUS. Malgré une requête de dominance ayant une complexité importante, les CP-net restent une représentation intéressante notamment grâce à la requête d'optimalité et au test d'ordre qui permet de classer des objets compatibles avec les préférences d'un utilisateur. En ce qui concerne l'apprentissage de cette représentation, on a vu qu'apprendre des CP-nets de façon passive était difficile même dans leur représentation la plus simple (les CP-nets séparables)[50]. Par contre en ce qui concerne l'apprentissage actif, on a vu que les CP-nets acycliques étaient apprenable en temps polynomial avec un nombre de requêtes raisonnable. Plus loin dans cette thèse, nous expliquerons pourquoi on a retenu les CP-nets afin de proposer une extension pour notre représentation multi-utilisateur.

Algorithm 4: apprentissage d'un CP-net

Input: $\mathcal{X}$: un ensemble de variables binaires,
p : le nombre maximum de parents,
q seuil de confiance minimal
Output: N : un CP-net

```
1  N ← ∅ ;
2  comparaison ← ∅ ;
3  certain ← ∅ ;
4  incertain ← V ;
5  for X_i ∈ X do
6      Question à l'utilisateur : x_i > x̄_i ou x̄_i > x_i ;
7      cpt(X_i) ← { réponse de l'utilisateur } ;
8      ajouter X_i dans N ;
9  repeat
10     for r ← 0 à p do
11         for X_i ∈ incertain do
12             (P, C) ← TrouveParent(X_i, r, q) ;
13             if C ≠ Fail then
14                 cpt(X_i) = C ;
15                 ajouter des arc pour chaque p de P vers X_i ;
16                 déplacer X_i d'incertain vers certain ;
17 until aucun parent à ajouter;
18 return N
```

Algorithm 5: Trouve Parent

Input: X_i : la variable que l'on traite,
r : le nombre de parents supposé,
q seuil de confiance minimal
Output: C : une CP-table pour X_i ou Fail, P : nouveaux parents pour X_i ou $\emptyset$

```
for P ∈ { Tous les sous-ensembles de parents possible d'une certaine taille r } do
    (C, tauxConfiance) ← CreeCPTable(v_i, P) ;
    while C ≠ FAIL ou tauxConfiance < q do
        (o_1, o_2) ← générer aléatoirement deux objets avec o[X_i] = o'[X_i] ;
        Poser la question : "Préfères tu o_1 à o_2 ?" ;
        Ajouter o_1 et o_2 dans comparaison selon la réponse de l'utilisateur. ;
        (C, tauxConfiance) ← CreeCPTable(v_i, P) ;
    if C ≠ FAIL then
        return (P,C)
return (∅,FAIL)
```

3.3 Extensions des CP-nets

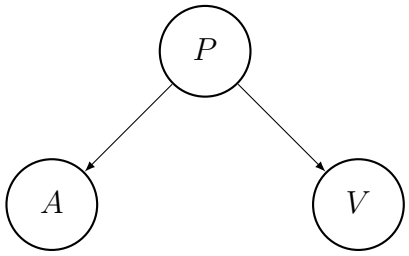
3.3.1 UCP-net

Les UCP-nets ("Utility CP-net") introduits par Boutilier *et al.* [12] remplacent la relation de préférence binaire des nœuds d'un CP-net par une fonction d'utilité sur les valeurs de ces nœuds. Ainsi, les dépendances entre les variables restent conditionnelles alors que les tables de préférences des variables deviennent quantitatives grâce aux fonctions d'utilité. Cette extension est motivée par le fait que les CP-nets donnent des ordres partiels et qu'il est donc difficile de comparer une paire d'objets. Ce problème est donc résolu par cette quantification.

Comme pour les GAI (chapitre 4 page 69), les fonctions d'utilité sont des réels associés aux valeurs des variables d'un nœud en fonction de l'instanciation de ses parents. Ils représentent le degré de préférence d'un attribut par un utilisateur. Plus la fonction d'utilité est importante plus la valeur concernée est préférée.

Afin de ne pas devenir contradictoires avec les CP-nets, les UCP-nets sont soumis à plusieurs contraintes [12], par exemple la somme des utilités d'une chaîne montante partant d'un nœud feuille à un nœud n (n différent de la racine) doit

être strictement inférieure à l'utilité des parents de n.



P	p	100
P	v	40
V	$p : b$	40
V	$p : r$	30
V	$v : b$	10
V	$v : r$	20
A	$p : z$	50
A	$p : f$	40
A	$v : z$	0
A	$v : f$	20

Figure 3.7 – *Exemple d'un UCP-net pour le choix d'un repas*

Exemple 3.10. *L'UCP-net présenté dans la Figure 3.7 reprend l'exemple du repas présenté dans la section CP-net. On peut à partir de cet UCP-net donner l'ordre total suivant :*

$$pbz(190) \succ prz(180) \sim pbf(180) \succ prf(170) \succ vrf(80) \succ vbf(70) \succ vrz(60) \succ vbz(50)$$

Si l'on reprend l'ordre de la Figure 3.5 donné par le CP-net de la Figure 3.4, on avait pbf est incomparable avec prz et vbf est incomparable avec vrz. Alors qu'ici grâce à la quantification des préférences, on a pbf qui est équivalent à prz et vbf qui est préféré à vrz.

On est donc capable comme pour les GAI, de comparer deux objets rapidement et facilement.

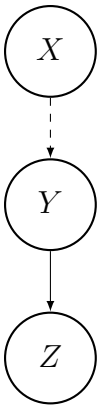
Le problème des UCP-nets est qu'on se sait pas comment appendre une tel structure. Cela est principalement du aux contraintes ajoutées aux UCP-net afin qu'ils ne deviennent pas contradictoires.

3.3.2 TCP-net

Les TCP-nets ("Tradeoffs-enhanced CP-net") proposés par Brafman et Domshlak [16] ajoute une notion d'importance entre les variables en rajoutant un type d'arc dans les graphes. Ces nouveaux arcs ne représentent pas des relations de préférences conditionnelles mais des relations d'importances conditionnelles permettant de mettre en avant une variable plutôt qu'une autre. Cette nouvelle

relation permet d'exprimer une préférence du type "Ma préférence sur les valeurs de X est plus importante que ma préférence sur les valeurs de Y".

Les TCP-nets permettent aussi de mélanger les relations de préférences conditionnelles et les relations d'importance conditionnelle pour avoir des préférences du type "Si l'attribut Z prend la valeur z, ma préférence sur les valeurs de X est plus importante que ma préférence sur les valeurs de Y."



X	$x > \bar{x}$
Y	$y > \bar{y}$
Z	$y : z > \bar{z}$
Z	$\bar{y} : \bar{z} > z$

Figure 3.8 – *Exemple d'un TCP-net sur 3 variables*

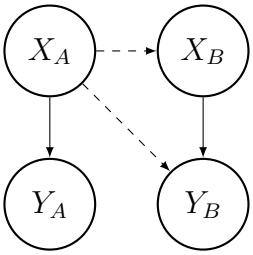
Exemple 3.11. *La Figure 3.8 nous montre un TCP-net à trois variables où il y a une relation d'importance conditionnelle entre les variables X et Y mais aucune relation de préférence entre ces deux variables.*

Le principal problème des TCP-nets est la difficulté à effectuer des requêtes avec cet représentation. De plus, il n'existe aucun apprentissage connu permettant d'apprendre de tel structure.

3.3.3 MCP-net

Les MCP-net ("multi-agents CP-net") proposés par Rossi *et al.* [61] sont une extension des CP-nets au cadre multi-agents. Ils ont été proposés principalement dans un cadre de vote entre agents où les préférences locales d'une variable dépendent, en plus des dépendances conditionnelles d'un agent, des choix des agents qui l'entourent. Il est ainsi possible d'exprimer des relations de préférences du type : "Si l'agent A choisit la valeur x pour la variable X alors je préfère la valeur $\bar{x}$ à x pour la variable X."

Un exemple d'utilisation de ce type de représentation est le choix d'un plat au restaurant pour un couple d'amoureux voulant partager leurs plats.



X_A	$x_A > \bar{x}_A$
X_B	$\bar{x}_B > x_B$
Y_A	$x_A : y_A > \bar{y}_A$
Y_A	$\bar{x}_A : \bar{y}_A > y_A$
Y_B	$x_A x_B : y_B > \bar{y}_B$
Y_B	$x_A \bar{x}_B : \bar{y}_B > y_B$
Y_B	$\bar{x}_A x_B : y_B > \bar{y}_B$
Y_B	$\bar{x}_A \bar{X}_B : y_B > \bar{y}_B$

Figure 3.9 – *Exemple d'un mCP-net sur 2 variables et 2 Agents*

Exemple 3.12. *La Figure 3.9 nous montre un MCP-net à deux variables Y et X et deux agents A et B. La préférence sur Y et X de l'agent B dépend du choix de A sur la variable X.*

Les mCP-nets souffrent des même problèmes que les UCP-nets et les TCP-nets, empêchant une utilisation efficace de ce formalisme.

3.4 Réseaux bayésiens

3.4.1 Représentation et formalisme

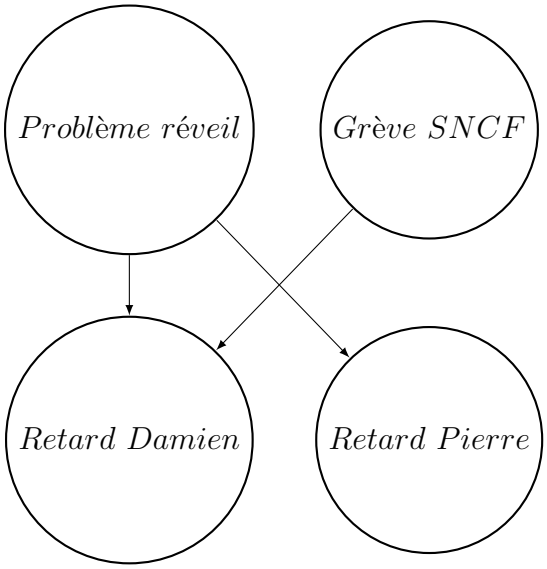
Un réseau bayésien est une représentation graphique représentant les connaissances d'un environnement et permettant de calculer la probabilité d'existence ou d'apparition d'un événement à partir de ces connaissances [24]. Dans ce graphe, les nœuds représentent des variables aléatoires et possèdent une table de probabilité. Les arcs eux représentent des dépendances conditionnelles entres les variables. Les réseaux bayésiens sont par définition acycliques. Les réseaux bayésiens sont principalement utilisés dans le diagnostic [44], l'analyse de risque [70], la modélisation de systèmes complexes [66] ou encore les préférences [58].

Définition 3.10. *Un réseau bayésien est un graphe orienté acyclique $G = (\chi, \mathcal{A})$ où χ est un ensemble de variables et $\mathcal{A}$ un ensemble d'arcs correspondant aux dépendances conditionnelles des variables. Á chaque nœud $X \in \chi$ est associé la table de probabilités conditionnelles suivante :*
$P(X|Pa(X))$, où $Pa(X)$ représente les parents de la variable X.

Un réseau bayésien permet de représenter de manière compacte une probabilité jointe sur un ensemble de variables [59] :

$$P(X_1, X_2, \ldots, X_n) = \prod_{i=1}^{n} P(X_i | Pa(X_i))$$

La décomposition de la fonction globale en un produit de termes locaux dépendant uniquement de la variable considérée et de ses parents est une propriété fondamentale des réseaux bayésiens. Comme on le verra dans la sous-section Apprentissage 3.4.2, c'est cette propriété qui a permis le développement des premiers algorithmes d'apprentissage.



Problème réveil	*¬Problème réveil*
0.6	0.4
Grève SNCF	*¬Grève SNCF*
0.9	0.1

	Problème réveil	*¬Problème réveil*
RetardPierre	0.95	0.4
¬RetardPierre	0.05	0.6

	Réveil Grève	*Réveil ¬Grève*	*¬Réveil Grève*	*¬Réveil ¬Grève*
RetardDamien	1	0.3	0.95	0.1
¬RetardDamien	0	0.7	0.05	0.9

Figure 3.10 – *Situation d'un semaine de travail au mois de juin 2014*

Exemple 3.13. *La Figure 3.10 représente le réseau bayésien d'une semaine de travail pour Pierre et Damien durant une grève de la SNCF. On y voit que le retard de Pierre dépend de Problème réveil et que le retard de Damien dépend de*

Problème réveil et de Grève SNCF. Ainsi la probabilité que Pierre soit en retard est égale à :

$$P(RetardPierre) = P(Retard\ Pierre|Problème\ réveil) * P(Problème\ réveil)$$

$$P(RetardPierre) = 0.95 * 0.6 + 0.4 * 0.4 = 0.73$$

Dans l'exemple, Pierre a donc 73% *de chances d'arriver en retard.*

3.4.2 Apprentissage

L'apprentissage d'un réseau bayésien à partir d'un jeu de données contenant un ensemble d'événements se fait en deux étapes :

- La première étape consiste à apprendre la structure du réseaux, c'est-à-dire déterminer les dépendances conditionnelles entre les variables.
- Une fois la structure connue, la seconde étape consiste à apprendre les bonnes distributions de probabilités conditionnelles des variables.

3.4.2.1 Apprentissage de structure

Apprendre la structure d'un réseau bayésien peut s'avérer complexe. Une première difficulté récurrente est un jeu de données incomplet, c'est-à-dire que les informations données par le jeu de données ne permettent pas d'apprendre correctement toutes les dépendances conditionnelles du réseau. Pour régler le problème deux méthodes existent.

- La première consiste à générer des données avant l'apprentissage grâce à une inférence statistique [15].
- La seconde est d'utiliser des algorithmes gérant ces données manquantes [33].

Une autre difficulté dans l'apprentissage des réseaux bayésiens est la gestion des variables dites latentes. Une variable latente est une variable dont dépendent d'autres variables mais qui n'est pas observée. C'est un problème bien connu entre autre dans le domaine de la génétique [4].

Enfin, la dernière difficulté vient des réseaux eux mêmes. En effet, le nombre de DAG possibles pour un ensemble de variables est exponentiel. Chickering et al. [20] ont montré qu'apprendre la structure exacte d'un réseau bayésien était un problème NP-complet.

Malgré toutes ces difficultés, l'apprentissage de structure de réseaux bayésiens est le sujet depuis 20 ans d'une multitude de méthodes pour approcher la solution optimale [34, 43, 51].

Recherche d'indépendance

La méthode la plus naturelle consiste à rechercher les indépendances entre les variables [48], ce qui permet ainsi de déduire les dépendances conditionnelles entre elles. Pour cela, il existe plusieurs façons de déterminer si deux variables X et Y sont indépendantes. Le test de dépendance chi 2 (χ^2) est l'une de ces méthodes. Ce test consiste à faire une hypothèse H_0 comme quoi ces deux variables sont indépendantes. Ensuite, le test estime si l'hypothèse est juste ou non avec un risque d'échec d'α. Pour évaluer cet hypothèse, on utilise la formule suivante :

$$\chi^2 = \sum_{i=1}^{\underline{X}} \sum_{j=1}^{\underline{Y}} \frac{(O[ij] - E[ij])^2}{E[ij]}$$

où $O[ij]$ représente le nombre d'observations où X a pour valeur i et Y l'instanciation j. E[ij] représente l'estimation d'observation de i et j dans l'hypothèse où H_0 est vraie. Afin d'approuver ou de rejeter H_0, on compare le résultat à la loi du χ^2 de degré de liberté k. Dans le cas de deux variables, k est défini par $k = (|\underline{X}| - 1)(|\underline{Y}| - 1)$. Il suffit d'utiliser la procédure suivante pour prendre la décision :

$$\begin{cases} \chi^2 < q_{\chi^2_k}^{1-\alpha} \rightarrow H_0 \; est \; acceptée \\ \chi^2 \geqslant q_{\chi^2_k}^{1-\alpha} \rightarrow H_0 \; est \; rejetée \end{cases}$$

où $q_{\chi^2_k}^{1-\alpha}$ représente le quantile à $(1-\alpha)$ de la loi de χ^2 à k degré de liberté. La valeur α est généralement mise à 0.05. Augmenter α a pour effet de rejeter plus souvent H_0 alors que le diminuer augmentera l'acceptation de H_0.

Score Maximum

Cette approche consiste à mesurer la vraisemblance d'un modèle sachant les observations par un score. L'un des algorithmes les plus connus a été proposé par Freidman [33] ; c'est l'un des premiers algorithmes déterministes efficaces pour apprendre la structure d'un réseau bayésien avec un ensemble de données incomplètes.

L'idée principale de l'algorithme 6 est de maximiser le score attendu des modèles. Pour cela, à chaque itération, il effectue deux opérations :

Algorithm 6: algorithme EM pour l'apprentissage de structure.

Input:
Output: G : une structure

k=0
choisir un graphe G^k et les paramètres $\Theta^{k,0}$ généré aléatoirement ou avec une heuristique ;
while *l'on a pas convergence et* $k \leqslant k_{max}$ **do**
    l=0 **while** *l'on a pas convergence et* $l \leqslant l_{max}$ **do**
        l=l+1 ;
        $\Theta^{k,l} = argmax_{\Theta} Q(G^n, \Theta : G^k, \Theta^{k,l-1})$;
    k=k+1 ;
    $G^k = argmax_G Q(G, . : G^{k-1}, \Theta^{k,l})$;
    $\Theta^{k,0} = argmax_{\Theta} Q(G^k, \Theta : G^{k-1}, \Theta^{k-1,l})$;
return *(*G^k*,*$\Theta^{k,0}$*)*

- la phase "Espérance", consiste à estimer les données inconnues sachant les données observées et la valeur des variables déjà connues grâce à l'itération précédente.
- La phase "Maximisation" , qui consiste à maximiser la vraisemblance, rendu possible en utilisant l'estimation des données inconnues obtenue pendant l'étape "Espérance" et met à jours les valeurs des variables pour l'itération suivante.

Méthodes hybrides

Enfin, il existe aussi des algorithmes mixant à la fois la recherche d'indépendance et le score maximum. Le principe de cette méthode est principalement de réduire l'espace de recherche grâce à la recherche d'indépendance, afin de faciliter la recherche du meilleur graphe grâce à la méthode des score.

3.4.2.2 Apprentissage de la distribution de probabilités

Cette phase d'apprentissage est plus ou moins compliquée en fonction du type de réseaux bayésiens avec lequel on travaille (réseaux bayésien à variables discretes/continues) ou encore le type d'observation que l'on possède (observation complète / partielle).

Dans le cas des réseaux bayésiens discrets, chaque distribution de probabilités conditionnelles $P(X_i|Pa(X_i)$ peut se définir par un ensemble de paramètres θ_i. Ces paramètres sont représentables par une matrice où chaque ligne k représente la $k^{ème}$ valeur de X_i et chaque colonne j à la $j^{ème}$ instanciation des valeurs de $Pa(X_i)$. La case correspondante prend alors la valeur :

$$\theta_{ijk} = P(X_i = k|pa(X_i) = j)$$

Ces paramètres peuvent être estimés de plusieurs façons, la plus connue étant le maximum de vraisemblance, qui ne suppose aucun a priori sur la valeur des paramètres comme le montrent Grossman et Domingos [41]. D'autre méthodes existent aussi dont certaines avec des à priori comme la loi de Dirichlet présentée par Geiger et Heckerman [36].

Maximum de vraisemblance

Avec cette méthode, on estime les paramètres de θ_i d'une variable X_i sachant que ses parents $Pa(X_i)$ comptent dans le jeu de données.

$$\theta_{ijk} = \frac{[X_i = k|Pa(X_i) = j]}{[Pa(X_i) = j]}$$

où $[X_i = k|Pa(X_i) = j]$ est le nombre de fois où X_i a la valeur k lorsque les parents de X_i ont l'instanciation j.

A priori de Dirichlet

Le principe "A priori de Dirichlet" est utilisé dans l'apprentissage de paramètres dans les cas où les données fournies empêchent d'estimer la probabilité d'une valeur d'une variable (par exemple la valeur x_3 pour la variable X n'apparaît pas dans le jeu de données). Dans ce cas, faire une supposition de Dirichlet sur les paramètres θ_{ijk} revient à fixer un ensemble de coefficients α_{ijk}. Ces nouveaux paramètres peuvent être interprétés comme des observations souhaitées de chaque configuration possible. Cela permet de lisser les variations entre les paramètres θ_{ijk} et ainsi ne pas apprendre une probabilité de 0 pour certaines variables. Cette méthode est principalement utilisée dans le cas où certaines instanciations de variables sont particulièrement rares et que le jeu de données est relativement petit, donnant le risque qu'elles n'apparaissent pas. Dans ce cas, au lieu d'avoir $\theta_{ijk} = 0$, ce paramètre prend une valeur à priori α_{ijk}.

L'estimation bayésienne permet de définir θ_{ijk} sachant la distribution de Dirichlet. On parle alors d'estimation par maximum à posteriori (MAP)

$$\theta_{ijk} = \frac{[X_i = k|Pa(X_i) = j] + \alpha_{ijk} - 1}{[Pa(X_i) = j] + \alpha_{ij} - 1} \; avec \; \alpha_{ij} = \sum_{k=1}^{r_i} \alpha_{ijk}.$$

3.5 Conclusion

Dans ce chapitre, nous avons parlé des principaux formalismes de préférences utilisé dans les domaines combinatoires ainsi que les différentes méthodes pour apprendre de telles représentations. Nous avons principalement parlé de représentations qualitatives comme les arbres lexicographiques ou les CP-nets. On a vu que lorsque la représentation était qualitative, il existait des algorithmes permettant de les apprendre. Nous verrons dans le prochain chapitre que le GAI est un formalisme quantitatif et qu'il n'existait pas avant nos travaux de méthodes permettant d'apprendre complètement un GAI (fonction d'utilité + structure). Nous avons donc décidé de consacrer la deuxième partie de cette thèse à l'apprentissage de GAI. De plus, les représentations étudiées dans ce chapitre ne permettent pas de prendre en compte un cadre multi-agents. Pour pallier ce problème, nous avons décidé de consacrer la troisième partie de cette thèse à un formalisme multi-utilisateurs ainsi que des méthodes pour apprendre une telle représentation.

Deuxième partie

Décomposition Additive Indépendante Généralisée et Réseaux GAI

Introduction

Le modèle des GAI-décompositions, et leurs représentations par des réseaux GAI [3, 39], sont des représentations quantitatives qui permettent d'offrir une alternative aux représentations qualitatives telles que les CP nets ou les arbres lexicographiques. Il permet de représenter la relation de préférence de l'utilisateur (supposée complète et transitive) par une fonction d'utilité, dont on exploite les éventuelles propriétés de décomposabilité. Ces travaux utilisent la notion d'indépendance additive généralisée (d'où le terme GAI, pour *Generalized Additive Independence*), introduite par Fishburn [28], pour décomposer la fonction d'utilité sur l'ensemble des caractéristiques des objets en une somme de petites fonctions d'utilité ne portant que sur certains groupes d'attributs. Ces GAI-décompositions permettent donc d'exprimer des interactions générales entre les attributs, tout en préservant une certaine décomposabilité du modèle. Une GAI décomposition peut être représentée par une structure graphique appelée réseau GAI [39].

Cette partie se compose de deux chapitres. Nous verrons dans le premier chapitre comment une GAI-décomposition ainsi qu'un réseau GAI peuvent représenter une relation de préférence. À partir de là, nous présenterons les principales requêtes utilisées dans les GAI-décompositions et les réseaux GAI.

Puis, nous nous intéresserons aux travaux déjà existants :

- Nous parlerons de la méthode proposée par Gonzales et Perny [39] afin de résoudre la requête d'optimalité.
- Nous nous intéresserons à l'élicitation de l'utilité d'un GAI. Pour cela, nous étudierons deux méthodes. La première proposée egalement par Gonzales et Perny [39], permet l'élicitation de la fonction d'utilité d'un GAI lorsque la structure est connue. La seconde, proposée par Braziunas [17], permet l'élicitation d'une fonction d'utilité pour un ensemble de variables locales (c'est-à dire un groupe de variables qui influe sur une clique).

Enfin dans le deuxième chapitre, nous nous intéresserons à notre contribution, l'apprentissage d'une GAI-décomposition [7]. Nous verrons comment on peut, en utilisant des systèmes d'équations linéaires, représenter une comparaison entre deux objets. Et comment à partir de ce système on peut apprendre à la fois la structure d'une GAI-décomposition et les fonctions d'utilités locales.

Chapitre 4

Présentation des GAI-Décompositions

Sommaire

4.1 Représentation et formalisme

Toute relation de préférence complète et transitive peut évidemment être représentée par une fonction d'utilité $u : \underline{\chi} \mapsto \mathbb{R}$ vérifiant $o \succeq o' \Leftrightarrow u(o) \geq u(o')$ pour tous $o, o' \in \underline{\chi}$. Cependant, l'ensemble χ étant combinatoire (il contient 2^n objets), il est impossible en pratique d'éliciter ou de mémoriser explicitement la relation $\succeq$, ou même la fonction u. Toutefois, dans certains cas, les préférences de l'utilisateur présentent une propriété d'indépendance forte entre attributs [25] ; il est alors possible de les représenter par un ensemble de fonctions d'utilité locales $u_i : \underline{X}_i \mapsto \mathbb{R}$, d'arité 1 (donc peu coûteuses en mémoire et faciles

à éliciter), et permettant de déduire directement l'utilité globale (et donc $\succeq$) : $u(o) = \Sigma_i u_i(o[X_i])$. La propriété correspondant à ces relations de préférence est appelée l'*indépendance additive*. Malheureusement, les relations de préférence la satisfont rarement.

Exemple 4.1. *Soit* $\chi = \{X_1, X_2, X_3\}$ *avec* $\underline{X}_1 = \{viande(v), poisson(p)\}$, $\underline{X}_2 = \{rouge(r), blanc(b)\}$ *et* $\underline{X}_3 = \{citron(c), moutarde(m)\}$. $\underline{\chi}$ *contient* 2^3=*8 objets possibles :*

$$\begin{array}{ll} o_1 = (v, r, c) & o_5 = (p, r, c) \\ o_2 = (v, r, m) & o_6 = (p, r, m) \\ o_3 = (v, b, c) & o_7 = (p, b, c) \\ o_4 = (v, b, m) & o_8 = (p, b, m) \end{array}$$

Considérons maintenant l'ordre de préférence suivant :

$$o_2 \succ o_1 \succ o_6 \succ o_5 \sim o_4 \succ o_3 \succ o_8 \succ o_7$$

Cet ordre peut être représentée par une fonction d'utilité additive $u(o) = \Sigma_i u_i(o[X_i])$, *en utilisant par exemple*

$$u_{\{X_1\}}(v) = 5, u_{\{X_1\}}(p) = 2$$

$$u_{\{X_2\}}(r) = 5, u_{\{X_2\}}(b) = 1$$

$$u_{\{X_3\}}(c) = 2, u_{\{X_3\}}(m) = 3$$

.

Exemple 4.2. *L'ordre de préférence suivant, sur les objets définis en exemple 4.1, ne peut pas être représenté par une somme de fonctions additives d'arité 1.*

$$o_2 \succ o_4 \sim o_7 \succ o_1 \succ o_8 \sim o_5 \succ o_3 \succ o_6$$

On voit ici que le menu $o_7 = (p, b, c)$ *est préféré au menu* $o_3 = (v, b, c)$, *et que le menu* $o_2 = (v, r, m)$ *est préféré au menu* $o_6 = (p, r, m)$ *; si les préférences étaient simplement additives, on en déduirait de la première comparaison que* $u_{\{X_1\}}(v) < u_{\{X_1\}}(p)$, *et de la seconde que* $u_{\{X_1\}}(v) > u_{\{X_1\}}(p)$, *ce qui constitue une contradiction.*

Pour représenter ce second exemple, on peut définir des fonctions d'utilité locales portant sur plusieurs *variables ; la première* $u_{\{X_1,X_2\}}$} *porte sur le plat principal* et *le vin, la seconde* u_{X_1,X_3}} *porte sur le plat principal* et *l'accompagnement — la fonction d'utilité globale étant définie par*

$u(o) = u_{\{X_1,X_2\}}(o[\{X_1, X_2\}]) + u_{\{X_1,X_3\}}(o[\{X_1, X_3\}])$:

$$u_{\{X_1,X_2\}}(v, r) = 5, \qquad u_{\{X_1,X_2\}}(v, b) = 2$$
$$u_{\{X_1,X_2\}}(p, r) = 1, \qquad u_{\{X_1,X_2\}}(p, b) = 4$$
$$u_{\{X_1,X_3\}}(v, c) = 3, \qquad u_{\{X_1,X_3\}}(v, m) = 7$$
$$u_{\{X_1,X_3\}}(p, c) = 5, \qquad u_{\{X_1,X_3\}}(p, m) = 2$$

On voit dans l'exemple 4.2 que pour certaines relations de préférence, des variables peuvent être dépendantes d'autres, et que dans ce cas la fonction d'utilité est décomposée, non pas en utilités élémentaires, mais selon des fonctions d'utilité liant les variables dépendantes. Une telle décomposition de la fonction u est appelée une *GAI-décomposition* de u :

Définition 4.1 (GAI-décomposition)**.** *Soit $\chi = \{X_1, ..., X_n\}$ un ensemble de variables, $\underline{\chi} = \underline{X}_1 \times \cdots \times \underline{X}_n$ un ensemble d'objets combinatoires et $u : \chi \mapsto \mathbb{R}$ une fonction d'utilité sur χ.*

Une GAI-décomposition *de u est un ensemble fini $G = \{u_{Z_1}, \ldots, u_{Z_m}\}$ de fonctions d'utilité portant sur des sous ensembles Z_i de χ (c'est-à-dire $u_{Z_i} : \underline{Z}_i \mapsto \mathbb{R}$) et telles que l'on ait $u(o) = \Sigma_{i=1,m}\ u_{Z_i}(o[Z_i])$ pour tout o de χ.*

Les fonctions d'utilité locales sont également appelées *GAI-tables*, car on les implémente typiquement par des tables.

Il est clair que, via une fonction d'utilité, une GAI-décomposition G représente exactement une relation de préférence sur χ :

Définition 4.2 (relation représentée)**.** *Soit $G = \{u_{Z_1}, \ldots, u_{Z_m}\}$ une GAI-décomposition d'une fonction d'utilité $u_G : \chi \mapsto \mathbb{R}$ (c'est-à-dire $u_G = \Sigma_{i=1,m} u_{Z_i}$). On dit que G* représente *la relation de préférence $\succeq$ sur χ (ou simplement que c'est une GAI-décomposition) si pour tous objets $o, o' \in \chi$ on a $(o \succeq o' \iff u_G(o) \geq u_G(o'))$.*

Dans la suite, on notera $\succeq_G$ la relation générée par une GAI-décomposition G donnée (c'est-à-dire $o \succeq_G o' \iff u_G(o) \geq u_G(o')$).

4.1.1 Réseau GAI

Une GAI-décomposition peut être représentée par une structure graphique appelée réseau GAI [39]. Cette représentation est similaire aux graphes de jonction des réseaux bayésiens [45].

Définition 4.3 (réseau GAI). *Soit* $\underline{\chi} = \prod_{i=1}^{n} X_i$. *Soit* $C_1; \ldots; C_k$ *des sous-ensembles de* $N = \{1; \ldots; n\}$ *tels que* $N = \bigcup_{i=1}^{k} C_i$. *Supposons que* $\succeq$ *soit représentable par une utilité GAI* $u(o) = \sum_{i=1}^{k} u_i(o[C_i]), \forall o \in \underline{\chi}$. *Alors un réseau GAI qui représente u(.) est un graphe non-orienté* $G = (C, \varepsilon)$ *qui satisfait les propriétés suivantes :*

- *Propriété 1 :* $C = \{X_{C_1}, \ldots, X_{C_k}\}$ *tel que* X_{C_i} *est un nœud étiqueté par le sous-ensemble* C_i.
- *Propriété 2 :* $(X_{C_i}, X_{C_j}) \in \varepsilon \Rightarrow C_i \cap C_j \neq \emptyset$.
- $\forall X_{C_i}, X_{C_j}$ *tels que* $C_i \cap C_j = T_{ij} \neq \emptyset$, *il existe un chemin dans G qui connecte* X_{C_i} *et* X_{C_j} *tel que tous ses nœuds contiennent tous les indices de* T_{ij} *(propriété d'intersection courante).*

Les nœuds de G sont appelés cliques. Chaque arête $(X_{C_i}, X_{C_j}) \in \varepsilon$ *est étiquetée par* $X_{T_{ij}} = X_{C_i \cap C_j}$ *et est appelée séparateur.*

Dans un réseau GAI, les ellipses représentent traditionnellement les cliques de la GAI décomposition alors que les rectangles représentent les séparateurs (c'est-à-dire les variables présentent dans plusieurs cliques). On représente les réseaux GAI par un graphe non orienté car les cliques d'un réseau représentent un ensemble de variables dépendantes par une fonction d'utilité et les arêtes reliant les cliques représentent un ensemble de variables présents dans plusieurs ensembles. Et comme l'intersection est une relation binaire commutative, il n'est pas nécessaire d'orienter le graphe.

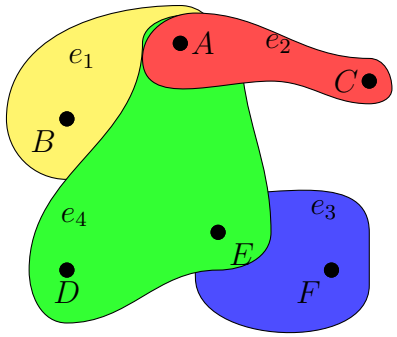
Exemple 4.3. *Soient un ensemble de variables* $\chi = \{A, B, C, D, E, F\}$, *et la fonction d'utilité représentée par la décomposition de la Figure 4.1* $u(abcde) = u_1(ab)+u_2(ac)+u_3(ade)+u_4(ef)$. *Les cliques sont donc :* $\{A, B\}$, $\{A, C\}$, $\{A, D, E\}$, $\{E, F\}$. *Le réseau GAI de la Figure 4.2 représente la GAI-décomposition de la Figure 4.1.*

4.2 Requêtes et complexités

4.2.1 Le test de dominance.

Soit une GAI-décomposition G et deux objets o et o'. On dit que $G \models o \succ o'$ (respectivement $G \models o \sim o'$) si et seulement si $u_g(o) > u_g(o')$ (respectivement $u_g(o) = u_g(o')$) .

Le test de dominance pour les GAI est linéaire en la taille du GAI ce qui est l'avantage de cette représentation par rapport à d'autres. En effet, pour un



AB	
ab	0.7
$a\bar{b}$	1.2
$\bar{a}b$	1.8
$\bar{a}\bar{b}$	0.5

(a) u_{AB}

AC	
ac	1.5
$a\bar{c}$	1
$\bar{a}c$	1.2
$\bar{a}\bar{c}$	0.2

(b) u_{AC}

EF	
ef	0.2
$e\bar{f}$	0.8
$\bar{e}f$	0.4
$\bar{e}\bar{f}$	1

(c) u_{EF}

ADE	
ade	0
$ad\bar{e}$	1
$a\bar{d}e$	0.5
$a\bar{d}\bar{e}$	0
$\bar{a}de$	2
$\bar{a}d\bar{e}$	1.2
$\bar{a}\bar{d}e$	1.8
$\bar{a}\bar{d}\bar{e}$	0.4

(d) u_{ADE}

Figure 4.1 – *Une GAI décomposition de 5 variables*

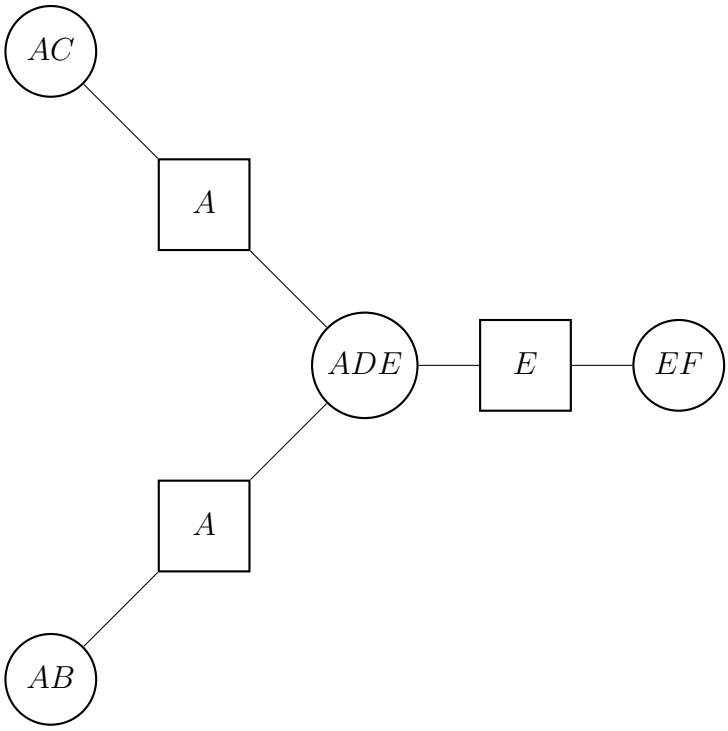


Figure 4.2 – *Réseau GAI*

objet o, l'utilité associée est égale à la somme des fonctions d'utilité sur les sous-ensembles Z de G.

Exemple 4.4. *Soit le GAI de la figure 4.1. Soit deux objets* $o_1 = abcdef$ *et* $o_2 = a\bar{b}cd\bar{e}f$. *L'utilité de l'objet* o_1 *est égale à* $u(o_1) = u(ab) + u(ac) + u(ade) + u(ef) = 0.7 + 1.5 + 0 + 0.2 = 2.4$ *et l'utilité de l'objet* o_2 *est égale à* $u(o_2) = u(a\bar{b})+u(ac)+u(ad\bar{e})+u(\bar{e}f) = 1.2+1.5+0+0.4 = 3.1$. *Comme on a* $u(o_2) > u(o_1)$ *alors on peut dire que* $o_2 \succ o_1$.

4.2.2 Requête d'optimalité.

Proposition 4.1. *Soit un GAI G et un objet o dans* $\underline{\chi}$. *On dit que :*

$$G \models o \text{ optimal} \;\Leftrightarrow\; \forall o' \in \chi, o' \neq o, u_g(o') \not\geq u_g(o)$$

.

Une méthode naïve pour trouver un objet optimal d'un GAI est de calculer l'utilité de chaque objet et de prendre celui qui a la plus grande utilité. Cette méthode n'est pas réaliste étant donné qu'un GAI sur n attributs a 2^n objets. Pour pallier à ce problème, un algorithme a été proposé par Gonzales *et al.* [40] en utilisant les réseaux GAI arborescents. Cela n'est pas gênant dans le cadre général car tout GAI-net sous forme de graphe peut être recompilé sous forme d'arbre [40]. Nous commencerons par présenter un exemple de cette méthode, puis nous donnerons les algorithmes proposés par Perny et Gonzales, ainsi que leurs complexités.

Exemple 4.5. *Considérons le problème consistant à déterminer la configuration maximale pour l'ensemble* $\chi = \{A, B, C, D, E, F\}$ *et la GAI décomposition* $u(ABCDEF) = u_1(AB) + u_2(AC) + u_3(ADE) + u_4(EF)$ *dont les tables sont présentées par la figure 4.1 et dont le GAI-net est présenté Figure4.2.*

Calculer une instanciation maximale revient à trouver $o^* \in \underline{\chi}$ *tel que* $u(o^*) = max_{o\in\underline{\chi}}u(o)$ *, ce qui revient à trouver :*

$$max_{x_a\in\underline{A}}max_{x_b\in\underline{B}}max_{x_c\in\underline{C}}max_{x_d\in\underline{D}}max_{x_e\in\underline{E}}max_{x_f\in\underline{F}}(u_1(x_ax_b)+u_2(x_ax_c)+u_3(x_ax_dx_e)+u_4(x_ex_f))$$

Pour cela, il suffit d'utiliser les propriétés de commutativité et d'associativité de l'addition et le fait que les sous-utilités ne sont pas définies sur l'intégralité des attributs du problème, afin de reformuler l'équation de la façon suivante :

$$u(o^*) = \max_{x_e \in \underline{E}} \max_{x_f \in \underline{F}} \Bigg(u_4(x_e x_f) + \max_{x_a \in \underline{A}} \left[\max_{x_d \in \underline{D}} u_3(x_a x_d x_e) + \max_{x_b \in \underline{B}} u_1(x_a x_b) + \max_{x_c \in \underline{C}} u_2(x_a x_c) \right] \Bigg)$$

En utilisant deux fonctions $\lambda_{i,j}$ et ψ_i, il est possible de réécrire cette formule de façon plus lisible :

- $\psi_2(x_a x_c) = u_2(x_a x_c)$
- $\lambda_{2,3}(x_a) = max_{x_c \in \underline{C}}\ \psi_2(x_a x_c)$
- $\psi_1(x_a x_b) = u_1(x_a x_b)$
- $\lambda_{1,3}(x_a) = max_{x_b \in \underline{B}}\ \psi_1(x_a x_b)$
- $\psi_3(x_a x_d x_e) = u_3(x_a x_d x_e) + \lambda_{1,3}(x_a) + \lambda_{2,3}(x_a)$
- $\lambda_{3,4}(x_e) = max_{x_d \in \underline{D}}\ max_{x_a \in A}\ \psi_3(x_a x_d x_e)$
- $\psi_4(x_e x_f) = u_4(x_e x_f) + \lambda_{3,4}(x_e)$
- $u(o^*) = max_{x_e \in \underline{E}} max_{x_f \in F} \psi_4(x_e x_f)$

En utilisant ces fonctions, les ψ_i permettent de représenter les sous-utilités ainsi que les sommes de l'expression. Les $\lambda_{i,j}$ sont des réductions des i sur un sous-ensemble d'attributs en utilisant la maximisation. Ainsi, Perny et Gonzales ont fait trois constatations :

- *ψ_i est définie sur les attributs de la clique qui définissent l'utilité u_i.*
- *$\lambda_{i,j}$ est définie sur les attributs du séparateur connectant la clique C_i à la clique C_j*
- *Ce séquençage suit un certain ordre le long du graphe.*

On peut représenter ce séquençage par un ensemble d'envois de messages le long des séparateurs du GAI-net. Cette phase s'appelle la phase de collecte et est représentée par la Figure 4.3 pour notre exemple.

On cherche maintenant à partir de ce système à trouver o^*. Pour cela, on continue le calcul de notre phase de collecte pour trouver $u(o^*)$ en maximisant $\psi_4(x_e x_f)$. Lors de celle-ci, on peut déterminer les valeurs de e^* et f^* ce qui ne suffit pas. En effet, en reprenant la façon dont on a construit notre 1^{er} système, on a $\psi_4(x_e x_f) = u_4(x_e x_f) + \lambda_{3,4}(x_e) = u_4(x_e x_f) + max_{x_d \in \underline{D}}\ max_{x_a \in \underline{A}}\ \psi_3(x_a x_d x_e)$. En développant cette expression, on peut obtenir ce nouveau système d'équations :

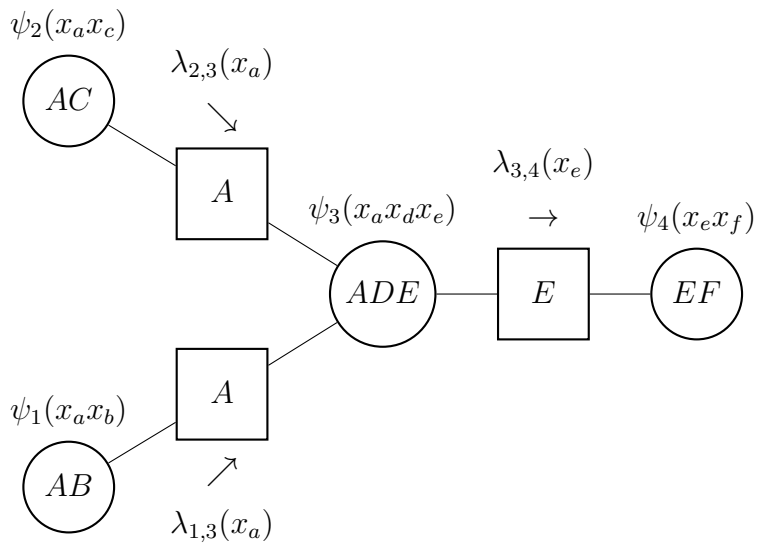


Figure 4.3 – *Envoi des messages de la méthode collecte dans le réseau GAI*

- $(e^*, f^*) = argmax_{(x_e,x_f)\in \underline{E}\times\underline{F}}\ \psi_4(x_e x_f)$
- $(d^*, a^*) = argmax_{(x_d,x_a)\in \underline{D}\times\underline{A}}\ \psi_3(x_a x_d x_e)$
- $b^* = argmax_{x_b\in\underline{B}}\ \psi_1(x_a x_b)$
- $c^* = argmax_{x_c\in\underline{C}}\ \psi_2(x_a x_c)$

Enfin, la dernière phase de collecte permet d'obtenir une instanciation partielle de o^*. Ensuite en remontant les messages envoyés lors de la phase de collecte comme le montre la figure 4.4, il suffit de transmettre la projection de l'instanciation partielle par le biais du séparateur. Ainsi, avec le message transitant et la fonction ψ_i calculée pendant la phase de collecte, on peut reconstruire au fur et à mesure l'objet optimal o^*. Cette dernière phase s'appelle la phase d'instanciation.

En utilisant ces deux phases sur notre exemple, et en calculant les valeurs intermédiaires de nos fonctions 4.5, on obtient :

- $(e^*, f^*) = (e\bar{f})$ *en effet d'après le tableau 4.5e la valeur la plus haute est 5.8 pour l'instanciation* $e\bar{f}$
- $(a^*, d^*) = (\bar{a}, d)$*, d'après le tableau 4.5c la valeur max est 5, on voit que cette valeur s'obtient grâce à la valeur d.*
- $(b^*) = b$ *si l'on a* $\bar{a}$*, la valeur optimale de la clique AB est de 1.8 pour l'instanciation b d'après le tableau 4.5a.*
- $(c^*) = c$ *si l'on a* $\bar{a}$*, la valeur optimale de la clique AC est de 1.2 pour l'instanciation b d'après le tableau 4.5b.*

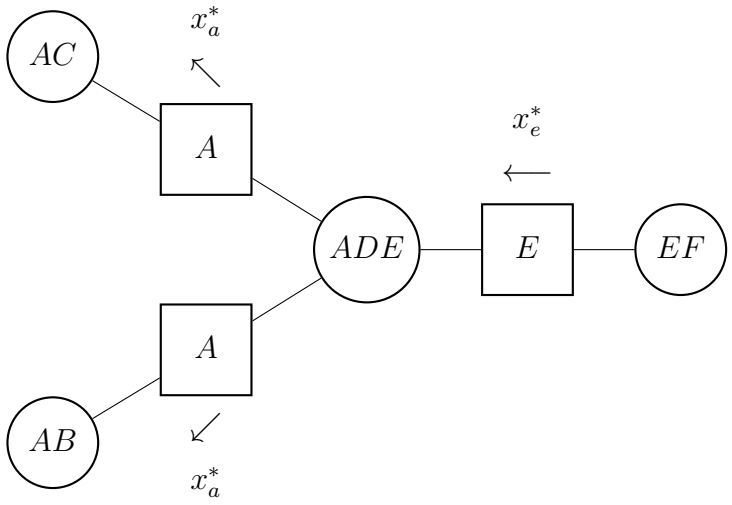


Figure 4.4 – *Réponse des messages de la phase Instanciation dans le réseau GAI*

Donc notre objet optimal est $o^* = \bar{a}bcde\bar{f}$ *et sa fonction d'utilité vaut 5.8.*

Cet exemple nous a permis de voir comment trouver l'objet optimal dans un réseau GAI sans calculer l'utilité de tous les objets possibles.
Nous allons maintenant donner la méthode pour trouver l'objet optimal. Pour cela, on utilise la fonction "ChoixOptimal" donnée par l'algorithme 7. Son principe est équivalent aux méthodes permettant de trouver l'événement le plus probable d'un réseau bayésien [23, 45]. Cet algorithme utilise deux fonctions :

- la fonction "Collecte" donnée par l'algorithme 8, c'est elle qui calcule les utilités maximales.
- la fonction "Instanciation" donnée par l'algorithme 9, c'est elle qui calcule la configuration optimale.

Dans ces fonctions, on ordonne les cliques $C = \{X_{C_1}, \ldots, X_{C_k}\}$ dans l'ordre inverse des appels à la fonction Collecte. Pour finir, dans la fonction "ChoixOptimal", la valeur de x_{C_k} donnée en argument dans l'appel de la fonction "Instanciation" est une instanciation quelconque des variables dans X_{C_k}.

La fonction "Collecte"(Algorithme 8 page suivante), comme dans l'exemple s'occupe de la collecte des cliques X_{C_i} tout en évitant la clique X_{C_r} afin d'éviter que le for des lignes 1-4 ne boucle indéfiniment.

Comme pour l'exemple, la fonction "Instanciation"(Algorithme 9 page 80) permet d'instancier les attributs. Comme pour Collecte, l'argument X_{C_r} permet

A	
a	1.2
$\bar{a}$	1.8

(a) $\lambda_{1,3}(a)$

A	
a	1.5
$\bar{a}$	1.2

(b) $\lambda_{2,3}(a)$

ADE	
ae	3.2
$a\bar{e}$	3.7
$\bar{a}e$	5
$\bar{a}\bar{e}$	4.2

(c) $\psi_3(ade)$

E	
e	5
$\bar{e}$	4.2

(d) $\lambda_{3,4}(e)$

EF	
ef	5.2
$e\bar{f}$	5.8
$\bar{e}f$	4.6
$\bar{e}\bar{f}$	5.2

(e) $\psi_4(ef)$

Figure 4.5 – *Calcul des fonctions intermédiaires*

Algorithm 7: ChoixOptimal

Input: $\mathcal{G}$
Output: o^* : l'objet optimal
Appeler Collecte(X_{C_k}, X_{C_k}) ;
Appeler Instanciation($X_{C_k}, X_{C_k}, x_{C_k}$) ;
return o^* *un objet optimal* ;

Algorithm 8: Collecte

Input: X_{C_i},X_{C_r}
foreach *les cliques* $X_{C_j} \in \{$*cliques adjacentes* à $X_{C_i}\} \setminus \{X_{C_r}\}$ **do**
    appeler Collecte(X_{C_j}, X_{C_i});
    foreach $x_{C_i} \in \underline{X}_{C_i}$ **do**
        $u_i(x_{C_i}) \leftarrow u_i(x_{C_i)} + u_j^*(x_{S_j})$ où x_{S_j} est la projection de x_{C_i} sur $X_{C_i \cap C_j}$;
if $X_{C_i} \neq X_{C_r}$ **then**
    foreach $x_{S_j} \in \underline{X}_{S_j}$ **do**
        $u_i^*(x_{S_i}) \leftarrow max_{x_{D_i}} u_i(x_{C_i})$;
return *location*;

de ne pas boucler indéfiniment sur le for des lignes 6-7. Enfin, l'argument $X^*_{C_r}$ contient les valeurs optimales des attributs de X_{C_r}

Algorithm 9: INSTANCIATION

Input: $X_{C_i}, X_{C_r}, x^*_{c_k}$
Output: les valeurs optimales des variables.

```
1 if X_{C_i} = X_{C_r} then
2     x*_{C_i} ← argmax{u_i(x_{C_i}) : x_{C_i} ∈ X_{C_i}};
3 else
4     x*_{S_i} ← La projection de x*_{C_r} sur X_{C_i ∩ C_R};
5     x*_{C_i} ← argmax{u_i(x*_{S_i}, x_{D_i}) : x_{D_i} ∈ X_{D_i}};
6 foreach les cliques X_{C_j} ∈ {cliques adjacentes à X_{C_i}} \ {X_{C_r}} do
7     appeler Instanciation(X_{C_j}, X_{C_i}, x*_{c_i});
8 return location;
```

Complexité

En ce qui concerne la phase de collecte, la complexité est en $O(|C| max_{Ci \in C} \prod_{X_i \in C_i} |X_i|)$. Pour la phase d'Instanciation, elle est en $O(|C| max_{Ci \in C} |C_i|)$.

4.2.3 Élicitation

Dans cette partie, nous allons voir deux méthodes proposées par Gonzales et Perny [39] et Braziunas [17] qui portent sur l'élicitation de l'utilité dans un GAI ainsi que l'élicitation l'utilité pour un ensemble de variable locale.

4.2.3.1 Élicitation de l'utilité

L'élicitation de l'utilité des GAI lorsque la structure est connue proposée par Gonzales et Perny [39] se situe dans le contexte de l'incertain basé sur la théorie des jeux :"standard gamble query" (SGQ). Le principe de cette méthode est de demander à un agent s'il préfère un objet o_i plutôt qu'un lot tel que l'objet optimal o_{max} pour l'agent arrive avec une probabilité p et le pire objet o_{min} arrive une probabilité 1-p. On se retrouve donc dans le cas où $o_{max} \succeq o_i \succeq o_{min}$. Lorsque l'on utilise une utilité, on obtient ainsi $u(o_{max}) \geqslant u(o_i) \geqslant u(o_{min})$. On obtient donc, grâce aux SGQ l'équation $u(o_i) = pu(o_{max}) + (1-p)u(o_{min})$, ou de façon

analogue, on fait le pari que $o_i \sim \langle p, o_{max}; 1-p, o_{min}\rangle$. Ainsi, connaissant p, o_{max}, o_{min}, il est possible de déterminer la valeur $u(o_i)$.

En utilisant simplement cette méthode, le nombre de questions à poser serait énorme étant donné qu'on se trouve dans un domaine combinatoire. Pour cela, on utilise la décomposabilité des GAI pour réduire le nombre de questions à poser à l'agent afin d'éliciter complètement la fonction d'utilité. De plus, les SGQ posées concernent toutes les variables, cependant le nombre d'attributs qui diffèrent entre les deux alternatives est relativement petit ce qui réduit encore plus la complexité des questions.

4.2.3.2 Élicitation locale de l'utilité

Suite aux travaux de Perny et Gonzales [39], Braziunas et Boutillier [17] ont cherché à réduire le nombre de questions à poser à l'utilisateur. Pour cela, ils ont montré que l'élicitation peut être simplifiée en posant des questions locales sur les attributs de la GAI décomposition et ses ensembles conditionnant.

Définition 4.4 (Ensemble conditionnant)**.** *Soit une GAI-décomposition* $G = \{Z_1, \ldots, Z_n\}$ *sur* $\chi = \{X_1, X_2, \ldots, X_n\}$*, où* Z_i *est une clique de* G*. Soit* $M_j = \bigcup_{i,j\in Z_i} Z_i$ *l'union de toutes les cliques possédant la variable* j*. L'ensemble conditionnant d'une clique* Z_i *est l'ensemble des voisins de* Z_i *moins la clique* Z_i *:*

$$EC_i = \bigcup_{j\in Z_i} M_j - Z_i$$

Ainsi, toutes les variables qui n'appartiennent ni à M_j ni à ECi n'auront pas d'influence sur l'élicitation. On peut donc proposer lors des SGQ des objets partiels afin de faciliter les questions posées à l'utilisateur. Une fois les EC_i définis, on donne pour chaque variable des EC_i une valeur de référence x^0. De plus, on définit $o_i^\top$ et $o_i^\perp$ comme respectivement le meilleur et le pire objet. On peut donc réécrire un objet o comme (o_i, o_{CI}, y), où y sont l'ensemble des variables qui ne sont ni dans ZI ni dans ECI. De plus, si les variables de EC_i sont à leur valeur de référence (x^0), on peut écrire les paris de la façon suivante : $(o_i, x^0_{EC_i}) \sim \langle p, (o_i^\top, x^0_{EC_i}); 1-p, (o_i^\perp, x^0_{EC_i})\rangle$.

Chapitre 5

Apprentissage de GAI-décomposition

Sommaire

5.1 Quelques nouvelles notions

Nous allons maintenant nous intéresser à l'apprentissage des GAI-décompositions. Pour cela, nous définissons deux nouvelles notions sur les GAI-décompositions. La première est la définition du degré d'un GAI : c'est la taille maximale d'une clique le composant.

Définition 5.1 (degré)**.** *Le* degré *d'une GAI-décomposition* $G = \{u_{Z_1}, \ldots, u_{Z_m}\}$*, noté* $deg(G)$*, est le cardinal du plus grand* Z_i *de* G*.*

On appelle donc une k-GAI-décomposition une GAI-décomposition dont la taille des cliques est au plus de taille k.

Une même relation de préférence peut être représentée par plusieurs fonctions d'utilité, par exemple, parce que toute transformation affine (à coefficients positifs) d'une fonction d'utilité représentant un ordre représente le même ordre. D'autre part, une fonction d'utilité donnée admet plusieurs GAI décompositions. Dans l'ensemble des GAI-décompositions représentant $\succeq$, on s'intéressera tout particulièrement aux représentations minimales au sens de la décomposition.

Définition 5.2 (minimalité)**.** *Une GAI-décomposition* $G = \{u_{Z_1}, \ldots, u_{Z_m}\}$ raffine *une autre GAI décomposition* $G' = \{u'_{Z'_1}, \ldots, u'_{Z'_{m'}}\}$ *si pour tout* Z_i*, il existe* $Z'_j \supseteq Z_i$*. Le raffinement est dit* strict *si l'inclusion est stricte pour au moins l'un des* Z_i*, ou s'il existe un* Z'_j *avec aucun* $Z_i \subseteq Z'_j$*. Une GAI-décomposition* G *d'une relation de préférence* $\succeq$ *est dite* minimale *si aucune autre GAI décomposition de* $\succeq$ *ne la raffine strictement.*

Ces notions sont résumées et illustrées dans l'exemple suivant.

Exemple 5.1. *Reprenons l'exemple 4.2 page 70 c'est-à-dire l'ordre de préférence suivant :*

$$o_2 \succ o_4 \sim o_7 \succ o_1 \succ o_8 \sim o_5 \succ o_3 \succ o_6$$

On peut représenter cette relation par la GAI-décomposition $G = \{u_{\{X_1,X_2,X_3\}}\}$ *comportant une seule fonction d'utilité* $u_{\{X_1,X_2,X_3\}}$ *dont la table est la suivante :*

o_1	v, r, c	8
o_2	v, r, m	12
o_3	v, b, c	5
o_4	v, b, m	9
o_5	p, r, c	6
o_6	p, r, m	3
o_7	p, b, c	9
o_8	p, b, m	6

Cette décomposition n'est pas minimale, puisqu'elle représente la même fonction d'utilité u *que la décomposition* $G' = \{u_{\{X_1,X_2\}}, u_{\{X_1,X_3\}}\}$ *donnée dans l'exemple 4.2 page 70. Or* G' *a un degré de 2 alors que* G *à un degré de 3.*

Nous allons maintenant nous intéresser à l'apprentissage de GAI-décomposition de degré k en transformant un ensemble d'exemples en un ensemble d'inéquations linéaires et en résolvant un système inéquation linéaire. Il est aussi possible d'apprendre des GAI-décompositions de degré 1 en utilisant des SVM [46] .

5.2 Transformation en inéquation linéaire

5.2.1 Rappel sur les systèmes d'équations linéaires

Une équation est dite linéaire si elle peut être représentée sous la forme $ax+b = 0$ où x est un vecteur, a un vecteur de coefficients et b un nombre donné. De même une inéquation est linéaire si elle s'écrit sous la forme $ax + c \geqslant b$. Par définition, un système d'équations linéaires (resp. système d'inéquations linéaires) est un ensemble d'équations(resp. d'inéquations) linéaires qui portent sur les mêmes inconnues. Dans cette partie, on s'intéresse aux inéquations linéaires et à leurs systèmes.

Exemple 5.2. *Voici l'exemple d'un système d'inéquations linéaires à 3 inconnues*

$$\begin{cases} 3x + 2y - 6z \geqslant 8 \\ 7y + 2z \geqslant 10 \\ 2x - y + 4z \geqslant 2 \end{cases}$$

Les systèmes d'inéquations linéaires sont souvent accompagnés d'une fonction objectif qui permet de maximiser ou de minimiser un problème. Par exemple on cherche à minimiser la somme de x, y, z. Afin de résoudre un tel système avec sa fonction objectif, on utilise la programmation linéaire. Il existe deux familles de méthodes pour résoudre ce problème. La première est la méthode des points intérieurs proposé par Nesterov *et al.* [55], cette méthode a permis de prouver que la programmation linéaire avait une complexité polynomiale lorsque les inconnues étaient des réels. La seconde méthode est la méthode du simplexe. C'est la plus utilisée malgré son algorithme pouvant être de complexité exponentielle, elle est souvent plus efficace que la méthodes des points intérieurs.

Par contre, la programmation linéaire en nombres entiers (c'est-à-dire. que l'ensemble des inconnues sont définies sur $\mathbb{N}$) utilise des algorithmes du type séparation et évaluation. C'est un problème NP-complet.

5.2.2 Transformation d'une relation de préférence en inéquation linéaire.

On montre dans cette section que l'ensemble des GAI-décompositions d'une relation de préférence donnée peut être caractérisé comme l'ensemble des solutions d'un système d'inégalités linéaires. Le nombre de variables du système dépend

exponentiellement du degré des décompositions visées, mais polynomialement (à degré fixé) du nombre d'attributs utilisés pour décrire les objets du domaine.

On verra dans la section 5.3 que cette construction permet d'apprendre des décompositions cohérentes avec un ensemble d'exemples donné en entrée.

Définition 5.3 (cohérence)**.** *Soit E un ensemble d'exemples. Une fonction d'utilité u est dite* cohérente *avec E si pour tout exemple $(o, R, o') \in E$, on a $u(o) > u(o')$ (respectivement $u(o) \geq u(o')$) si R est la relation $\succ$ (respectivement $\succeq$).*

Dans la suite, on supposera que l'ensemble d'exemples est *sain*, c'est-à-dire qu'il existe une relation $\succeq^*$, complète et transitive, telle que pour tout exemple $(o, \succ, o') \in E$ (resp. tout exemple $(o, \succeq, o') \in E$), on ait $o \succ^* o'$ (resp. $o \succeq^* o'$). On utilisera ainsi la même notation, $\succeq$, pour la relation cible et pour sa restriction aux exemples.

Définition 5.4 (inégalité linéaire associée à un exemple)**.** *Soit $e = (o, R, o')$ un exemple d'une relation de préférence $\succeq$ sur $\underline{\chi} = \underline{X}_1 \times \cdots \times \underline{X}_n$, et soit $k \in \{1, \ldots, n\}$. Soit $\sigma > 0$ une constante réelle, strictement positive mais arbitraire.*

Pour tout sous-ensemble de variables $Z \subseteq \chi$ et toute affectation z de Z, on introduit une variable U_z à valeurs réelles.

*L'*inégalité linéaire *$ineq_k(e)$ associée à $e = (o, R, o')$ et k est définie comme*

$$\sum_{Z \subseteq \chi, 0 < |Z| \leq k} U_{o[Z]} \geq \sum_{Z \subseteq \chi, 0 < |Z| \leq k} U_{o'[Z]} + \sigma$$

si R est la relation $\succ$, comme

$$\sum_{Z \subseteq \chi, 0 < |Z| \leq k} U_{o[Z]} \geq \sum_{Z \subseteq \chi, 0 < |Z| \leq k} U_{o'[Z]}$$

si R est la relation $\succeq$.

Si E est un ensemble d'exemples de $\succeq$, le système linéaire *associé à E et k est défini comme le système d'inégalités linéaires $\Sigma_k(E) = \bigwedge_{e \in E} ineq_k(e)$.*

Les variables $U_{o[Z]}$ codent les éléments des GAI-tables de la décomposition G à apprendre : on définira G par $u_Z(z) = U_{o[Z]}$ pour $|Z| \leq k$ et $o \in O_E$, et par $u_Z(z) = 0$ sinon. D'autre part, nous utilisons une constante σ (un pas) pour coder une inégalité stricte avec comme but d'utiliser la programmation linéaire (la proposition 5.1, ci-dessous, montre que ceci est sans perte de généralité) : si l'on considérait des inégalités strictes (et, partant de là, l'ensemble de toutes les GAI-décompositions cohérentes avec E), nous considérerions un espace topologique ouvert.

Exemple 5.3. *Soient* $o = x_1x_2\bar{x_3}$ *et* $o' = \bar{x_1}x_2x_3$*. L'inégalité linéaire associée à l'exemple* $e = (o, \succ, o')$ *pour* $k = 2$ *et* $\sigma = 0, 1$ *est :*

$$\begin{aligned} & U_{x_1} + U_{x_2} + U_{\bar{x}_3} + U_{x_1x_2} + U_{x_1\bar{x_3}} + U_{x_2\bar{x_3}} \\ \geq\ & U_{\bar{x_1}} + U_{x_2} + U_{x_3} + U_{\bar{x_1}x_2} + U_{\bar{x_1}x_3} + U_{x_2x_3} + 0.1 \end{aligned}$$

Notons que le système $\Sigma_k(E)$ a au plus $\sum_{i=0}^{k} 2^i \binom{n}{i}$ inconnues (au plus autant que d'affectations possibles pour chaque sous-ensemble d'au plus k variables de χ). En réalité, cependant, ce nombre de variables est moindre : la variable U_z n'apparaît que s'il existe un objet $o \in O_E$ avec $o[Z] = z$. De fait, le nombre d'inconnues dans une inégalité est au plus $2.\sum_{i=0}^{k} \binom{n}{i}$, et le nombre total d'inconnues dans le système $\Sigma_k(E)$ est au plus $\sum_{i=0}^{k} \binom{n}{i}.|O_E|$.

On montre maintenant que le système linéaire de la définition 5.4 caractérise les GAI-décompositions de degré au plus k cohérentes avec E. Pour des raisons techniques cependant, nous nous restreignons aux fonctions d'utilités *de pas au moins* σ, c'est-à-dire aux fonctions u vérifiant $|u(o)-u(o')| \geq \sigma$ pour tous o, o' avec $u(o) \neq u(o')$. Ceci est sans perte de généralité en vertu de la proposition suivante (qui invoque essentiellement l'invariance de l'ordre induit par une transformation affine à coefficient positif).

Proposition 5.1. *Soit* E *un ensemble fini d'exemples et* $\sigma > 0$ *un réel arbitraire. Il existe une fonction d'utilité* u *et une GAI-décomposition de* u *de la forme* $(u_{Z_1}, \ldots, u_{Z_m})$ *et cohérente avec* E *si et seulement s'il existe une fonction* u' *avec une GAI-décomposition* $(u'_{Z_1}, \ldots, u'_{Z_m})$ *tel que* u' *soit cohérente avec* E *et que* $|u(o) - u(o')| \geq \sigma$ *pour chaque exemple* (o, R, o')*.*

Démonstration. Il suffit d'observer que si u est de pas $\sigma_u < \sigma$, alors la fonction u' définie par la décomposition $((\sigma/\sigma_u)u_{Z_1}, \ldots, (\sigma/\sigma_u)u_{Z_m})$ représente la même relation de préférence que u, et est donc également cohérente avec E tout en ayant clairement un pas supérieur ou égal à σ. □

Proposition 5.2. *Soit* $\succeq$ *une relation de préférence sur* $\chi = X_1 \times \cdots \times X_n$*, soit* E *un ensemble d'exemples de* $\succeq$*, soit* $k \in \{1, \ldots, n\}$*, et soit* $\sigma > 0$ *arbitraire. Alors les GAI-décompositions de degré au plus* k*, de pas au moins* σ*, et cohérentes avec* E *sont en bijection avec les solutions du système* $\Sigma_k(E)$ *(considéré comme possédant toutes les variables* U_z*,* $|z| < k$*, même si elles n'apparaissent pas dans les inégalités).*

Démonstration. La preuve est directe en utilisant, pour une GAI-décomposition telle que dans l'énoncé, l'affectation s des variables $U_{o[Z]}$ du système linéaire défini par $s(U_{o[Z_i]}) = u_{Z_i}(o[Z_i])$ pour tous $i \in \{1, \ldots, m\}, o \in \underline{\chi}$. □

5.3 Apprendre un GAI grâce à un solveur d'équations linéaires

On montre dans cette section qu'il est possible d'apprendre de façon passive une GAI-décomposition d'une relation de préférence $\succeq^*$, en utilisant un nombre d'exemples (de la forme (o, R, o')) qui croît exponentiellement avec le degré k de cette décomposition, mais qui, à k fixé, croît polynomialement avec le nombre d'attributs. En particulier, si l'on note G_k la classe de toutes les relations de préférence $\succeq$ qui ont une GAI-décomposition de degré au plus k, alors la classe G_k est apprenable à partir d'exemples.

Pour cela, nous donnons tout d'abord une borne théorique sur le nombre d'exemples nécessaires (section 5.3.1), puis nous donnons un algorithme pour le cas d'un degré k borné et connu (section 5.3.1.1). Dans les grandes lignes, l'algorithme maintient l'*espace des versions* de toutes les GAI-décompositions de degré k *et de pas au moins* σ cohérentes avec les exemples reçus, en le représentant par le système linéaire $\Sigma_k(E)$. La VC-dimension de la classe G_k permet de conclure que l'algorithme converge dans le cadre de l'apprentissage PAC vers $\succeq^*$ après réception d'un nombre polynomial d'exemples.

5.3.1 VC-dimension

Proposition 5.3. *Soit $C_{\succ}^{n,k}$ la classe des concepts binaires de la forme $C_{\succ}$ où $\succeq$ est une relation de préférence sur $\underline{\chi} = \underline{X}_1 \times \cdots \times \underline{X}_n$ représentable par une GAI-décomposition de degré k. La VC-dimension pour des attributs binaires de $C_{\succ}^{n,k}$ est en $O(2^k n^{k+1})$.*

Plus généralement, si les domaines des variables ne sont pas booléens alors la VC-dimension est en $O(|D|^k n^{k+1})$ où D est le domaine des variables.

Démonstration. Soit $K = \sum_{i=0}^{k} 2^i \binom{n}{i}$.Nous montrons qu'aucun ensemble de couples d'objets $O \subseteq \underline{\chi} \times \underline{\chi}$ de taille strictement supérieure à K n'est éclaté par $C_{\succ}^{n,k}$, ce qui conclura.

Soit donc $O \subseteq \underline{\chi} \times \underline{\chi}$ un ensemble de couples d'objets de taille au moins $K+1$. Pour chaque couple $(o, o') \in O$, dans l'esprit de la définition 5.4 nous définissons la somme (combinaison linéaire) formelle

$$V_{o,o'}^{k} = \sum_{Z \subseteq \chi, 0 < |Z| \leq k} U_{o[Z]} - U_{o'[Z]}$$

qui correspond donc à la combinaison des membres gauche et droit de toute inégalité linéaire associée à o et o'.

Tous les $V^k_{o,o'}$ (pour $(o,o') \in O$) portent sur les mêmes K variables. Donc, si O contient au moins $K+1$ couples, il existe au moins un couple (ω,ω') tel que le vecteur $V^k_{\omega,\omega'}$ est une combinaison linéaire des autres vecteurs, c'est-à-dire qu'il existe une famille de coefficients $\lambda_{o,o'}$ (pour $(o,o') \in O \setminus \{(\omega,\omega')\}$) vérifiant

$$V^k_{\omega,\omega'} = \sum_{(o,o')\in O\setminus\{(\omega,\omega')\}} \lambda_{o,o'} V^k_{o,o'}$$

Soit O^- (resp. O^+) le sous-ensemble de $O \setminus \{(\omega,\omega')\}$ constitué des couples (o,o') avec $\lambda_{o,o'} \leq 0$ (resp. $\lambda_{o,o'} > 0$).

Si $O^+ \neq \emptyset$, nous montrons alors qu'aucun concept $C \in C^{n,k}_{\succ}$ n'est cohérent avec la partition $O^1 = O^+, O^0 = O^- \cup \{(\omega,\omega')\}$, c'est-à-dire avec les étiquettes $o \succ_C o'$ pour tout $(o,o') \in O^+$, $o \preceq_C o'$ pour tout $(o,o') \in O^-$, et $\omega \preceq_C \omega'$. Dans le cas $O^+ = \emptyset$, il est facile de voir que le système implique $\omega \preceq_C \omega'$, et c'est alors $\omega \succ_C \omega'$ qui est impossible.

En utilisant les propositions 5.1 et 5.2 nous obtenons immédiatement que le système linéaire suivant doit avoir une solution (pour un $\sigma > 0$ arbitraire) pour qu'il existe une fonction d'utilité u ayant une GAI-décomposition de degré k et consistante avec les étiquettes susmentionnées :

$$\begin{aligned} V^k_{o,o'} &\geq \sigma \quad (\forall (o,o') \in O^+) \\ V^k_{o,o'} &\leq 0 \quad (\forall (o,o') \in O^-) \end{aligned}$$

Du fait des signes des $\lambda_{o,o'}$, il s'ensuit que les équations suivantes doivent être vérifiées :

$$\begin{aligned} \lambda_{o,o'} V^k_{o,o'} &\geq \lambda_{o,o'}\sigma \quad (\forall (o,o') \in O^+) \\ \lambda_{o,o'} V^k_{o,o'} &\geq 0 \quad (\forall (o,o') \in O^-) \end{aligned}$$

Toute solution de ce système doit donc satisfaire

$$\sum_{(o,o')\in O\setminus\{(\omega,\omega')} \lambda_{o,o'} V^k_{o,o'} \geq \sigma \sum_{(o,o')\in O^+} \lambda_{o,o'}$$

c'est-à-dire

$$V_{\omega,\omega'} \geq \sigma \sum_{(o,o')\in O^+} \lambda_{o,o'}$$

En utilisant $O^+ \neq \emptyset$ et à nouveau la proposition 5.2, nous obtenons $\omega > \omega'$, donc $\omega \preceq_C \omega'$ est impossible et donc, O n'est pas éclaté par $C^{n,k}_{\succ}$.

Si par contre $O^+ = \emptyset$, nous montrons qu'aucun concept $c \in C_{\succ}^{n,k}$ n'est cohérent avec une partition (O^1, O^0) où O^1 contient $O^- \cup \{(\omega, \omega')\}$: en effet si on peut avoir $O^- \subseteq O^1$, alors les $V_{o,o'}^k$ pour $(o, o') \in O^-$ sont > 0, et donc

$$V_{\omega,\omega'}^k = \sum_{(o,o') \in O^-} \lambda_{o,o'} V_{o,o'}^k \leq 0$$

et donc forcément $(\omega, \omega') \in O^0$.

Dans les deux cas, on conclut que O n'est pas éclaté par $C_{\succ}^{n,k}$, et puisque O est arbitraire de taille $K + 1$, que la VC-dimension de $C_{\succ}^{n,k}$ est au plus K.

□

5.3.1.1 PAC-Apprenabilité

On montre maintenant que C^k est PAC-apprenable. Pour cela, nous exhibons un algorithme polynomial permettant d'apprendre une GAI-décomposition cohérente avec un ensemble d'exemples donnés.

Pour montrer que les GAI-décompositions de degré k connu et borné sont PAC-apprenables, on utilise une approche standard, connue sous le nom d'« apprentissage cohérent » (*consistent learning*). Dans cette approche, l'apprenant maintient à tout instant un concept cohérent avec tous les exemples reçus jusqu'alors ; en l'occurrence, notre apprenant les maintient essentiellement tous (au *step* σ près) en intention, via le système $\Sigma_k(E)$. Des résultats génériques permettent de conclure sur le nombre m d'exemples nécessaires en utilisant la VC-dimension.

L'algorithme est représenté ci-dessous. La valeur de m est donnée plus loin (Proposition 5.4). Notons que les exemples étant sans bruit, l'approche par consistance retourne toujours une solution, en l'occurrence, le système $\Sigma_k(E)$ a nécessairement une solution.

Proposition 5.4. *Pour toute constante $k > 0$, la classe C^k des GAI-décompositions de degré au plus k est PAC-apprenable. Le nombre m d'exemples requis par l'algorithme* GAI-Learning *est en* $O(\max(\frac{1}{\varepsilon}\log\frac{1}{\delta}, \frac{2^k n^{k+1}}{\varepsilon}\log\frac{1}{\varepsilon}))$.

Démonstration. La proposition 5.2 montre que la GAI-décomposition retournée par l'algorithme est cohérente avec l'ensemble de tous les exemples reçus.

Une conséquence directe d'un résultat générique de Blumer *et al.* [9] est qu'un nombre d'exemples m en $O(\max(\frac{1}{\varepsilon}\log\frac{1}{\delta}, \frac{2^k n^k}{\varepsilon}\log\frac{1}{\varepsilon}))$ ($2^k n^{k+1}$ étant la VC-dimension du problème) suffit pour apprendre une GAI-décomposition G telle que l'ordre associé $\succeq_G$ soit ε-correct (même si la VC-dimension telle que donnée dans

Algorithm 10: GAI-Learning

Input: Un ensemble d'exemples E
Output: G

```
Requit k < n ; Σ_k(E) = ∅ ;
for i = 1, ..., m do
    demander un exemple e de la forme (o, R, o') ; Ajouter ineq_k(e) à Σ_k(E) ;
résoudre Σ_k(E) ;
return retourner une GAI-décomposition construite à partir d'une solution de Σ_k(E) ;
```

la proposition 5.3 concerne les concepts binaires $C_{\succ}$ et $C_{\prec}$, les mêmes exemples sont utilisés pour apprendre à la fois le concept binaire $C_{\succ}$ et le concept binaire $C_{\prec}$, via le système $\Sigma_k(E)$; pour plus de détails sur ce type de constructions, on renvoie le lecteur à [5]).

En outre, m étant polynomial en $n, 1/\epsilon, 1/\delta$ (rappelons que k est considéré comme constant), le système $\Sigma_k(E)$ est de taille polynomiale, et en tant que système linéaire, il peut être résolu en temps polynomial.

□

5.3.2 Différentes fonctions objectifs

Dans la section précédente, nous avons considéré le problème d'apprentissage d'une GAI-décomposition d'une fonction d'utilité cible, sans autre restriction qu'une borne sur le degré maximal k, connu à l'avance.

Nous montrons maintenant comment généraliser ce résultat en calculant des GAI-décompositions aussi « simples » que possible. Pour cela, nous proposons différents problèmes d'optimisation, basés sur le système linéaire $\Sigma_k(E)$. Nous explorons deux notions de « simplicité » d'une GAI-décomposition, toujours en supposant une borne k, connue, sur le degré de la décomposition cible.

Nous cherchons tout d'abord à maximiser le nombre d'entrées nulles dans les tables de la GAI-décomposition apprise, c'est-à-dire à maximiser le nombre de couples (Z, z) avec $u_Z(z) = 0$. Optimiser cette mesure permet en particulier de réduire l'espace nécessaire pour stocker la décomposition apprise.

Pour atteindre cet objectif, étant donné un degré k fixé et un ensemble

d'exemples E, on définit le problème d'optimisation linéaire suivant :

$$\text{(P1)} \begin{cases} \text{minimiser} \sum_{Z \subseteq \chi, 0<|Z| \leq k, o \in O_E} U_{o[Z]} \\ \text{sous les contraintes} \\ \bullet\ ineq_k(e) \text{ pour tout } e \in E \\ \bullet\ U_{o[Z]} \geq 0 \text{ pour tous } Z, o \end{cases}$$

Bien que (P1) permette d'obtenir une certaine forme de minimalité, il ne suffit pas à minimiser le nombre de coefficients non nuls dans les tables, comme le montre l'exemple suivant.

Exemple 5.4. *Soient deux variables booléennes X_1, X_2, et $E = \{x_1x_2 \succ x_1\bar{x}_2\ ,\ x_1\bar{x}_2 \succ \bar{x}_1\bar{x}_2\ ,\ \bar{x}_1\bar{x}_2 \succ \bar{x}_1x_2\}$. On fixe $k = 2$, $\sigma = 1$, et on considère les décompositions (u_1, u_{12}) et (u'_{12}), toutes deux consistantes avec E :*

$$u_1 : \begin{array}{|c|c|} \hline x_1 & 1 \\ \hline \bar{x}_1 & 0 \\ \hline \end{array} \quad u_{12} : \begin{array}{|c|c|} \hline x_1x_2 & 2 \\ \hline x_1\bar{x}_2 & 1 \\ \hline \bar{x}_1\bar{x}_2 & 1 \\ \hline \bar{x}_1x_2 & 0 \\ \hline \end{array} \quad u'_{12} : \begin{array}{|c|c|} \hline x_1x_2 & 3 \\ \hline x_1\bar{x}_2 & 2 \\ \hline \bar{x}_1\bar{x}_2 & 1 \\ \hline \bar{x}_1x_2 & 0 \\ \hline \end{array}$$

La décomposition (u'_{12}) a moins de coefficients non nuls que (u_1, u_{12}), pourtant cette dernière a une meilleure valeur dans (P1).

Une solution pour tenter de minimiser le nombre de coefficients non nuls consiste alors à effectuer un post-traitement peu coûteux aux solutions de (P1) : on reporte les valeurs des coefficients des tables de certains Z_i vers les tables de $Z_j \supset Z_i$.

Un autre objectif naturel est de minimiser le degré de la GAI-décomposition apprise (sous la borne k). Notons tout d'abord que cet objectif n'est pas réalisé, dans le cas général, par l'objectif précédent, autrement dit, que trouver un optimum des problèmes (P1) ne garantit pas qu'on a une GAI-décomposition de degré minimal :

Exemple 5.5. *Soient $\chi = \{X_1, X_2\}$, $D_1 = \{x_1, \bar{x}_1\}$ et $D_2 = \{x_2, \bar{x}_2\}$. Soit $E = \{x_1\bar{x}_2 \succ \bar{x}_1x_2, x_1x_2 \succ \bar{x}_1x_2, x_1\bar{x}_2 \succ x_1x_2\}$, donc $O_E = \{x_1\bar{x}_2, \bar{x}_1x_2, x_1x_2\}$. En cherchant une GAI-décomposition de degré au plus 2, on a le problème d'optimisation suivant :*

$$\begin{cases} \textit{minimiser} \sum_{Z\subseteq\chi, 0<|Z|\leq k, o\in O_E} U_{o[Z]} \\ \textit{sous les contraintes} \\ \bullet\ U_{x_1} + U_{\bar{x}_2} + U_{x_1\bar{x}_2} \geq U_{\bar{x}_1} + U_{x_2} + U_{\bar{x}_1 x_2} + \sigma \\ \bullet\ U_{x_1} + U_{x_1 x_2} \geq U_{\bar{x}_1} + U_{\bar{x}_1 x_2} + \sigma \\ \bullet\ U_{\bar{x}_2} + U_{x_1\bar{x}_2} \geq U_{x_2} + U_{x_1 x_2} + \sigma \\ \bullet\ U_{o[Z]} \geq 0 \textit{ pour tous } Z, o \end{cases}$$

En prenant par exemple $\sigma = 0.1$, on obtient comme solution optimale $U_{x_1x_2} = 0.1$ et $U_{x_1\bar{x}_2} = 0.2$, toutes les autres valeurs étant nulles. On a donc une décomposition de degré 2, alors qu'il en existe une de degré 1, définie par $U_{x_1} = U_{\bar{x}_2} = 0.1$, toutes les autres valeurs étant nulles (d'après les exemples, les deux variables X_1 et X_2 sont indépendantes, avec x_1 comme valeur préférée pour X_1 et $\bar{x}_2$ pour X_2).

En pondérant les $U_{o[Z]}$ dans la fonction objectif, on peut pousser la minimisation et donc la mise à zéro de certains d'entre eux, et donc favoriser *a contrario* certaines cliques – les plus petites – dans la décomposition. On introduit donc des poids w_i, où chaque w_i pondérera, dans la fonction objectif, les utilités de degré i (i.e. les $U_{o[Z]}$ tels que $|Z| = i$). Formellement, on cherche à minimiser la fonction objectif suivante :

$$\sum_{Z\subseteq\chi \& |Z|\leq k \& o\in O_E} w_{|Z|} \times U_{o[Z]}$$

On doit fixer les w_is de manière à rendre un $U_{o[Z]}$ de degré i plus coûteux que l'ensemble des $U_{o[Z']}$ pour $Z' \subset Z$ et $|Z| = i - 1$; les w_is doivent alors vérifier

$$w_i \ \times\ \min(U_{o[Z]}) > i \ \times\ 2^{i-1} \ \times\ w_{i-1} \ \times\ \max(U_{o[Z]})$$

où le $\min(U_{o[Z]})$ est pris sur l'ensemble des $U_{o[Z]}$ non nuls. On peut imposer $U_{o[Z]} \leq 1$ pour tous les o, Z, ce qui conduit à la constante σ suffisamment petite pour ne pas rendre le système d'équations linéaires artificiellement inconsistant. Pour imposer que les $U_{o[Z]}$ non nuls aient un minimum non nul, on peut introduire de nouvelles variables booléennes $V_{o[Z]}$, on alors le problème de programmation linéaire mixte suivant :

$$\text{(P2)} \begin{cases} \text{minimiser } (\sum_{Z\subseteq\chi} \text{t.q. } {}_{|Z|\leq k, o\in O_E} w_{|Z|} \times U_{o[Z]}) \\ \text{sous les contraintes} \\ \text{1) } \textit{ineq}_k(e) \text{ pour tout } e \in E \\ \text{2) } \alpha \ .\ V_{o[Z]} \leq U_{o[Z]} \leq V_{o[Z]} \\ \qquad\qquad \text{pour tout } Z \subseteq X, o \in O_E \\ \text{3) } V_{o[Z]} \in \{0, 1\} \end{cases}$$

Puisque $V_{o[Z]} \in \{0,1\}$, les contraintes P2 impliquent que $U_{o[Z]} \in \{0\} \cup [\alpha, 1]$. Le minimum des $U_{o[Z]}$ pris sur l'ensemble des $U_{o[Z]}$ non nuls est alors α, et il suffit d'avoir $w_i > (i2^{i-1}/\alpha)w_{i-1}$ pour avoir un $U_{o[Z]}$ de degré i plus coûteux que l'ensemble des $U_{o[Z']}$ pour $Z' \subset Z$ et $|Z| = i-1$. Il faut choisir la constante σ qui apparaît dans certaines des $ineq_k(e)$ suffisamment petite afin que la restriction à l'intervalle $[0,1]$ des valeurs possibles pour les $U_{o[Z]}$ ne risque pas de supprimer artificiellement certaines solutions du problème.

Une autre solution consiste à minimiser directement la somme des V_z pondérés par les w_is afin d'apprendre une GAI-décomposition minimale au sens de la décomposition (définition 5.2). Ici V_z représente une GAI-table de Z et w_i représente le coût de la création d'une table de degré i. On a alors le programme linéaire mixte suivant, en choisissant les w_is tels que $w_k = k * w_{k-1} + 1$ avec $w_1 = 1$:

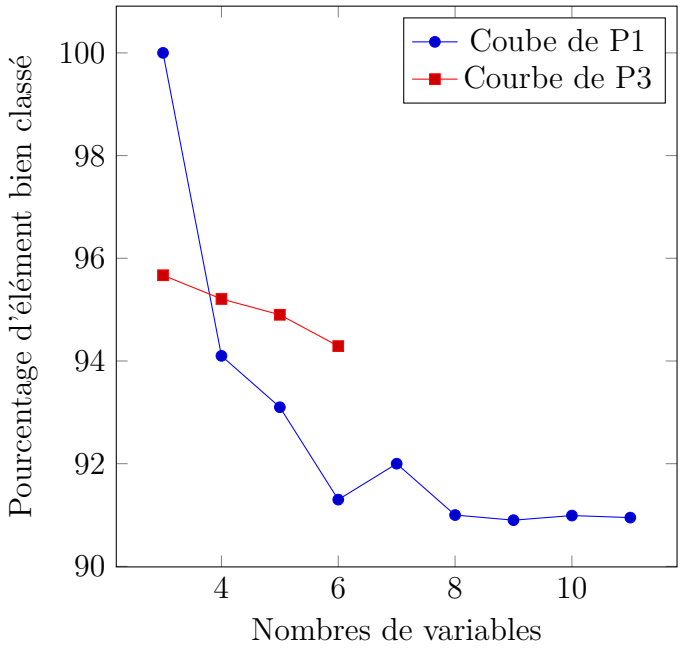
$$\text{(P3)} \begin{cases} \text{minimiser } (\sum_{Z \subseteq X} \text{ t.q. } {}_{|Z| \leq k} w_{|Z|} \times V_Z) \\ \text{sous les contraintes} \\ \bullet\ ineq_k(e) \text{ pour tout } e \in E \\ \bullet\ V_Z \geq U_{o[Z]} \geq 0 \\ \qquad\qquad \text{pour tout } Z \subseteq \chi, o \in O_E \\ \bullet\ V_Z \in \{0,1\} \end{cases}$$

5.4 Expérimentations

On s'est intéressé dans cette partie à apprendre des GAI à partir de la fonction objective P1 c'est-à-dire celle qui minimise la somme des utilités apprises. Puis on s'est intéressé à P3 qui apprend un GAI minimal au sens de la définition 5.2. Pour cela, on a généré aléatoirement des ordres sur les objets. Cette génération d'ordre a été effectuée par un générateur d'ordre aléatoire développé par Nicolas Schmidt [64] lors de son stage de M2R effectué à l'Irit. Une fois l'ordre généré, on a tiré deux ensembles d'exemples du type (o, R, o') l'un pour l'apprentissage E_a dont taille dépend du résultat de la PAC-apprenabilité à savoir $0.21n^2$, l'autre pour tester notre GAI apprit E_t de taille 50. Puis on a généré le système d'équations linéaires à partir de E_a que l'on a résolu grâce à Cplex. Pour ces résultats, nous avons configuré ϵ à 0.95 et δ à 0.5, ϵ et δ étant les paramètres de la PAC-apprenabilité. De plus, on apprend ici des GAI de degrés au plus 2.

La figures 5.1 nous montre la moyenne du nombre d'exemples E_t bien classés sur 100 exécutions en fonction du nombre de variables. On voit sur ces graphes

Figure 5.1 – *Pourcentage d'objets bien classé*



que le GAI apprit arrive à classer correctement l'ensemble d'objets qui lui est présenté. On voit que pour la courbe P1. Le pourcentage se stabilise vers 92% à partir de 6 variables. Pour la courbe P3, on voit qu'elle classe globalement mieux que P1 mais la complexité de ce système l'empêche de dépasser 6 variables.

De plus, la figures 5.2 nous montre le nombre d'exemples nécessaires pour apprendre un GAI, qui classe correctement 90% des exemples, est en moyenne de 90% du nombre d'exemples théoriques $|E_a|$. En ce qui concerne la courbe P3, on voit que le nombre d'exemples nécessaires explose par rapport au nombre d'exemples théoriques ce qui montre qu'il est extrêmement difficile d'apprendre un GAI avec la notion de minimalité de la définition 5.2.

Enfin les figures 5.3, nous donnent le temps d'exécution moyen de la phase d'apprentissage. On n'a pas pu tester l'apprentissage sur plus de 11 variables étant donné que la génération complète d'un d'ordre sur un domaine combinatoire est longue et difficile. De ce fait, le générateur à notre disposition s'arrête à 11 variables.

Avec ces différentes fonctions objectives, les résultats théoriques sur la PAC-apprenabilité indiquent qu'il est difficile d'apprendre un GAI dès que le degré augmente. En effet, pour 5 variables apprendre un GAI de degrés au plus 4

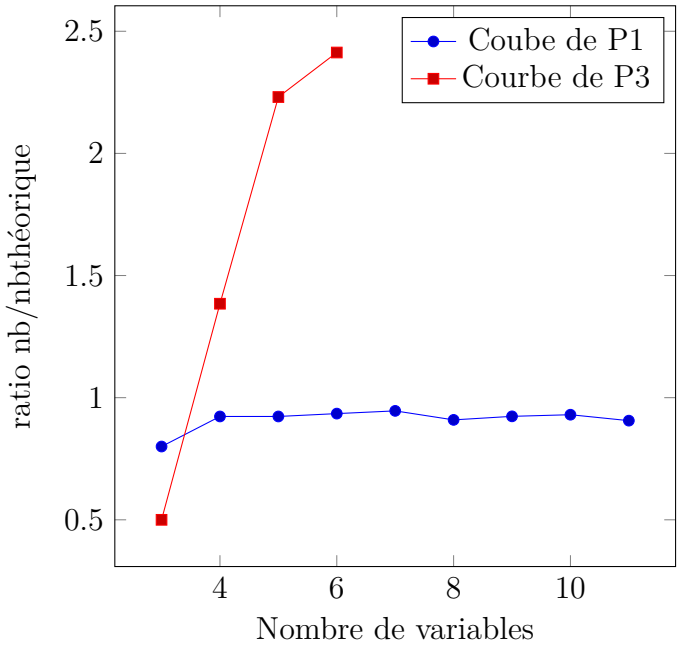


Figure 5.2 – *Ratio nombre d'exemples sur nombre d'exemples théoriques*

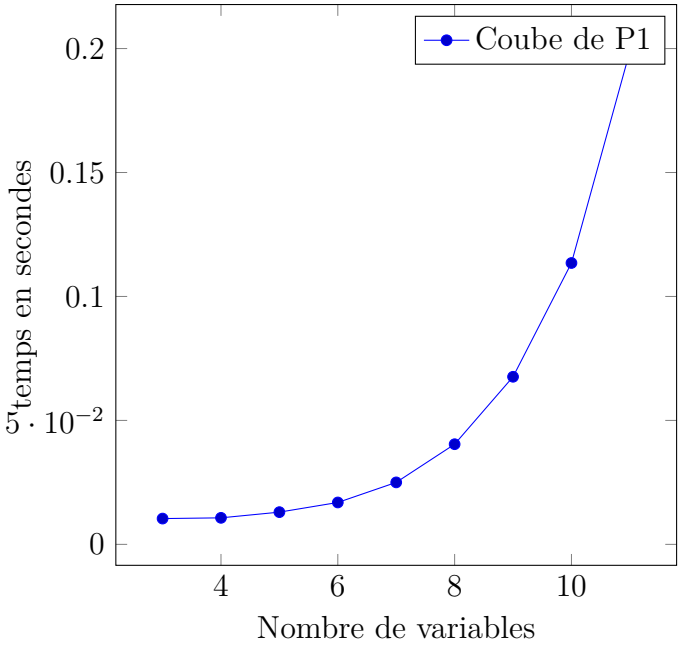


Figure 5.3 – *Moyenne du temps d'apprentissage sur 500 exécutions*

nécessite 6717 exemples en théorie donc la création d'un nombre de variables énormes lors de la génération du système linéaire. Pour pallier ce problème, nous pouvons imaginer apprendre un GAI dont toutes les clauses ont une taille de k. Cela permet ainsi d'apprendre des GAI plus importants même s'ils ne sont pas minimaux au sens de la définition 5.2.

5.5 Conclusion

Nous nous sommes intéressés dans ce chapitre aux GAI-décompositions. Nous avons d'abord commencé à les présenter. Puis nous avons parlé des travaux déjà existants avec cette représentation. Ensuite, nous avons montré que tout préordre complet sur un ensemble d'objets combinatoires représentable par une GAI-décomposition de degré k peut également être encodé par un système d'inégalités linéaires, dont les solutions correspondent essentiellement aux GAI-décompositions de la relation initiale. Le principe de notre algorithme est de générer, étant donné un ensemble d'exemples et un degré k fixé, un système linéaire, dont les variables sont les valeurs des GAI-tables recherchées. Le choix d'une fonction objectif pertinente permet alors d'utiliser un algorithme de programmation linéaire (éventuellement mixte) pour générer une GAI-décomposition qui soit de degré ou de taille minimale.

Nous avons supposé que toutes les variables sont binaires, mais les résultats se généralisent aisément dans le cas où on a des variables dont les domaines ont au plus d valeurs, pour un certain d fixé : on vérifie que la VC-dimension de la classe de GAI correspondante est bornée par $d^k n^{k+1}$.

Dans la version de l'algorithme que nous proposons dans ce chapitre, nous cherchons à couvrir tous les exemples. Si cela n'est pas possible avec le degré choisi, ou si l'ensemble d'exemples n'est pas sain, le système linéaire généré n'a pas de solution. Il est possible de relaxer cette exigence en introduisant dans les parties gauches des inégalités des variables de relaxation ζ, dont la somme est à minimiser (la version courante correspondant à des ζ nuls).

Le second développement nécessaire de ce travail consiste à établir une stratégie intelligente de choix du degré du GAI à apprendre. On peut procéder de manière croissante (supposer les variables indépendantes, puis augmenter incrémentalement le degré jusqu'à obtenir une bonne couverture des exemples). Une stratégie plus fine consisterait à analyser le graphe appris pour $k = 2$, afin d'augmenter notre connaissance sur les dépendances de variables (une clique sur un ensemble de variables Y induisant la nécessité d'une utilité locale u_Y). On pourrait

plus généralement faire collaborer une procédure d'apprentissage (actif ou passif) du degré du GAI, ou mieux, de sa structure, et une procédure d'apprentissage des tables fondées sur leur représentation par un système d'équations linéaires.

Enfin, Y. Chevaleyre [19] a proposé lors de ses travaux d'utiliser les noyaux afin de réduire la taille des inéquations linéaires construites et donc de réduire ainsi la complexité du problème.

Nous avons donc montré dans ce chapitre qu'il était possible d'apprendre une GAI-décomposition de manière passive malgré le fait que cet apprentissage reste difficile. Par la suite, nous nous sommes intéressés dans cette thèse à la représentation des préférences dans les cadres multi-agents ou bien lorsque ces préférences sont changeantes à cause de conditions inconnues. Pour cela, nous nous sommes intéressés aux CP-nets et avons proposé une extension de ceux-ci : les CP-nets probabilistes.

Troisième partie

CP-nets probabilistes

Chapitre 6

Introduction aux CP-nets probabilistes

Sommaire

6.1 Introduction

Comme nous l'avons dit au tout début de cette thèse, lors d'un problème de recommandation ou de configuration de produits avec un utilisateur, il est nécessaire de connaître les préférences de celui-ci. De plus, ces préférences peuvent être mal connues ou encore changeantes au cours du temps ou à cause de variables d'environnement qui ne sont pas toujours connues. Enfin, apprendre ces préférences prend du temps qui, associé avec le temps de la recommandation / configuration, peut être décourageant pour l'utilisateur. Afin de pallier ces problèmes, nous proposons non pas de représenter un ordre qui représente les préférences d'un utilisateur mais de représenter une distribution de probabilité sur des ordres permettant ainsi de représenter soit des préférences mal connues d'un utilisateur soit les préférences d'un ensemble d'utilisateurs. Plus précisément, nous proposons une extension des préférences conditionnelles [14] en ajoutant des probabilité aux règles de préférence locale [6]. La notion de CP-net probabiliste a été introduite par Amo *et al.* [1], sans que les auteurs ne s'intéressent ni à la sémantique ni à leurs propriétés de calcul. Une forme plus générale des PCP-nets que la notre a aussi été proposée indépendamment par Cornelio [21] qui prouve que trouver l'objet qui a la plus grande probabilité d'être optimal est équivalent à un problème d'optimisation dans un réseau bayésien.

Dans ce chapitre, nous commencerons à présenter les CP-nets probabilistes en donnant tout d'abord les motivations de cette représentation, puis la sémantique de celle-ci et enfin les différents raisonnements possibles. Puis, nous nous intéresserons à l'algorithmique et à la complexité des requêtes. Dans la section sur le test de dominance 6.3, nous montrerons que la complexité de la requête de dominance est un problème $\#P$-difficile dans le cas des PCP-nets acycliques. Ensuite, nous montrerons comment transformer une expression de dominance en formule logique dans le cas des PCP-nets arborescents. Puis nous donnerons un algorithme FPT pour résoudre la requête de dominance dans le cas des PCP-nets arborescents. Enfin, nous dériverons un résultat pour le test de dominance pour les CP-nets. Finalement, dans la section optimisation 6.4 nous parlerons de la complexité de celle-ci puis nous donnerons un algorithme polynomial permettant de résoudre ce problème dans le cas de PCP-nets arborescents.

6.2 Présentation

Il arrive souvent, lorsque l'on cherche à effectuer de la recommandation ou à configurer un produit avec un utilisateur, que la relation de préférence de cet utilisateur soit mal connues. Cela peut-être dû à un manque de temps ou d'informations lors de la phase d'apprentissage de ses préférences ou encore parce que certaines relations de préférence dépendent de la valeur d'un ensemble de variables d'état que l'on ne contrôle pas ou qui sont inconnues. Dans ces conditions, on souhaite tout de même pouvoir répondre à certaines requêtes afin de faciliter la recommandation ou la configuration comme : " Quelle est la probabilité qu'un objet o soit préférable à un objet o' ? ". Dans ce contexte, les CP-nets probabilistes permettent de représenter de façon compacte une distribution de probabilité sur des CP-nets afin de répondre à de telles requêtes. Dans le cadre de la recommandation, les préférences d'un utilisateur anonyme seront extrapolées à partir des informations partielles recueillies sur cet individu ou encore à partir des renseignements recueillis sur d'anciens clients afin d'estimer la probabilité qu'un nouveau client fasse un choix donné.

6.2.1 Sémantique

Définition 6.1. *Un réseau de préférences conditionnelles probabilistes $\mathcal{N}$, ou PCP-net, sur un ensemble de variables χ, est défini par un graphe orienté $G = (\chi, E)$ et, pour chaque sommet $X \in \chi$, une table de préférences conditionnelles probabilistes, écrite PCP-Table(X). Une PCP-table sur X donne, pour chaque instanciation $u \in \underline{pa(X)}$, une distribution de probabilité sur l'ensemble des ordres totaux sur $\underline{X}$. On note $p^u_{\mathcal{N},X}$ pour cette distribution. Nous appelons également G la structure de $\mathcal{N}$.*

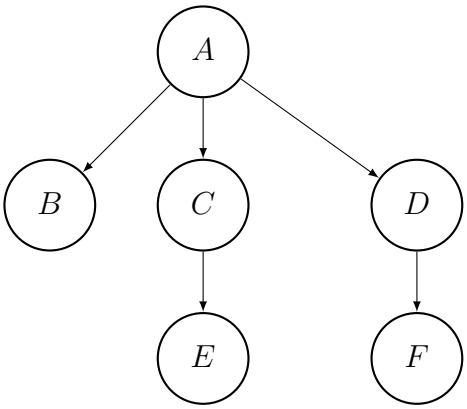
En particulier, lorsque toutes les variables sont binaires, une PCP-table sur X donne pour chaque instanciation $u \in \underline{pa(X)}$ une distribution de probabilité sur l'ensemble des deux ordres $\{x > \bar{x}, \bar{x} > x\}$. Par souci de concision, on écrit $u : x > \bar{x}$ (p) pour la distribution qui attribue une probabilité p à la préférence locale $u : x > \bar{x}$ et 1-p à la préférence locale $u : \bar{x} > x$.

Pris dans son ensemble, un PCP-net $\mathcal{N}$ n'est pas destiné à représenter une relation de préférence. Il s'agit plutôt d'une distribution de probabilité sur un ensemble de CP-nets (déterministes), à savoir ceux qui sont compatibles avec $\mathcal{N}$.

Définition 6.2. *Un CP-net (déterministe) N est dit compatible avec un PCP-net $\mathcal{N}$, ou $\mathcal{N}$-compatible, si il a la même structure que $\mathcal{N}$. Dans ce cas, on écrira $N \propto$*

$\mathcal{N}$. Si N est $\mathcal{N}$-compatible, on définit la probabilité de N selon $\mathcal{N}$ par $p_{\mathcal{N}}(N) = \prod_{X \in V, u \in pa(X)} p^u_{\mathcal{N},X}(>^u_{N,X})$ où $(>^u_{N,X})$ est la relation de préférence sur X sachant u pour le CP-net N.

On en déduit facilement que $p_{\mathcal{N}}$ est une distribution de probabilité sur l'ensemble des CP-nets déterministes $\mathcal{N}$-compatibles.



A	$a > \bar{a}$ 0.9
B	$a : b > \bar{b}$ 0.2
B	$\bar{a} : b > \bar{b}$ 0.7
C	$a : c > \bar{c}$ 0.9
C	$\bar{a} : c > \bar{c}$ 0.7
D	$a : d > \bar{d}$ 0.8
D	$\bar{a} : d > \bar{d}$ 0.5
E	$c : e > \bar{e}$ 1
E	$\bar{c} : e > \bar{e}$ 0.1
F	$d : f > \bar{f}$ 0
F	$\bar{d} : f > \bar{f}$ 0.8

Figure 6.1 – *Exemple d'un PCP-net arborescent à 6 variables*

Exemple 6.1. *La figure 6.1 nous montre un PCP-net sur l'ensemble de variables $\{A, B, C, D, E, F\}$. Par exemple, la première règle sur D nous dit que si l'objet possède la valeur a, on aura une probabilité de 0.8 que l'on préfère la valeur $\bar{d}$ à la valeur d, c'est à dire qu'on a aussi une probabilité de 0.2 que ce soit d qui soit préféré à $\bar{d}$. De façon indépendante, il y a une probabilité de 0.5 d'avoir la règle $\bar{a} : d > \bar{d}$. On aura donc une probabilité de $0.8 \times 0.5 = 0.4$ d'avoir les deux règles dans un CP-net.*

Enfin , le CP-net $\mathcal{N}$-compatible qui a le plus de chance d'apparaître aura une probabilité de :

$$\begin{aligned} p = p(a > \bar{a}) \times (1 - p(a : b > \bar{b})) \times p(\bar{a} : b > \bar{b}) \times p(a : c > \bar{c}) \times p(\bar{a} : c > \bar{c}) \\ \times p(a : d > \bar{d}) \times p(\bar{a} : d > \bar{d}) \times p(c : e > \bar{e}) \times (1 - p(\bar{c} : e > \bar{e})) \\ \times (1 - p(d : f > \bar{f})) \times p(\bar{d} : f > \bar{f}) \end{aligned}$$

$$p = 0.9 \times (1-0.2) \times 0.7 \times 0.9 \times 0.7 \times 0.8 \times 0.5 \times 1 \times (1-0.1) \times (1-0) \times 0.8 = 0.09$$

Lorsque la structure de $\mathcal{N}$ est cyclique, il peut arriver que $p_{\mathcal{N}}$ contiennent des CP-nets inconsistant qui possèdent quand même une probabilité non nulle, ce qui n'est pas désirable. De plus, décider si un CP-net cyclique est inconsistant est un problème PSPACE-difficile [38], alors que ce problème se résout en temps polynomiale dans la classe des CP-nets acycliques. Le reste de cette thèse ne traitera donc que des structures acycliques.

6.2.2 Motivation

La motivation d'étudier les PCP-nets découle directement de plusieurs cas applicatifs différents. Le premier cas est celui où les relations de préférence d'un utilisateur sont inconnues, mais où le système connaît les relations de préférence d'un ensemble d'individus m proches de notre utilisateur, par exemple d'anciens clients. Pour tous ces anciens clients, les relations de préférence sont représentées par des CP-nets N_i ayant tous une structure commune G. Dans ce cas, un CP-net probabiliste $\mathcal{N}$ est défini par les probabilités $p^u_{\mathcal{N},X} = |\{i|((X, u :>) \in N_i)\}|/m$ (c'est la proportion de CP-nets N_i qui contiennent cette règle, indépendamment de toutes les autres règles) et fournit une représentation compacte de l'ensemble des préférences individuelles.

Il est évident que ce genre d'agrégation introduit une certaine approximation de la véritable distribution des préférences de la population. Notamment, la probabilité d'avoir un CP-net N donné calculée à partir d'un PCP-net $\mathcal{N}$ est souvent différente de la véritable proportion de personnes qui ont donné leur relation de préférence. Plus précisément, lors de l'agrégation des CP-nets on fait l'hypothèse forte que les relations de préférence sont indépendantes, entraînant une approximation sur la probabilité des règles. Par exemple, si dans un groupe de 100 personnes, on a 50 personnes qui préfèrent la valeur a à $\bar{a}$ et la valeur b à $\bar{b}$, et que les 50 autres personnes ont des préférences opposées on aura alors un CP-net probabiliste avec la règle $a > \bar{a}$ 0.5 et $b > \bar{b}$ 0.5. Dans cette situation, il sera possible de trouver des CP-nets qui ne reflètent pas les préférences du groupe, par exemple a est préféré à $\bar{a}$ et $\bar{b}$ est préféré à b.

Un second cadre dans lequel les PCP-nets trouvent leur utilité est celui d'un système qui interagit avec un grand nombre d'individus tel que chaque individu donne seulement quelques relations de préférence. Par exemple, lorsqu'un utilisateur compare deux objets qui ne diffèrent que d'un seul flip, celui-ci nous donne implicitement sa préférence sous la forme $u : x > \bar{x}$. Ce cas se retrouve souvent dans des cas réels notamment lors de la configuration de produit [37]. Par exemple lors de la configuration interactive d'une voiture, lorsque l'utilisa-

teur choisira la couleur de sa voiture, ses préférences s'exprimeront sous la forme $u : x > \bar{x}$, où u est l'ensemble des variables qu'il a déjà configuré et X est la couleur choisie. Ces règles obtenues à partir des différents individus, et en absence d'autre information, peuvent être agrégées de façon indépendante afin de faciliter la recommandation pour des clients futurs.

Enfin, un troisième cadre dans lequel on peut utiliser les PCP-nets est celui où un agent exprime ses préférences, mais dans lequel ses préférences peuvent être changeantes. Ces changements peuvent être dus à un ensemble de variables exogènes, c'est-à-dire des variables non contrôlables ou encore inconnues à un instant donné (le temps, le jour), ou encore à des événements particuliers. Par exemple on peut préférer la viande au poisson mais avoir envie de changer de temps en temps.

Dans la suite de ce chapitre, nous nous intéresserons aux différentes requêtes possibles avec les PCP-nets. Nous verrons aussi que raisonner avec eux peut poser des problèmes de calcul non triviaux même en se restreignant à des PCP-nets de structures simples, comme les PCP-nets arborescents.

6.2.3 Raisonnement sur les requêtes

Étant donné qu'un PCP-net représente une distribution de probabilités sur des CP-nets, deux probabilités naturelles peuvent être définies.

Définition 6.3. *Étant donné un PCP-net $\mathcal{N}$ et deux objets o et o', la probabilité que $o \succ o'$, notée $p_{\mathcal{N}}(o \succ o')$, est définie par la somme des probabilités des CP-nets $\mathcal{N}$-compatibles qui satisfont $o \succ o'$.*

$$p_{\mathcal{N}}(o \succ o') = \sum_{N \propto \mathcal{N}, N \models o \succ o'} p_{\mathcal{N}}(N)$$

Clairement, la probabilité de $o \succ o'$ étant donné $\mathcal{N}$ est précisément la probabilité, lors du tirage d'un CP-net N au hasard selon $p_{\mathcal{N}}$, d'obtenir $o \succ_{\mathcal{N}} o'$.

La seconde requête consiste à calculer la probabilité qu'un objet o donné soit optimal.

Définition 6.4. *Étant donné un PCP-net acyclique $\mathcal{N}$ et un objet o, la probabilité que o soit optimal, notée $p_{\mathcal{N}}(o)$, est définie comme étant la somme des probabilités des CP-nets $\mathcal{N}$-compatibles pour lesquels o est optimal.*

Fait intéressant, malgré l'approximation importante induite par l'agrégation d'un ensemble de CP-nets en un PCP-net, certaines probabilité peuvent être calculées de façon efficace avec cette approximation.

Par ailleurs, nous insistons sur le fait que les PCP-nets peuvent servir à d'autres fins que l'agrégation des préférences, comme par exemple la modélisation des préférences mal connues d'un seul utilisateur, et que dans ce cas cette approximation se produit aussi.

Proposition 6.1. *Soit un PCP-net acyclique $\mathcal{N}$ et une paire d'objets $\{o, o'\}$ qui diffèrent seulement sur la valeur de X. La probabilité $p_{\mathcal{N}}(o, o')$ est précisément la proportion de CP-nets qui impliquent $(X, u : o[X] > o'[X])$.*

Démonstration. Cette propriété découle directement du fait qu'un CP-net déterministe N implique $o \succ o'$ si et seulement si il contient la règle o[pa(X)] : o[X] >o'[X] [Koriche et Zanuttini, 2009, Lemme 1] □

Une autre propriété intéressante est la préservation des gagnants de Condorcet locaux [Xia et al. 2008, Li et al., 2011], aussi appelés "hypercubewise Condorcet winner" par Conitzer et al. [2011] : ce sont les objets o qui sont préférés à tous leurs voisins, c'est-à-dire tous les objets qui diffèrent de o par un seul flip, par au moins la moitié des CP-nets déterministes. La proposition 6.1 prouve que les gagnants de Condorcet locaux sont les objets qui dominent chacun de leurs voisins dans le PCP-net cumulé avec une probabilité d'au moins 0,5.

6.3 Complexité de la requête de dominance

Nous étudions maintenant dans cette partie la complexité du problème du calcul de la probabilité d'avoir $o \succ o'$ étant donné un PCP-net $\mathcal{N}$. Nous limitons principalement nos recherche aux PCP-nets arborescents. Nous montrerons dans un premier temps comment transformer le calcul du test de dominance en une requête de satisfaction d'une formule logique. Puis, grâce à cette formule, nous montrerons comment calculer de manière efficace la probabilité de dominance entre deux objets dans le cas des PCP-nets arborescents. A partir de ces résultats, nous donnerons deux résultats dérivés dans le cadre des CP-nets. Le premier permet de réduire la complexité du test de dominance pour les arbres. Le second résultat propose un algorithme d'apprentissage passif qui permet d'apprendre des CP-nets arborescents lorsque la structure est donnée. Enfin, nous étendrons ces résultats en donnant la complexité du test de dominance dans le cas des PCP-nets acycliques.

6.3.1 Expression de la dominance en formule logique pour les arbres

Le composante fondamentale de nos résultats est une caractérisation de tous les CP-nets de structure arborescente G donnée pour lesquels il est possible de dégrader o en o'. Cette caractérisation est donnée par une formule propositionnelle pour chaque feuille G, notée $worsen_{o,o'}(X)$. Les variables sont les règles possibles $y : x > \bar{x}, \bar{y} : x > \bar{x}$, etc telle que Y est le parent de X dans G. Une assignation de $y : x > \bar{x}$ à $\top$ signifie que le CP-net correspondant contient cette règle. Une instanciation complète de toutes les variables définit ainsi un CP-net déterministe avec la structure G.

Plus précisément, étant donné une forêt $G = (\chi, E)$ et deux objets (o, o'), pour chaque variable X qui ne possède pas de parents dans G (pour chaque racine), on introduit une variable propositionnelle $x > \bar{x}$ et on note sa négation $\bar{x} > x$ (étant donné que $>$ est total, on a $x > \bar{x}$ à vrai si et seulement si $\bar{x} > x$ est faux). De façon similaire, pour chaque variable X avec $Pa(X) = \{Y\}$, on introduit les variables propositionnelles $y : x > \bar{x}$ et $\bar{y} : x > \bar{x}$ et on note respectivement $y : \bar{x} > x$ et $\bar{y} : \bar{x} > x$ leurs négations.

Boutilier et al. [14] montrent qu'une détérioration d'un objet o en un objet o' peut comprendre jusqu'à n changements de la valeur d'une variable, même dans le cas des CP-nets arborescents binaires. Nous exploitons ce résultat dans la construction de notre formule.

Précisément, la formule $change_k^{o,o'}(X)$ signifie qu'il existe une séquence de flips dans laquelle la variable X change de valeur au moins k fois, à partir de sa valeur en o et jusqu'à sa valeur en o'. Par exemple, $change_3^{x\ldots,\bar{x}\ldots}(X)$ signifie qu'il existe une séquence de flips dans laquelle X prend successivement les valeurs $x, \bar{x}, x, \bar{x}$ (au moins 4 valeurs et trois alternances). La formule $change_k^{o,o'}(X)$ est définie par induction dans le tableau 6.1, où Y désigne le parent de X. Nous donnons les formules pour le cas où o[X]=x, o[Y]=y, les autres cas peuvent être obtenus par symétrie. Ensuite, $worsen^{o,o'}(X)$ est défini comme suit :

- $worsen^{o,o'}(X) = change_1^{o,o'}(X)$ si o[X]=o'[X].
- $worsen^{o,o'}(X) = change_0^{o,o'}(X)$ sinon.

Dans la suite, on écrira $o[\geq X]$ pour o restreint aux variables qui sont les ascendants de X dans G (X inclus).

Proposition 6.2. *Soit un CP-net N de structure arborescente G. Soient deux objets o et o'. Il existe une séquence de flips de* $o[\geq X]$ *à* $o'[\geq X]$ *dans la-*

cas de base ($\mathbf{Pa(X)} = \emptyset$ ou $\mathbf{k} \leq 1$)			
$\mathbf{Pa(X)}$	$\mathbf{o}, \mathbf{o'}$	$\mathbf{k}$	$\mathsf{change}_{\mathbf{k}}^{\mathbf{o},\mathbf{o}'}(\mathbf{X})$
$\emptyset$	$o[X] = o'[X]$	0	$\top$
$\emptyset$	$o[X] = o'[X]$	> 0	$\bot$
$\emptyset$	$o[X] = x, o'[X] = \bar{x}$	0	$\mathsf{change}_1^{o,o'}(X)$
$\emptyset$	$o[X] = x, o'[X] = \bar{x}$	1	$x > \bar{x}$
$\emptyset$	$o[X] = x, o'[X] = \bar{x}$	> 1	$\bot$
$\{Y\}$	$o[X] = o'[X]$	0	$\mathsf{change}_0^{o,o'}(Y)$
$\{Y\}$	$o[X] \neq o'[X]$	0	$\mathsf{change}_1^{o,o'}(X)$
$\{Y\}$	$o[X] = o'[X]$	1	$\mathsf{change}_2^{o,o'}(X)$
$\{Y\}$	$o[X] = x, o'[X] = \bar{x}, o[Y] = o'[Y] = y$	1	$(y : x > \bar{x} \wedge \mathsf{change}_0^{o,o'}(Y)) \vee (\bar{y} : x > \bar{x} \wedge \mathsf{change}_2^{o,o'}(Y))$
$\{Y\}$	$o[X] = x, o'[X] = \bar{x}, o[Y] = y, o'[Y] = \bar{y}$	1	$(y : x > \bar{x} \vee \bar{y} : x > \bar{x}) \wedge \mathsf{change}_1^{o,o'}(Y)$

Etape inductive ($\mathbf{Pa(X)} \neq \emptyset$ et $\mathbf{k} > 1$)			
Rule	$\mathbf{k}$	$\mathbf{o[X], o'[X], o[Y], o'[Y]}$	$\mathsf{change}_{\mathbf{k}}^{\mathbf{o},\mathbf{o}'}(\mathbf{X})$
0	odd	$x, x,$ indifferent, indiff.	$\mathsf{change}_{k+1}^{o,o'}(X)$
1	even	x, x, y, y	$((y : x > \bar{x} \wedge \bar{y} : \bar{x} > x) \vee (y : \bar{x} > x \wedge \bar{y} : x > \bar{x})) \wedge \mathsf{change}_k^{o,o'}(Y)$
2	even	$x, x, y, \bar{y}$	$(y : x > \bar{x} \wedge \bar{y} : \bar{x} > x \wedge \mathsf{change}_{k-1}^{o,o'}(Y)) \vee (y : \bar{x} > x \wedge \bar{y} : x > \bar{x} \wedge \mathsf{change}_{k+1}^{o,o'}(Y)$
3	even	$x, \bar{x},$ indifferent, indiff.	$\mathsf{change}_{k+1}^{o,o'}(X)$
4	odd	$x, \bar{x}, y, y$	$(y : x > \bar{x} \wedge \bar{y} : \bar{x} > x \wedge \mathsf{change}_{k-1}^{o,o'}(Y)) \vee (y : \bar{x} > x \wedge \bar{y} : x > \bar{x} \wedge \mathsf{change}_{k+1}^{o,o'}(Y)$
5	odd	$x, \bar{x}, y, \bar{y}$	$((y : x > \bar{x} \wedge \bar{y} : \bar{x} > x) \vee (y : \bar{x} > x \wedge \bar{y} : x > \bar{x})) \wedge \mathsf{change}_k^{o,o'}(Y)$

Table 6.1 – *Définition récursive de la formule* $\mathsf{change}_k^{o,o'}(X)$ *pour une structure donnée arborescente* G

quelle X change de valeur au moins k fois si et seulement si N est un modèle de $change_k^{o,o'}(X)$.

Démonstration. La preuve est faite par induction sur le parent de X à partir de la définition de la formule.

Dans le cas de base où X est une racine, si X à la même valeur dans o et o' alors la formule sera toujours vraie si $k = 0$ (on n'a aucun changement à faire) et sera toujours fausse si $k > 0$ (comme X est une racine elle ne possède qu'une règle elle ne pourra donc pas faire la séquence $x\bar{x}x$). Dans le cas où X a une valeur différente pour o et o', la formule ne peut être vraie que dans le cas où $k = 1$ et où le CP-net possède la règle $o[X] > o'[X]$.

Pour l'étape d'induction , on commence par donner la preuve pour la première règle, c'est-à-dire $o[X] = o'[X] = x$ et $o[Y] = o'[Y] = y$. Les autres règles peuvent être prouvées exactement de la même façon. On suppose tout d'abord que N satisfait la formule de la règle 1 et donc N satisfait $change_k^{o,o'}(Y)$. Par hypothèse d'induction, il y a une séquence détériorante de la forme :

$$w_1 y, w_2 \bar{y}, \ldots, w_k \bar{y}, w_{k+1} y$$

où les w_i sont des instanciations des ascendants de Y et telle que $o[\geq Y] = w_1 y$ et $o'[\geq Y] = w_{k+1} y$. De plus, si N satisfait $y : x > \bar{x} \wedge \bar{y} : \bar{x} > x$, comme la valeur de X n'influe pas sur les préférences sur Y, on peut construire la séquence suivante :

$$w_1 yx > w_1 y\bar{x} > w_2 \bar{y}\bar{x} > w_2 \bar{y}x > \cdots > w_k \bar{y}x > w_k \bar{y}\bar{x} > w_{k+1} y\bar{x} > w_{k+1} yx$$

qui est une séquence de flips de $o[\geq X]$ à $o[\geq X]$, où X change de valeur k+1 fois. De même, si N satisfait $(y : \bar{x} > x \wedge \bar{y} : x > \bar{x})$ on peut construire la séquence suivante où X change de valeur k fois.

$$w_1 yx > w_2 \bar{y}x > w_2 \bar{y}\bar{x} > \cdots > w_k \bar{y}x > w_k \bar{y}\bar{x} > w_{k+1} y\bar{x} > w_{k+1} yx$$

Réciproquement, on montre que s'il existe une séquence de flips, alors N satisfait la formule de la règle 1. Soit :

$$w_1 x, w_2 \bar{x}, \ldots, w_k \bar{x}, w_{k+1} x$$

une séquence de $o[\geq X]$ à $o'[\geq X]$ dans laquelle X change de valeur au moins $k \geq 2$ fois. On a forcément deux règles opposées sur X, sinon X ne peut pas

alterner de valeur, d'où la disjonction dans la définition de $change_k^{o,o'}(X)$. En outre, ces règles doivent changer de valeur au moins k fois, donc Y doit prendre au moins k valeurs différentes dans la séquence $w_1, w_2, \ldots, w_k, w_{k+1}$, qui doit changer au moins k-1 fois. Mais étant donné que Y doit se terminer par la même valeur et que k-1 est impaire, on doit le changer au moins k fois. Ainsi, par hypothèse d'induction N doit aussi satisfaire $change_k^{o,o'}(Y)$

□

Proposition 6.3. *Soit un CP-net N de structure arborescente G. Soit deux objets o et o'. Il existe une séquence de flips de o vers o' si et seulement si N satisfait la formule* $\bigwedge_X worsen^{o,o'}(X)$*, où X parcourt toute les feuilles de G.*

Démonstration. Dans le cas où G est une chaîne, la proposition 6.2 suffit à prouver la proposition 6.3. Dans un cas plus général, si on a deux branches avec un sous-arbre commun au-dessus de X (inclus), on note $\mathcal{G}'$ l'ensemble des variables au-dessus de X (inclus), $\mathcal{Y}$ pour la chaine en dessous de X avec Y la feuille de $\mathcal{Y}$, et $\mathcal{Z}$ la chaine en dessous de X avec Z la feuille de $\mathcal{Z}$.

S'il y a une séquence de flips de o à o', alors N doit satisfaire les formules $worsen^{o,o'}(Y)$ et $worsen^{o,o'}(Z)$ pour les deux branches (Proposition 6.2).

Réciproquement, si N satisfait les deux formules, alors selon la proposition 6.2 il y a une séquence de flip de l'objet $o[\geq Y] = o[\mathcal{X}]o[\mathcal{Y}]$ à l'objet $o'[\geq Y] = o'[\mathcal{X}]o'[\mathcal{Y}]$ ainsi qu'une séquence de $o[\geq Z] = o[\mathcal{X}]o[\mathcal{Z}]$ à $o'[\geq Z] = o'[\mathcal{X}]o'[\mathcal{Z}]$.

Ainsi, par construction de la formule $worsen^{o,o'}(.)$, il existe une de ces séquences dans laquelle les valeurs des variables au dessus de X change le plus, par exemple $\mathcal{Y}$. Ensuite, étant donné que $\mathcal{Y}$ et $\mathcal{Z}$ sont indépendants l'un de l'autre, tous les flips sur $\mathcal{Z}$ peuvent être effectués avant ou de façon entrelacée avec ceux de $\mathcal{Y}$. De cette manière, on obtient une séquence détériorante de o à o' comme on le souhaite.

□

Exemple 6.2. *Soit le PCP-net de la Figure 6.1 et soient deux objets* $o = abcdef$ *et* $o' = \bar{a}bc\bar{d}\bar{f}$*. La formule correspondante est donnée par la table 6.2.*

6.3.2 Un algorithme FPT pour les PCP-nets arborescents.

A partir des propositions 6.2 et 6.3, on en déduit un algorithme "fixed-parameter tractable" (FPT) pour le test de dominance dans le cas des PCP-nets arborescents. Un algorithme FTP est un algorithme qui tourne en $O(f(k).n^c)$, où n est la

$$
\begin{aligned}
\mathsf{worsen}^{o,o'}(F) &= (d : f > \bar{f} \wedge \mathsf{change}_1^{o,o'}(D)) \vee \\
&\quad (\bar{d} : f > \bar{f} \wedge \mathsf{change}_1^{o,o'}(D) \\
\mathsf{change}_1^{o,o'}(D) &= (a : d > \bar{d} \wedge \mathsf{change}_1^{o,o'}(A)) \vee \\
&\quad (\bar{a} : d > \bar{d} \wedge \mathsf{change}_1^{o,o'}(A) \\
\mathsf{worsen}^{o,o'}(E) &= \mathsf{worsen}^{o,o'}(C) \\
\mathsf{worsen}^{o,o'}(C) &= \mathsf{change}_1^{o,o'}(A) \\
\mathsf{worsen}^{o,o'}(B) &= \mathsf{change}_1^{o,o'}(A) \\
\mathsf{change}_1^{o,o'}(A) &= a > \bar{a}
\end{aligned}
$$

Table 6.2 – *Formule de la Figure 6.1 sur les objets* $o = abcdef$ *et* $o' = \bar{a}bc\bar{d}e\bar{f}$

taille de l'entrée, c est une constante, f est une fonction calculable et k est une mesure autre que l'entrée, nommée paramètre et supposée petite [31]. Ainsi, le temps d'exécution de ce type d'algorithme est essentiellement un polynôme modulo un facteur qui peut être exponentiel par rapport à la valeur du paramètre.

Dans le cas du problème de dominance pour les PCP-nets arborescents, nous avons choisi comme paramètre k le nombre de variables qui ont une valeur différente dans les objets à comparer o et o'. Ce choix nous semble logique car généralement, on compare rarement des objets qui sont complètement différents les uns des autres. Par exemple, dans un système de recommandation, la comparaison entre certains produits se fait lorsque certaines variables sont déjà instanciées. Cette comparaison est donc susceptible de se faire sur des objets assez proches (je souhaite une voiture 5 portes avec un moteur diesel, un grand coffre et un GPS. On me proposera donc des voitures possédants ces caractéristiques et qui différencient sur quelques attributs comme la couleur, la radio ou la marque de la voiture) .

Définition 6.5. *Le problème de dominance paramétrée pour les PCP-nets arborescents, écrit p-arborescente-PDominance, est résolu par l'algorithme 11*

Théorème 6.1. *Le problème de dominance paramétrée pour les PCP-nets arborescents est un problème "fixed-parameter tractable". Plus exactement, il existe un algorithme qui tourne en* $O(2^{2k^2}n)$ *et qui résout ce problème.*

Démonstration. Pour chaque variable feuille X dans l'arbre de $\mathcal{N}$, l'algorithme commence d'abord par la formule $worsen^{o,o'}(X)$. Chaque fois qu'il trouve deux

Algorithm 11: Algorithme du test de dominance pour les PCP-nets arborescents

Input: Un PCP-net arborescent $\mathcal{N}$,
deux objets o et o',
$k = |X \in \mathcal{X}$ tel que $o[X] \neq o'[X]|$
Output: La probabilité que $o \succ o'$ selon $\mathcal{N}$

```
p=0;
f = ∅ ;
foreach feuille X de N do
    ajouter à f la formule worsen^{o,o'}(X);
calculer p à partir de f;
return p
```

appels récursifs différents (par exemple, sur $k-1$ et $k+1$ dans la deuxième règle), il divise la formule en deux parties. Par construction, l'algorithme se termine par :

$$\Phi^X = \{\varphi_1^X, \varphi_2^X, \ldots, \varphi_p^X\}$$

Les φ_i^X sont mutuellement incompatibles, puisque les appels récursifs à chaque règle sont conditionnés sur des formules mutuellement contradictoires sur le nœud courant. En outre, d'après la Proposition 6.2 page 109, un CP-net $N \propto \mathcal{N}$ satisfait $o[\geq X] \succ o'[\geq X]$ si et seulement si il satisfait l'une de ces formules.

Maintenant, on définit Φ comme étant égale à :

$$\Phi = \{\varphi^{X_1}, \varphi^{X_2}, \ldots, \varphi^{X_k} | X_i \text{ est une feuille}, \varphi^{X_i} \in \Phi^{X_i}\}$$

Φ représente l'ensemble des Φ^X (sur l'ensemble des feuilles). Par construction, les conjonctions sur Φ sont mutuellement incompatibles, et un CP-net $N \propto \mathcal{N}$ satisfait $o \succ o'$ si et seulement si il vérifie l'une d'entre elles (Proposition 6.3 page 112). Il s'ensuit que la probabilité recherchée peut être calculée en temps $O(|\Phi|.n)$: où le poids de chaque conjonction de Φ peut être obtenue en multipliant les probabilités des règles correspondant à N, en temps O(n). Et grâce à l'incompatibilité mutuelle des conjonctions, le résultat s'obtient en additionnant les conjonctions de Φ. Il est possible d'observer que certains éléments de Φ peuvent être des formules incohérentes, mais cela ne pose pas de problème car ils peuvent être détectés efficacement étant donné que par construction ce sont des conjonctions de littéraux.

Pour finir cette preuve il faut limiter la taille de Φ. Considérons d'abord Φ^X pour une variable X. Par construction, $|\Phi^X|$ est inférieur à 2^l où l est le nombre de règles utilisées qui donnent lieu à deux appels récursifs différents. Ce cas apparaît seulement pour les règles deux et quatre du tableau 6.1, c'est-à-dire lorsque soit X soit Y ont des valeurs différentes dans o et o'. On a donc $l \leq 2k$ et donc $|\Phi^X| \leq 2^{2k}$ pour chaque nœud feuille X et donc on en déduit que $|\Phi| \leq (2^{2k})^k = 2^{2k^2}$.

□

6.3.3 Un résultat dérivé pour les CP-nets

Un sous-résultat intéressant peut être déduit du test de dominance pour les PCP-nets. Nous allons maintenant proposer un algorithme en temps linéaire pour le test de dominance dans le cas des CP-nets arborescents. Cet algorithme permet ainsi d'améliorer le temps d'exécution de l'algorithme quadratique de Boutillier [14], et n'est pas forcement intuitif étant donné que la séquence de flips la plus petite du test de dominance peut être de taille quadratique. Malgré cela, ce n'est pas contradictoire étant donné que notre résultat indique seulement si un objet o domine un objet o' sans donner la séquence explicitement (ce que fait l'algorithme TreeDT [14]).

Théorème 6.2. *Le problème de dominance pour les Tree-CP-nets sur un ensemble de n variable peut être résolu en O(n).*

Démonstration. L'algorithme consiste simplement à décider si N satisfait la formule $\bigwedge_X worsen^{o,o'}(X)$, pour toutes les feuilles X de la structure de N. Ceci peut être réalisé de manière efficace en suivant ces quelques règles générales.

On doit nécessairement satisfaire au plus l'une des deux disjonctions et donc, un seul appel récursif est impliqué à chaque étape. Le seul point étant de vérifier si l'algorithme peut éviter d'examiner les mêmes variables à plusieurs reprises le long de différentes branches.

Pour ce faire, l'algorithme déroule les formules $worsen^{o,o'}(X)$ en parallèle. Par construction, ces appels sont tous de la forme $change_{k_i}^{o,o'}(X)$. Par construction et grâce à la Proposition 6.2, ils doivent tous être satisfaits.

Rappelons que $change_{k_i}^{o,o'}(X)$ se lit "X change de valeur au moins k_i fois". L'algorithme a besoin tout simplement de remplacer tous les appels récursifs par un unique, à savoir, $change_{max_i(k_i)}^{o,o'}(X)$, où $max_i(k_i)$ est la valeur maximale rencontré dans les différents $change_{k_i}^{o,o'}(X)$. Ainsi, chaque variable est visitée une fois, et l'algorithme est en effet linéaire en temps. □

Fait intéressant, un algorithme "top-down" est également possible : en partant des nœuds racine de la structure et de façon inductive, on peut calculer pour chaque nœud X la plus grande valeur k telle que N satisfait la formule $change_k^{o,o'}(X)$. Cet algorithme permet de prouver le résultat suivant sur des CP-nets partiels.

On dit qu'un CP-net N est partiel avec une structure donnée, s'il est donné avec un graphe G, mais pour certaines variables X et instanciation u des parents de X, N ne contient ni la règle $u : x > \bar{x}$ ni la règle inverse $u : \bar{x} > x$. Les CP-nets partiels interviennent naturellement dans le processus d'élicitation [49], et plus généralement lorsque l'utilisateur est indifférent pour certains attributs d'un objet ou certains objets (par exemple : "Je n'ai pas de préférence sur les desserts à base de noisette car je suis allergique aux noisettes. ").

Théorème 6.3. *Le problème de décider s'il existe un instanciation d'un CP-net incomplet N qui impliquerait $o \succ o'$ étant donné o,o' et N se calcule en O(n).*

Démonstration. Comme évoqué plus haut, on utilise un algorithme "top-down" pour parcourir l'arborescence. Par le calcul, pour chaque nœud X on obtient le plus grand k pour lequel il existe une instanciation satisfaisant $change_k^{o,o'}(X)$. Pour ce faire, il suffit de remplir toutes les règles manquantes d'une manière gloutonne. Par exemple, si le nœud courant Y et son enfant X sont dans le cadre de la règle inductive 2 du tableau 6.1, et N ne contient aucune règle sur X, on décide d'ajouter à N les règles de la première disjonction. De cette façon, pour la valeur k de Y on obtient $k + 1$ pour X.

Finalement, étant donné que $change_k^{o,o'}(X)$ se lit "au moins k fois", plus la valeur de k à chaque nœud est élevée, plus il y a de chance de trouver une instanciation complète de N qui implique $o \succ o'$. En arrivant à une feuille X, si le k maximum trouvé pour $change_k(X)$ est strictement plus petit que celui demandé pour $worsen(X)$, alors on peut dire que le CP-net n'implique pas $o \succ o'$. On peut donc dire que l'algorithme est correct car les règles choisies pour les variables en dessous de X ne dépendent pas de celles choisies pour X.

□

6.3.4 Un problème #P-difficile dans le cas général

Nous terminons cette section en donnant un résultat qui met en évidence la difficulté des tests de dominance sur PCP-net avec une structure plus générale qu'un arbre. Ce résultat n'est pas choquant étant donné que le test de dominance pour le CP-net est un test difficile. En effet, dans le cas des CP-nets acycliques,

on sait que le test de dominance est NP-difficile. La difficulté étant qu'il est très difficile de déterminer si la taille d'une solution peut être exponentielle. Ici, nous allons voir qu'on peut effectuer une transformation polynomiale du problème $\#Monotone\ (2, 4\mu)\ bipartite\ CNF$ en notre problème, montrant ainsi que notre problème est au moins $\#P$-difficile.

Théorème 6.4. *Le problème du calcul de $p_{\mathcal{N}}(o \succ o')$, étant donné deux objets o et o' et un PCP-net acyclique $\mathcal{N}$, est $\#P$-difficile. Cela est vrai même si la structure est acyclique, avec le chemin le plus long de longueur trois, et où chaque nœud a au plus un fils et au plus quatre parents.*

Démonstration. Nous donnons une réduction polynomiale de $\#Monotone\ (2, 4\mu)\ bipartite\ CNF$, qui est un problème $\#P$-difficile [67].

Soit $\mathcal{X}$ et $\mathcal{Y}$ deux ensemble disjoints de variables.
Un problème $\#Monotone\ (2, 4\mu)\ bipartite\ CNF$ prend une conjonction de clauses de la forme $X \vee Y$ avec $X \in \mathcal{X}$ et $Y \in \mathcal{Y}$, et où chaque variable apparaît au plus 4 fois dans la formule et conclue combien de solutions satisfont la formule. Étant donnée une formule ϕ, on construit un PCP-net $\mathcal{N}$ sur l'ensemble de variables $\mathcal{V} = \mathcal{X} \cup \mathcal{Y} \cup \mathcal{Z}$, où $\mathcal{Z}$ contient une nouvelle variable, notée Z_Y, pour chaque $Y \in \mathcal{Y}$. Les variables de $\mathcal{Z}$ ne possèdent pas de parents, et chaque variable $Y \in \mathcal{Y}$ a un unique parent Z_Y. Enfin, chaque variable $X \in \mathcal{X}$ a pour parents l'ensemble des Y qui apparaissent dans les clauses de la forme $X \vee Y$ dans la formule ϕ (X a au plus 4 parents). Cette structure, ainsi que les probabilité associées, est donnée dans la Figure 6.2, où l'on voit une partie du PCP-net qui correspond aux clause $X \vee Y_1, \ldots, X \vee Y_p$.

On considère maintenant deux objets o et o' définis par $o[X] = x, o'[X] = \bar{x}$ pour chaque $X \in \mathcal{X}$, $o[Y] = o'[Y] = y$ pour chaque $Y \in \mathcal{Y}$, et $o[Z] = z, o'[Z] = \bar{z}$ pour chaque $Z \in \mathcal{Z}$. On va montrer maintenant que $p_{\mathcal{N}}(o \succ o')$ est exactement la proportion d'interprétations de $\mathcal{V}$ telle que ϕ est vraie.

Soit I une interprétation de $\mathcal{X} \cup \mathcal{Y}$ et qui définit le CP-net $N_I \propto \mathcal{N}$ comme suit :

1. Pour chaque $\mathbf{Z} \in \mathcal{Z}$, N_I possède la règle $z > \bar{z}$
2. Pour chaque $\mathbf{Y} \in \mathcal{Y}$:
   (a) N_I possède la règle $\bar{z}_Y : \bar{y} > y$
   (b) Si $I(Y) = \top$ alors N_I possède la règle $z_Y : y > \bar{y}$, sinon il contient la règle opposée
3. pour chaque $\mathbf{X} \in \mathcal{X}$:

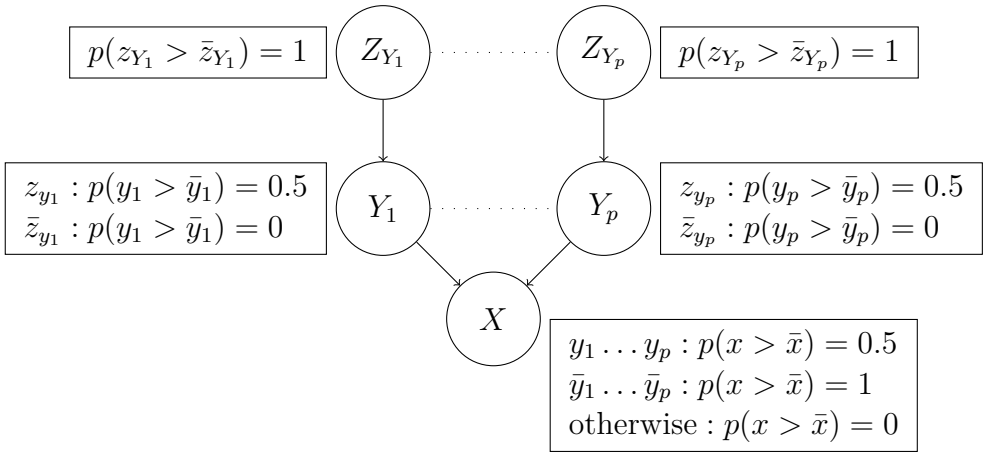


Figure 6.2 – *Schéma de réduction*

(a) N_I possède la règle $\bar{y}_1 \dots \bar{y}_p : x > \bar{x}$,

(b) Si $I(X) = \top$ alors N_I possède la règle $y_1 \dots y_p : x > \bar{x}$, sinon il contient la règle opposée

(c) pour chaque instanciation restante u de $\mathsf{pa}(X)$, N_I possède la règle $u : \bar{x} > x$.

On montre maintenant que N_I implique $o \succ o'$ si et seulement si I satisfait ϕ. Clairement, on peut raisonner sur l'ensemble $\{X, Y_1, \dots, Y_p, Z_{Y_1}, \dots, Z_{Y_p}\}$ de façon indépendante. On suppose en premier lieu que I satisfait $(X \vee Y_1) \wedge \dots \wedge (X \vee Y_p)$.

Si I satisfait X alors N_I implique $o \succ o'$ grâce à la séquence de flips $z_{Y_1} > \bar{z}_{Y_1}, \dots, z_{Y_p} > \bar{z}_{Y_p}$ et $y_1 \dots y_p : x > \bar{x}$. Sinon I doit satisfaire $Y_1 \wedge \dots \wedge Y_p$, d'où I satisfait $o \succ o'$ en utilisant la séquence de flips $z_{Y_1} : y_1 > \bar{y}_1, \dots, z_{Y_p} : y_p > \bar{y}_p$, ainsi que le flip $\bar{y}_1 \dots \bar{y}_p : x > \bar{x}$, puis le flip $z_{Y_1} > \bar{z}_{Y_1}, \dots, z_{Y_p} > \bar{z}_{Y_p}$, et enfin le flip $\bar{z}_{Y_1} : \bar{y}_1 > y_1, \dots, \bar{z}_{Y_p} : \bar{y}_p > y_p$.

La réciproque est prouvé de façon similaire, et finalement on obtient N_I implique $o \succ o'$ si et seulement si I satisfait ϕ. Maintenant, par construction, chaque N_I construit de cette manière a une probabilité de $1/2^n$ selon $\mathcal{N}$. Donc, la probabilité que $\mathcal{N}$ implique $o \succ o'$ est $m/2^n$ si et seulement si ϕ a exactement m modèles. Ce qui complète cette réduction.

□

6.4 Complexité de l'optimisation

Nous nous intéréssons maintenant au problème d'optimisation dans le cas des PCP-nets arborescents. Nous verrons dans cette section que ce problème se résout de façon simple et facile. Le premier résultat donné nous permet de connaître la probabilité qu'un objet donné soit optimal dans le cas des PCP-nets acycliques.

Proposition 6.4. *Soit un objet o et un PCP-net $\mathcal{N}$ acyclique. La probabilité que o soit optimal selon $\mathcal{N}$, notée $p_{\mathcal{N}}(o)$, se calcule en $O(n)$.*

Démonstration. Dans l'esprit de la procédure "forward sweeping" de Boutilier *et al.* [14], il est facile de montrer que o est optimal dans le cas d'un CP-net $N \propto \mathcal{N}$ si et seulement si N contient :

1. La règle $o[X] > \bar{o}[X]$ si X est une racine.
2. La règle $o[\mathsf{pa}(X)] : o[X] > \bar{o}[X]$ sinon.

Il s'ensuit que la probabilité recherchée est le produit de toutes ces règles, ce qui se calcule en $O(n)$.

□

Exemple 6.3. *Soit le PCP-net de la figure 6.3 et l'objet $o = a\bar{b}cde\bar{f}$. o est optimal dans tous les CP-nets $N \propto \mathcal{N}$ qui possèdent les règles suivantes :*

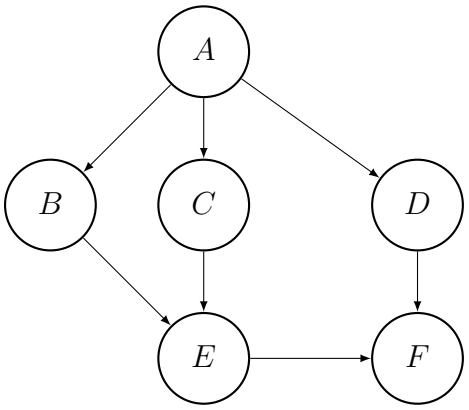
- $a > \bar{a}$
- $a : \bar{b} > b$
- $a : c > \bar{c}$
- $a : d > \bar{d}$
- $\bar{b}c : e > \bar{e}$
- $de : \bar{f} > f$

La probabilité que o soit optimal est donc le produit de ces règles soit $p_{\mathcal{N}}(o) = 0.9 \times 0.8 \times 0.9 \times 0.8 \times 0.7 \times 1 = 0,3629$

Proposition 6.5. *L'objet qui a la plus grande probabilité d'être optimal étant donné un PCP-net arborescent $\mathcal{N}$ peut être calculé en $O(n)$.*

Démonstration. L'algorithme utilisé pour trouver cet objet est un algorithme de programmation dynamique "bottom-up" dans les arbres.

Pour commencer, étant donné un nœud feuille X avec pour parent Y, l'algorithme détermine l'instanciation optimale de X étant donné $Y = y$ en prenant la



(a) *PCP-net pour un repas*

A	$a > \bar{a}$ 0.9
B	$a : b > \bar{b}$ 0.2
B	$\bar{a} : b > \bar{b}$ 0.7
C	$a : c > \bar{c}$ 0.9
C	$\bar{a} : c > \bar{c}$ 0.7
D	$a : d > \bar{d}$ 0.8
D	$\bar{a} : d > \bar{d}$ 0.5
E	$bc : e > \bar{e}$ 1
E	$b\bar{c} : e > \bar{e}$ 0.1
E	$\bar{b}c : e > \bar{e}$ 0.7
E	$\bar{b}\bar{c} : e > \bar{e}$ 0.4
F	$de : f > \bar{f}$ 0
F	$d\bar{e} : f > \bar{f}$ 0.8
F	$\bar{d}e : f > \bar{f}$ 0.6
F	$\bar{d}\bar{e} : f > \bar{f}$ 0.5

(b) *Table de préférences conditionnelles probabilistes*

Figure 6.3 – *Exemple d'un PCP-net acyclique à 6 variables*

plus haute probabilité entre les règles $y : x > \bar{x}$ et $y : \bar{x} > x$, et de façon similaire on prend la plus haute probabilité de X lorsque $Y = \bar{y}$.

Maintenant dans le cas général, lorsque l'on a une variable Y avec un parent Z et un ensemble d'enfants $X_1, \ldots, X_k$, l'algorithme considère d'abord le cas où $Z = z$. Étant donné cette valeur, on cherche l'instanciation la plus probable de $Y, X_1, \ldots, X_k$ et de leurs descendants. Cette recherche peut être effectuée de façon efficace en comparant 1. et 2. telle que :

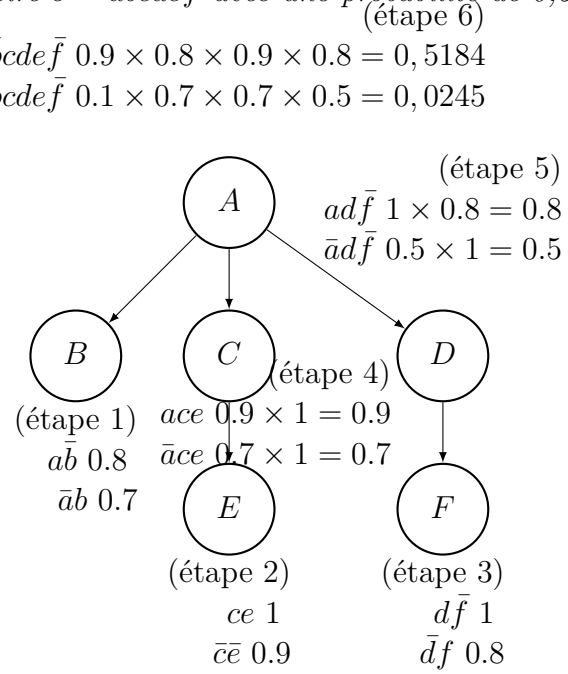
1. $p_y \times p_{y1} \times \cdots \times p_{yk}$, où p_y est la probabilité de la règle $z : y > \bar{y}$ et p_{yi} $(i = 1, \ldots, k)$ la probabilité précédemment calculée de la meilleure valeur de X_i et de ses descendants connaissant $Y = y$
2. $p_{\bar{y}} \times p_{\bar{y}1} \times \cdots \times p_{\bar{y}k}$.

De la même manière, l'algorithme calcule la probabilité de la meilleur valeur pour $Z = \bar{z}$, puis continue avec le fils de Z jusqu'à arriver à la racine. Finalement, lorsque l'algorithme a visité toutes les variables, il retourne l'objet désiré.

□

Exemple 6.4. *On voit sur le PCP-net de la Figure 6.4 repris de la Figure 6.1*

la meilleure instanciation ainsi que la probabilité de celle-ci au dessus de chaque variable. Le nombre entre parenthèses représente l'ordre dans lequel les variables ont été traitées. A la fin de l'exécution, l'algorithme nous donne comme objet désiré $o = a\bar{b}cde\bar{f}$ avec une probabilité de 0,5184.



(a) *PCP-net pour un repas*

A	$a > \bar{a}$ 0.9
B	$a : b > \bar{b}$ 0.2
B	$\bar{a} : b > \bar{b}$ 0.7
C	$a : c > \bar{c}$ 0.9
C	$\bar{a} : c > \bar{c}$ 0.7
D	$a : d > \bar{d}$ 0.8
D	$\bar{a} : d > \bar{d}$ 0.5
E	$c : e > \bar{e}$ 1
E	$\bar{c} : e > \bar{e}$ 0.1
F	$d : f > \bar{f}$ 0
F	$\bar{d} : f > \bar{f}$ 0.8

(b) *Table de préférences conditionnelles probabilistes*

Figure 6.4 – *Exemple d'exécution de l'algorithme bottom-up sur un PCP-net arborescent*

6.5 Une évolution possible des PCP-nets

En ce qui concerne les PCP-nets, le plus gros problème est la notion d'indépendance entre les préférences locales. Elle empêche de représenter de façon fidèle les préférences d'un ensemble d'utilisateurs. En effet, il est impossible de spécifier la proportion de personnes qui ont la préférence $a > \bar{a}$ sachant qu'ils ont la préférence $b > \bar{b}$. Afin de pallier ce problème,nous allons voir dans cette section perspective, une nouvelle extension des PCP-nets permettant de répondre à ce type de question. L'extension imaginée met en relation les CP-nets et les réseaux bayésiens, permettant ainsi de représenter des dépendances entre les relations de

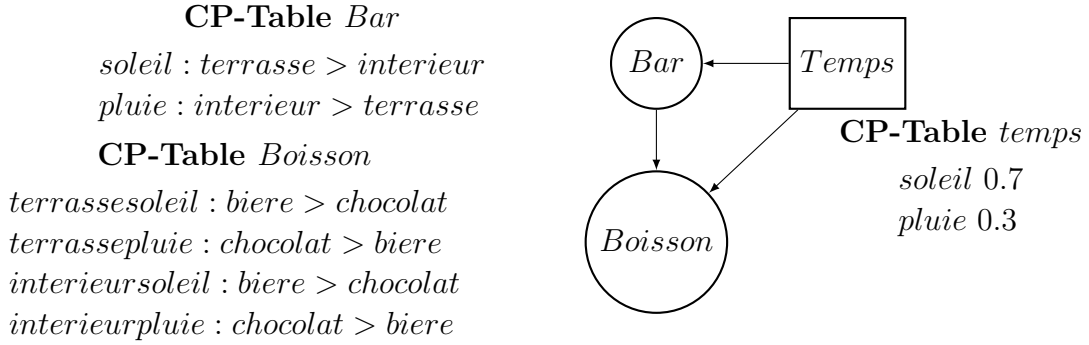


Figure 6.5 – *un CP-net probabiliste généralisé*

préférence. Le CP-net représente ainsi un ordre partiel sur un ensemble d'objet dans un monde donné. Le réseau bayésien permet quant à lui de représenter une distribution de probabilités sur des modèles/mondes. Il peut ainsi représenter des caractéristiques d'un ensemble d'utilisateurs comme le nombre d'enfants, la profession, etc. ou encore des variables extérieures qui influent sur les préférences d'un utilisateur mais qui ne sont pas des caractéristiques des objets concernés.

Définition 6.6. *Soit E un ensemble de variables exogènes, c'est à dire un ensemble de variables non contrôlables. Soit C un ensemble de variables de décision. Un GPCP-net est une CP-net déterministe N sur C et un réseau bayésien B sur E tel qu'un arc représente soit une dépendance préférentielle entre deux éléments de C, soit une dépendance conditionnelle entre deux variables de E soit une dépendance entre une variable X de E vers un variable Y C indiquant que les préférences de Y dépendent de X. Le GPCP-nets de la Figure 6.5 associé à E et C représente une distribution de probabilités sur le produit cartésien des domaines des variables de E, chaque combinaison est appelé un monde et où à chaque monde on associe un cp-net déterministe sur C.*

Exemple 6.5. *Soit $C = \{Bar, Boisson\}$ l'ensemble des variables contrôlables où $Bar = \{interieur, terrasse\}$, $Boisson = \{chocolat, biere\}$. Soit $E = \{Temps\}$ avec $Temps = \{soleil, pluie\}$. Le GPCP-net associé à E et C est donné Figure 6.5. Ainsi si on se place dans le monde où le temps est à beau, on obtient un cp-net déterministe ayant une probabilité de 0.7 et où l'objet optimal est boire une bière en terrasse. Alors que si le temps est à pluie le cp-net obtenu à une probabilité de 0.3 et l'objet optimal est boire un chocolat à l'intérieur.*

Cette représentation serait un cadre généralisant les CP-nets et les PCP-nets. En effet, un CP-net est un cas particulier du GPCP-net où le monde est connu.

Un PCP-net est un GPCP-net où chaque variables du monde est indépendante aux autres et où chaque variable du CP-net dépend au plus d'une variable du monde.

Les premières requêtes possibles consistent à se placer dans le cadre où un monde m est instancié. Ces requêtes se placent donc dans le cas connu des CP-nets déterministes. Ainsi les premières requêtes sont de la forme : "Connaissant un monde m, quel est l'objet optimal ?, ou encore l'objet o domine t-il l'objet o' ?"

Nous proposons maintenant une suite de requêtes que nous pensons pertinentes d'étudier dans un futur proche. Pour certaines de ces ces requêtes, nous donnerons une piste pour essayer d'y répondre.

Requête 6.1. *Soit un GPCP-net N,soit un monde $m \in 2^E$. Quel est l'objet optimal $o \in 2^C$ selon N.*

Lorsque le monde est connu, on se place dans le cadre des CP-nets déterministes donc la complexité de savoir quel objet est optimal pour un CP-net donné est linéaire si le CP-net est acyclique.

Requête 6.2. *Soit un GPCP-net N,soit un monde $m \in 2^E$ et deux objets o et $o' \in 2^C$. Est ce que le CP-net impliqué par m implique $o \succ o'$*

Lorsque le monde est connu, on se place dans le cadre des CP-nets déterministe donc la complexité du test de dominance est PSPACE-complet dans le cas général, NP-dur dans le cas des CP-nets acyclique et polynomial dans le cas des arbres.

Nous nous intéressons maintenant à des mondes connus partiellement. Deux nouvelles requêtes consistent à chercher la probabilité qu'un objet o soit optimal ou la probabilité que o domine o'.

Requête 6.3. *Soit un GPCP-net N,soit un monde $m \in 2^{E'}$ et $E' \subseteq E$. Soit un objet $o \in 2^C$. Quel est la probabilité que o soit optimal selon N.*

Il est possible que l'utilisation de la requête MAR des réseaux bayésiens suffise pour répondre à une tel requête.

Requête 6.4. *Soit un GPCP-net N,soit un monde $m \in 2^{E'}$ et $E' \subseteq E$. Soit deux objets $o, o' \in 2^C$. Quel est la probabilité que $o \succ o'$.*

Pour les GPCP-nets arborescents, on construit une formule ϕ correspondant aux possibilités des CP-net satisfaisant $o \succ o'$ comme on le fait pour les PCP-nets. Une fois la formule construite, on pose pour chaque variable de la formule (2^{2k^2}) une requête de probabilité marginal (MAR) qui consiste à calculer la probabilité à posteriori $p(V|E' = e)$ pour toute les variables du réseau bayésien $V\{E'\}$

Requête 6.5. *Sachant un objet $o \in 2^c$. quel est la probabilité qu'il soit optimal ?*

Requête 6.6. *Sachant un objet $o \in 2^C$ et un monde $e \in 2^E$. Ètant donné un utilisateur/monde inconnu possédant o comme objet optimal, quelle est la probabilité que u=e.*

Requête 6.7. *Sachant un objet partiel $o \in 2^{C'}$ avec $C' \subset C$. Quel est la complétion de o la plus probablement optimale.*

Il est possible que les requêtes MAP(rechercher une affectation partielle la plus probable) ou MPE (rechercher une affectation complète la plus probable) suffisent pour répondre à ces requêtes.

6.6 Conclusion

Dans ce chapitre, nous avons proposé une version probabiliste des CP-nets qui permet la représentation des relations de préférence d'un groupe d'utilisateurs sur un ensemble d'objets combinatoires, ou qui peut aussi représenter les relations de préférence d'un individu lorsque celles-ci sont mal connues. A travers ce chapitre, nous avons étudié la version probabiliste des principales tâches de raisonnement qui sont utilisées dans les CP-nets, plus précisément la requête de dominance ainsi que les requêtes d'optimisation. On a proposé des algorithmes efficaces dans le cas des PCP-nets arborescents, de plus nous avons obtenu un algorithme qui permet de décider en temps linéaire la dominance entre deux objets dans le cas des CP-nets arborescents.

En plus de ces algorithmes, nous avons vu que le test de dominance des PCP-nets dans le cas général était difficile, notamment en prouvant que le problème devenait $\#P$-difficile lorsque l'on avait un PCP-net de hauteur 3 et où chaque variable avait au plus 4 parents.

Dans le prochain chapitre, nous reprendrons les PCP-nets puis nous verrons comment apprendre de telles structures, que ce soit grâce à de l'apprentissage passif ou de l'apprentissage actif.

Malgré ces résultats intéressants, les PCP-nets ont une expressivité limitée. Premièrement, les algorithmes efficaces que nous avons proposés ne fonctionnent que pour des structures arborescentes ce qui est peu probable dans des cas de relation de préférence réelles. La deuxième limitation vient du fait que les tables de préférence locale des PCP-nets sont indépendantes les unes des autres. Par exemple, si 70 agents sur 100 préfèrent a à $\bar{a}$ et 80 agents préfèrent b à $\bar{b}$, il est

impossible de donner de façon exacte le nombre d'agents qui préfèrent à la fois a et b.

Chapitre 7

Apprentissage de PCP-net

Sommaire

7.1 Introduction

Depuis quelques années, plusieurs méthodes ont été proposées afin d'apprendre des relations de préférence, une bonne partie de ces méthodes sont présentées dans le livre de Fürnkranz et Hüllermeier [35]. Ces approches, vont de l'apprentissage de fonction d'utilité [7, 47, 72] pour les représentations de préférences quantitatives, à l'apprentissage de règles structurées [10, 26, 49, 50]. Ces travaux supposent qu'on donne ou qu'on élicite un ensemble de comparaisons représenté par des paires d'objets comme effectué dans la partie apprentissage du chapitre 4 des GAI de cette thèse, afin de construire un modèle qui généralise ces comparaisons.

Toutefois, dans plusieurs cadres, il est intéressant d'apprendre non pas le modèle d'un seul utilisateur, mais plutôt un modèle probabiliste qui peut représenter de manière compacte les préférences d'un groupe d'utilisateurs - ce modèle peut alors être finement ajusté pour s'adapter à un utilisateur particulier. Même dans les contextes où un utilisateur n'est pas anonyme, ses préférences sont généralement mal connues, car elles peuvent dépendre de la valeur d'un ensemble de variables d'état non contrôlables. Dans un tel contexte, la question n'est plus de savoir quel est l'objet optimal ou encore est-ce qu'un objet o est préféré à un objet o' mais "Quelle est la probabilité qu'un objet o soit optimal ?" ou encore "Quelle est la probabilité que o soit préféré à un objet o' ? ".

Dans le chapitre 6, nous avons présenté les réseaux de préférences conditionnelles probabilistes [6, 22]. Cette représentation de préférences permet de représenter de manière compacte une distribution sur des ordres partiels et de répondre à ce genre de requête. Dans son travail de master [21], Cornelio montre qu'il existe un lien étroit entre les PCP-nets et les réseaux bayésiens. Plus particulièrement, elle prouve que le problème de trouver l'objet le plus probablement optimal est équivalent à un problème d'optimisation d'un réseau bayésien. Toutefois, un PCP-net code une distribution de probabilités sur des ordres partiels et pas seulement une distribution sur des objets.

Nous allons étudier dans ce chapitre comment les PCP-nets peuvent être appris, à la fois à partir d'un apprentissage off-line et d'un apprentissage on-line avec différents paramètres. Contrairement à l'apprentissage de GAI présenté plus haut dans le Chapitre 4 ou dans l'article mentionné sur l'apprentissage de préférences, nous n'utiliserons pas ici un ensemble de comparaisons mais plutôt une liste d'objets qui représente des objets optimaux pour un ensemble d'utilisateurs. Cette liste peut être obtenue par exemple par un historique de vente. Nous prouverons au cours de ce chapitre qu'une telle liste suffit pour apprendre une distribution de probabilités sur des ordres partiels sur l'ensemble des objets pos-

sibles, lorsque ceux-ci ont une structure combinatoire. L'intérêt de cette approche est d'essayer d'apprendre des préférences sur tout les objets à partir seulement d'exemples d'objets optimaux.

7.2 Apprentissage d'un PCP-net hors ligne

Dans de nombreux cas, notamment dans un système de recommandation, le système peut enregistrer une liste d'objets que l'on peut supposer avoir été optimaux pour des utilisateurs à un instant donné. Il peut s'agir par exemple d'une liste d'objets vendus. Notons $\mathcal{L}$ cette liste, supposons maintenant que les relations de préférence des utilisateurs peuvent être représentées par un PCP-net $\mathcal{N}$, les fréquences des objets de cette liste correspondent ainsi aux probabilités d'optimalité des objets dans le réseau bayésien correspondant. Cette correspondance laisse à penser qu'il est possible d'apprendre un PCP-net à partir de cette liste d'objets vendus.

7.2.1 Apprendre les paramètres

Supposons que nous connaissons la structure du PCP-net que l'on cherche à apprendre, et que nous voulons estimer les probabilités dans les tables. Lorsque les variables sont binaires, ces probabilités sont exactement les probabilités qui apparaissent dans les tables du réseau bayésien qui codent les probabilités d'optimalité.

En particulier, observer les probabilités d'optimalité peut être suffisant pour estimer les probabilités des règles : pour toute variable binaire $X \in \mathcal{V}$, pour chaque $u \in \underline{\mathsf{pa}(X)}$, nous avons :

$$\begin{aligned} p(X, u : x > \bar{x}) &= P(o_{opt}[X] = x \mid o_{opt}[U] = u) \\ &\sim |\{o \in \mathcal{L}, o[UX] = ux\}|/|\{o \in \mathcal{L}, o[U] = u\}| \end{aligned}$$

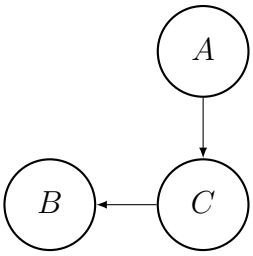
si $\{o \in \mathcal{L}, o[U] = u\}$ n'est pas vide, c'est-à-dire quand $P(\text{opt}[U] = u) \neq 0$ dans le PCP-net que l'on cherche à apprendre.

Plus généralement, nous pouvons utiliser des méthodes qui ont été utilisées pour les réseaux bayésiens pour apprendre ces probabilités.

Si $P(\text{opt}[U] = u) = 0$, on peut encore être en mesure d'estimer $p(X, u : x > \bar{x})$ à partir des probabilités des objets sous-optimaux, si nous avons une liste des objets qui ont été choisis par certains utilisateurs sous certaines contraintes, par exemple une liste des objets vendus au cours d'une période où certaines options

ne sont pas disponibles. Le seul effet d'une telle contrainte est de restreindre le domaine de certaines variables, mais ne modifie pas les probabilités des autres variables pour les combinaisons restantes des valeurs des parents. Soit $\mathcal{L}_{V=v}$ une liste d'éléments optimaux sous la contrainte $V = v$ pour certains $V \subseteq \mathsf{asc}(U)$ et $v \in \underline{V}$ tel que $v(U) = u(V)$, et tel que " $\mathrm{opt}_{V=v}[U] = u$ " désigne le cas où l'élément qui est optimal parmi tous les objets qui ont une valeur v pour les variables de V, et une valeur u pour les variables de U, alors :

$$p(X, u : x > \bar{x}) = P(\mathrm{opt}_{V=v}[X] = x \mid \mathrm{opt}_{V=v}[U] = u)$$
$$\sim |\{o \in \mathcal{L}_{V=v}, o[UX] = ux\}|/|\{o \in \mathcal{L}_{V=v}, o[U] = u\}|$$



A	$a > \bar{a}$ 0.9
C	$a : c > \bar{c}$ 0
C	$\bar{a} : c > \bar{c}$ 0
B	$c : b > \bar{b}$ 0.2
B	$\bar{c} : b > \bar{b}$ 0.7

(b) *Exemple de table locale 2*

A	$a > \bar{a}$ 0.9
C	$a : c > \bar{c}$ 0.9
C	$\bar{a} : c > \bar{c}$ 0.7
B	$c : b > \bar{b}$ 0.2
B	$\bar{c} : b > \bar{b}$ 0.7

(a) *Exemple de table locale 1*

A	$a > \bar{a}$ 0
C	$a : c > \bar{c}$ 0.8
C	$\bar{a} : c > \bar{c}$ 0
B	$c : b > \bar{b}$ 0.2
B	$\bar{c} : b > \bar{b}$ 0.7

(c) *Exemple de table locale 3*

Figure 7.1 – *Exemple d'un PCP-net à 3 variables et 3 exemples de tables de préférences locales*

Exemple 7.1. *Considérons un PCP-net qui a la même structure que le PCP-net de la figure 7.1. La probabilité de la règle $B, c : b > \bar{b}$ peut être estimée des façons suivantes :*

1. *Dans les tables de préférences locales 7.1a on a $P(\mathrm{opt}[C] = c) \neq 0$ on peut donc calculer notre règle selon la formule suivante :*

$$p(B, c : b > \bar{b}) \sim |\{o \in \mathcal{L}, o[CB] = cb\}|/|\{o \in \mathcal{L}, o[C] = c\}|;$$

2. *Dans le cas des table 7.1b, on a $P(\mathrm{opt}[C] = c) = 0$, avec $p(a > \bar{a}) > 0$ on peut donc apprendre la règle en forcant $C = c$:*

$$p(B, c : b > \bar{b}) \sim |\{o \in \mathcal{L}_{C=c}, o[B] = b\}|/|\mathcal{L}_{C=c}|;$$

3. *ou, encore dans le cas des tables 7.1c, on a* $P(\text{opt}[C] = c) = 0$*, car* $p(\bar{a} : c > \bar{c}) = 0$ *et* $p(a > \bar{a}) = 0$*, on doit donc apprendre la règle en forcant la valeur de* A *:*

$$p(B, c : b > \bar{b}) \sim |\{o \in \mathcal{L}_{A=a}, o[CB] = cb\}|/|\{o \in \mathcal{L}_{A=a}, o[C] = c\}|;$$

Les équations 2 et 3 ci-dessus donnent deux façons différentes de calculer $p(B, c : b > \bar{b})$ *lorsque la probabilité d'avoir* $C = c$ *dans un objet optimal est égale à zéro, correspondant à deux observations différentes : 2. correspond à l'observation des objets optimaux lorsque* $C = c$ *est forcé, et 3. à l'observation des objets optimaux lorsque* $A = a$ *est forcé.*

7.2.2 Apprentissage de la structure

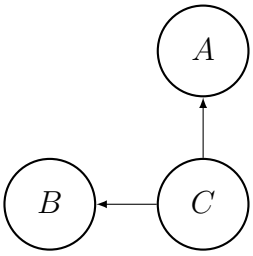
Là encore, les algorithmes utilisés pour apprendre les réseaux bayésiens peuvent être utilisés pour apprendre un PCP-net. L'algorithme présenté dans l'état de l'art sur les réseaux bayésien et rappelé avec l'algorithme 6 page 61 permet d'apprendre un réseau bayésien à partir d'une liste d'événements. Dans notre cas, la liste d'événements est une liste d'objets optimaux.

Un obstacle majeur, cependant, est que plusieurs PCP-nets avec des structures différentes peuvent donner la même distribution de probabilités d'optimalité : cela est dû au fait que les dépendances préférentielles encodées dans un PCP-net sont orientées, alors que, dans un réseau bayésien, les dépendances probabilistes sont symétriques.

Considérons par exemple le réseau bayésien qui code les probabilités d'optimalité pour le PCP-net de l'exemple 7.1a Il est possible de construire un autre réseau bayésien en inversant par exemple la dépendance entre A et C. Ce qui donne $p(c) = p(c|a)p(a) + p(c|\bar{a})p(\bar{a}) = 0.79$, $p(a|c) = (p(c|a) * p(a))/p(c) = 0.911$ et $p(\bar{a}|c) = (p(c|\bar{a}) * p(\bar{a}))/p(c) = 0.089$

On peut donc construire le PCP-net de la figure 6.3 qui possède la même distribution de probabilités d'optimalité que le PCP-net de l'exemple 7.1.

Par conséquent, les méthodes d'apprentissage de réseaux bayésiens ne permettent pas d'apprendre complètement un PCP-net. Cependant, beaucoup d'informations sont obtenues de cette façon. Si un ordre topologique sur les variables du PCP-net est donné, alors la bonne orientation de chaque arc peut être déduite, et les paramètres du PCP-net peuvent être calculées à partir du réseau bayésien. Ce genre de cas où l'ordre topologique est connu à l'avance arrive assez régulièrement. C'est notamment le cas des systèmes de configuration interactive



A	$c : a > \bar{a}$ 0.911
A	$\bar{c} : a > \bar{a}$ 0.089
C	$c > \bar{c}$ 0.79
B	$c : b > \bar{b}$ 0.2
B	$\bar{c} : b > \bar{b}$ 0.7

Figure 7.2 – *Un autre PCP-net donnant la même distribution de probabilité sur les objets optimaux que le PCP-net de la figure 7.1*

où l'ordre des variables à configurer est fixé (le système vous demande de choisir une valeur pour X_1 , puis une valeur pour X_2 , etc). On peut supposer dans ce cas que les relations de préférence de l'utilisateur seront exprimées par rapport à cet ordre.

7.3 Apprentissage en ligne

7.3.1 Mise à jour des paramètres d'un PCP-net

On peut supposer avoir déjà appris un PCP-net $\mathcal{N}$ (à partir d'une liste d'objets optimaux comme vu dans la section 7.2), que l'on souhaite mettre à jour en temps réel à chaque fois qu'un nouvel utilisateur se présente et nous donne son objet optimal suite à une phase de recommandation. Ainsi, mettre à jour le PCP-net revient à mettre à jour les probabilités d'un ensemble de règles du PCP-net.

Par exemple, dans le cas où le PCP-net $\mathcal{N}$ correspond à un groupe d'utilisateur, lorsqu'un nouvel utilisateur se présente, il est possible que le PCP-net ne reflète pas complètement ses préférences. Dans cette situation, il est cohérent de vouloir modifier les probabilités du PCP-net afin de prendre en compte les préférences de cet utilisateur. Il est aussi possible d'imaginer que le PCP-net reflète les préférences d'un groupe d'utilisateur à un temps donné ou dans un contexte particulier (certains produits sont indisponibles), et qu'il faille mettre à jour ces probabilités suite à un nouveau contexte.

Pour cela, on ne met pas à jour complètement le PCP-net mais seulement un ensemble de règles correspondant aux règles représentant l'objet optimal observé par le système. Ainsi, à chaque fois que le système reçoit un nouvel objet optimal o, il mettra à jour ses paramètres, à savoir les probabilités des règles de son PCP-net $\mathcal{N}$ actuel comme suit :

Algorithm 12: Mise à jour d'un PCP-net

Input: un PCP-net $\mathcal{N}$
Output: un PCP-net $\mathcal{N}$

```
1 foreach objet optimal o observé do
2     foreach X ∈ χ do
3         u = o[pa(U)] ;
4         X_o = 1 si o[X] = x, 0 sinon ;
5         p(X, u : x > x̄) = (η_t × X_o) + (1 − η_t)p(X, u : x > x̄) ;
6 return N
```

L'algorithme 12 met à jour seulement certaine règles (étape 1) : les règles qui font que l'objet donné est optimal. La règle de mise à jour est commune dans un contexte d'apprentissage stochastique [11]. Le paramètre η_t est le taux d'apprentissage qui varie dans le temps. Plus η est grand, plus l'apprentissage se fait rapidement au détriment de la précision, alors que plus η est petit plus l'apprentissage est précis mais plus long. Lors de l'apprentissage actif, on fait varier η au cours du temps. Généralement, on le diminue afin d'assurer la convergence vers le modèle que l'on cherche à apprendre. Cette convergence est garantie si :

- $\sum_t \eta_t = \infty$
- $\sum_t \eta_t^2 < \infty$

Dans nos expérimentations, nous avons choisi de prendre $\eta_t = 1/k(t, X, u)$ où $k(t, X, u)$ est le nombre de fois où la règle correspondant à (X, u) a été mise à jour à un instant donné. Dans ce cas, $p(X, u : x > \bar{x})$ est la fréquence avec laquelle les éléments optimaux o avec $o[U] = u$ et $o[X] = x$ ont été rencontrés jusqu'à présent.

7.4 Expérimentation

Afin de vérifier que notre formule de mise à jour des probabilités des règles dans le cadre d'un apprentissage en ligne est efficace, nous avons effectué plusieurs expériences. Pour chacunes de ces expérimentations, on suppose que le PCP-net $\mathcal{N}$ que l'on cherche à apprendre possède une structure acyclique connue sur un ensemble de n variables. Pour cela, on commence l'expérimentation avec un PCP-net $\mathcal{N}$ que l'on tire aléatoirement, représentant les préférences d'un groupe d'utilisateurs. Ensuite, on construit un PCP-net initial $\bar{\mathcal{N}}$ possédant la structure

de $\mathcal{N}$ et dont les probabilités de chaque règle sont initialisées à 0.5 ; le but de l'expérimentation étant de faire converger $\bar{\mathcal{N}}$ vers $\mathcal{N}$.

Afin de vérifier si le PCP-net $\bar{\mathcal{N}}$ converge bien vers $\mathcal{N}$ et combien d'utilisateurs sont nécessaires pour bien apprendre $\mathcal{N}$ nous avons décidé d'observer comment la distance évolue entre les deux PCP-nets. Comme mesures de distances, nous avons décidé d'utiliser :

- La somme des distances au carré des paramètres

$$d_2(\mathcal{N}, \bar{\mathcal{N}}) = \sum_{(X,u)} (\, p_{\mathcal{N}}(X, u : x > \bar{x}) - p_{\bar{\mathcal{N}}}(X, u : x > \bar{x}) \,)^2;$$

- La divergence de Kullback-Lieber, ou l'entropie relative, sur la distributions de probabilité des objets optimaux définis par $\mathcal{N}$ et $\bar{\mathcal{N}}$:

$$d_{KL}(\mathcal{N} \| \bar{\mathcal{N}}) = \sum_{o \in \underline{\mathcal{V}}} \log(\, P_{\mathcal{N}}(\text{opt} = o) / P_{\bar{\mathcal{N}}}(\text{opt} = o) \,) \times P_{\mathcal{N}}(\text{opt} = o).$$

  Plus précisément, on calcule une estimation de cette distance en tirant de façon uniforme un ensemble d'objets . Cette estimation est faite en prenant 2^{n-2} objets tiré aléatoirement.

Pour ces expérimentations , nous avons proposé deux protocoles permettant de générer les objets optimaux à chaque étape.

7.4.1 Protocole

7.4.1.1 Protocole 1

L'idée est de simuler un cadre interactif, dans lequel, pour une paire d'objets o_1 et o_2, on observe quel objet est le plus souvent optimal. Si notre PCP-net courant n'est pas d'accord avec cette observation, on le met à jour en fonction de l'objet préféré. Plus précisément, pour chaque nouvelle itération, nous générons deux objets de la manière suivante :

- On génére deux CP-nets N_1 et N_2 selon la distribution de probabilités de notre hypothèse courante $\bar{\mathcal{N}}$
- Puis on prend les deux objets o_1 et o_2 qui sont respectivement optimaux dans N_1 et N_2.

- Ensuite on calcule les probabilités $\bar{p}_i = p_{\bar{\mathcal{N}}}(\text{opt} = o_i)$ qu'o_1 et o_2 soient optimaux selon notre PCP-net courant $\bar{\mathcal{N}}$.
- On observe ensuite qui de o_1 et o_2 est l'objet le plus souvent optimal selon le PCP-net $\mathcal{N}$ que l'on cherche à apprendre en calculant les probabilités $p_i = p_{\mathcal{N}}(\text{opt} = o_i)$ qu'o_1 et o_2 soient optimaux selon notre PCP-net.
- Finalement, si $\bar{p}_1 > \bar{p}_2$ alors que $p_2 > p_1$, on met à jour $\bar{\mathcal{N}}$ afin d'augmenter la probabilité que o_2 soit plus souvent optimal.

7.4.1.2 Protocole 2

Le second protocole consiste à mettre à jour le PCP-net après qu'un nouvel utilisateur ait donné un objet, en supposant que cet objet soit son objet optimal. Pour cela, à chaque itération on génère un CP-net N selon $\mathcal{N}$ qui représente la relation de préférence d'un utilisateur, et on calcule son objet optimal.

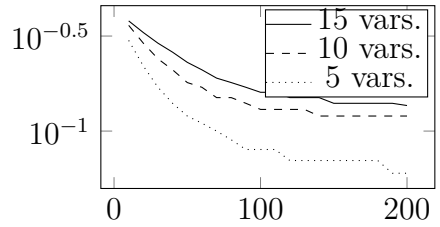
7.4.2 Résultats

Pour chaque protocole, nous avons effectué 500 essais avec des PCP-nets arborescent tirés de façon aléatoire pour 5, 10 et 15 variables avec au maximum n/2 fils. Pour chaque essai, on a généré un PCP-net cible, mis en place notre protocole et l'algorithme d'apprentissage. Puis toutes les 10 itérations, on a calculé la distance entre le PCP-net cible et le PCP-net appris. Les graphiques des figures 7.3 représentent l'évolution de cette moyenne en fonction du nombre d'itérations effectuées. Ainsi, chaque point représente une moyenne de la distance sur les 500 essais pour chaque nombre de variables.

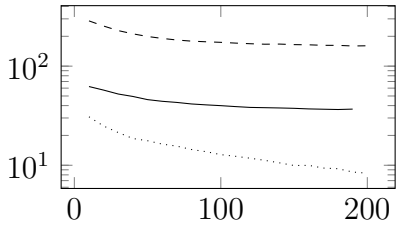
On observe qu'une bonne approximation du PCP-net cible est atteint après environ 100 itérations.

7.5 Conclusion

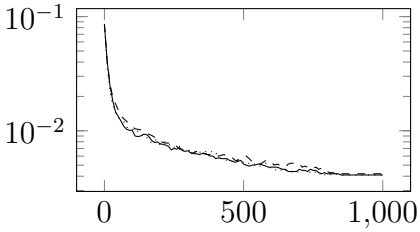
Après avoir présenté les PCP-nets dans le chapitre 6, on a présenté dans ce chapitre comment il est possible d'apprendre une représentation probabiliste des relations de préférence d'un groupe d'utilisateurs dans un domaine combinatoire et comment il est possible d'affiner ces préférences pour un utilisateur en particulier. De façon générale, les CP-nets sont une représentation des préférences efficace pour rechercher les objets les plus privilégiés. Notre méthode d'apprentissage de PCP-nets prend avantage de cette propriété en supposant que l'apprenant utilise



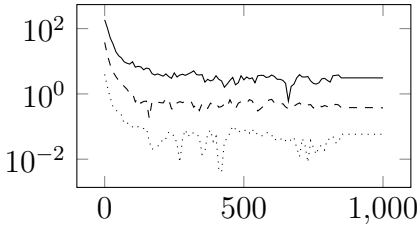
Premier protocole :
somme au carré sur les paramètres.



Premier protocole :
divergence de KL.



Second protocole :
somme au carré sur les paramètres.



Second protocole – divergence de KL

Figure 7.3 – *Graphes représentant l'évolution, entre le nombre d'observations de résultats optimaux (axe des x), et la mesure de la distance entre le PCP-net cible et l'apprenant (axe des y) pour le protocole 1 et 2*

une liste de ces objets pour apprendre de façon efficace les relations de préférence d'un groupe d'utilisateurs.

Chapitre 8

Conclusions et perspectives

Sommaire

La présentation de cette thèse touchant à sa fin, nous allons maintenant faire un résumé conclusif de celle-ci, puis nous donnerons plusieurs directions scientifiques qui pourront être poursuivies après cette thèse.

8.1 Résumé conclusif

Cette thèse a principalement porté sur de la représentation de relation de préférence ainsi que leurs méthodes d'apprentissage que l'on peut associer à ces formalismes. Nos principales contributions sont :

- Une méthode d'apprentissage passif pour les réseaux GAI.
- Une nouvelle extension des CP-nets : le PCP-net. Il permet de représenter les relations de préférence d'un ensemble d'utilisateurs, ou d'un utilisateur dont les préférences changent au cours du temps.
- Un algorithme d'apprentissage passif permettant d'apprendre un PCP-net.
- Plusieurs méthodes d'interactions possibles avec un utilisateur permettant un apprentissage actif plus précis que l'apprentissage passif.

Notre première contribution, traitée dans le chapitre 5, consiste en la définition d'une méthode d'apprentissage passif de la structure d'une GAI-décomposition ainsi que les fonctions d'utilités qui lui sont associées. Cet apprentissage se fait ainsi en transformant des exemples de la forme $o \succ o'$ en équations linéaires et à résoudre un système d'équations linéaires. Nous avons ainsi prouvé que le formalisme des GAI était PAC-apprenable mais qu'il nécessitait énormément

d'exemples pour un apprentissage correct. Puis nous avons appuyé ces résultats par des expérimentations qui ont confirmé les résultats théoriques sur le nombre d'exemples nécessaires.

Le chapitre 6, présentant notre seconde contribution, ne s'intéresse plus à l'apprentissage de représentations de préférences mono-agent, mais propose une extension probabiliste des CP-nets permettant une représentation de préférence multi-utilisateurs. Cette nouvelle représentation introduite par Bigot *et al.* [6] et Cornelio *et al.* [22] permet de représenter une distribution de probabilités sur des CP-nets. Cette distribution est construite en ajoutant à chaque règle de préférences locales une probabilité d'existence. À partir de cette nouvelle représentation, nous avons proposé un ensemble d'algorithmes linéaires permettant de trouver l'objet qui a la plus haute probabilité d'être optimal dans un PCP-net arborescent ou encore quel est la probabilité qu'un objet o donné soit optimal selon un PCP-net. Nous avons aussi donné des résultats intéressants en ce qui concerne la probabilité de dominance notamment un algorithme FPT dans le cas des PCP-nets arborescents ainsi qu'un algorithme linéaire pour les CP-nets dérivés directement de celui des PCP-nets. Finalement, nous avons prouvé que le problème de la probabilité de dominance était un problème $\#P$-difficile dans le cas général.

Enfin, notre troisième contribution, présenté dans le Chapitre 7, est une suite logique du Chapitre 6 puisque l'on propose différentes façon d'apprendre un PCP-net, que ce soit de façon hors-ligne à partir d'une liste d'objets optimaux, ou encore en-ligne en posant différentes requêtes à l'utilisateur. Ces deux méthodes peuvent être utilisées de façon indépendante mais dans certains cas, elles peuvent être utilisées de façon complémentaire. C'est notamment le cas lorsque l'on souhaite apprendre la structure d'un PCP-net. Dans ce cas, on commencera par un apprentissage hors-ligne permettant d'apprendre la structure du PCP-net sans connaître l'orientation correcte des arcs. Une fois cette structure apprise, on continue avec un apprentissage actif afin d'orienter correctement les arcs. Nous avons appuyé l'algorithme de mise à jour d'un PCP-net par une série d'expérimentations.

8.2 Perspectives

Ces travaux effectués durant ces trois années ont permis de répondre à notre problématique mais on a aussi ouvert de nouvelles perspectives.

Une première perspective intéressante serait de trouver une stratégie pour le

choix du degré d'un GAI à apprendre. Une approche possible consiste à suivre une stratégie de "plus en plus" ; on suppose d'abord que toutes les variables sont indépendantes puis si on ne trouve pas de solution, on augmente le degré à k=2 et ainsi de suite jusqu'à trouver le bon k. Une autre stratégie est d'apprendre un GAI de taille exactement k, c'est-à-dire que toute les cliques du GAI appris possèdent la même taille. Puis une fois ce GAI appris, on essaye d'affiner localement les cliques afin de réduire la taille de la décomposition.

Une autre perspective intéressante serait de mettre en place pour les GAI un algorithme d'apprentissage actif proche de ceux utilisés dans l'apprentissage de CP-net. L'idée est de poser à l'utilisateur une série de questions, dont la réponse permet à l'apprenant de déduire des dépendances entre les variables [49]. Dans ce contexte, les équations et les variables du système linéaire se présenteraient seulement lorsque cela est nécessaire, ce qui conduit à une procédure beaucoup plus efficace en termes de mémoire.

Enfin, une dernière perspective intéressante serait d'appendre une GAI décomposition approchée de degré k lorsqu'il n'existe pas de représentation de degré k cohérente avec l'ensemble d'exemples. Pour cela, on peut imaginer l'introduction d'un ensemble de variables de relaxation ζ dans chaque membre gauche de chaque équation. La mise en place de cette relaxation n'est pas si simple puisqu'il faut introduire une constante de régularisation qui permet de trouver un compromis entre le nombre d'exemples "violés" et le degré maximum du GAI.

En ce qui concerne les PCP-nets, une perspective importante consiste à continuer les travaux commencés sur les CP-nets probabilistes généralisés. Ces travaux peuvent être découpés en deux sous-parties. Tout d'abord, il faudrait trouver comment répondre aux requêtes développées dans la section 6.5 page 121. Enfin, il serait intéressant d'étudier la complexité de ces requêtes.

Enfin, on laisse entrevoir deux perspectives intéressantes pour l'apprentissage de PCP-nets. La première perspective est à court terme et vise à expérimenter les différents algorithmes qui n'ont pas encore été testés. Une seconde perspective intéressante est d'essayer de transposer ces méthodes d'apprentissages au cas des CP-nets probabilistes généralisés.

Bibliographie

[1] Amo, S. D., Bueno, M. L. P., Alves, G. et da Silva, N. F. F. (2012). Cprefminer : An algorithm for mining user contextual preferences based on bayesian networks. *In Proceedings of the ICTAI*, pages 114–121. (Cité en page 103.)

[2] Angluin, D. (1988). Queries and concept learning. *Machine learning*, 2(4): 319–342. (Cité en page 30.)

[3] Bacchus, F. et Grove, A. (1995). Graphical models for preference and utility. *In Proceedings of UAI'95*, pages 3–10. (Cité en pages 14, 22 et 67.)

[4] Beal, M. J., Falciani, F., Ghahramani, Z., Rangel, C. et Wild, D. L. (2005). A bayesian approach to reconstructing genetic regulatory networks with hidden factors. *Journal of Bioinformatics*, 21(3):349–356. (Cité en page 59.)

[5] Ben-David, S., Cesa-Bianchi, N., Haussler, D. et Long, P. M. (1995). Characterizations of learnability for classes of {0,. . .,n}-valued functions. *Journal Of Computer and System Sciences*, 50:74–86. (Cité en page 91.)

[6] Bigot, D., Fargier, H., Mengin, J. et Zanuttini, B. (2013). Probabilistic conditional preference networks. *In Proceedings of the UAI*. (Cité en pages 17, 47, 103, 129 et 140.)

[7] Bigot, D., Fargier, H., Zanuttini, B. et Mengin, J. (2012). Using and Learning GAI-Decompositions for Representing Ordinal Rankings. (regular paper). *In* Fürnkranz, J. et Hüllermeier, E., éditeurs : *Proceedings of the ECAI'2012 workshop on Preference Learning (PL), Monpellier, 28/08/2012*, pages 5–10, http ://www.ke.tu-darmstadt.de/events/PL-12/workshop.html. TU Darmstadt. (Cité en pages 17, 67 et 129.)

[8] BIGOT, D., MENGIN, J. et ZANUTTINI, B. (2014). Learning probabilistic cp-nets from observations of optimal items. *In Proceedings of Stairs 2014*. (Cité en page 17.)

[9] BLUMER, A., EHRENFEUCHT, A., HAUSSLER, D. et WARMUTH, M. K. (1989). Learnability and the vapnik-chervonenkis dimension. *Journal ok the ACM*, 36(4):929–965. (Cité en pages 33 et 90.)

[10] BOOTH, R., CHEVALEYRE, Y., LANG, J., MENGIN, J. et SOMBATTHEERA, C. (2010). Learning conditionally lexicographic preference relations (regular paper). *In Proceedings of ECAI'10*, pages 269–274, http ://www.iospress.nl/. IOS Press. (Cité en pages 14, 39, 40, 42 et 129.)

[11] BOTTOU, L. (2004). Stochastic learning. *In Proceedings of the Advanced lectures on machine learning*, pages 146–168. (Cité en page 134.)

[12] BOUTILIER, C., BACCHUS, F. et BRAFMAN, R. I. (2001). Ucp-networks : A directed graphical representation of conditional utilities. *In Proceedings of UAI'01*, pages 56–64. (Cité en pages 14, 21 et 54.)

[13] BOUTILIER, C., BRAFMAN, R., HOOS, H. et POOLE, D. (1999). Reasoning with conditional ceteris paribus preference statem. *In Proceedings of UAI'99*, pages 71–80, San Francisco, CA. (Cité en pages 13 et 14.)

[14] BOUTILIER, C., BRAFMAN, R. I., DOMSHLAK, C., HOOS, H. H. et POOLE, D. (2004). Cp-nets : A tool for representing and reasoning with conditional ceteris paribus preference statements. *Journal of AIR*, 21:135–191. (Cité en pages 13, 14, 21, 44, 46, 47, 103, 109, 115 et 119.)

[15] BOYEN, X., FRIEDMAN, N. et KOLLER, D. (1999). Discovering the hidden structure of complex dynamic systems. *In Proceedings of the UAI*, pages 91–100. Morgan Kaufmann Publishers Inc. (Cité en page 59.)

[16] BRAFMAN, R. et DOMSHLAK, C. (2002). Tcp-nets for preference-based product configuration. *In Proceedings of the Fifth Workshop on Configuration, ECAI-2002*, pages 101–106. (Cité en pages 21 et 55.)

[17] BRAZIUNAS, D. (2005). Local utility elicitation in gai models. *In Proceedings of UAI'05*, pages 42–49. (Cité en pages 14, 67, 80 et 81.)

[18] BRÄUNING, M. et HÜLLERMEYER, E. (2012). Learning conditional lexicographic preference trees. *In Proceedings of ECAI'2012 workshop on Preference Learning (PL)*, pages 11–15. (Cité en pages 11, 39, 42 et 43.)

[19] CHEVALEYRE, Y. (2012). Learning gai networks. *In Proceedings of the DA2PL2012*. (Cité en page 98.)

[20] CHICKERING, D. M., HECKERMAN, D. et MEEK, C. (2004). Large-sample learning of bayesian networks is np-hard. *J. Mach. Learn. Res.*, 5:1287–1330. (Cité en page 59.)

[21] CORNELIO, C. (2012). Dynamic and probabilistic cp-nets. Mémoire de D.E.A., University of Padova. (Cité en pages 103 et 129.)

[22] CORNELIO, C., GOLDSMITH, J., MATTEI, N., ROSSI, F. et VENABLE, K. B. (2013). Updates and uncertainty in CP-nets. *In Proceedings of the Advances in Artificial Intelligence*, pages 301–312. (Cité en pages 15, 129 et 140.)

[23] COWELL, R., DAWID, A., LAURITZEN, S. et SPIEGELHALTER, D. (1999). Probabilistic networks and expert systems springer. *New York*. (Cité en page 78.)

[24] DARWICHE, A. (2009). *Modeling and reasoning with Bayesian networks*. Cambridge University Press. (Cité en page 57.)

[25] DEBREU, G. (1960). Topological methods in cardinal utility theory. *In* ARROW, K., KARLIN, S. et SUPPES, P., éditeurs : *Mathematical Methods in the Social Sciences*, pages 16–26. Stanford University Press, Stanford. (Cité en page 69.)

[26] DIMOPOULOS, Y., MICHAEL, L. et ATHIENITOU, F. (2009). Ceteris paribus preference elicitation with predictive guarantees. *In Proceedings of IJCAI 2009*, pages 1–6. (Cité en pages 14 et 129.)

[27] DOMBI, J., IMREH, C. et VINCZE, N. (2007). Learning lexicographic orders. *European Journal of Operational Research*, 183(2):748–756. (Cité en pages 39 et 42.)

[28] FISHBURN, P. (1970). *Utility Theory for Decision Making*. Wiley, New York. (Cité en pages 14 et 67.)

[29] FISHBURN, P. C. (1979). *Utility theory for decision making*. Publications in operations research. R. E. Krieger Pub. Co. (Cité en page 25.)

[30] FLACH, P. A. et MATSUBARA, E. T. (2007). On classification, ranking, and probability estimation. *Probabilistic, Logical and Relational Learning-A Further Synthesis*, 7161. (Cité en pages 14 et 37.)

[31] FLUM, J. et GROHE, M. (2006). *Parameterized Complexity Theory (Texts in Theoretical Computer Science. An EATCS Series)*. Springer-Verlag. (Cité en page 113.)

[32] FRASER, N. (1994). Ordinal preference representations. *Theory and Decision*, 36(1):45–67. (Cité en pages 14, 21, 37 et 39.)

[33] FRIEDMAN, N. (1998). The bayesian structural em algorithm. *In Proceedings of the UAI*, pages 129–138. Morgan Kaufmann Publishers Inc. (Cité en pages 59 et 60.)

[34] FRIEDMAN, N., NACHMAN, I. et PEÉR, D. (1999). Learning bayesian network structure from massive datasets : The &#171 ;sparse candidate&#171 ; algorithm. *In Proceedings of UAI*, UAI'99, pages 206–215, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc. (Cité en page 60.)

[35] FÜRNKRANZ, J. et HÜLLERMEIER, H., éditeurs (2011). *Preference learning*. Springer. (Cité en page 129.)

[36] GEIGER, D. et HECKERMAN, D. (1995). A characterization of the dirichlet distribution with application to learning bayesian networks. *In Proceedings of UAI*, UAI'95, pages 196–207, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc. (Cité en page 62.)

[37] GELLE, E. et WEIGEL, R. (1996). Interactive configuration using constraint satisfaction techniques. *In Proceedings of the PACT*, pages 37–44. (Cité en page 106.)

[38] GOLDSMITH, J., LANG, J., TRUSZCZYNSKI, M. et WILSON, N. (2008). The computational complexity of dominance and consistency in cp-nets. *Journal of AIR*, 33:403–432. (Cité en page 106.)

[39] GONZALES, C. et PERNY, P. (2004). Gai networks for utility elicitation. *In Proceedings of KR'04*, pages 224–234. (Cité en pages 13, 14, 67, 71, 80 et 81.)

[40] GONZALES, C., PERNY, P. et QUEIROZ, S. (2007). Réseaux GAI pour la prise de décision. *Revue d'Intelligence Artificielle*, 21(4):555–587. NAT LIP6 DECISION. (Cité en page 75.)

[41] GROSSMAN, D. et DOMINGOS, P. (2004). Learning bayesian network classifiers by maximizing conditional likelihood. *In Proceedings of ICML*, ICML '04, pages 46–, New York, NY, USA. ACM. (Cité en page 62.)

[42] GUERIN, J., ALLEN, T. et GOLDSMITH, J. (2013). Learning cp-net preferences online from user queries. *In* PERNY, P., PIRLOT, M. et TSOUKIÀS, A., éditeurs : *Algorithmic Decision Theory*, volume 8176 de *Lecture Notes in Computer Science*, pages 208–220. Springer Berlin Heidelberg. (Cité en pages 35, 47 et 51.)

[43] HECKERMAN, D., GEIGER, D. et CHICKERING, D. (1995). Learning bayesian networks : The combination of knowledge and statistical data. *Machine Learning*, 20(3):197–243. (Cité en page 60.)

[44] JANSEN, R., YU, H., GREENBAUM, D., KLUGER, Y., KROGAN, N. J., CHUNG, S., EMILI, A., SNYDER, M., GREENBLATT, J. F. et GERSTEIN, M. (2003). A bayesian networks approach for predicting protein-protein interactions from genomic data. *Journal of Science*, 302(5644):449–453. (Cité en page 57.)

[45] JENSEN, F. V. (1996). *An introduction to Bayesian networks*, volume 210. UCL press London. (Cité en pages 71 et 78.)

[46] JOACHIMS, T. (2002a). Optimizing search engines using clickthrough data. *In Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 133–142. ACM. (Cité en page 84.)

[47] JOACHIMS, T. (2002b). Optimizing search engines using clickthrough data. *In Proceedings of the ACM SIGKDD*, pages 133–142. (Cité en page 129.)

[48] KIM, H., LEE, J. K. et PARK, T. (2007). Boolean networks using the chi-square test for inferring large-scale gene regulatory networks. *Journal of BMC bioinformatics*, 8(1):37. (Cité en page 60.)

[49] KORICHE, F. et ZANUTTINI, B. (2009). Learning conditional preference networks with queries. *In Proceedings of IJCAI'09*, pages 1930–1935. (Cité en pages 14, 35, 47, 49, 51, 116, 129 et 141.)

[50] LANG, J. et MENGIN, J. (2009). The complexity of learning separable ceteris paribus preferences (regular paper). *In Proceedings of the IJCAI'09, Pasaden, Californie*, pages 848–853, http ://www.aaai.org/Press/press.php. AAAI Press. (Cité en pages 31, 47, 48, 52 et 129.)

[51] LARRANAGA, P., KUIJPERS, C., MURGA, R. et YURRAMENDI, Y. (1996). Learning bayesian network structures by searching for the best ordering with genetic algorithms. *Systems, Man and Cybernetics, Part A : Systems and Humans, IEEE Transactions on*, 26(4):487–493. (Cité en page 60.)

[52] LITTLESTONE, N. (1988). Learning quickly when irrelevant attributes abound : A new linear-threshold algorithm. *Machine learning*, 2(4):285–318. (Cité en page 30.)

[53] MAILHARRO, D. (1998). A classification and constraint-based framework for configuration. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, pages 383–397. (Cité en page 13.)

[54] MICHALSKI, R. S. et STEPP, R. E. (1983). Learning from observation : Conceptual clustering. *In Machine learning*, pages 331–363. Springer. (Cité en page 30.)

[55] NESTEROV, Y., NEMIROVSKII, A. et YE, Y. (1994). *Interior-point polynomial algorithms in convex programming*, volume 13. SIAM. (Cité en page 85.)

[56] OLIVER, R. M. et SMITH, J. Q. (1990). *Influence diagrams, belief nets, and decision analysis*. John Wiley & Sons Inc. (Cité en page 22.)

[57] ONO, C., KUROKAWA, M., MOTOMURA, Y. et ASOH, H. (2007a). A context-aware movie preference model using a bayesian network for recommendation and promotion. *In* CONATI, C., MCCOY, K. et PALIOURAS, G., éditeurs : *User Modeling 2007*, volume 4511 de *Lecture Notes in Computer Science*, pages 247–257. Springer Berlin Heidelberg. (Cité en page 15.)

[58] ONO, C., KUROKAWA, M., MOTOMURA, Y. et ASOH, H. (2007b). A context-aware movie preference model using a bayesian network for recommendation and promotion. *In User Modeling 2007*, pages 247–257. Springer. (Cité en page 57.)

[59] PEARL, J. (1988). *Probabilistic reasoning in intelligent systems : networks of plausible inference*. Morgan Kaufmann. (Cité en page 57.)

[60] RAO, A. S. et GEORGEFF, M. P. (1991). Modeling rational agents within a bdi-architecture. *In Proceedings of KR'92*, pages 473–484. (Cité en page 22.)

[61] ROSSI, F., VENABLE, K. B. et WALSH, T. (2004). mcp nets : Representing and reasoning with preferences of multiple agents. *In Proceedings of the 19th National Conference on Artifical Intelligence*, AAAI'04, pages 729–734. AAAI Press. (Cité en pages 21 et 56.)

[62] SCHIEX, T., FARGIER, H. et VERFAILLIE, G. (1995b). Valued constraint satisfaction problems : Hard and easy problems . *In Proceedings of IJCAI'95)*, pages 631–637. (Cité en page 13.)

[63] SCHIEX, T., FARGIER, H., VERFAILLIE, G. *et al.* (1995a). Valued constraint satisfaction problems : Hard and easy problems. *Proceedings of the IJCAI*, 95:631–639. (Cité en page 22.)

[64] SCHMIDT, N., FARGIER, H., LANG, J. et MENGIN, J. (2012). Issue-by-issue voting : an experimental evaluation (regular paper). *In Proceedings of the MPREF*, page (on line). LIP6. (Cité en page 94.)

[65] SCHMITT, M. et MARTIGNON, L. (2006). On the complexity of learning lexicographic strategies. *The Journal of Machine Learning Research*, 7:55–83. (Cité en page 39.)

[66] TORRES-TOLEDANO, J. G. et SUCAR, L. E. (1998). Bayesian networks for reliability analysis of complex systems. *In Progress in Artificial Intelligence—IBERAMIA 98*, pages 195–206. Springer. (Cité en page 57.)

[67] VADHAN, S. P. (2002). The complexity of counting in sparse, regular, and planar graphs. *SIAM J. Comput.*, 31(2):398–427. (Cité en page 117.)

[68] VALIANT, L. G. (1984). A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142. (Cité en pages 30 et 32.)

[69] VIAPPIANI, P., FALTINGS, B. et PU, P. (2006). Preference-based search using example-critiquing with suggestions. *Journal of AIR*, 27:465–503. (Cité en page 13.)

[70] WEBER, P., MEDINA-OLIVA, G., SIMON, C. et IUNG, B. (2012). Overview on bayesian networks applications for dependability, risk analysis and maintenance areas. *Engineering Applications of Artificial Intelligence*, 25(4):671–682. (Cité en page 57.)

[71] WILSON, N. (2009). Efficient inference for expressive comparative preference languages. *In Proceedings of the IJCAI*, pages 961–966. (Cité en page 22.)

[72] XU, J. et LI, H. (2002). Adarank : a boosting algorithm for information retrieval. *In Proceedings of the ACM SIGIR*, pages 391–398. (Cité en page 129.)

[73] YAMAN, F., WALSH, T. J., LITTMAN, M. L. et DESJARDINS, M. (2008). Democratic approximation of lexicographic preference models. *In Proceedings of the ICML*, pages 1200–1207. ACM. (Cité en page 39.)

Chapitre 9

Notations

Table 9.1 – *Notation*

Notion	Notation
Élement de base	
Variable	$A, B, \ldots, X, Y, Z(X_0, X_1, \ldots)$
Valeur d'une variable	$a_1, a_2, \ldots, a_n$
Fonction d'utilité	u
Énsemble d'élément de base	
Domaine d'une Variable	$\underline{A}, \underline{B}, \ldots, \underline{X}, \underline{Y}, \underline{Z}$
Un CP-net	N
Un CP-net probabiliste	$\mathcal{N}$
Un ensemble de variables	χ
Une instanciation complète de χ (un objet)	o, o', o_i
Produit cartésien des domaines de χ	$\underline{\chi}$
Ensemble des réels	$\mathbb{R}$
Relation Binaire	
Une relation binaire	R
Un ordre sur une préférence local ou sur $\mathbb{R}$	$\geq$
Un ordre strict sur une préférence local ou sur $\mathbb{R}$	$>$
Graphe	
Un graphe	G
Un ensemble d'arêtes/arcs	$\mathcal{E}$
Un ensemble de sommets/nœuds	$\mathcal{V}$

Un sommet/nœud	S, S'
Parent d'un nœud S	$Pa(S)$
Apprentissage	
Un exemple (paire ordonnées)	(o, R, o')
Un exemple d'objet optimal	o_{opt}
Un ensemble d'exemple	E
Taux d'apprentissage	η
Paramètre de confiance	δ
Paramètre d'erreur	ϵ
Nombres d'exemple	m
Préférence	
Relation de préférence large sur des objets	$\succsim$
Relation de préférence stricte sur des objets	$\succ$
Relation d'indifférence	$\sim$
Parents d'une variable X	$Pa(X)$
Une règle de préférence locale sur X	$(Xu : x > \bar{x})$

Résumé

La modélisation des préférences par le biais de formalismes de représentation compacte fait l'objet de travaux soutenus en intelligence artificielle depuis plus d'une quinzaine d'années. Ces formalismes permettent l'expression de modèles suffisamment flexibles et riches pour décrire des comportements de décision complexes. Pour être intéressants en pratique, ces formalismes doivent de plus permettre l'élicitation des préférences de l'utilisateur, et ce en restant à un niveau admissible d'interaction. La configuration de produits combinatoires dans sa version *business to customer* et la recherche à base de préférences constituent de bons exemples de ce type de problème de décision où les préférences de l'utilisateur ne sont pas connues *a priori*.

Dans un premier temps, nous nous sommes penchés sur l'apprentissage de GAI-décompositions. Nous verrons qu'il est possible d'apprendre une telle représentation en temps polynomial en passant par un système d'inéquations linéaires. Dans un second temps, nous proposerons une version probabiliste des CP-nets permettant la représentation de préférences multi-utilisateurs afin de réduire le temps nécessaire à l'apprentissage des préférences d'un utilisateur. Nous étudierons les différentes requêtes que l'on peut utiliser avec une telle représentation, puis nous nous pencherons sur la complexité de ces requêtes. Enfin, nous verrons comment apprendre ce nouveau formalisme, soit grâce à un apprentissage hors ligne à partir d'un ensemble d'objets optimaux, soit grâce à un apprentissage en ligne à partir d'un ensemble de questions posées à l'utilisateur.

Mots clefs : Représentation de préférences, apprentissage, multi-utilisateurs, CP-net probabiliste, GAI-décomposition