Algorithmes efficaces pour l'apprentissage de réseaux de préférences conditionnelles à partir de données bruitées

Fabien Labernia

THÈSE DE DOCTORAT

de l'Université de recherche Paris Sciences et Lettres
PSL Research University

Préparée à l'Université Paris-Dauphine

Algorithmes efficaces pour l'apprentissage de réseaux de préférences conditionnelles à partir de données bruitées

École doctorale de Dauphine – ED 543

Spécialité INFORMATIQUE

COMPOSITION DU JURY :

M. Jamal ATIF
Université Paris-Dauphine
Directeur de thèse

M. Brice MAYAG
Université Paris-Dauphine
Co-encadrant de thèse

M. Antoine CORNUEJOLS
AgroParisTech
Rapporteur

M. Frédéric KORICHE
Université d'Artois
Rapporteur

Mme Isabelle BLOCH
Télécom ParisTech
Présidente du jury

M. Bruno ZANUTTINI
Université de Caen Basse-Normandie
Examinateur

Soutenue par **Fabien LABERNIA**
le **27.09.2018**

Dirigée par **M. Jamal ATIF**

DAUPHINE UNIVERSITÉ PARIS | PSL RESEARCH UNIVERSITY PARIS

REMERCIEMENTS

Et bien voilà, quasiment quatre ans de thèse se terminent. Ce fut une superbe aventure que je n'oublierai jamais. Je ne veux pas vous faire patienter plus longtemps, commençons sans plus tarder les remerciements [1].

Je remercie tout d'abord très chaleureusement Frédéric Koriche, ainsi que Antoine Cornuejols, pour avoir accepté de rapporter, à plusieurs reprises, ce manuscrit. Vos nombreuses remarques n'ont fait qu'améliorer considérablement mon travail de recherche, et ma vision des différents problèmes abordés. Ce travail fut complexe, à plusieurs niveaux, mais le résultat final est plus que satisfaisant. Un grand merci également à Bruno Zanuttini, et à Isabelle Bloch, pour avoir accepté de faire partie de mon jury. Bruno, un grand merci pour nos nombreux échanges, qui m'ont énormément inspiré dans mes recherches, et pour ta bonne humeur permanente.

Ma deuxième vague de remerciements se dirige vers mes deux (co-)directeurs de thèse Jamal Atif et Brice Mayag. Jamal, tu as pris la décision de m'accepter en tant que nouveau doctorant alors que tu arrivais à peine au LAMSADE, sans vraiment me connaître. Tu m'as fait confiance, et pour cela je t'en serai à jamais reconnaissant. Un énorme merci pour ces –presque– quatre années d'encadrement, aussi bien scientifiquement qu'humainement. Tu as su me remotiver, m'encourager, et m'aider dans les périodes compliquées, avec des réunions de travail toujours très enrichissantes. Brice, un très très grand merci également pour toute ton aide. Malgré une orientation de la thèse s'éloignant un peu de ton domaine de prédilection, tu as toujours été présent, à l'écoute, je n'oublierai jamais nos nombreuses discussions et réunions.

Je veux également remercier Miguel Couceiro, sans qui cette thèse n'aurait probablement jamais pu avoir lieu. Un grand merci à Amedeo Napoli pour ses nombreuses aides et suggestions. Florian (Yger), nous n'avons pas pu travailler ensemble autant que nous l'aurions voulu. Malgré tout, je suis vraiment content de t'avoir connu, et

1. Les remerciements seront assez décousus : c'est un choix de ma part. Je veux éviter au maximum tout phénomène de privilège, car vous m'avez tous apporté, qui que vous soyez, quelque chose !

je suis sûr que les futures collaborations ne manqueront pas !

Je souhaite ensuite adresser un remerciement tout spécial à Olivier Raynaud, Maître de Conférences au LIMOS, laboratoire rattaché à l'Université Clermont Auvergne, qui m'a donné, il y a déjà six ans, l'envie de faire de la recherche grâce à ses cours d'algorithmique, et aux nombreux stages de recherche qui ont ponctué mon cursus universitaire. Je ne t'oublie pas, et j'espère que de nouvelles occasions de collaboration se présenteront. J'en profite également pour remercier mes anciens professeurs, qui m'ont tous apporté quelque chose : Yannick Loiseau, Nourine Lhouari, Engelbert Mephu NGuifo, Farouk Toumani, Anne Berry, et Hervé Kerivin.

Merci au LAMSADE, en particulier au personnel scientifique avec qui j'ai déjà pu discuter (Vangelis, Stefano, Meltem, Jérôme, Olivier, Benjamin, Virginie, Sonia, Daniela, Khalid, et beaucoup d'autres) et au personnel administratif (Juliette, Mireille, Eleni, Hawa, et Marie-Clotilde). Un grand merci à Igor pour l'ensemble de ses aides durant mon long processus de soumission. Mention spéciale à mon cher Olivier Rouyer avec qui j'ai passé un nombre incalculable d'heures à discuter de tout et de rien, et qui a été un excellent professeur en administration systèmes☺

Je garde bien entendu le meilleur pour la fin, tous mes camarades docteurs et doctorants ! Nous le savons tous, Paris-Dauphine manque cruellement de place, ce qui apporte beaucoup de contraintes, mais aussi de nombreux points positifs non négligeables : rares sont, je pense, les laboratoires possédant des doctorants aussi soudés que nous, où tout le monde se connait. Alors oui, le travail en pâtit bien souvent, mais la bonne humeur persistante est une source de motivation et d'envie permanente de venir travailler. Cette thèse n'aurait pas pu s'achever sans vous tous et votre soutien inconscient ! De ce fait, je souhaite remercier chaleureusement [2] : Thomas notre cher.e.s président.e.s des doctoranx (vive les genderfluides), Ian (les pots vont nous manquer !), Myriam (à quand la saison 8 de GoT ?), Khalil (*wild wolf biker* toujours motivé, tu vas me manquer), Céline (courage avec les compilations LaTeX), Ioannis (don't smoke too much !), Boris (ne joue pas trop), Yassine (embête pas trop tes voisins :p), Paul (faut passer nous voir de temps en temps !), Linda (as-tu retrouvé Henry Leroy ?), Raja (on prend soin de ton laurier), Justin (fais ta carte vitale !), Diana (continue to find compatible colors for your clothes), Amin (don't drink too much tea !), Rafael (c'était bien le Japon ?), Mehdi (faut pas prendre Olivier au sérieux :p), Romain (ça fait longtemps qu'on t'a pas vu !), Marek (ton optimisme nous manque ☺), Tom (vive Staline !), Nathanaël (c'est cool le Japon ?), Marcel (c'est quand que tu rev...ah mais tu es déjà de retour !), George (arrête de courir partout :p), Louis (tu dois passer plus souvent !), Manel (on se reverra aux journées du LAMSADE !), Oussama (profite de cette année à Dauphine !), et Sylvia (once again, congratulations for your wedding). Je pense aussi aux (déjà) anciens Youcef (j'espère que ton postdoc se passe bien), Sami (tu te sers encore de mes livres ?), Renaud (la brasserie te manque ?), Florian (vive Fontainebleau !), Edouard (où es-tu maintenant ?), Lydia (comment vas-tu ?), Lyes (ta musique classique manque au

2. Les remerciements ayant été écrits en deux fois, certains commentaires ont été reformulés pour être plus au goût du jour !

bureau), et Amine (j'espère que nous pourrons nous revoir). Tiens...il ne manquerait pas des gens ? Mais oui ! Je tiens à remercier tout spécialement tous les doctorants de mon bureau, avec qui j'ai passé quatre magnifiques années : Pedro (on est champions du monde haha !), Anaelle (nos commérages vont me manquer), Maude (j'espère que le nouveau venu va bien ☺), Satya (IRCJZLDBOSNFEDEX : Sauras-tu retrouver la technique de chifFrement utiLisée, et décrypteR le code ? Je te laisse chercher [3] ☺), Paul-Henri (je pense très fort à toi), Anne (que t'est-il encore arrivé ? Repose-toi, c'est bientôt ton tour ☺), Hiba (...aller on mange plus vite !), Hossein (be careful with your phone my friend !), Mehdi (learn French !), Stéphane (il est vraiment pratique ton Mac mini), Antoine (le DJ du labo ☺), Saeed (nouveau polythechnicien !), et Zahra (prends bien soin de Saeed ☺). Si des gens manquent à l'appel, tout d'abord je m'excuse, ensuite je vous remercie aussi, et enfin prévenez-moi, je ferai une V2 !

Petit aparté, je tiens à vous présenter toutes mes excuses pour les trop nombreuses soirées auxquelles je n'ai pas participé, et croyez-moi ça n'a rien à voir avec vous, c'est certainement l'un de mes plus gros regrets. Vous resterez tous dans mon cœur, merci pour tout ! Et même si je ne suis plus là, n'hésitez pas à me demander de l'aide par mail, je me ferai un plaisir de vous rendre service haha ☺

Un énorme merci, également, à mon meilleur ami Valer, pour tous les délires que l'on a eu, même de loin : sans toi je n'aurai peut-être pas non plus démarré cette aventure ! Vincent, on s'est un peu perdu de vue ces dernières années, mais crois-moi je ne t'oublie pas ! Renaud, merci aussi pour tes nombreux encouragements malgré mes nombreux désistements pour venir te voir, mais je ne désespère pas ☺ Merci aussi à Ludo Sensei, Tommy, Damien, et Piwi mes amis de toujours. Un grand 謝謝 à Kass, Marina, Zhilong, Xiaomao, Shiyuan, Yiheng, Siyu, Huyu, Tianliang, Weitian, Yang Gege, Ciel, Lin, Tianya, Hongyue, et à tous ceux que j'aurai oublié, 愛你們 !

J'ai dit le meilleur pour la fin ? Merci Papa, Maman pour votre soutien dans tous les domaines pour ces longues études que je ne regrette pas d'avoir faites ! Merci aussi à mon frère, Quentin, pour son expertise scientifique depuis le Japon. En espérant que toi aussi, tu aies l'occasion d'expérimenter ce merveilleux voyage qu'est la thèse. 我想寫一個很大的謝謝，阿姨，叔叔，你們是最棒的，你們經常鼓勵我！非常感謝舅媽，舅舅跟阿非，我很愛你們！如果沒有你們，這四年就會大不相同。我很快來見你們，後來我們大家一起喝茶！ Je fais également de grands « miaous » à mes deux amours Luke et Meimei 🐾

Pour finir, je ne serais jamais parti à Paris, une ville pas forcément attirante pour le provincial que je suis, sans une bonne raison, celle qui rythme ma vie depuis maintenant presque cinq ans : Zhidan, merci pour ton soutien et ton amour de tous les instants ! Cette année a été riche en événements, une nouvelle aventure démarre, différente de celle à laquelle nous pensions il y a un an, mais qui n'en demeure pas moins intéressante et pleine d'ambitions. 王芷丹，我超級開心我們在一起，謝謝妳，我愛妳。

3. Site web source et solution : `https://owncloud.lamsade.dauphine.fr/index.php/s/b3NIDRcO6gOl7R6`.

TABLE DES MATIÈRES

LISTE DES FIGURES

LISTE DES TABLEAUX

LISTE DES ALGORITHMES

INTRODUCTION

Sommaire

Les travaux en intelligence artificielle se sont énormément développés ces dernières années, notamment grâce à la croissance exponentielle des données personnelles. De nombreuses branches de l'informatique sont concernées par ces recherches, telles que l'apprentissage automatique, l'aide à la décision, la robotique, etc. Ainsi, il est possible de se rendre compte des avancées de tous ces travaux dans des situations aussi diverses que l'utilisation d'un smartphone (assistants personnels), d'un service de flux musicaux ou vidéos (recommandation de musiques et de films), d'un site d'e-commerce (recommandation de nouveaux articles), de la médecine (aide au diagnostic), des jeux vidéos (simulation de joueurs par un ordinateur), ou même de vote (mise en pratique de règles empêchant la manipulation des résultats). Cette liste n'est bien-entendu pas exhaustive et ces recherches en intelligence artificielle touchent de plus en plus de situations de la vie quotidienne, dont le point commun concerne l'apprentissage de préférences.

Contexte et motivations

Une préférence correspond classiquement à une comparaison entre deux objets prenant la forme « je préfère cet objet à celui-là ». Elles peuvent être obtenues de façon explicite avec l'expression de la préférence désirée telle que « je préfère ce parfum-ci à celui-là », ou implicite grâce à des actions ou des décisions prises sans expression réelle des préférences, à l'image d'articles similaires achetés ou non. Ces préférences implicites sont les plus nombreuses et sont très utilisées par les publicitaires pour cibler au mieux nos besoins : le choix d'un produit par rapport à un autre

implique une préférence pour le premier. Les actions, ou choix effectués peuvent donc être directement transformés en préférences. Aussi, le simple fait de ne pas prendre de décision peut impliquer une sorte de préférence. À titre d'exemple, en utilisant le moteur de recherche Google, le fait de cliquer sur le premier lien plutôt que sur le deuxième peut indiquer une préférence du premier par rapport au deuxième. Si le deuxième lien n'est jamais visité, cela peut signifier qu'il se trouvera être le moins préféré de tous (tout en gardant à l'esprit que les liens retournés impliquent une relation de préférence générée par un algorithme de recherche, la préférence étant alors conditionnée par la confiance donnée à l'algorithme exposant les résultats). Une telle vision des préférences permet de se rendre compte de leur nombre, et de ce que cela peut impliquer en stockage de données. Il est alors nécessaire de trouver un moyen de représenter et de sauvegarder efficacement toutes ces préférences.

Les préférences peuvent notamment être mises à profit pour aider les personnes dans leurs vies quotidiennes, en faisant ressortir les choses qu'elles pourraient préférer. Cette aide peut notamment être effectuée par des **systèmes de recommandations**. Un des problèmes posé est le suivant : sachant qu'une personne a acheté l'article A, que lui suggérer comme autre article B pouvant potentiellement lui plaire ? Une solution pourrait alors être de proposer des articles achetés par des utilisateurs ayant des goûts similaires avec cette personne. Cette idée de recommandation à base de mesure de similarité est également présente sur des sites comme Netflix pour les contenus vidéo, Deezer et Spotify pour les contenus audio. Ainsi, des titres musicaux ou des films pourront être proposés à un utilisateur en fonction de ses visionnages ou écoutes précédents, et plus son historique sera alimenté, plus la recommandation sera pertinente. Nous pouvons aussi citer les recommandations de personnes sur le réseau social professionnel LinkedIn, basées sur une distance variant en fonction de la proximité de chaque contact que l'on possède.

La recommandation peut être vue comme un moyen d'**induire** des règles de préférences à partir des exemples de préférences observés, qui pourraient alors se généraliser à des exemples qui n'ont pas encore été observés. On parle alors d'**apprentissage de préférences**, qui sera explicité dans le prochain chapitre au travers de trois grands types de problèmes illustrés comme suit :

1.
   - Un ensemble de modules de cours.
   - Un groupe d'étudiants caractérisés par leurs notes et possédant chacun un classement de ces modules de cours (par ordre d'importance).

   $\Rightarrow$ Déterminer un classement des modules de cours pour un étudiant ne faisant pas parti du groupe.
2.
   - Un ensemble ordonné de catégories de puissances.
   - Un ensemble de véhicules caractérisés par leur taille, leur couleur, etc, dont la catégorie de puissance est connue.

   $\Rightarrow$ Déterminer la puissance d'un véhicule nouvellement arrivé.
3.
   - Un groupe d'étudiants caractérisés par leurs notes.

⇒ Déterminer un classement de ces étudiants à partir de leurs caractéristiques.

Nous pouvons remarquer que les deux premiers exemples font intervenir deux ensembles de données : tout d'abord un ensemble d'**étiquettes** (les modules de cours, ainsi que les catégories de puissance), puis un ensemble d'**individus** (les étudiants) ou d'**objets** (les véhicules). Le troisième problème travaille uniquement sur un groupe d'étudiants. De plus, le deuxième problème introduit une notion d'ordre entre les différentes catégories de puissance, là où il n'est pas possible d'ordonner *a priori* les différents modules de cours sans connaitre les préférences de la personne. Nous verrons dans la suite de ce manuscrit que ces différences fondamentales entraînent une approche de résolution différente pour ces trois problèmes.

On distingue deux types de systèmes de recommandation :

1. Les systèmes basés sur l'étude des caractéristiques de chaque objet, faisant alors ressortir une structure de préférence, généralement graphique (arbre de décision, réseau bayésien, ...), pouvant être comprise par le grand publique, et sur laquelle il est possible de mener d'autres types de raisonnements ;
2. Les systèmes basés des fonctions mathématiques permettant de faire ressortir, aux moyens d'optimisation d'une fonction objectif, l'objet qui sera recommandé. Nous pouvons citer comme exemples les systèmes à base de factorisation de matrice, de bandits manchots, ou de similarité entre objets (nous référons le lecteur au livre *Recommender systems handbook* [RRS15] pour plus de précisions).

Ce deuxième type de système de recommandation, bien que très efficace, ne permet pas une explicabilité claire pour le grand public des recommandations ou des décisions effectuées en raison d'un manque de structure exploitable graphiquement, au contraire d'une structure lisible et graphique plus apte à être comprise par la majorité des personnes. Cette idée de proposer un système interprétable et explicable accessible à tous, et utilisable en pratique, a motivé notre recherche d'une structure de préférences intuitive capable de représenter et de raisonner sur des préférences complexes.

Notre choix s'est alors porté sur une structure appelée **réseau de préférences conditionnelles**, et souvent abrégé en **CP-net**. Nous verrons dans le chapitre suivant que cette structure propose un formalisme intuitif des préférences basé sur un graphe orienté permettant une factorisation des préférences au sein de **règles de préférences**. Elle permet, de plus, la représentation de préférences complexes basées sur une notion de conditionnement entre les caractéristiques d'objets. L'exemple classiquement cité pour représenter ce conditionnement est celui du choix de type de vin en fonction du contenu de l'assiette d'un client dans un restaurant. Ainsi, traditionnellement, le vin rouge est souvent suggéré par les restaurateurs afin d'accompagner une viande rouge, alors que le vin blanc est généralement préféré comme accompagnement d'un poisson. Une fois cette structure graphique apprise, il devient alors possible de travailler directement sur les préférences qu'elle représente

afin de mener de nouveaux types de raisonnements, ou d'effectuer de la fouille de préférences, comme nous l'introduirons à la fin de ce manuscrit.

L'une des particularités des CP-nets est le fait que les règles de préférences qu'ils contiennent s'expriment entre deux objets quasiment identiques. Seule une caractéristique les distingue, nous parlons alors de préférences *ceteris paribus*, ou « toutes choses étant égales par ailleurs ». Ce type de préférence possède l'avantage d'être très simple à appréhender. Il fait cependant parti des limites fortes de ce modèle, car les préférences *ceteris paribus* semblent être difficiles à observer en pratique.
Les CP-nets nous semblent cependant être des structures adéquates pour servir de base à des systèmes de recommandation utilisant leurs propriétés afin de recommander de nouveaux objets. Nous présentons un exemple de CP-net dans la Figure 1.

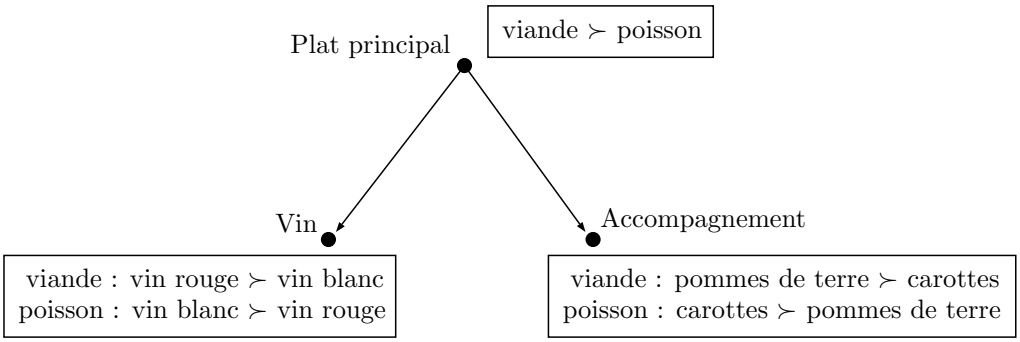


FIGURE 1 – Exemple de CP-net représentant les préférences d'un repas au restaurant. Le symbole « ≻ » peut être traduit par « est préféré à ».

Les travaux de cette thèse de doctorat s'inscrivent ainsi comme une étude préliminaire aux systèmes de recommandations, avec comme support de travail principal les réseaux de préférences conditionnelles, qui ont été pensés en premier lieu comme un moyen de représenter efficacement les préférences conditionnelles, mais qui, grâce à leur structure graphique, pourraient également être utilisés à des fins de recommandations, une fois appris correctement (voir la Figure 2 ci-dessous). Leur utilisation dans la vie courante suppose cependant de pouvoir prendre en compte des problématiques pratiques.

Les défis à relever

L'apprentissage de préférences est un important domaine en forte expansion compte tenu du grand nombre de préférences récoltées sur la toile. L'automatisation de la création de telles procédures est alors nécessaire, notamment en induisant des règles de préférences censées représenter les avis d'une (ou d'un groupe de) personne(s).

De nombreux travaux traitent des CP-nets. Il s'agit principalement d'études théoriques relatives à leur complexité, et leur implémentation a donné lieu à peu

FIGURE 2 – Groupement d'utilisateurs en fonction de leurs ressemblances afin d'établir des **profils types** pour une recommandation plus précise.

d'algorithmes efficaces testés sur de vraies données. L'objectif de cette thèse est donc d'apporter des réponses à cette insuffisance, en proposant un algorithme d'apprentissage de CP-nets permettant de résoudre deux types de problèmes :

1. Un traitement du **bruit** présent dans les préférences ;
2. Son **passage à l'échelle**.

La première problématique concerne le **caractère bruité** des préférences prenant la forme de préférences erronées, c'est-à-dire de préférences ayant subit une transformation avant d'être observées. Les causes de ce bruit sont multiples :

- Des préférences **contradictoires** au sein d'une base de données multi-utilisateurs, dont la cause de ces contradictions est due à des opinions divergentes entre les utilisateurs, entraînant alors la présence simultanée d'une préférence et de son inverse ;
- Des **erreurs** lors de la **transmission** des préférences, dont la cause peut être par exemple due à l'utilisation d'un mauvais réseau, ou d'une mauvaise connexion à Internet, entraînant une inversion de la préférence initialement envoyée.

La conséquence d'un tel bruit dans les bases de données est, dans notre cas, identique, quelque soit la cause sélectionnée : la présence de **cycles** au sein des préférences (je préfère aussi bien le premier objet au deuxième, que le deuxième au

premier) à cause des préférences bruitées, ce qui entraîne des incohérences. Les algorithmes d'apprentissage auront alors pour objectif d'éviter de telles préférences. Notons que le modèle de bruit étudié dans ce manuscrit n'implique aucune suppression et/ou modifications des données, il ne fait qu'inverser, de manière aléatoire, certaines préférences entre deux objets. L'apprentissage en milieu bruité reste une question encore peu traitée dans l'apprentissage de CP-nets, et constitue l'une des deux principales problématiques de cette étude.

La deuxième problématique concerne l'apprentissage de CP-nets à partir d'un grand nombre de préférences, récoltées par exemple lors d'une utilisation quotidienne d'applications ou de visites sur des sites en ligne. Cet apprentissage requiert alors de s'assurer d'un **passage à l'échelle** de l'algorithme, c'est-à-dire, d'être capable de traiter le plus rapidement possible chaque préférence d'une base de données en contenant des millions. Cette propriété, de plus en plus demandée dans les méthodes actuelles, est peu utilisée dans l'apprentissage de CP-nets. Nous allons donc concevoir un algorithme en ligne pouvant traiter un nombre infini d'observations, dont la structure apprise est exploitable à tous moments.

En résumé, nous proposons un algorithme d'apprentissage d'une structure de préférences compacte et intuitive, à partir d'un grand nombre de préférences pouvant être bruitées. L'accent sera mis sur un nombre important d'expériences prouvant empiriquement l'intérêt des algorithmes proposés. Ces expériences seront effectuées aussi bien sur des données synthétiques (*i.e.*, générées par ordinateur), que réelles (grâce à plusieurs jeux de données récupérés sur Internet). Les résultats sur les données réelles seront en outre étudiés afin d'expliquer en quoi les limites des CP-nets empêchent toujours, après plusieurs années de développement et de recherche, leur utilisation à des fins de recommandations.

La Figure 3 résume les différentes étapes présentées précédemment : les différentes préférences (potentiellement bruitées) sont récupérées. Elles sont ensuite généralisées au sein de règles de préférences, qui seront alors apprises par un algorithme afin de générer un réseau de préférences conditionnelles, qui peut par la suite être utilisé à des fins de recommandations, ou être utilisé pour mener d'autres types de raisonnements, comme de la fouille de préférences ou de la déduction de nouvelles préférences.

Plan

Ce manuscrit se découpera en trois chapitres principaux (un chapitre d'état de l'art et deux chapitres de contributions), et sera accompagné d'une annexe traitant de travaux initiés durant la thèse et n'ayant pas encore aboutis à des résultats définitifs.

Le Chapitre 1 est consacré à la notion d'apprentissage de préférences. Il introduira les principaux problèmes liés à ce domaine : le classement d'étiquettes, d'exemples, et d'objets. Il se tournera ensuite vers les principales méthodes de résolution de

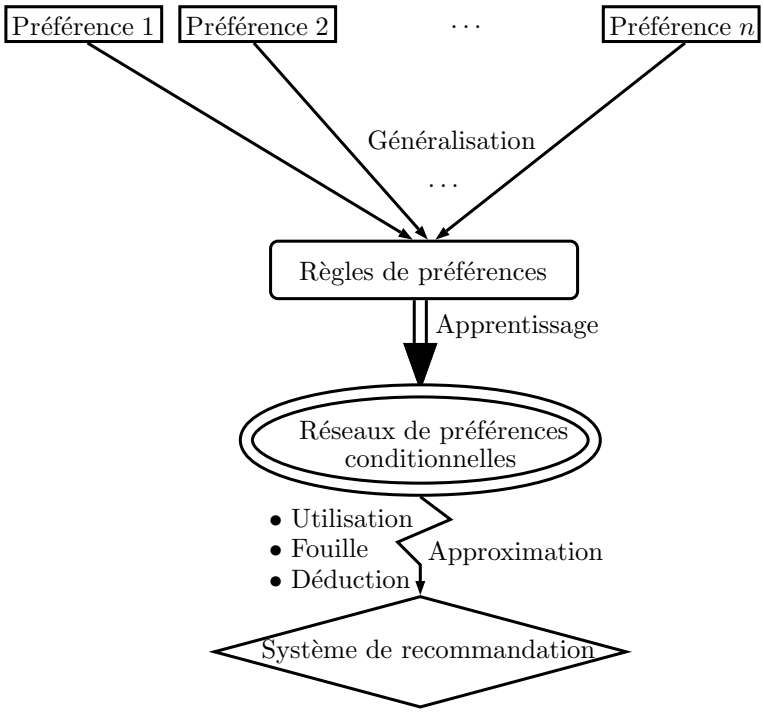


FIGURE 3 – Fonctionnement général d'une procédure de création d'un système de recommandation à travers l'utilisation d'un CP-net.

tels problèmes. Le cas spécifique des CP-nets y sera abordé, avec une introduction formelle de ces structures, de leurs extensions les plus connues, ainsi que de quelques algorithmes permettant de les apprendre.

Notre première contribution à l'apprentissage de CP-nets est l'objet du Chapitre 2. Nous présenterons ici un algorithme hors ligne apprenant ces structures via des requêtes à partir de préférences bruitées. Cet algorithme sera capable d'apprendre efficacement des CP-nets même en présence de contradictions au niveau des préférences, répondant ainsi à l'un de nos objectifs. Nous démontrerons l'efficacité de notre algorithme à travers des expérimentations sur des données réelles et synthétiques. Nous le comparerons également à quelques algorithmes de la littérature.

Le Chapitre 3 présentera notre deuxième contribution permettant de répondre aux deux objectifs fixés. Nous proposerons un algorithme d'apprentissage décliné en une version hors ligne, capable d'apprendre efficacement un CP-net à partir de préférences bruitées, et en ligne, capable de passer à l'échelle, tout en maintenant une certaine efficacité d'apprentissage. Tout comme au chapitre précédent, son efficacité sera abordé à travers plusieurs expériences sur des données synthétiques et réelles.

Nous terminerons avec l'Annexe A, représentative de nombreuses séances de travail initiées pendant cette thèse, et n'ayant pas encore abouties à des résultats assez pertinents pour être publiés en l'état. Elle abordera une vision légèrement différente des CP-nets, avec une étude plus orientée théorie des graphes et des treillis. Nous

exposerons notamment quelques transformations structurelles mettant en lumière certaines propriétés des CP-nets relatives à la suppression de variables notamment. Nous essayerons également de créer des liens avec l'analyse formelle des concepts, et plus particulièrement sur les contextes de préférence.

CHAPITRE 1

APPRENTISSAGE DE PRÉFÉRENCES

Sommaire

Résumé

L'apprentissage de préférences [FH10a] constitue l'un des nombreux nœuds de l'aide à la décision et de l'apprentissage automatique. Ses objectifs sont multiples, et touchent de nombreux domaines allant de la qualité des résultats renvoyés par les moteurs de recherche, jusqu'aux recommandations des sites d'e-commerce, grâce à l'utilisation de préférences récoltés de façon explicite (via l'expression directe d'une préférence de la part des utilisateurs), et implicite (via les choix effectués lors de l'utilisation quotidienne de logiciels). Dans ce chapitre, nous commencerons par introduire les principales notations utilisées dans ce manuscrit. Puis nous présenterons trois des grands problèmes de l'apprentissage de préférence, avec le classement d'étiquettes, le classement d'exemples, et le classement d'objets. Nous nous pencherons ensuite sur une structure de préférences qui a reçu une attention particulière cette dernière décennie, à savoir le réseau de préférences conditionnelles. Ce réseau est représenté par un graphe orienté où chaque sommet correspond à une caractéristique d'un objet (appelée ici variable). Ces sommets sont de plus accompagnés d'une table représentant l'ordre total des préférences sur les valeurs de leurs caractéristiques. Nous étudierons en détail cette structure, en exposant rapidement ses différentes extensions et généralisations. Nous nous focaliserons ensuite sur le problème d'apprentissage des CP-nets, vu comme un problème de classement d'objets (nous verrons qu'il est possible, à partir d'un CP-net, de déduire un ordre sur les objets), en décrivant quelques algorithmes d'apprentissage de cette structure, avec leurs forces et faiblesses. Enfin, nous proposerons le modèle principal étudié sur lequel reposeront nos contributions.

1.1 Introduction

Depuis maintenant une dizaine d'années, le sujet des préférences prend de plus en plus d'importance au sein des grandes conférences en intelligence artificielle. L'ensemble de ces travaux a alors été regroupé dans ce que l'on nomme « apprentissage de préférences ». Bien que les contours de ce nouveau sous-domaine de l'apprentissage automatique soient encore flous, il est possible le décrire de la manière suivante : l'apprentissage de préférences est un ensemble de méthodes d'apprentissage supervisées, c'est-à-dire de méthodes se basant sur un ensemble d'observations afin d'induire des informations ou une structure particulière. La particularité de ces méthodes est le fait que ces observations, récupérées implicitement ou explicitement, révèlent des informations sur les préférences d'un individu (classiquement, l'utilisateur d'un système informatique). L'objectif de telles méthodes est alors d'arriver à généraliser les préférences observées.

Le nombre de préférences en libre accès ayant connu une forte hausse ces dernières années, il est nécessaire de développer des algorithmes qui passent à l'échelle. De nombreux problèmes issus de l'apprentissage de préférences existent, il est cependant possible de les placer, pour la plupart, au sein de trois grands paradigmes. On

dit qu'un **objet** est caractérisé par un ensemble d'attributs (par exemple, un étudiant sera caractérisé par un ensemble de notes obtenues à l'issue de ses examens). Chaque objet peut alors être associé à une donnée, appelée ici **étiquette** (chaque étudiant peut par exemple est associé à un module de cours spécifique, ou à plusieurs modules), correspondant à la classe potentielle de l'objet. Enfin, un **exemple** correspond à l'observation d'un objet associé à son étiquette.

Nous avons illustré, dans l'introduction générale, ces paradigmes par trois exemples concrets. Chacun de ces exemples est une instanciation d'un paradigme plus général :

1. Problème du classement de modules de cours pour un nouvel étudiant : classement d'étiquettes [VG10] qui associe à chaque objet un ordre total sur un ensemble d'étiquettes ;
2. Problème d'attribution d'une catégorie de puissance d'un nouveau véhicule : classement d'exemples [HLY08] qui associe à chaque objet une étiquette prise parmi un ensemble totalement ordonné d'étiquettes (problème de classification) ;
3. Problème de classement d'étudiants : classement d'objets [KKA10] qui ordonne totalement un ensemble d'objets grâce à leur vecteur de caractéristiques (problème de régression).

Il existe plusieurs modèles de représentations de préférences. Ils se distinguent entre-eux pour les propriétés qu'ils vérifient (transitivité, ordre, etc...) [BL04, Raj79, San07, Tso08, Wal07]. Nous avons choisi de nous intéresser aux réseaux de préférences conditionnelles (ou CP-nets) [BBD$^+$04] pour leur compacité et leur capacité à représenter un grand nombre de préférences uniques, ainsi que pour leur interprétabilité et leur utilisation des préférences entre objets proches. Ces structures restent encore maintenant étudiées, aussi bien théoriquement [AMZ16] qu'algorithmiquement [AGJ$^+$17]. L'apprentissage de ces structures reste cependant difficile [CKL$^+$10] à cause de leur forte combinatoire (le nombre de règles de préférences pouvant être représentées est exponentiel en le nombre de valeurs différentes que chaque variable peut prendre). Nous présenterons ci-dessous quelques algorithmes d'apprentissage de telles structures [KZ10, GAG13, LXW$^+$14].

1.2 Préliminaires

Cette section présente les notions de base nécessaires à la bonne compréhension de la suite de ce manuscrit.

1.2.1 Notations préliminaires

Définition 1.1 (Variable, domaine, objet, état)**.**
Soit $\mathbf{V} = \{V_1, V_2, \ldots, V_n\}$ *un ensemble de* n ***variables****. Chaque variable* $V_i \in \mathbf{V}$ *est associée à un* ***domaine*** *de valeurs* $Dom(\{V_i\}) = \{v_{i1}, \ldots, v_{im_i}\}$*, où* m_i *correspond*

*au nombre de valeurs prises pour la variable V_i (par la suite, lorsqu'aucune confusion n'est possible, nous omettrons les accolades de $Dom(\{V_i\})$ en notant simplement $Dom(V_i)$). Soit $\mathbf{V'} = \{V_1, \ldots, V_k\} \subseteq \mathbf{V}$ un sous-ensemble de $\mathbf{V}$, nous notons par $Dom(\mathbf{V'}) = Dom(V_1) \times \ldots \times Dom(V_k)$ le domaine des valeurs possibles de toutes les variables de $\mathbf{V'}$. On appelle **objet** (resp. **état**) $\mathbf{x} \in Dom(\mathbf{V})$ (resp. $\mathbf{x} \in Dom(\mathbf{V'})$) un vecteur de valeurs de toutes les variables de $\mathbf{V}$ (resp. d'un sous-ensemble $\mathbf{V'} \subseteq \mathbf{V}$).*

Nous introduisons l'exemple suivant qui servira de fil conducteur aux deux premiers chapitres de ce manuscrit, et illustrera les différentes notions qui y seront abordées.

Exemple 1.1. *Plaçons nous dans le contexte suivant : quelles seraient les préférences vestimentaires d'une personne en fonction d'une activité donnée ? Pour cela, nous définissons 3 variables descriptives binaires :*

- *La variable A pour **activité**, pouvant prendre deux valeurs ma et sp qui désignent respectivement le fait d'aller à un mariage, et le fait d'aller faire du sport. Cela signifie que $Dom(A) = \{ma, sp\}$;*
- *La variable H pour le **haut de vêtement** correspondant au vêtement porté par la personne pour le haut du corps. Elle prend deux valeurs tsh (tee-shirt) et ch (chemise). Nous avons alors $Dom(H) = \{tsh, ch\}$;*
- *La variable B pour le **bas de vêtement** correspondant à ce que portera la personne pour le bas de corps. On considérera $Dom(B) = \{sh, pa\}$ avec sh correspondant au short, et pa correspondant au pantalon.*

Pour résumer, on aura $\mathbf{V} = \{A, H, B\}$ détaillé comme suit :

Variable	Valeurs
A : Activité	sp : sport ma : mariage
H : haut de vêtement	tsh : t-shirt ch : chemise
B : bas de vêtement	sh : short pa : pantalon

Il est alors possible de définir un état $\mathbf{x} = (ma, pa)$ signifiant qu'une personne se rend à un mariage en portant un pantalon, et un objet $\mathbf{o} = (ma, ch, pa)$ signifiant qu'une personne se rend à un mariage en portant une chemise et un pantalon.

Définition 1.2 (Concaténation, extensions, projection)**.**
*Soient deux états $\mathbf{x} \in Dom(\mathbf{X})$ et $\mathbf{y} \in Dom(\mathbf{Y})$ avec $\mathbf{X}, \mathbf{Y} \subseteq \mathbf{V}$ et $\mathbf{X} \cap \mathbf{Y} = \emptyset$. Nous introduisons la notation $\mathbf{xy}$ comme étant la **concaténation** des deux états $\mathbf{x}$ et $\mathbf{y}$. Cette concaténation forme alors un nouvel état $\mathbf{u} = \mathbf{xy} \in Dom(\mathbf{X} \cup \mathbf{Y})$. $x\mathbf{y}$ désigne l'**extension** d'un état $\mathbf{y} \in Dom(\mathbf{Y})$ par la valeur $x \in Dom(X)$ de la variable X.*

Finalement, nous notons par $\mathbf{o}[\mathbf{X}]$ *la* ***projection*** *de l'objet* $\mathbf{o}$ *sur les variables de* $\mathbf{X}$*, et* $\bar{\mathbf{o}}[\mathbf{X}]$ *la projection de l'inverse des valeurs de l'objet* $\mathbf{o}$ *sur les variables de* $\mathbf{X}$ [1]*.*

Exemple 1.2. *Reprenons l'Exemple 1.1 page 12 et considérons le sous-ensemble de variables* $\mathbf{X} = \{A, B\} \subset \mathbf{V}$*. Soit* $\mathbf{o} = (ma, ch, pa)$*, alors la projection de* $\mathbf{o}$ *sur le sous-ensemble* $\mathbf{X}$ *est notée* $\mathbf{o}[\mathbf{X}] = (ma, pa)$*, et la projection de* $\bar{\mathbf{o}}$ *sur le sous-ensemble* $\mathbf{X}$ *est notée* $\bar{\mathbf{o}}[\mathbf{X}] = (sp, sh)$*. De plus, une concaténation* $\mathbf{x}ch = (ma, ch, pa)$ *correspond à l'ajout de la valeur* $ch \in Dom(H)$ *sur l'état* $\mathbf{x} = (ma, pa) \in Dom(\mathbf{X})$ *(remarquons que cette concaténation correspond également à l'extension de* $\mathbf{x}$ *par la valeur* ch*).*

Soit $\mathbf{o} \in Dom(\mathbf{V})$ un objet et x une valeur de la variable $X \in \mathbf{V}$. Nous introduisons également la notation $\mathbf{o}[x]$ signifiant que l'objet $\mathbf{o}$ prend la valeur x à la place de la valeur d'origine de la variable X. Plus généralement, nous notons par $\mathbf{o}[\mathbf{x}]$ l'objet $\mathbf{o}$ possédant les valeurs de l'état $\mathbf{x} \in Dom(\mathbf{X})$, avec $\mathbf{X} \subseteq \mathbf{V}$.

Exemple 1.3. *Reprenons l'Exemple 1.1 page 12 :*

Variable	Valeurs
A : Activité	sp : sport ma : mariage
H : haut de vêtement	tsh : t-shirt ch : chemise
B : bas de vêtement	sh : short pa : pantalon

Soit $\mathbf{o} = (sp, tsh, sh)$ *un objet, et* $\mathbf{x} = (ma, pa) \in Dom(A, B)$ *un état. Nous obtenons alors* $\mathbf{o}[\mathbf{x}] = (ma, tsh, pa)$*.*

Enfin, nous introduisons quelques notions élémentaires de théorie des graphes. On appelle ainsi **graphe** G une paire d'ensembles $(\mathbf{S}, \mathbf{A})$ de **sommets** $\mathbf{S} = \{s_1, \ldots, s_n\}$ et d'**arêtes** $\mathbf{A} = \{a_1, \ldots, a_m\}$, où $a_i = (s_j, s_k)$, avec $s_j, s_k \in \mathbf{S}$. Un graphe est dit **non orienté** si $(s_j, s_k) \in \mathbf{A} \Leftrightarrow (s_k, s_j) \in \mathbf{A}$. Dans le cas contraire, on parle de **graphe orienté** (ou graphe dirigé), et l'ensemble des arêtes est alors nommé ensemble d'**arcs**.

Dans un graphe (orienté ou non), un **cycle** est une suite d'arêtes ou d'arcs consécutifs dont les deux sommets situés aux extrémités sont identiques. Un graphe possédant au moins un cycle en son sein est appelé **graphe cyclique**. Un graphe n'en possédant pas est quant à lui appelé **graphe acyclique**.

1.2.2 Relation binaire, d'ordre, et de préférences

Soit $\mathbf{V} = \{V_1, \ldots, V_n\}$ un ensemble de n variables.

1. Cette notation est utilisée dans un cadre où chaque $X \in \mathbf{X}$ est binaire, *i.e.*, $|Dom(X)| = 2$, $\forall X \in \mathbf{X}$.

Définition 1.3 (Relation binaire)**.**
On appelle ***relation binaire*** *une relation R sur un ensemble d'objets $\mathbf{O} = \{\mathbf{o}_1, \ldots, \mathbf{o}_m\}$ un sous-ensemble du produit cartésien $R \subseteq Dom(\mathbf{V})^2$. Lorsqu'une relation existe entre $\mathbf{o} \in Dom(\mathbf{V})$ et $\mathbf{o}' \in Dom(\mathbf{V})$, nous la noterons $\mathbf{o}R\mathbf{o}'$.*

Étudions maintenant les quelques propriétés d'une telle relation.

Définition 1.4 (Propriétés d'une relation binaire)**.**
Soient $\mathbf{V}$ un ensemble de variables et R une relation binaire sur $Dom(\mathbf{V})^2$, avec $\mathbf{V}$ un ensemble de variables. Cette relation est dite :

1. ***Réflexive*** *si* $\mathbf{o}R\mathbf{o},\ \forall \mathbf{o} \in Dom(\mathbf{V})$ *;*
2. ***Complète*** *ou* ***totale*** *si* $\mathbf{o}R\mathbf{o}'$ *ou* $\mathbf{o}'R\mathbf{o},\ \forall \mathbf{o}, \mathbf{o}' \in Dom(\mathbf{V})$ *;*
3. ***Transitive*** *si* $\mathbf{o}R\mathbf{o}'$ *et* $\mathbf{o}'R\mathbf{o}'' \Rightarrow \mathbf{o}R\mathbf{o}'',\ \forall \mathbf{o}, \mathbf{o}', \mathbf{o}'' \in Dom(\mathbf{V})$ *;*
4. ***Antisymétrique*** *si* $\mathbf{o}R\mathbf{o}'$ *et* $\mathbf{o}'R\mathbf{o} \Rightarrow \mathbf{o} = \mathbf{o}',\ \forall \mathbf{o}, \mathbf{o}' \in Dom(\mathbf{V})$ *;*
5. ***Symétrique*** *si* $\mathbf{o}R\mathbf{o}' \Rightarrow \mathbf{o}'R\mathbf{o},\ \forall \mathbf{o}, \mathbf{o}' \in Dom(\mathbf{V})$ *;*
6. ***Asymétrique*** *si* $\mathbf{o}R\mathbf{o}' \Rightarrow \neg(\mathbf{o}'R\mathbf{o}),\ \forall \mathbf{o}, \mathbf{o}' \in Dom(\mathbf{V})$.

Il est alors possible de définir plusieurs types d'ordres à partir de telles propriétés.

Définition 1.5 (Ordres et relation de préférence)**.**
Soient $\mathbf{V}$ un ensemble de variables et R une relation binaire sur $Dom(\mathbf{V})$. Cette relation peut être :

1. *Un* ***préordre*** *si R est réflexive et transitive ;*
2. *Un* ***préordre total*** *(ou* ***complet****) si R est totale (ou complète), réflexive, et transitive ;*
3. *Un* ***ordre*** *si R est réflexive, antisymétrique et transitive ;*
4. *Un* ***ordre total*** *(ou* ***ordre complet****) si R est total (ou complète), réflexive, antisymétrique et transitive ;*
5. *Une* ***relation d'équivalence*** *si R est réflexive, symétrique et transitive.*

La relation de ***préférence*** *est un préordre sur un ensemble de variables $\mathbf{V}$ noté $\succsim$, dont la partie asymétrique de $\succsim$ sera notée $\succ$ et appelée* ***préférence stricte****, et la partie symétrique sera notée $\sim$ et appelée* ***indifférence****.*

De la définition précédente découle alors naturellement la notion de **classement** (ou rangement).

Définition 1.6 (Classement/rangement)**.**
Soit $\mathbf{V}$ un ensemble de variables. On appelle ***classement*** *(ou* ***rangement****) un ordre total dont la relation de préférence est décrite par rapport aux préférences d'une personne, d'une structure ou d'un ensemble. Notons U cette entité, on écrit alors $\mathbf{o} \succ_U \mathbf{o}'$, pour $\mathbf{o}, \mathbf{o}' \in Dom(\mathbf{V})$ et signifiant que $\mathbf{o}$ est strictement préféré à $\mathbf{o}'$ pour l'entité U.*

1.2.3 Les modèles d'apprentissage supervisé étudiés

Trois types d'apprentissage distincts seront abordés dans ces travaux. Nous les détaillons dans cette section.

Un problème d'apprentissage supervisé se compose d'un **espace d'observations**, que l'on notera $\mathbf{O}$, ainsi qu'un **ensemble de sorties**, aussi appelé **classes**, que l'on notera Y. On appelle ainsi tout élément $z \in \mathbf{Z} = \mathbf{O} \times Y$ un **exemple**, et toute séquence $S = \{z_1, \ldots, z_m\}$ est appelée **ensemble d'entraînement**. Une observation $\mathbf{o} \in \mathbf{O}$ est décrite par un ensemble de n variables, et l'ensemble des sorties (ou classes) est ici décrit par 0 ou 1.

En supplément de ces deux ensembles $\mathbf{O}$ et Y, un problème d'apprentissage supervisé contient également une **classe d'hypothèses**, notée $\mathcal{H} = \{h \mid h : \mathbf{O} \to Y\}$. Chaque fonction $h : \mathbf{O} \to Y$ est appelée **hypothèse**, et permet d'associer à chaque observation $\mathbf{o}$, une classe prédite notée $\hat{y}$. L'apprenant cherche ainsi à prédire son environnement à l'aide d'une hypothèse $h \in \mathcal{H}$. Son objectif est alors de trouver, à partir d'un ensemble d'entraînement S, l'hypothèse $h_S \in \mathcal{H}$ (signifiant que l'hypothèse h a été déterminée à partir de l'ensemble S) la plus proche de l'hypothèse cible $h^* \in \mathcal{H}$.

Exemple 1.4. *Plaçons nous dans un contexte de cinéma, où l'objectif est de déterminer les films qu'un individu pourrait aimer. Les observations correspondront ainsi à des films décrits par leurs genres, et une sortie (ou classe) pourra être* 1 *pour « j'aime ce film », ou* 0 *pour « je n'aime pas ce film ». Une observation pourrait alors être* $\mathbf{o} = (\textit{romantique, fantastique, horreur})$*, associée à la classe* 1*. Un exemple serait donc* $(\mathbf{o}, y) = ((\textit{romantique, fantastique, horreur}), 1)$*.*
L'ensemble d'apprentissage serait donc un ensemble de m *exemples* $S = \{(\mathbf{o}_1, y_1), (\mathbf{o}_2, y_2), \ldots, (\mathbf{o}_m, y_m)\}$*, et une hypothèse* h_S *correspondra à une fonction (parmi un ensemble de fonctions* $\mathcal{H}$*) capable d'associer, à tout film* $\mathbf{o}$*, le fait de l'apprécier (*$\hat{y} = 1$*) ou non (*$\hat{y} = 0$*).*

Apprentissage hors ligne

Un algorithme d'apprentissage hors ligne désigne un algorithme ayant un accès permanent à la base d'entraînement. La phase d'apprentissage peut donc s'effectuer à l'aide de plusieurs parcours de cette base. Ainsi, la performance de chaque hypothèse $h_S \in \mathcal{H}$ (pour un ensemble d'entraînement S donné) est mesurée à l'aide d'une fonction de perte $\ell : \mathcal{H} \times \mathbf{O} \times Y \to \mathbb{R}^+$. Cette fonction permet d'évaluer l'écart entre la prédiction $\hat{y}$ de $h(\mathbf{o})$, et la classe réelle y d'une observation $\mathbf{o}$. Il est également possible de calculer la performance d'une hypothèse sur l'ensemble d'entraînement $S = \{z_1, \ldots, z_m\}$ à l'aide d'une perte cumulée, notée $L(h_S, S) = \sum_{i=1}^{m} \ell(h_S, z_i)$. Un bon apprentissage correspond donc à un apprentissage minimisant cette perte cumulée.

Soit S une base d'entraînement. Il existe de nombreuses mesures de perte différentes, dont l'utilité dépend du problème à résoudre. Nous pouvons par exemple

citer la perte $0-1$ définie par

$$\ell_{0-1}(h_S, (\mathbf{o}, y)) = \begin{cases} 1 & \text{si } h_S(\mathbf{o}) = y, \\ 0 & \text{sinon.} \end{cases} \tag{1.1}$$

Nous pouvons également citer la perte logarithmique définie par

$$\ell_{\log}(h_S, (\mathbf{o}, y)) = -y \log h_S(\mathbf{o}) - (1-y) \log(1 - h_S(\mathbf{o})). \tag{1.2}$$

Résumons l'apprentissage hors ligne de la manière suivante :

Étant donnés :	• un ensemble d'entraînement $S = \{z_1, \ldots, z_m\}$ où un exemple $z_i = (\mathbf{o}_i, y_i)$ est composé d'un objet $\mathbf{o}_i \in \mathbf{O}$ et d'une classe $y_i \in Y$; • une mesure de perte cumulée L.
Nous voulons :	apprendre une hypothèse $h_S \in \mathcal{H}$ capable de minimiser la mesure de perte L sur S.

Apprentissage en ligne

Les algorithmes d'apprentissage en ligne, au contraire de leurs homologues hors ligne, n'ont pas un accès illimité à l'ensemble d'entraînement, car cet ensemble est généralement considéré comme un flux potentiellement infini de données, ne pouvant alors pas décemment être stocké en mémoire pour être réutilisé. Toute la difficulté de ce type d'apprentissage réside alors dans le fait de pouvoir utiliser l'information reçue le plus intelligemment possible afin d'apprendre la structure désirée. Ces algorithmes sont de ce fait moins efficaces que leurs équivalents hors ligne. Ils permettent, en revanche, une exécution plus rapide de l'apprentissage. L'effort est notamment mis sur une exécution rapide d'une itération, *i.e.*, sur le temps d'apprentissage de chaque observation. Enfin, un tel type d'algorithme doit pouvoir retourner, à tout moment, la structure apprise, qui doit alors représenter au mieux les données aperçues jusqu'alors. Aucune hypothèse sur une séquence spécifique du flux de données n'est faite dans l'apprentissage en ligne. Elle peut ainsi être déterministe, aléatoire, ou bien adversariale.

L'apprentissage en ligne est classiquement défini comme un jeu répétitif au sein duquel l'apprenant récupère une observation $\mathbf{o}$, puis lui attribut une classe $\hat{y}$ définie grâce à son hypothèse h_S, où S désigne l'ensemble des exemples déjà observés. La classe réelle y de l'observation $\mathbf{o}$ est alors révélée, et la différence entre les deux classes est calculée suivant une notion de regret. Cet écart de prédiction permet alors à l'apprenant d'actualiser son hypothèse h_S, qui devient de plus en plus précise (*i.e.*, en réduisant la différence qu'il y a entre l'hypothèse h_S, et l'hypothèse cible h^*) au fur et à mesure des observations rencontrées.

Soient $\mathbf{O}$ l'espace des observations, Y l'ensemble des classes possibles, et $\ell : \mathbf{O} \times Y \to \mathbb{R}^+$ une fonction de perte. La notion de regret est alors définie formellement

par l'équation suivante :

$$\text{regret}(h_S, h^*, j) = \sup_{\mathbf{z}\in(\mathbf{O}\times Y)^j} \left(\sum_{t=1}^{j} \ell(h_{S^{(t)}}, z_t) - \ell(h^*, z_t) \right), \tag{1.3}$$

avec $h_S^{(t)}$ l'hypothèse de l'apprenant après $t-1$ observations, et h^* l'hypothèse cible. Ainsi, un algorithme d'apprentissage en ligne est jugé comme performant si, lorsqu'un nombre infini d'observations a été traité, les deux hypothèses h_S et h^* prédisent les mêmes classes, *i.e.*,

$$\lim_{j\to+\infty} \left(\frac{1}{m} \text{regret}(h_S, h^*, j) \right) = 0. \tag{1.4}$$

Apprentissage par requêtes

L'apprentissage par requête, aussi appelé apprentissage exact, est un modèle permettant d'apprendre des connaissances en interagissant avec un utilisateur, ou avec une machine. Un tel modèle d'apprentissage peut s'apparenter à un jeu au cours duquel des questions sont posées à un utilisateur afin d'apprendre, au fur et à mesure de ses réponses, la structure désirée. La difficulté de ce modèle réside alors dans le fait de minimiser le nombre de questions posées à cet utilisateur, que l'on nomme, dans ce modèle, oracle.

Définition 1.7 (Oracle)**.**
On appelle ***oracle*** *un utilisateur ou une machine, ayant une hypothèse cible $h^* \in \mathcal{H}$, et que l'on suppose capable de répondre aux deux requêtes suivantes :*

- *Requête d'**appartenance** $\texttt{MQ}(x, h^*)$: l'apprenant envoie à l'oracle une instance x, qui retourne vrai si $h^*(x) = 1$, et faux sinon ;*
- *Requête d'**équivalence** $\texttt{EQ}(h, h^*)$: l'apprenant propose à l'oracle une hypothèse h, qui retourne vrai si $h^* = h$. Dans le cas contraire, l'oracle retourne faux, ainsi qu'un contre-exemple prenant la forme d'une instance $(x, 1)$ (resp. $(x, 0)$) si $h^*(x) = 1$ (resp. $h^*(x) = 0$).*

Dans l'apprentissage par requêtes, les requêtes d'appartenance correspondent à une forme d'apprentissage dit « actif », car l'apprenant cherche à affiner son hypothèse en posant directement des questions à l'oracle. Les requêtes d'équivalence correspondent plutôt à une forme d'apprentissage dit « passif », car l'apprenant va patienter avec sa structure courante tant qu'elle n'est pas remise en cause par l'oracle grâce à des contre-exemples.

Définition 1.8 (Apprentissage par requête ou apprentissage exact [KZ10])**.**
Soit $\mathbf{O}$ un espace de représentation sur n variables. Un algorithme A est un ***algorithme d'apprentissage par requête*** *pour une classe $\mathcal{H}$ d'hypothèses si, pour toute hypothèse $h \in \mathcal{H}$, il existe un polynôme p tel que, après $p(n)$ requêtes d'appartenance et d'équivalence, A trouve une représentation de h dans l'espace $\mathbf{O}$.*

Pour plus de détails sur ces différents types d'apprentissages, nous référons le lecteur au Chapitre 9 du livre de Marquis *et al.* [MPP14].

1.3 Apprentissage de préférences : les principaux paradigmes

Cette section présente les principaux paradigmes de l'apprentissage de préférences qui, rappelons le, sont le classement d'étiquettes [VG10], le classement d'exemples [WB10], et le classement d'objets [HLY08, KKA10]. Deux grandes approches sont généralement utilisées afin de résoudre ces trois grands problèmes :

- Agréger et comparer, via des fonctions d'utilité $f : Dom(\mathbf{V}) \rightarrow \mathbb{R}$, avec $Dom(\mathbf{V})$ un ensemble d'objets ($1 \leq |Dom(\mathbf{V})| \leq m$) [Kee88, Raj79]. Ces fonctions vont attribuer un score à cet ensemble d'objets. Il ne restera donc plus qu'à trier chaque ensemble en fonction de son score afin d'obtenir un classement de celui-ci ;
- Comparer et agréger, au moyen de relations entre les préférences, typiquement les comparaisons par paires [FH03] qui sont également très utilisées. Elles permettent d'effectuer un classement relatif en agrégeant les choix des utilisateurs.

Nous proposons ici un survol de chaque problème, avec les principales méthodes de résolution existantes associées (voir la Figure 1.1 pour une explication schématisée).

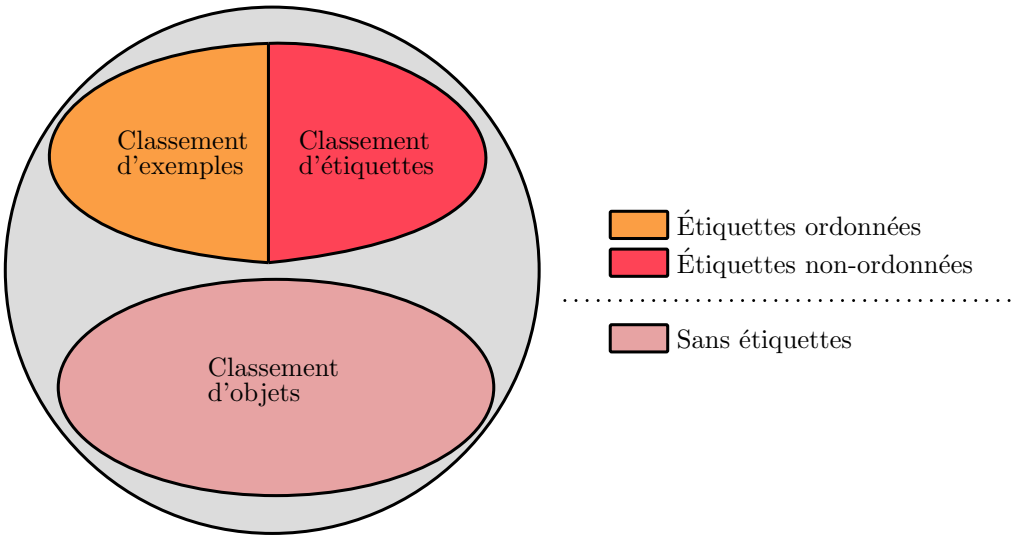


FIGURE 1.1 – Les trois grandes classes de problèmes en apprentissage de préférences.

1.3.1 Le problème de classement d'étiquettes

Commençons par définir formellement le terme « étiquette ».

Définition 1.9 (Étiquette).
Une ***étiquette*** y*, dans le domaine de l'apprentissage, est une classe potentielle pour un objet. On note* $Y = \{y_1, y_2, \ldots, y_k\}$ *un ensemble de* k *étiquettes.*

Problème 1 (Classement d'étiquettes)**.**

Données en entrée :	• *un ensemble* $\mathbf{O} = \{\mathbf{o}_1, \ldots, \mathbf{o}_m\} \subseteq Dom(\mathbf{V})$ *de* m *objets définis sur les variables de l'ensemble* $\mathbf{V} = \{V_1, \ldots, V_n\}$ *;* • *un ensemble* $Y = \{y_1, \ldots, y_k\}$ *de* k *étiquettes ;* • *un ensemble de* m' *relations de préférences* $\{y_i \succ_{\mathbf{o}_l} y_j\}$*, avec* $i \neq j$*,* $i, j \in \{1, \ldots, k\}$ *et* $l = 1, \ldots, m'$ *signifiant que* y_i *est préférée à* y_j *pour* $\mathbf{o}_l$*.*
Objectif :	*trouver une fonction de classement* f *faisant correspondre un classement* $\succ_{\mathbf{o}}$ *des étiquettes de* Y *à chaque objet* $\mathbf{o} \in \mathbf{O}$*.*
Mesures de performance :	• *erreur de classement entre le classement prédit et le classement cible ;* • *erreur de position entre le classement prédit et l'étiquette cible.*

Exemple 1.5. *Supposons que chaque étudiant en France soit en mesure d'exprimer ses préférences sur les candidats d'une élection présidentielle. Supposons de plus que ce votant soit caractérisé par les attributs âge, sexe, métier (i.e.,* $\mathbf{V} = \{\hat{a}ge, sexe, métier\}$*), et que les candidats sont au nombre de trois* $Y = \{Sarkozy, Hollande, Lepen\}$*. Le Tableau 1.1 illustre l'exemple précédent. L'objectif d'un tel problème sera alors de déterminer, à partir des préférences des étudiants* $\mathbf{O} = \{e_1, e_2, e_3\}$*, les préférences d'un quatrième étudiant* e_4 *sur les candidats de l'ensemble* Y*.*

	Âge	Sexe	Métier	Classement
e_1	18	M	étudiant	Sarkozy $\succ_1$ Hollande, Hollande $\succ_1$ Lepen
e_2	30	F	esthéticienne	Hollande $\succ_2$ Lepen, Lepen $\succ_2$ Sarkozy
e_3	65	M	retraité	Lepen $\succ_3$ Sarkozy, Sarkozy $\succ_3$ Hollande
e_4	50	F	DRH	?

TABLEAU 1.1 – Exemple de classement d'étiquettes. À partir des trois premiers individus, comment inférer les préférences politiques de la quatrième personne ?

Le problème de classement d'étiquettes se résout en deux phases [FH03, FH10b] : la première consiste à créer un modèle d'apprentissage de préférences, la deuxième calcule un ordre total des étiquettes de Y à partir des préférences précédemment prédites de chaque instance.

Approche par paires de préférences

La première étape de résolution du classement d'étiquettes est l'élaboration d'un modèle de classification qui servira à l'apprentissage des préférences. Parmi les méthodes existantes, la plus utilisée est la **classification par paires**.

L'idée principale de la **classification par paires de préférences** [HFCB08, FH10b, CQL+07] est de réduire un problème ayant k classes en entrée, en plusieurs problèmes à 2 classes. De multiples raisons poussent à faire cette transformation :

- Les performances d'apprentissage sont notamment sensiblement meilleures sur des classifications binaires [FH03] ;
- La nature même du problème de départ qui demande la recherche d'un classement souvent plus difficile à trouver dans un problème de classification sur k étiquettes, qui nous retournera seulement l'étiquette préférée ;
- Le fait de découper des problèmes par paires de préférences va permettre d'obtenir une classification entre 2 étiquettes induisant naturellement la préférence de l'une par rapport à l'autre, d'autant que cette réduction possède un faible coût en complexité temporelle, car les entrées du problème sont des paires de préférences.

Reprenons l'exemple du Tableau 1.1. Nous pouvons voir que les préférences de la première personne sont Sarkozy $\succ_1$ Hollande $\succ_1$ Lepen. Il est possible d'isoler chaque préférence : Sarkozy $\succ_1$ Hollande, Sarkozy $\succ_1$ Lepen, et Hollande $\succ_1$ Lepen. L'objectif sera donc, dans un premier temps, d'apprendre, à partir d'exemples, des modèles contenant uniquement deux étiquettes : le modèle (Sarkozy, Hollande), le modèle (Sarkozy, Lepen), et le modèle (Hollande, Lepen). Chaque modèle devra ensuite, pour un nouvel individu, retourner sa préférence entre les deux étiquettes concernées. La Définition 1.10 formalise cette notion de modèle d'apprentissage par paires de préférences.

Définition 1.10 (Modèle d'apprentissage [FH10b])**.**
Soit Y un ensemble d'étiquettes et $\mathbf{V}$ un ensemble de variables. On appelle ***modèle d'apprentissage*** *entre deux étiquettes $y_i \in Y$ et $y_j \in Y$ la fonction $\mathcal{M}_{ab} : Dom(\mathbf{V}) \to [0;1]$, i.e., la fonction $\mathcal{M}_{ij}(\mathbf{o})$ prenant en paramètre un individu $\mathbf{o} \in Dom(\mathbf{V})$, et retournant la probabilité que $\mathbf{o}$ préfère y_i à y_j, i.e., $y_i \succ_{\mathbf{o}} y_j$.*

L'objectif étant de pouvoir comparer chaque paire de préférences, deux méthodes de découpe peuvent être envisagées [Für01, Für02] :

- Réduire le problème principal ayant k étiquettes en $\frac{k(k-1)}{2}$ problèmes binaires. On appelle cette transformation la **round robin classification**. Dans ce cas on ne distingue pas l'ordre et $\mathcal{M}_{ij} = \mathcal{M}_{ji}$ (dans l'exemple précédent, le modèle (Sarkozy, Hollande) sera identique au modèle (Hollande, Sarkozy)) ;
- Réduire le problème principal ayant k étiquettes en $k(k-1)$ problèmes 2 classes. On appelle cette transformation la **double round robin classification**. Dans ce cas, $\mathcal{M}_{ij} \neq \mathcal{M}_{ji}$. Il y a deux fois plus de problèmes à résoudre, mais cela permet de notamment mieux repérer des erreurs de classification (dans l'exemple

précédent, le modèle (Sarkozy, Hollande) est différent du modèle (Hollande, Sarkozy)).

La transformation de ces problèmes permet de se ramener à des problèmes classiques de classification. Toutes les méthodes de classification existantes peuvent donc être utilisées [Bis07]. Résoudre le problème de classement d'étiquettes général revient alors à agréger les résultats retournés par les différents sous-problèmes. On peut distinguer trois grandes techniques permettant de déduire un classement à partir de comparaisons par paires [HF04] :

1. L'utilisation de fonctions de vote : soit Y un ensemble d'étiquettes et $\mathbf{O}$ un ensemble d'objets. Une **fonction de vote** est une fonction $v : Dom(\mathbf{V}) \times Y \rightarrow \mathbb{R}$ permettant d'attribuer à un objet et à une étiquette donnés un score définit par $v(\mathbf{o}, y_i) = \sum_{y_i \neq y_j} \mathcal{M}_{ij}(\mathbf{o})$, avec $y_i, y_j \in Y$, et $\mathcal{M}_{ij}(\mathbf{o})$ la probabilité d'avoir $y_i \succ y_j$ pour un individu $\mathbf{o} \in Dom(\mathbf{V})$. Le classement recherché sera alors induit par la fonction $v(.,.)$ en ordonnant les étiquettes par ordre décroissant des valeurs retournées ;
2. L'utilisation de l'**approche *slater***, qui consiste à traduire les résultats des sous-problèmes, pour l'objet recherché, sous la forme d'un graphe orienté $G = (\mathbf{S}, \mathbf{A})$, où $\mathbf{S}$ correspond à l'ensemble des étiquettes. Un arc $a = (y_i, y_j) \in \mathbf{A}$ ($\mathbf{A}$ correspondant à l'ensemble des arcs du graphe) induit alors une préférence $y_i \succ y_j$, *i.e.*, l'étiquette y_i est préférée à l'étiquette y_j. L'objectif de cette approche est de repérer, puis de corriger, les problèmes de transitivité du graphe, afin d'obtenir un classement sans cycle au sein des préférences ;

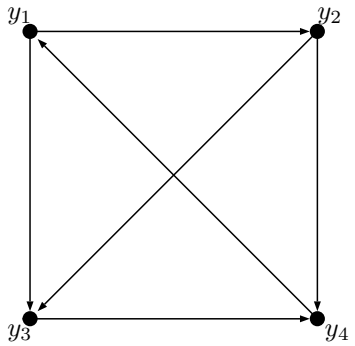


FIGURE 1.2 – Graphe orienté représentant les comparaisons des étiquettes. Chaque sommet représente une étiquette y_i et un arc (y_i, y_j) signifie que y_i est préférée à y_j.

La Figure 1.2 permet de mettre en évidence le problème de transitivité : nous avons les arcs (y_1, y_2) et (y_2, y_4) mais pas (y_1, y_4). Le graphe n'est pas transitif, il y a donc une contradiction des préférences ne permettant pas de définir un classement entre ces étiquettes [2]. Cette représentation n'est pas idéale à des fins de résolution du problème, mais elle permet

2. Trouver un classement des étiquettes revient à trouver un chemin hamiltonien entre les sommets du graphe (**nécessairement transitif**), il s'agit d'un problème NP-Complet.

(i) de repérer les problèmes de classification des étiquettes,

(ii) de corriger ces problèmes en se ramenant à des problèmes connus de la théorie des graphes dont le but est de modifier le graphe afin de le rendre transitif [DF03] [3].

3. L'**approche probabiliste** est une approche gloutonne qui va chercher l'étiquette la plus préférée parmi l'ensemble des étiquettes disponibles en calculant sa probabilité de préférence. L'étiquette préférée est alors exclue de la liste des étiquettes disponibles pour être placée en tête du classement. L'opération est alors réitérée jusqu'au traitement de toutes les étiquettes.

Une généralisation du problème de classement d'étiquettes peut être obtenue en partitionnant, pour chaque objet $\mathbf{o}$, l'ensemble des étiquettes Y en deux ensembles Y^+, rassemblant l'ensemble des étiquettes pertinentes pour $\mathbf{o}$, et Y^-, rassemblant l'ensemble des étiquettes non pertinentes pour $\mathbf{o}$. On appelle ce problème le **problème de classement multiétiquettes** [BH06, BFH06, EW01].

1.3.2 Le problème de classement d'exemples

Problème 2 (Classement d'exemples)**.**

Données en entrée :	• *un ensemble* $\mathbf{O} = \{\mathbf{o}_1, \ldots, \mathbf{o}_m\} \subseteq Dom(\mathbf{V})$ *de* m *objets définis sur les variables de l'ensemble* $\mathbf{V} = \{V_1, \ldots, V_n\}$ *;* • *un ensemble d'étiquettes* $Y = \{y_1, \ldots, y_k\}$ *totalement ordonné* $y_k \succ y_{k-1} \succ \ldots \succ y_1$ *;* • *chaque exemple* $\mathbf{o}$ *est associé à une étiquette* y_j*.*
Objectif :	*trouver une fonction de classement* $f : Dom(\mathbf{V}) \to \mathbb{R}$ *ordonnant les objets d'un nouvel ensemble* $\mathbf{O}'$ *suivant la propriété suivante :* $\forall \mathbf{o}, \mathbf{o}' \in \mathbf{O}' \ : \ \mathbf{o} \succ \mathbf{o}'$ *si* $f(\mathbf{o}) > f(\mathbf{o}')$. (1.5)
Mesures de performance :	• *aire sous la courbe ROC [WB10] (lorsque* $k = 2$*) ;* • *C-index [GH05] (lorsque* $k \geq 3$*).*

Exemple 1.6. *Considérons une étudiante en littérature qui désire lire un maximum de livres (caractérisés par un ensemble de variables telles que l'auteur du livre, son année de parution, ainsi que son genre principal). Soit B l'ensemble de tous les livres existant. Ayant déjà lue un grand nombre de livres (que l'on note par l'ensemble* B_1*), elle a pu établir des niveaux d'intérêts et a pu classer ses différentes lectures dans des niveaux qui vont de une étoile (mauvais) à 5 étoiles (très intéressant). Formellement,*

3. Cela revient, dans notre contexte, à remettre en cause le modèle d'apprentissage en proposant de nouvelles classifications, l'objectif étant de modifier le moins possible le graphe.

nous avons alors un ensemble d'étiquettes $Y = \{1, 2, 3, 4, 5\}$ *totalement ordonné de façon croissante (*$1 < 2 < 3 < 4 < 5$*) représentant les étoiles. Notre étudiante désire, maintenant qu'elle a pu définir ses différents niveaux d'intérêts, connaître dans quelle catégorie se trouvent les livres encore non lus (correspondant à l'ensemble* $B_2 = B \setminus B_1$*), afin de ne sélectionner que les plus intéressants de son point de vue, c'est-à-dire les livres de* B_2 *qui seront chacun étiquetés par une étiquette de* Y *(cet exemple est tiré de [FH01]). Le Tableau 1.2 illustre cet exemple.*

	Auteur	Année	Genre	Note
1	J.K. Rowling	2000	Fantastique	5
2	J.R.R. Tolkien	1955	Fantastique	4
3	G.R.R. Martin	200	Fantastique	5
4	Isaac Asimov	1967	Science-fiction	4
5	Bernard-Henri Lévy	1985	Romantique	?

TABLEAU 1.2 – Exemple de classement d'exemple. L'objectif est de déterminer, à partir des quatre premiers livres, la note du cinquième.

Deux principes peuvent être employés pour réaliser cette tâche :

1. Une **fonction de mapping** $h : Dom(\mathbf{V}) \to Y$ permettant d'étiqueter chaque exemple. Le classement de ceux-ci s'effectuera naturellement avec le classement des étiquettes. Ceci va cependant engendrer un grand nombre d'égalités (tous les exemples identiquement étiquetés) qu'il faudra ordonné (par exemple de façon aléatoire).
2. Une **fonction de régression** $f : Dom(\mathbf{V}) \to \mathbb{R}$, qui va attribuer un score à chaque objet, induisant un classement naturel de ceux-ci.

Le problème du classement d'exemples est également appelé **problème du classement** k**-parti** ou **multiparti** [FHV09, RA05] lorsque l'on cherche une fonction de mapping h. Dans le cas où l'on a deux classes, nous parlons d'un problème de classement biparti [Aga05].

Ce problème peut être aussi vu comme un problème de régression ordinale, c'est-à-dire un problème de régression de variables discrètes ordonnées [WB10]. Le problème de régression consiste alors à déterminer une fonction $h : Dom(\mathbf{V}) \to Y$ associant chaque nouvel objet $\mathbf{o} \in \mathbf{O} \subseteq Dom(\mathbf{V})$ à une étiquette Y et ayant la forme générale suivante :

$$h(\mathbf{o}) = \begin{cases} y_1, & \text{si } f(\mathbf{o}) < b_1, \\ y_i, & \text{si } b_{i-1} < f(\mathbf{o}) \leq b_i,\ i = 2, \ldots, k-1, \\ y_k, & \text{si } f(\mathbf{o}) > b_{k-1}, \end{cases} \tag{1.6}$$

avec b_i le seuil de passage de l'étiquette y_{i-1} à y_i et f une fonction donnant un score à l'objet $\mathbf{o}$.

Trois types d'approches sont utilisés pour résoudre ce problème : la comparaison par paires [FH01, FHV09], les *support vector ordinal regression* (ou SVOR) [GST+15,

LL06, CK05], et les méthodes de *learning to rank* [Tro05]. Nous exposons rapidement les deux premières approches.

Comparaison par paires (approche par réduction)

Frank *et al.* dans [FH01] proposent une méthode de simplification du classement d'exemples en se servant des informations complémentaires de l'ordonnancement des étiquettes : soit un problème comportant k étiquettes. Ce problème est réduit en créant $k-1$ problèmes de classification binaire. Le classement naturel des étiquettes induit alors automatiquement une bipartition de celles-ci (voir la Figure 1.3).

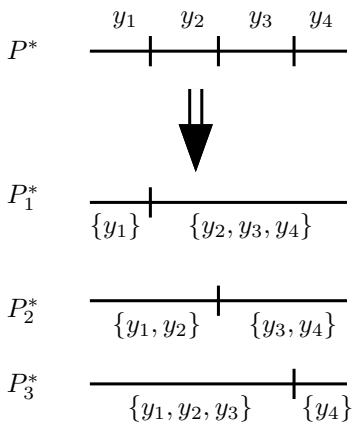


FIGURE 1.3 – La division (P_i^*) du problème de départ (P^*) alliée à l'ordonnancement des étiquettes (y_j) permet d'obtenir une bipartition de celles-ci.

Chacun des $k-1$ modèles créés va calculer une probabilité $\mathbb{P}(y_{\mathbf{o}} \succ y_i)$ avec $y_{\mathbf{o}}$ l'étiquette de $\mathbf{o}$. Frank *et al.* appliquent ensuite les ensembles de règles suivantes en fonction de l'étiquette recherchée :

$$\mathbb{P}(l_{\mathbf{o}} = y_i) = \begin{cases} 1 - \mathbb{P}(l_{\mathbf{o}} \succ y_i) & \text{pour } i = 1, \\ \mathbb{P}(l_{\mathbf{o}} \succ y_{i-1}) - \mathbb{P}(l_{\mathbf{o}} \succ y_i) & \text{pour } 2 \leq i \leq k-1, \\ \mathbb{P}(l_{\mathbf{o}} \succ y_i) & \text{pour } i = k-1. \end{cases} \tag{1.7}$$

Lorsqu'un nouvel objet $\mathbf{o}$ arrive, chacune des probabilités précédentes est calculée et l'objet sera alors étiqueté par l'étiquette correspondant à la probabilité la plus élevée.

Fürnkranz *et al.* dans [FHV09] comparent cette approche à un dérivé d'une approche similaire par comparaison de paires [FH10b] utilisée notamment pour la résolution du classement d'étiquettes (voir la Section 1.3.1 page 18). Le problème de k étiquettes est cette fois divisé en $\frac{k(k-1)}{2}$ problèmes de 2 étiquettes. De la même manière que pour l'apprentissage par paires de préférences dans le classement d'étiquettes, on désigne par $\mathcal{M}_{ij}$ le modèle attribuant un « vote » pour l'étiquette y_i par rapport à l'étiquette y_j. Une critique de ce modèle est qu'un objet ne possède qu'une seule étiquette. Il peut donc arriver qu'un objet initialement étiqueté par y_4 se voit

appliquer le modèle $\mathcal{M}_{23}$. Ce modèle n'est alors pas pertinent (on dit que le modèle n'est pas **compétent**). Cependant l'ordonnancement naturel des étiquettes apporte une certaine sémantique à ces modèles car si $y_3 \succ y_2 \succ y_1$, nous aurons $\mathcal{M}_{31} > \mathcal{M}_{32}$.

Approche par vecteurs de support (SVOR)

Rappelons que le classement d'exemples peut également être vu comme un problème de régression ordinale. Il est alors possible d'utiliser l'approche exprimée par Shashua *et al.* dans [SL02] et légèrement modifiée par Chu *et al.* dans [CK05] et [CK07], que l'on appelle **régression ordinale par vecteurs de support** (SVOR). Il s'agit d'un dérivé des **machines à vecteurs de support** (ou SVM), où les classes (ici les étiquettes) sont ordonnées. L'algorithme utilise donc ici une base d'apprentissage ayant $Y = \{y_1, y_2, \ldots, y_k\}$ étiquettes ordonnées. Pour chaque étiquette $y_j \in Y$, le i^{e} exemple étiqueté par y_j est noté $\mathbf{o}_i^j$ et le nombre total d'exemples étiquetés par y_j est noté n^j.

Le problème de base des SVM est la recherche d'un hyperplan séparateur entre deux classes. Nous avons ici un problème multi-classes, nous allons donc avoir $r - 1$ hyperplans parallèles permettant de séparer chacune des classes de type $\mathbf{w}^T \cdot \phi(\mathbf{o}_i^j)$ discriminés par $r - 1$ seuils b_j ($\mathbf{w}^T$ est un vecteur de poids appliqué sur les caractéristiques de l'objet $\mathbf{o}_i^j$, et $\phi(\mathbf{o}_i^j)$ est une fonction noyau appliquée à $\mathbf{o}_i^j$). La Figure 1.4 résume le principe.

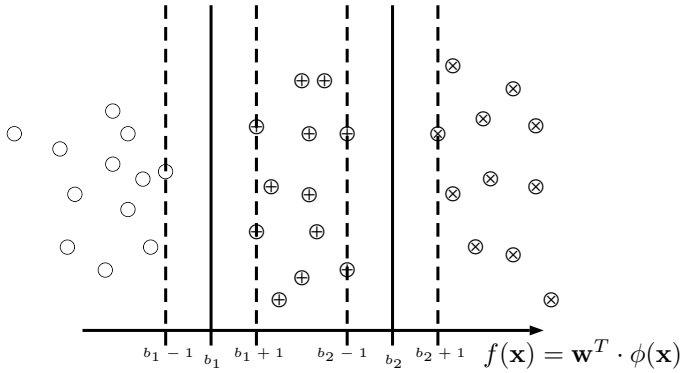


FIGURE 1.4 – Nous avons ici 3 étiquettes (◯, ⊕ et ⊗). L'ensemble des étiquettes est totalement ordonné, nous pouvons alors introduire 2 seuils définissant 2 surfaces de décision parallèles séparant chaque étiquette.

Notons enfin que le problème de classement d'exemples est également connu sous le nom de problème de tri en aide à la décision multicritères [Roy13]. La méthode ELECTRE-TRI [Roy68, FMR05, CMM12, LM16] constitue une méthode supplémentaire, issue de l'aide à la décision multicritères, permettant de résoudre ce problème.

1.3.3 Le problème de classement d'objets

Problème 3 (Classement d'objets)**.**

Données :	• *un ensemble* $\mathbf{O} = \{\mathbf{o}_1, \ldots, \mathbf{o}_m\} \subseteq Dom(\mathbf{V})$ *de* m *objets définis par les variables de l'ensemble* $\mathbf{V} = \{V_1, \ldots, V_n\}$ *;*
	• *un ensemble fini de paires de comparaison* $\mathbf{o}_i \succ \mathbf{o}_j$*.*
Résultat :	*trouver le classement d'un sous-ensemble* $\mathbf{O}' \subseteq \mathbf{O}$ *d'objets au travers d'une fonction de classement (ou de permutation)* $f : \mathbf{O}' \to \{1, \ldots, \lvert\mathbf{O}'\rvert\}$*.*
Mesures de performance :	• *erreur de classement entre le classement prédit et le classement cible ;*
	• *les mesures top-k [IBS08].*

Le classement d'objets (Problème 3) est différent des autres problèmes de l'apprentissage de préférences car il ne fait pas intervenir d'étiquettes. Soit un ensemble d'objets $\mathbf{O} = \{\mathbf{o}_1, \mathbf{o}_2, \ldots, \mathbf{o}_m\}$ et un ensemble de paires de préférences sur ces objets de type $\mathbf{o}_i \succ \mathbf{o}_j$ signifiant que $\mathbf{o}_i$ est préféré à $\mathbf{o}_j$. L'objectif est, étant donné un sous-ensemble $\mathbf{O}' \subseteq \mathbf{O}$, de trouver un classement des objets de $\mathbf{O}'$.

Les exemples pratiques sont nombreux à ce problème, comme le classement d'objets personnels. Nous pouvons reprendre l'exemple donné en introduction de la Section 1.3.2 expliquant le désir d'une étudiante de classer les livres qui lui restent à lire dans des catégories sachant qu'elle a déjà pu classer ceux qu'elle a auparavant lus. Le classement d'objets peut être ici interprété comme une pré-procédure au classement d'exemples permettant de classer les livres déjà lus (et d'en déterminer leurs catégories) sachant que l'étudiante est capable d'en comparer certains paire à paire. Les méthodes de résolution de tels problèmes sont également nombreuses, et nous orientons à ce propos le lecteur vers les excellentes études de Kamashima *et al.* dans [KKA05, KKA10].

Learning to order things est un travail de Cohen et al. [CSS99] exposant une méthode se basant sur un algorithme glouton et dynamique qui va chercher un ordre total des objets, puis va modifier cet ordre en fonction des retours, à l'image de l'apprentissage par renforcement [JLC00].

Plus formellement, nous allons ici considérer le problème suivant : un ensemble $\mathbf{O}$ d'objets, un ensemble de N experts et une fonction de préférence $PREF : \mathbf{O} \times \mathbf{O} \to [0, 1]$ telle que

$$PREF(\mathbf{o}, \mathbf{o}') \begin{cases} > \frac{1}{2} & \Rightarrow \mathbf{o} \text{ devrait être classé avant } \mathbf{o}', \\ < \frac{1}{2} & \Rightarrow \mathbf{o} \text{ devrait être classé après } \mathbf{o}', \\ = \frac{1}{2} & \Rightarrow \text{abstention de prendre une décision.} \end{cases}$$

Nous représentons l'ensemble d'apprentissage comme un ensemble de fonctions primitives $R_1, \ldots, R_N$ relatives à chaque expert. Les fonctions $f_i : \mathbf{O} \to S$ (avec S un ensemble totalement ordonné), appelées **fonctions d'ordonnancement**, sont ensuite introduites. Ces fonctions f_i induisent les fonctions de préférence R_{f_i} définies de la façon suivante :

$$R_{f_i}(\mathbf{o}, \mathbf{o}') = \begin{cases} 1 & \text{si } f_i(\mathbf{o}) > f_i(\mathbf{o}'), \\ 0 & \text{si } f_i(\mathbf{o}) < f_i(\mathbf{o}'), \\ \frac{1}{2} & \text{sinon.} \end{cases}$$

On appelle alors R_{f_i} un **ordre de classement** de $\mathbf{O}$ dans S pour l'expert i. On dira que l'objet $\mathbf{o}$ est préféré à $\mathbf{o}'$, ou que $\mathbf{o}$ a un classement plus haut que $\mathbf{o}'$ pour l'expert i lorsque $R_{f_i}(\mathbf{o}, \mathbf{o}') = 1$. L'objectif de l'algorithme de [CSS99] est d'apprendre une fonction de préférence $PREF$ en utilisant les fonctions primitives R_{f_i}. La fonction de préférence prend ici la forme suivante :

$$PREF(\mathbf{o}, \mathbf{o}') = \sum_{i=1}^{N} w_i R_{f_i}(\mathbf{o}, \mathbf{o}'). \tag{1.8}$$

Apprendre la fonction $PREF$ revient alors à apprendre les différents poids w_i relatifs aux différents experts, et représentant l'importance de chaque expert. L'algorithme utilisé sera un algorithme dynamique, basé sur l'algorithme "Hedge" de Freund et Schapire [FS97], au sein duquel les poids seront mis à jour incrémentalement, et dont la perte associée au retour d'un utilisateur noté F est définie par :

$$Loss(R, F) = 1 - \frac{1}{|F|} \sum_{(\mathbf{o}, \mathbf{o}') \in F} R(\mathbf{o}, \mathbf{o}'). \tag{1.9}$$

Cette fonction de perte peut être interprétée comme une probabilité que R soit en désaccord avec les retours de F, qu'il faut alors minimiser.

Soit N un ensemble d'experts. Le fonctionnement de cet algorithme peut être résumé de la façon suivante :

1. Initialiser aléatoirement les poids w_i de chaque expert $i = \{1, \ldots, N\}$ de sorte que $\sum_{i=i}^{N} w_i = 1$;
2. Un ensemble d'objets $\mathbf{O}$, ainsi qu'un ensemble de fonctions d'ordonnancement f_i (pour chaque expert $i = \{1, \ldots, N\}$) sont reçus ;
3. Calculer l'Équation (1.8) pour chaque paire d'objets ;
4. Ordonner $\mathbf{O}$ en fonction des résultats obtenus ;
5. Évaluer la perte à l'aide de l'Équation (1.9) ;
6. Mettre à jour le vecteur de poids $w_i \leftarrow \frac{w_i \cdot \beta^{Loss(R_i, F)}}{Z}$ avec Z une constante de normalisation et $\beta \in [0; 1]$ un paramètre fixé manuellement ;
7. Réitérer t fois à partir de l'étape 2 (avec t un paramètre fixé manuellement).

Nous renvoyons le lecteur à l'article de Cohen *et al.* [CSS99] pour une description détaillée de l'algorithme d'apprentissage de poids.

La méthode générale de Cohen *et al.* propose donc un fonctionnement en ligne permettant de remettre en cause ses propres choix grâce aux retours des différents utilisateurs. Elle se base notamment sur un graphe comportant des objets combinatoires. Ce graphe peut de fait être exponentiel en la taille du vecteur caractérisant chaque objet.

Remarquons que le problème du classement d'exemples [FH10a, WB10, ZBHH10] vu dans la section précédente se situe à mi-chemin entre le classement d'étiquettes et le classement d'objets en ce sens qu'il va chercher à classer un ensemble de nouveaux exemples entre eux, sachant que chaque exemple est associé à une étiquette (encore inconnue) parmi un ensemble totalement ordonné d'étiquettes.

1.4 Réseaux de préférences conditionnelles

Il existe de nombreuses manières de représenter les préférences [DHKP11]. Chacune d'elles possède des propriétés spécifiques. Cette section aborde les **réseaux de préférences conditionnelles** (plus simplement appelés **CP-nets**), introduits par Boutilier *et al.* dans [BBD+04].

1.4.1 Formalisme

Nous commençons par poser les bases du formalisme des CP-nets.

1.4.1.1 Notion de *ceteris paribus* et de préférence conditionnelle

Les CP-nets ont la particularité de se baser sur deux notions fondamentales permettant une lecture intuitive des préférences présentes au sein de cette structure : la notion de *préférences conditionnelles*, permettant une modification des préférences en fonction d'un contexte donné, ainsi que la notion de *ceteris paribus*, ou « toute chose étant égale par ailleurs », n'autorisant le changement que d'une seule valeur au sein d'une comparaison entre deux objets.

Exemple 1.7. *Nous continuons de nous baser sur l'Exemple 1.1 page 12. Les préférences conditionnelles peuvent être aperçues lorsque, par exemple, nous préférons habituellement porter des tee-shirts plutôt que des chemises dans la vie de tous les jours. Cependant, si ce contexte est modifié en supposant que nous nous rendons à un mariage, alors nous préférerons porter une chemise plutôt qu'un tee-shirt.*
Une comparaison ceteris paribus pourrait alors être, si l'on reprend les variables données dans l'Exemple 1.1, une comparaison entre les objets (ma, ch, sh) *(aller à un mariage en chemise et en short) et* (ma, ch, pa) *(aller à un mariage en chemise et en pantalon).*

La notion de préférence conditionnelle est liée à celle d'**indépendance préférentielle**. Cette dernière stipule que le fait que si nous avons deux ensembles de variables $\mathbf{X}$ et $\mathbf{Y}$ avec une préférence sur les valeurs [4] de $\mathbf{X}$ (représentée par un ordre total sur ses valeurs), alors cette préférence n'est pas dépendante de la présence de telle ou telle valeur de Y. Cela permet de mettre en exergue l'intuition que certaines variables sont indépendantes entre elles, et le fait d'observer que telle ou telle valeur de ces variables n'aura aucune incidence sur l'ordre de préférence des valeurs des autres variables. La Définition 1.11 formalise cette notion.

Définition 1.11 (Indépendance préférentielle [BBD+04, Section 2.1])**.**
Soit $\mathbf{V}$ *un ensemble de variables, et* $\mathbf{X} \subseteq \mathbf{V}$ *et* $\mathbf{Y} \subseteq \mathbf{V} \setminus \mathbf{X}$ *une partition de* $\mathbf{V}$*. Nous disons que* $\mathbf{X}$ ***est indépendante préférentiellement à*** $\mathbf{Y}$ *si et seulement si,* $\forall \mathbf{o}, \mathbf{o}' \in Dom(\mathbf{X})$ *et* $\forall \mathbf{y}, \mathbf{y}' \in Dom(\mathbf{Y})$*, nous avons*

$$\mathbf{oy} \succ \mathbf{o'y} \iff \mathbf{oy'} \succ \mathbf{o'y'}. \tag{1.10}$$

On dit alors que la préférence entre $\mathbf{o}$ *et* $\mathbf{o}'$ *ne dépend pas des valeurs de* $\mathbf{y}$*.*

Exemple 1.8. *Continuons d'utiliser l'Exemple 1.1 page 12. Rappelons notre ensemble de variables* $\mathbf{V} = \{A, H, B\}$ *:*

Variable	Valeurs
A : Activité	sp : sport ma : mariage
H : haut de vêtement	tsh : t-shirt ch : chemise
B : bas de vêtement	sh : short pa : pantalon

Nous disons que les variables H *et* B *sont* ***indépendantes*** *au sens des préférences, si* $(ch, pa) \succ (tsh, pa) \iff (ch, sh) \succ (tsh, sh)$*. Nous pouvons expliquer cette propriété par le fait que le haut et le bas de corps peuvent être portés indépendamment du choix de l'un ou de l'autre.*

Considérons deux objets $\mathbf{o}$ et $\mathbf{o}'$. Il peut sembler plus aisé d'exprimer ses préférences entre $\mathbf{o}$ et $\mathbf{o}'$ lorsque ces deux objets se ressemblent, surtout si cette ressemblance prend la forme de valeurs identiques pour certaines variables.

Définition 1.12 (Variable conditionnée et variable parente [BBD+04, Section 2.1])**.**

Soient $X \in \mathbf{V}$ *et* $Y \in \mathbf{V}$ *deux variables d'un ensemble* $\mathbf{V}$*. On dit que* X ***est conditionnée par*** Y *(ou* Y ***conditionne*** X*) si,* $\exists x, x' \in Dom(X)$ *et* $y, y' \in Dom(Y)$*, et il existe un état* $\mathbf{z} \in Dom(\mathbf{V} \setminus \{X, Y\})$ *satisfaisant*

$$xy\mathbf{z} \succ x'y\mathbf{z} \text{ et } x'y'\mathbf{z} \succ xy'\mathbf{z}. \tag{1.11}$$

4. Il est à noter qu'aucun ordre *a priori* n'est attribué entre les valeurs d'une variable.

On dit alors que Y est la ***variable parente*** *de X, et que X* ***est conditionnée par*** *Y.*

Exemple 1.9. *Reprenons notre ensemble $\mathbf{V} = \{A, H, B\}$ issu de l'Exemple 1.1 page 12 :*

Variable	Valeurs
A : Activité	sp : sport ma : mariage
H : haut de vêtement	tsh : t-shirt ch : chemise
B : bas de vêtement	sh : short pa : pantalon

Dans le contexte où il est nécessaire de se rendre à un mariage, il est bien évidemment conseillé de porter une chemise avec un pantalon. Cependant, il est préférable, pour faire du sport, de porter un tee-shirt avec un short. Cela signifie que l'activité effectuée (la variable A) ***conditionne*** *ce que l'on porte (les variables H et B). Cela signifie, pour la variable H :*

- $(ma, ch, pa) \succ (ma, tsh, pa)$ *et* $(sp, tsh, pa) \succ (sp, ch, pa)$,
- $(ma, ch, sh) \succ (ma, tsh, sh)$ *et* $(sp, tsh, sh) \succ (sp, ch, sh)$.

Et pour la variable B :

- $(ma, ch, pa) \succ (ma, ch, sh)$ *et* $(sp, ch, sh) \succ (sp, ch, pa)$,
- $(ma, tsh, pa) \succ (ma, tsh, sh)$ *et* $(sp, tsh, sh) \succ (sp, tsh, pa)$.

On dit alors que la variable A est la variable parente des variables H et B.

1.4.1.2 CP-règles, CP-tables et CP-nets

Définition 1.13 (CP-règle)**.**
Soit $V \in \mathbf{V}$ une variable avec $v, v' \in Dom(V)$ et $\mathbf{u} \in Dom(\mathbf{U})$ ($\mathbf{U} \subset \mathbf{V}$) un sous-ensemble de variables de $\mathbf{V}$. Une CP-règle $(\mathbf{u} : v \succ v')$ signifie que v est préférée à v' lorsque l'ensemble de variables $\mathbf{U} \subseteq \mathbf{V} \setminus \{V\}$ possède la valeur $\mathbf{u}$ [5]*.*

Nous définissons alors l'ensemble $\mathbf{U}$ de la définition précédente comme étant l'**ensemble des variables parentes** de V, que l'on note $Pa(V)$. Cela signifie qu'un changement de valeur au sein d'une (ou plusieurs) variable(s) de l'ensemble $Pa(V)$ aura un impact sur les préférences des valeurs de la variable V. Nous notons également $\overline{Pa}(V)$ l'**ensemble des variables non parentes** de V, *i.e.*, $\overline{Pa}(V) = \{V \in \mathbf{V} \mid V \notin Pa(V)\}$. La définition suivante permet d'introduire l'inverse d'une CP-règle, et sera principalement utilisée dans les deux chapitres suivants.

5. Il est possible que $\mathbf{U} = \emptyset$, la CP-règle sera donc une règle du type $(\emptyset : v \succ v')$.

Définition 1.14 (Inverse d'une CP-règle)**.**
Soient $\mathbf{V}$ *un ensemble de variables,* $V \in \mathbf{V}$ *une variable de* $\mathbf{V}$ *(avec* $Dom(V) = \{v, v'\}$*),* $\mathbf{u} \in Dom(Pa(V))$ *une valeur sur les variables parentes de* V*, et* $(\mathbf{u} : v \succ v')$ *une CP-règle. Nous définissons l'****inverse d'une CP-règle*** *comme étant la préférence inverse sur les valeurs de* V*, avec la même valeur* $\mathbf{u}$ *des parents de* V*, i.e.,* $(\mathbf{u} : v' \succ v)$*.*

Définition 1.15 (CP-table)**.**
L'ensemble des CP-règles d'une variable V *(identifiées par la valeur de l'état de ses variables parentes) est réuni au sein d'une* ***table de préférences conditionnelles****, ou* ***CP-table****, notée* $CPT(V)$*. On dit qu'une CP-table associée à une variable* V *est* ***complète*** *s'il existe une CP-règle pour chaque état* $\mathbf{u} \in Dom(\mathbf{U})$*, avec* $\mathbf{U}$ *l'ensemble des variables parentes de* V*. On note par* $CPT(\mathbf{X}) = \bigcup_{X \in \mathbf{X}} CPT(X)$ *l'ensemble des CP-tables des variables* $\mathbf{X} \subseteq \mathbf{V}$*.*

Un exemple de CP-table, basé sur l'Exemple 1.9, est donné dans le Tableau 1.3. Dans la suite de ce manuscrit, $|CPT(V)|$ correspond à la **taille** de la CP-table de la variable V, *i.e.*, le nombre de règles qu'elle possède. De manière équivalente, la **taille d'un CP-net** $\mathcal{N}$ correspond à la somme de la taille de toutes ses CP-tables. On note cette taille par $|\mathcal{N}|$ ou par $|CPT(\mathbf{V})| = \sum_{V \in \mathbf{V}} |CPT(V)|$.

sp	:	$tsh \succ ch$
ma	:	$ch \succ tsh$

TABLEAU 1.3 – Exemple de CP-table pour l'ensemble $\mathbf{V} = \{A, H, B\}$ (Exemple 1.1 page 12). Cette CP-table $CPT(H)$ est associée à la variable H, avec $Pa(H) = \{A\}$.

Exemple 1.10. *L'Exemple 1.9 a montré que* $Pa(H) = Pa(B) = \{A\}$ *et* $Pa(A) = \emptyset$*. Une CP-règle peut être* $(ma : ch \succ tsh)$ *signifiant qu'une personne allant à un mariage préférera porter une chemise plutôt qu'un tee-shirt.*

Nous avons désormais toutes les notions nécessaires à la définition formelle des réseaux de préférences conditionnelles.

Définition 1.16 (Réseau de préférences conditionnelles [BBD+04])**.**
Soit $\mathbf{V}$ *un ensemble de variables. Un* ***réseau de préférences conditionnelles****, aussi appelé* ***CP-net****, est un graphe orienté* $\mathcal{N} = (\mathbf{V}, \mathbf{A}, CPT(\mathbf{V}))$*, où* $\mathbf{V}$ *est un ensemble de variables,* $\mathbf{A}$ *correspond à l'ensemble des arcs du graphe, avec* $(V, V') \in \mathbf{A} \iff V \in Pa(V')$ *(*$V, V' \in \mathbf{V}$*), et* $CPT(\mathbf{V})$ *est l'ensemble des CP-tables de toutes les variables de* $\mathbf{V}$*. On dit qu'un CP-net est* ***complet*** *lorsque toutes ses CP-tables sont complètes.*

Une illustration du CP-net relatif à l'Exemple 1.9 est donnée dans la Figure 1.5. Nous remarquons qu'une variable X conditionnant les préférences d'une autre variable Y est « plus importante » que cette dernière (*i.e.*, supposons que $x \succ x'$,

Variable	Valeurs
A : Activité	sp : sport ma : mariage
H : haut de vêtement	tsh : t-shirt ch : chemise
B : bas de vêtement	sh : short pa : pantalon

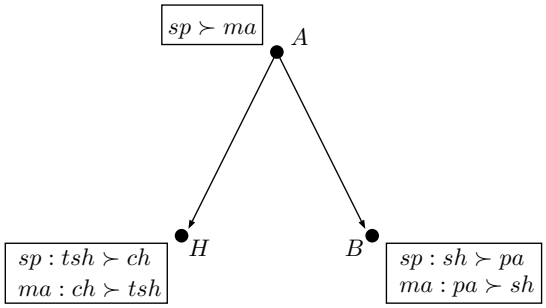


FIGURE 1.5 – Exemple de CP-net pour l'ensemble $\mathbf{V} = \{A, H, B\}$ de l'Exemple 1.9.

alors toutes les préférences faisant intervenir la valeur x seront préférées à toutes celles faisant intervenir la valeur x') : soit A ($Dom(A) = \{a, a'\}$) parente de B ($Dom(B) = \{b, b'\}$). Si $a \succ a'$ et $a : b \succ b'$, alors $a' : b' \succ b$. Nous en déduisons que $ab \succ ab' \succ a'b' \succ a'b$. Cependant, une variable Z conditionnant X conditionnant elle-même Y ($Pa(X) = \{Z\}$ et $Pa(Y) = \{X\}$) ne va pas forcément conditionner Y. Cela signifie que les CP-nets ne sont pas forcément transitifs.

Définition 1.17 (CP-net acyclique)**.**
Un ***CP-net acyclique*** *est un CP-net dont le graphe est acyclique. De plus, si l'ensemble des variables* $\mathbf{V}$ *du CP-net sont des variables binaires, alors nous parlerons de* ***CP-net acyclique à variables binaires****, ou plus simplement CP-net acyclique binaire.*

Définition 1.18 (CP-net non borné)**.**
Soit $\mathcal{N}$ *un CP-net à* n *variables.* $\mathcal{N}$ *est dit* ***non borné*** *si le nombre maximum de variables parentes par variable est égal à* $n - 1$*.*

1.4.1.3 Ordre partiel, problèmes de cycles et complexité

Les CP-nets, par l'intermédiaire de leurs CP-règles, permettent de représenter les préférences des valeurs de variables, grâce à cette notion de *ceteris paribus*, qui suppose que seule une variable verra ses valeurs changées. Les CP-règles induisent donc des préférences entre paires d'objets *ceteris paribus*, où seule la variable concernée par la CP-règle sera différente entre les deux objets. Lorsque l'ensemble des préférences des CP-règles sont mises bout-à-bout, un ordre partiel naturel sur les objets est alors défini sur l'ensemble des variables du CP-net [BBD+04].

Ordre partiel des objets

Définition 1.19 (Préférence *ceteris paribus* et swap)**.**
Soit $\mathbf{V}$ *un ensemble de variables. Soient* $\mathbf{o}, \mathbf{o}' \in Dom(\mathbf{V})$ *deux objets. On appelle* $\mathbf{o} \succ \mathbf{o}'$ *une* ***préférence ceteris paribus*** *si la projection* $\mathbf{o}[V]$ *est différente de la projection* $\mathbf{o}'[V]$ *pour une variable* $V \in \mathbf{V}$*, et* $\mathbf{o}[X] = \mathbf{o}'[X], \forall X \in \mathbf{V} \setminus \{V\}$*.*
On appelle de plus ***swap*** *(ou* ***flip****), que l'on note par* $(\mathbf{o}, \mathbf{o}')_V$*, la préférence ceteris*

paribus $\mathbf{o} \succ \mathbf{o}'$ *avec* $\mathbf{o}[V] \neq \mathbf{o}'[V]$ *pour une variable* $V \in \mathbf{V}$ *et* $\mathbf{o}[X] = \mathbf{o}'[X], \forall X \in \mathbf{V} \setminus \{V\}$. *La variable* V *est alors appelée* ***variable de swap***.

Définition 1.20 (Séquence de swaps)**.**
Soient $\mathcal{N}$ *un CP-net, et* $\mathbf{o}, \mathbf{o}' \in Dom(\mathbf{V})$ *deux objets quelconques. Une* ***séquence de swaps*** *correspond à un chemin de swaps de type* $(\mathbf{o}_1, \mathbf{o}_2)_{V_1}, (\mathbf{o}_2, \mathbf{o}_3)_{V_2}, \ldots, (\mathbf{o}_{m-1}, \mathbf{o}_m)_{V_{m-1}}$ *démarrant à* $\mathbf{o}_1$ *pour arriver à* $\mathbf{o}_m$ *(avec* $V_1, \ldots, V_{m-1} \in \mathbf{V}, m \leq |V|$*).*

Exemple 1.11. *Reprenons l'Exemple 1.1 ayant un ensemble* $\mathbf{V} = \{A, H, B\}$*, décrit par le CP-net de la Figure 1.5, dont l'ordre partiel est donnée dans la Figure 1.6. Considérons deux objets* $\mathbf{o} = (sp, tsh, sh)$ *et* $\mathbf{o}' = (sp, ch, pa)$*. Une séquence de swaps permettant de passer de* $\mathbf{o}$ *à* $\mathbf{o}'$ *correspond par exemple à* $(\mathbf{o} = (sp, tsh, sh), (sp, tsh, pa))_B, ((sp, tsh, pa), \mathbf{o}' = (sp, ch, pa))_H$.

Définition 1.21 (Relation de dominance et ordre partiel des objets [BBD+04])**.**
Soit $\mathcal{N}$ *un CP-net sur un ensemble* $\mathbf{V}$*. Soient deux objets* $\mathbf{o}$ *et* $\mathbf{o}'$*. On dit que* $\mathbf{o}$ ***domine*** $\mathbf{o}'$ *si et seulement si il existe une séquence de swaps au sein de* $\mathcal{N}$ *partant de* $\mathbf{o}$ *et se terminant à* $\mathbf{o}'$*. Cette relation de dominance est alors notée* $\mathbf{o} \succ_{\mathcal{N}} \mathbf{o}'$ *et définit l'****ordre partiel des objets*** *induit par le CP-net* $\mathcal{N}$.

Variable	Valeurs
A : Activité	sp : sport ma : mariage
H : haut de vêtement	tsh : t-shirt ch : chemise
B : bas de vêtement	sh : short pa : pantalon

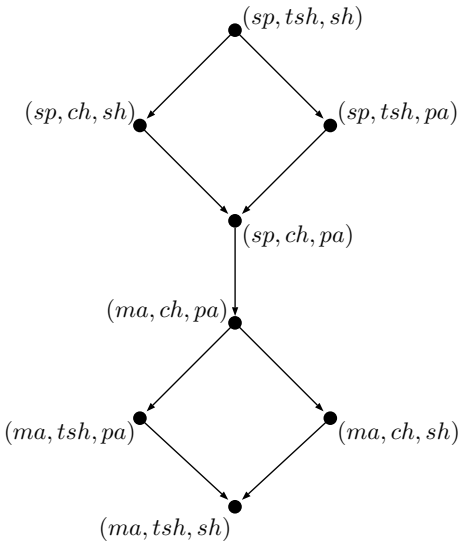
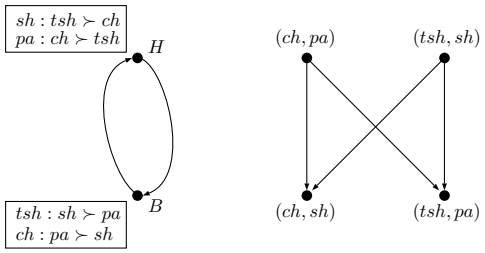


FIGURE 1.6 – Ordre partiel des objets induit par le CP-net de la Figure 1.5.

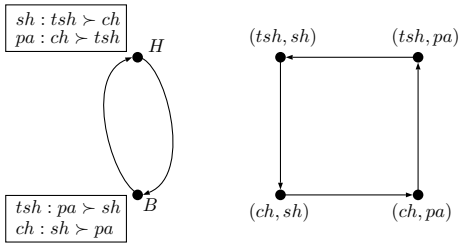
Problèmes de cycles

Les CP-nets illustrés jusqu'à présent sont des CP-nets ayant un graphe acyclique (voir la Figure 1.5 page 32), dont l'ordre partiel des objets associé est également acyclique. Cette propriété n'est pas systématique, et il est possible d'obtenir un CP-net contenant un cycle dans le conditionnement de ses variables. Cela peut

arriver à la suite d'une erreur lors de la transmission de l'information, d'une erreur de la personne exprimant ses préférences, ou bien même par une incohérence des vraies préférences, ce qui aura pour incidence de faire apparaitre dans le CP-net des variables se conditionnant mutuellement. Pour reprendre l'Exemple 1.9 page 30, nous pouvons imaginer une sorte de « cercle vicieux » où une personne n'arrivera pas à se décider sur la tenue à porter (chemise avec short, ou chemise avec pantalon, etc...). Cela induit que $Pa(H) = \{B\}$ et $Pa(B) = \{H\}$. Ce cycle au sein du graphe du CP-net peut entraîner un cycle au sein de l'ordre partiel associé (voir la Figure 1.7).



(a) Considérons les préférences suivantes : « je préfère porter un pantalon lorsque je mets une chemise » (et « je préfère porter un short lorsque je mets un tee-shirt ») et « je préfère porter une chemise lorsque je mets un pantalon » (et « je préfère mettre un short lorsque je porte un tee-shirt »). L'ordre partiel reste acyclique car les préférences restent cohérentes (la cohérence est, dans ce cas, renforcée par une « équivalence » entre les préférences).



(b) Considérons les préférences suivantes : « je préfère porter un tee-shirt lorsque je mets un short » (par contre « de temps en temps j'aime porter une chemise lorsque je mets un short ») et « je préfère porter une chemise lorsque je mets un pantalon » (par contre « en été, je préfère porter un tee-shirt lorsque je mets un pantalon »). L'ordre partiel fait apparaitre un cycle car les préférences ne sont plus cohérentes.

FIGURE 1.7 – Nous reprenons ici l'ensemble **V** issu de l'Exemple 1.1 page 12 privé de sa variable A. Cette fois, les deux variables se conditionnent mutuellement.

Cette incohérence des préférences exprimée dans la Figure 1.7b est connue sous le nom de **paradoxe de Condorcet** [Arr12], qui considère les objets inclus dans un cycle comme étant tous équivalents du fait qu'il ne soit pas possible d'en sélectionner un préféré aux autres. Ce paradoxe est relativement fréquent dans un contexte de comparaison par paire, car la personne les exprimant ne possède qu'une vision réduite de ses préférences, dont les incohérences sont alors difficilement visibles.

Définition 1.22 (Cohérence)**.**
*On dit qu'un CP-net $\mathcal{N}$ est **cohérent** lorsque son ordre partiel associé est acyclique. Dans le cas contraire, le CP-net $\mathcal{N}$ est **incohérent**.*

Nous avons vu au travers de la Figure 1.7 qu'un CP-net cyclique peut être incohérent du fait qu'il n'est pas toujours possible d'obtenir un ordre partiel associé acyclique. Il est cependant connu de [BBD+04] que tout CP-net acyclique est cohérent, car il assure une acyclicité de son ordre partiel associé, quelles que soient les préférences présentes dans les CP-tables du CP-net.

Définition 1.23 (Séparabilité)**.**
Soit $\mathcal{N}$ un CP-net sur un ensemble $\mathbf{V}$. On dit que $\mathcal{N}$ est ***séparable*** *si et seulement si $Pa(V) = \emptyset, \forall V \in \mathbf{V}$.*

Un CP-net séparable est donc par définition acyclique, et possède un nombre de règles (non conditionnelles) égal au nombre de variables. Enfin, nous définissons l'équivalence entre deux CP-nets, signifiant que deux CP-nets aux structures différentes peuvent malgré tout représenter les mêmes préférences.

Définition 1.24 (Équivalence)**.**
Soient $\mathcal{N}$ et $\mathcal{N}'$ deux CP-nets sur le même ensemble de variables $\mathbf{V}$. $\mathcal{N}$ et $\mathcal{N}'$ sont dits ***équivalents****, que l'on note par $\mathcal{N} \equiv \mathcal{N}'$, si et seulement si toutes les préférences de l'un sont représentables dans l'autre (et vice versa) :*

$$\forall \mathbf{o}, \mathbf{o}' \in Dom(\mathbf{V}),\ \mathbf{o} \succ_{\mathcal{N}} \mathbf{o}' \iff \mathbf{o} \succ_{\mathcal{N}'} \mathbf{o}'. \tag{1.12}$$

Complexité

En raison de l'intuitivité de leur structure, de nombreux chercheurs se sont attelés à analyser la complexité des CP-nets, aussi bien d'un point de vue purement combinatoire, que d'un point de vue apprenabilité du modèle. Nous pouvons citer entre autres les travaux de Boutilier *et al.* [BBD+04], Lang *et al.* [LM08, GLTW08, CKL+10], de Domshlak et Brafman [DB02], et de Alanazi *et al.* [AMZ16].

Une notion primordiale dans une structure de préférences concerne la détermination de la dominance entre deux objets $\mathbf{o}$ et $\mathbf{o}'$ comme expliqué dans la Définition 1.21 page 33. La dominance entre deux objets *ceteris paribus* $\mathbf{o}$ et $\mathbf{o}'$ est immédiate dans le cas des CP-nets acycliques : après avoir isolé l'état des variables parentes $\mathbf{u}$ de V, la CP-règle identifiée par V et $\mathbf{u}$ permet de savoir directement si $\mathbf{o} \succ \mathbf{o}'$, $\mathbf{o}' \succ \mathbf{o}$, ou si ces deux objets sont incomparables ou équivalents. Cette dominance devient cependant PSPACE-Complète lorsque $\mathbf{o}$ et $\mathbf{o}'$ sont deux objets quelconques [GLTW08, AGJ+17]. Effectivement, les CP-règles ne permettent pas de déterminer immédiatement la dominance, il est alors nécessaire de chercher une séquence de swaps (voir la Définition 1.20 page 33) partant du premier pour arriver au second objet. Cette séquence peut être exponentielle si l'on imagine deux objets se trouvant aux extrémités d'un ordre total, qu'il est en plus nécessaire de stocker dans le cas où certaines préférences soient incohérentes (voir la Section 1.4.1.3 page 32).

Le problème de la recherche de l'**objet le plus préféré** (pouvant être vu comme le problème de la recherche de l'objet **non-dominé**) est également très lié au problème de dominance, et est linéaire dans le cas des CP-nets acycliques : il suffit de sélectionner, pour chaque variable en commençant par celle n'ayant aucun parent, la valeur préférée de chacune, puis parcourir le CP-net en tenant compte de la valeur des différentes variables parentes afin de construire l'objet le plus préféré. Cette recherche devient cependant NP-Complète dans le cas d'un CP-net cyclique [DRVW09].

Une **classe** de CP-nets, notée $\mathcal{H}$, correspond à l'ensemble des CP-nets respectant une propriété donnée. La définition suivante introduit notamment la classe des CP-nets acycliques à degré borné, correspondant à la classe des CP-nets étudiée dans ce manuscrit.

Définition 1.25 (Classe des CP-nets acycliques à degré borné)**.**
La classe des CP-nets acycliques de degré borné k, notée $\mathcal{H}^k_{acy}$, correspond à l'ensemble des CP-nets acycliques dont le nombre de variables parentes par variable est inférieur ou égal à k. De manière plus formelle, soit $\mathbf{V}$ un ensemble de variables, alors $\mathcal{H}^k_{acy} = \{\mathcal{N} \mid \mathcal{N}$ est acyclique, et $|Pa(V)| \leq k,\ \forall V \in \mathbf{V}\}$.

Enfin, citons les travaux de Chevaleyre *et al.* [CKL$^+$10] et de Alanazi *et al.* [AMZ16] portant sur l'apprenabilité des CP-nets. Ces travaux utilisent la **dimension de Vapnik-Chervonenkis** (ou **VC-dimension**) [VC15] afin de démontrer de nombreuses propriétés de complexité. La VC-dimension est notée $Dim(\mathcal{H})$, où $\mathcal{H}$ correspond à une **classe** (voir la Définition 1.25) de CP-nets. Elle nous renseigne sur la taille du plus grand ensemble d'exemples (ici de swaps) pour lequel toutes les données sont bien classées. Ainsi, la VC-dimension de la classe des CP-nets acycliques à variables binaires ayant au plus k variables parentes par variable, notée $Dim(\mathcal{H}^k_{\text{acy}})$, peut notamment être bornée par $(n-k)2^k+2^k-1 \leq Dim(\mathcal{H}^k_{\text{acy}}) \leq \min\{2^n - 1, n2^k + (O(kn\log(n-1))\}$ [AMZ16], avec n le nombre de variables du CP-net.

1.4.2 Extensions et généralisations des CP-nets

Le modèle de CP-net, bien qu'étant compact et intuitif, présente des restrictions comme

(i) le modèle *ceteris paribus* obligeant à avoir des préférences entre objets quasi identiques,

(ii) la notion d'**importance** entre variables, qu'il est difficile de modéliser au-delà d'une seule variable parente (comme nous avons pu le voir précédemment).

Nous présentons ici quatre modèles permettant de généraliser les CP-nets. Le premier modèle, appelé **PCP-net**, ajoute une dimension probabiliste aux CP-nets, où chaque préférence de chaque CP-table est décrite au travers d'une probabilité d'apparition. Le deuxième modèle, appelé **TCP-net**, étend les CP-nets en ajoutant la notion d'importance entre variables. Le troisième modèle, appelé **CI-net** se base sur la propriété de monotonie afin de construire des règles de préférences. Enfin, le quatrième modèle présenté, appelé **CP-théorie**, est une généralisation du modèle de préférences conditionnelles.

1.4.2.1 PCP-net

Comme nous venons de le voir, les CP-nets possèdent l'avantage de pouvoir synthétiser les préférences conditionnelles dans un graphe de façon intuitive. Cependant le côté déterministe des préférences qu'il réunit rentre en conflit avec la réalité des

préférences, qui ne sont pas forcément fiables comme le laisse supposer le modèle classique des CP-nets. En effet, il est possible qu'une personne ne soit pas certaine des préférences qu'elle exprime, et souhaite alors donner un ordre de grandeur dans la confiance qu'il octroie à ses préférences. De façon analogue, suite à une erreur dans la réception automatique de préférences via un programme, il peut être important d'accorder un degré de confiance sur les préférences que l'on récupère. Enfin, dans un cadre multi-utilisateurs, où des préférences peuvent logiquement entrer en conflit, il peut être délicat de représenter de manière déterministe ces préférences.

Le modèle étudié dans cette partie s'inscrit dans les nombreux (et non-exhaustifs) cas cités précédemment, où il devient nécessaire d'attribuer une notion de **probabilité** aux préférences stockées. Pour répondre à ce besoin, plusieurs recherches ont été menées. Nous recensons à l'heure actuelle deux grands modèles de CP-nets probabilistes ayant été étudiés en parallèle : une version menée par Bigot *et al.* dans [BZFM13, Big15] s'attachant à proposer un modèle de CP-net où chaque variable possède une distribution de probabilité relative à sa CP-table, et une version proposée par Cornelio *et al.* [CGM+12, CGM+13, CGG+15, Cor16] dans laquelle les auteurs ajoutent une distribution de probabilité sur les conditions entre variables, en plus de la distribution sur les préférences. Cornelio *et al.* proposent également, dans leur version, une transformation des PCP-nets en réseaux bayésiens.

Définition 1.26 (PCP-net [BZFM13])**.**
Soit $\mathbf{V}$ *un ensemble de variables. Un* ***réseau de préférences conditionnelles probabilistes****, ou* ***PCP-net****, est un graphe orienté* $\mathcal{N}p = (\mathbf{V}, \mathbf{A}, PCPT(\mathbf{V}))$*, où* $\mathbf{V}$ *correspond à l'ensemble des variables du PCP-net,* $\mathbf{A}$ *correspond à l'ensemble des arcs du graphe, avec* $(V, V') \in \mathbf{A} \iff V \in Pa(V')$ *(*$V, V' \in \mathbf{V}$*), et* $PCPT(\mathbf{V})$ *correspond à l'ensemble des PCP-tables de toutes les variables de* $\mathbf{V}$*. Une* ***PCP-table*** *sur* $V \in \mathbf{V}$*, notée* $PCPT(V)$*, est un ensemble de* ***PCP-règles*** *du type* $(\mathbf{u} : v \succ v', p)$*, avec* $\mathbf{u} \in Dom(Pa(V))$*,* $v, v' \in Dom(V)$*, et* $p \in [0, 1]$ *une probabilité de confiance sur la préférence décrite par la CP-règle.*

Un exemple de PCP-net reprenant l'Exemple 1.9 page 30 est donné dans la Figure 1.8.

Variable	Valeurs
A : Activité	sp : sport ma : mariage
H : haut de vêtement	tsh : t-shirt ch : chemise
B : bas de vêtement	sh : short pa : pantalon

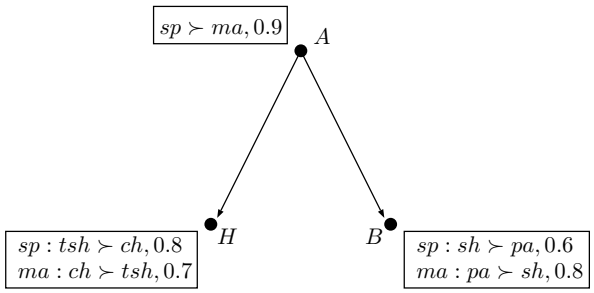


FIGURE 1.8 – Exemple de PCP-net sur l'ensemble $\mathbf{V} = \{A, H, B\}$ de l'Exemple 1.9 page 30.

Dans le cas où les variables du PCP-net sont binaires ($|Dom(V)| = 2, \forall V \in \mathbf{V}$), il est facile de déterminer la probabilité de la préférence inverse d'une CP-règle. Soit

$(\mathbf{u} : v \succ v', p)$ avec $\mathbf{u} \in Dom(Pa(V))$, $Dom(V) = \{v, v'\}$, et p une probabilité de confiance sur la CP-règle. Si la probabilité que $v \succ v'$ est p, alors la probabilité que $v' \succ v$ est $1 - p$. Considérons par exemple la règle $(\emptyset : a \succ a', 0.8)$, alors la règle inverse aura comme probabilité $(\emptyset : a' \succ a, 0.2)$.

Cette définition d'un PCP-net est identique aussi bien chez Bigot *et al.* [BZFM13] que chez Cornelio *et al.* [CGM+13]. La différence entre les deux modèles apparaît chez Cornelio *et al.* au niveau du lien entre variables, où une probabilité peut également être présente. De plus, Cornelio propose dans [Cor16] une transformation directe d'un CP-net vers un réseau bayésien, permettant alors d'utiliser toute les méthodes d'apprentissage issues de la littérature des réseaux bayésiens.

1.4.2.2 TCP-net

Une faiblesse des CP-nets est leur difficulté (voir leur impossibilité dans certains cas) à donner de l'importance à une variable plutôt qu'à une autre [BD02]. Ainsi, en reprenant l'exemple des tenues à porter, il est compliqué de signifier que toutes les situations où nous portons un tee-shirt, par exemple, seront préférées à toutes celles où nous portons une chemise. Brafman *et al.* ont donc entrepris de chercher dans [BD02, BDS06] une extension aux CP-nets résolvant, entre autres, cette contrainte. Cette extension se traduit par l'ajout de types d'arcs légèrement différents au sein du graphe dirigé du CP-net classique : un arc d'importance et un arc d'importance conditionnelle.

Nous commençons par introduire l'importance d'une variable par rapport à une autre.

Définition 1.27 (Importance)**.**
Soit $\mathbf{V}$ *un ensemble de variables. Soient* $V, V' \in \mathbf{V}$ *une paire de variables et* $\mathbf{W} = \mathbf{V} \setminus \{V, V'\}$*. On dit que* V *est* ***plus importante*** *que* V'*, noté par* $V \triangleright V'$*, si pour tout état* $\mathbf{w} \in Dom(\mathbf{W})$ *et pour tout* $v_1, v_2 \in Dom(V)$*,* $v'_1, v'_2 \in Dom(V')$ *tels que pour* $v_1 \succ v_2$ *et* $v'_1 \succ v'_2$*, on a*

$$v_1 v'_1 \mathbf{w} \succ v_2 v'_2 \mathbf{w}.$$

Introduisons ensuite la notion d'importance conditionnelle.

Définition 1.28 (Importance conditionnelle)**.**
Soit $\mathbf{V}$ *un ensemble de variables. Soient* $V, V' \in \mathbf{V}$ *une paire de variables et* $\mathbf{Z} = \mathbf{V} \setminus \{V, V'\}$*. On dit que* V *est* ***plus importante*** *que* V' ***conditionnellement*** *à un état* $\mathbf{z} \in Dom(\mathbf{Z})$ *(ceteris paribus), noté par* $V \triangleright_{\mathbf{z}} V'$*, si et seulement si, pour tout état* $\mathbf{w} \in Dom(\mathbf{W})$ *on a*

$$v_1 v_2 \mathbf{z}\mathbf{w} \succ v'_1 v'_2 \mathbf{z}\mathbf{w},$$

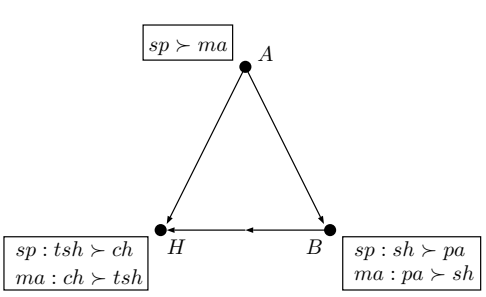
pour tout $v_1 \succ v_2$ *sachant* $\mathbf{z}\mathbf{w}$ *et* $v'_1 \succ v'_2$ *sachant* $\mathbf{z}\mathbf{w}$ *(*$v_1, v_2 \in Dom(V)$ *et* $v'_1, v'_2 \in Dom(V')$*).*

Ceci nous permet de définir formellement un TCP-net.

Définition 1.29 (TCP-net [BD02])**.**
Un ***TCP-net*** *(ou* ***Tradeoffs-enhanced CP-net****)* $\mathcal{TN}$ *est un sextuple* $(\mathbf{V}, \mathbf{A}, \mathbf{I}, \mathbf{CI}, CPT, CIT)$ *:*

- $\mathbf{V} = \{V_1, V_2, \ldots, V_n\}$ *est un ensemble de variables ;*
- $\mathbf{A}$ *est un ensemble d'arcs représentant les liens de parenté :* $\forall A = (V_j, V_i) \in \mathbf{A}$*,* $V_j = Pa(V_i)$ *;*
- $\mathbf{I}$ *est un ensemble d'*i*-arcs représentant l'importance d'une variable par rapport à une autre :* $\forall i = (V_j, V_i) \in \mathbf{I}$*,* $V_j \triangleright V_i$ *;*
- $\mathbf{CI}$ *est un ensemble de ci-arêtes représentant l'importance entre deux variables conditionnée par un ensemble d'états :* $\forall CI = (V_j, V_i, \mathbf{Z}) \in \mathbf{CI}$*, l'importance de* V_j *sur* V_i *est conditionnée par* $\mathbf{Z} \subseteq \mathbf{V} \setminus \{X_i, X_j\}$ *;*
- CPT *est l'ensemble des CP-tables du TCP-net ;*
- CIT *est une table associée à chaque ci-arête* (V_j, V_i)*, qui représente l'ensemble des règles de la forme* $\mathbf{z} : V_j \triangleright V_i$ *(ou* $\mathbf{z} : V_i \triangleright V_j$*), avec* $\mathbf{z} \in Dom(\mathbf{Z})$*.*

La Figure 1.9 donne une représentation d'un TCP-net pour l'exemple des tenues à porter.



Variable	Valeurs
A : Activité	sp : sport ma : mariage
H : haut de vêtement	tsh : t-shirt ch : chemise
B : bas de vêtement	sh : short pa : pantalon

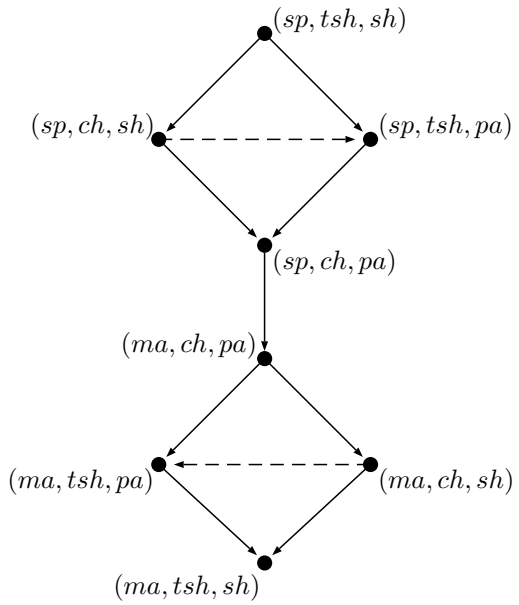


FIGURE 1.9 – Exemple de TCP-net $\mathcal{TN}$ ayant trois variables binaires $V = \{A, H, B\}$ à gauche (l'arc avec la double flèche représente un i-arc), et son ordre partiel des objets associé à droite (les arcs en pointillés représentent les nouvelles préférences liées à l'importance entre les deux variables).

1.4.2.3 CI-nets

Les CI-nets, pour *Conditional Importance Networks*, introduits par Bouveret *et al.* dans [BEL09], permettent d'exprimer l'idée qu'« ajouter plus d'éléments ne

fait que renforcer la préférence de l'objet courant ». Ces réseaux de préférences sont constitués de variables binaires définissant chacune un objet, et permettant de savoir si l'objet est présent ou absent. Ainsi, la propriété énoncée plus tôt permet de déduire un principe de **monotonie** de la structure de préférences.

Soit S un ensemble d'objets. Cet ensemble est divisé en sous-ensembles disjoints : l'ensemble $S^+ \subseteq S$ des objets **intéressants** (*i.e.*, ceux que l'on souhaite prendre), et l'ensemble $S^- \subseteq S$ des objets **inutiles** (*i.e.*, ceux que l'on souhaite éviter).

Définition 1.30 (Assertion d'importance conditionnelle)**.**
On appelle ***assertion d'importance conditionnelle*** *sur un ensemble d'objets S un quadruplet $\gamma = (S^+, S^-, S_1, S_2)$ de paires disjointes de sous-ensembles sur S décrits par $S^+, S^- : S_1 \triangleright S_2$. Cela signifie que « lorsque nous prenons les objets du sous-ensemble S^+ mais pas ceux de S^-, S_1 est* ***plus important*** *(i.e., est préféré) que S_2 ».*

La définition précédente permet d'introduire formellement un CI-net.

Définition 1.31 (CI-Net [BEL09])**.**
Un ***CI-Net*** *sur S est un ensemble $\mathcal{N} = \{\varphi_1, \ldots, \varphi_m\}$ d'assertions d'importance conditionnelle $\varphi_i = (S^+, S^- : S_1 \triangleright S_2)$ sur S.*

Nous pouvons observer que les objets issus des CI-nets sont des objets ayant une valeur cachée représentant leur utilité. Cette valeur n'étant jamais négative, la propriété de monotonie est donc très importante : $X \triangleright Y, \forall X, Y \in S$ avec $Y \subsetneq X$. Illustrons les CI-nets avec un exemple.

Exemple 1.12. *Prenons $\mathbf{S} = \{A, B, C\}$ avec $Dom(A) = \{a, \bar{a}\}, Dom(B) = \{b, \bar{b}\}$ et $Dom(C) = \{c, \bar{c}\}$, où chaque variable représente un objet, et $\bar{c}$ signifie que l'objet c n'est pas sélectionné. Soit $\mathcal{N} = \{\varphi_1 = (a, \emptyset, b, c)\}$ un CI-net comportant une seule assertion d'importance conditionnelle. Nous pouvons voir que $b \triangleright c$ et $ab \triangleright ac$ avec cette assertion, et que $ab \triangleright b$ avec la propriété de monotonicité.*

Les CI-nets et les TCP-nets sont similaires car ils permettent de comparer l'importance de variables (resp. d'objets) entre elles (resp. eux). L'un des avantages des CI-nets par rapport aux TCP-nets est qu'ils ne sont pas limités à des comparaisons par swaps (*ceteris paribus*). Cependant, ils ne peuvent pas exprimer des préférences entre les valeurs d'objets comme le font les CP-nets ou les TCP-nets (du fait qu'ils traitent ici d'objets présents ou non). Nous donnons dans la Figure 1.10 un exemple graphique de CI-net.

1.4.2.4 CP-théories

Etudions maintenant une généralisation des CP-nets et des TCP-nets. Les CP-théories, introduites en 2011 par Wilson [Wil11], ajoutent une dimension logique aux CP-nets, et permettent d'exprimer l'importance d'une (ou d'un groupe de) variable(s) directement au sein de règles de préférences.

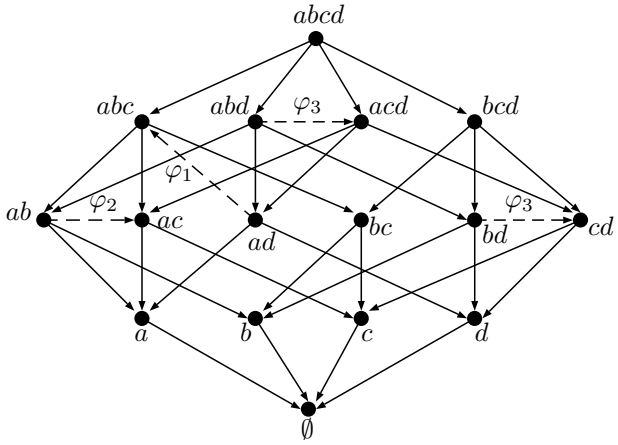


FIGURE 1.10 – Un CI-net $\mathcal{N} = \{\varphi_1, \varphi_2, \varphi_3\}$ pour quatre objets $\mathbf{S} = \{A, B, C, D\}$ avec $\varphi_1 = (a : d \triangleright bc), \varphi_2 = (a\bar{d} : b \triangleright c)$ et $\varphi_3 = (d : b \triangleright c)$. Les arcs pleins représentent la propriété de monotonie, alors que les arcs en pointillés représentent les différentes assertions de $\mathcal{N}$.

Définition 1.32 (CP-théorie)**.**
Une ***CP-théorie*** *est un ensemble d'assertions (théories)* $\mathcal{N} = \{\varphi_1, \dots \varphi_m\}$ *de la forme* $\varphi_i = (\mathbf{u} : x \succ x'[\mathbf{W}])$*, où* $\mathbf{u} \in Dom(\mathbf{U})$ *est un état,* $x, x' \in Dom(X)$ *deux valeurs de la variable* $X \notin \mathbf{U}$*, et* $\mathbf{W} \subseteq \mathbf{V} \setminus (\mathbf{U} \cup \{X\})$*. Un état vide est noté par* $\top$*.*

Nous illustrons cette règle avec l'Exemple 1.13 et la Figure 1.11.

Exemple 1.13. *Soit* $\mathbf{V} = \{A, H, B\}$ *:*

Variable	Valeurs
A : Activité	sp : sport ma : mariage
H : haut de vêtement	tsh : t-shirt ch : chemise
B : bas de vêtement	sh : short pa : pantalon

L'assertion $\top : sp \succ ma[\{H, B\}]$ *signifie que la variable* A *est* ***plus importante*** *que les variables* H *and* B *(*$A \triangleright \{H, B\}$*).*

1.4.3 Apprentissage de CP-nets

La compacité des CP-nets, et leur façon intuitive de représenter les préférences avec des règles, a permis l'émergence de nombreux algorithmes d'apprentissage. Nous pouvons notamment citer des algorithmes de régression, des algorithmes se basant sur une réduction vers 2-SAT [DMA09], des algorithmes se basant sur l'apprentissage par requête [KZ10, CKL$^+$10], des algorithmes se basant sur des tests d'hypothèses [LYX$^+$13], des algorithmes se basant sur l'apprentissage de l'ordre partiel

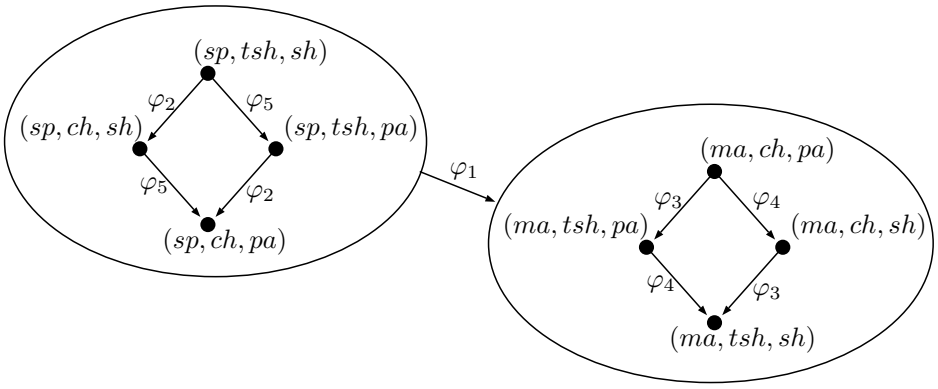


FIGURE 1.11 – Une CP-théorie $\mathcal{N} = \{\varphi_1, \ldots, \varphi_5\}$ sur trois variables $\mathbf{V} = \{A, H, B\}$ avec $\varphi_1 = (\top : sp \succ ma[\{H, B\}]), \varphi_2 = (\top : tsh \succ ch[\emptyset]), \varphi_3 = (ma : ch \succ tsh[\emptyset]), \varphi_4 = (ch : pa \succ sh[\emptyset])$ et $\varphi_5 = (sp : sh \succ pa[\emptyset])$.

des objets [LXW+14], et des algorithmes utilisant le test du χ^2 [LZLZ18]. La suite de cette partie se focalisera sur trois d'entre-eux : le premier algorithme est un apprentissage hors ligne proposé par Liu *et al.* [LXW+14] qui présente la particularité d'apprendre des CP-nets à partir de préférences incohérentes. Le deuxième algorithme présenté est proposé par Koriche *et al.* [KZ10]. Il s'agit d'un algorithme d'apprentissage par requête actif où le CP-net sera construit au fur et à mesure des retours d'un utilisateur à des questions posées par l'algorithme. Cet algorithme présente l'intérêt de pouvoir apprendre le CP-net avec un nombre réduit de requêtes. Enfin, le troisième algorithme étudié, proposé par Guerin *et al.* [GAG13], permet un apprentissage en ligne à partir de requêtes légèrement différentes de celui de Koriche *et al.* [KZ10].

1.4.3.1 Apprentissage de l'ordre partiel des objets

Nous nous plaçons ici dans le cadre particulier d'une base de données S incohérente (*i.e.*, dont l'ordre partiel des objets contient des cycles). La procédure proposée par Liu *et al.* [LXW+14] tente de supprimer, de la façon la plus efficace possible, ces cycles, tout en maximisant le nombre de comparaisons de S valides dans le CP-net appris.

Dans ce travail, le CP-net n'est pas vu comme la structure à apprendre. L'objectif est, à partir d'une base de données incohérente S et d'un ordre partiel des objets associé à cette base, contenant alors des cycles, de déterminer une base S' cohérente dans laquelle un nombre minimum de préférences a été supprimé afin d'obtenir un ordre partiel acyclique.

Considérons ici une matrice de données A de taille $l \times l$, où l représente le nombre d'objets différents présents dans S. Un élément de la matrice A, noté p_{ij}, représente une intensité $p \in \mathbb{N}$ sur la préférence $\mathbf{o}_i \succ \mathbf{o}_j$ (plus l'intensité est élevée, plus la

confiance sur la préférence est élevée).

$$A = \begin{pmatrix} 0 & p_{12} & \cdots & p_{1l-1} & p_{1l} \\ p_{21} & 0 & & & p_{2l} \\ \vdots & & \ddots & & \vdots \\ p_{11-1} & & & 0 & p_{l-1l} \\ p_{l1} & p_{2l} & \cdots & p_{ll-1} & 0 \end{pmatrix}.$$

Nous savons via la Section 1.4.1.3 page 32 que le test de dominance entre deux objets quelconques est PSPACE-Complet au sein des CP-nets. Les auteurs de [LXW+14] calculent cette dominance en utilisant l'exponentielle de la matrice A [ML03].

Définition 1.33 (Exponentielle de matrice)**.**
Soit A une matrice. L'exponentielle de la matrice A, notée e^A, est déterminée par :

$$e^A = \sum_{k=0}^{+\infty} \frac{A^k}{k!}, \tag{1.13}$$

$$(e^A - I) = \sum_{k=1}^{+\infty} \frac{A^k}{k!}, \tag{1.14}$$

avec $k \in \mathbb{N}$.

L'équation (1.13) permet, pour un k et une matrice A donnée, de calculer tous les chemins de taille k partant de l'objet $\mathbf{o}_i$, jusqu'à l'objet $\mathbf{o}_j$. Il devient alors possible de déterminer les **incohérences** induites par la transitivité de la relation $\succ_S$ en vérifiant $(e^A - I)_{ii} \neq 0, \forall i \in \{1, \ldots, l\}$. Afin de simplifier la matrice obtenue, Liu *et al.* proposent d'utiliser la fonction de seuil suivante :

$$th((e^A - I))_{ij} = \begin{cases} 1 & \text{si } (e^A - I)_{ij} \neq 0, \\ 0 & \text{sinon.} \end{cases} \tag{1.15}$$

Ces différentes opérations effectuées sur la matrice de préférences A associée à S permettent de détecter les cycles de préférences entre les objets. Il faut donc rechercher une matrice d'adjacence A', représentant l'ordre partiel acyclique (et donc cohérent) de S, à partir de la matrice de poids (à maximiser) A pouvant contenir des cycles. Cette matrice A' est calculée grâce à un un programme linéaire dont la fonction objectif est

$$\max g(A') = tr(A^T th(e^{A'} - I)), \tag{1.16}$$

avec $tr(X)$ la trace d'une matrice X.

Ce programme linéaire est ensuite résolu à l'aide d'un algorithme de *branch-and-bound*, puis le CP-net appris est reconstruit à partir de cet ordre via un algorithme de transformation. Ce dernier algorithme se base sur les deux points suivants :

(i) La propriété d'inversion de la préférence sur les valeurs d'une variable V par rapport aux valeurs de chacune de ses variables parentes Q (les variables conditionnées, voir la Définition 1.12) ;

(*ii*) Les préférences *ceteris paribus* (voir la Définition 1.19 page 33).

L'algorithme de *branch-and-bound* procède en ajoutant dans un premier temps les parents de chaque variable, puis met à jour la CP-table de chaque variable dans un second temps. Il est intéressant de noter que cet algorithme peut générer des CP-nets cycliques. Cependant, étant donné que programme linéaire précédent rend l'ordre partiel des objets acyclique, le CP-net appris $\mathcal{N}_L$ assure la cohérence des préférences.

La complexité de l'algorithme de *branch-and-bound* est relativement élevée du fait qu'il se base sur l'ordre partiel des objets, qui est de taille exponentielle par rapport au nombre de variables. Ainsi, considérons k comme le nombre d'arcs présents dans l'ordre partiel. L'algorithme de transformation de l'ordre partiel en CP-net admet alors une complexité en $O(k^2)$. La procédure de *branch and bound* admet quant à elle une complexité de l'ordre de $O(2^k)$ du fait d'un parcours complet, dans le pire des cas, de l'arbre de recherche.

Plaçons-nous maintenant du point de vue du nombre de variables n et supposons que toutes les variables sont binaires. L'ordre partiel des objets contient alors 2^n objets, et le nombre d'arcs maximum peut donc être exprimé par $k' = \frac{2^n(2^n-1)}{2}$. Dans le pire des cas, l'algorithme de *branch-and-bound* admet alors une complexité en $O(2^{2^n})$.

Une telle complexité ne permet pas à cet algorithme d'être utilisable sur un nombre élevé de préférences, ou sur des objets possédant une combinatoire trop importante. Il est alors nécessaire de trouver une méthode capable, elle aussi, de manipuler des préférences potentiellement bruitées, et admettant une complexité ne dépendant pas –ou peu– de la combinatoire des objets servants aux comparaisons, tels que les algorithmes d'apprentissage par requêtes.

1.4.3.2 Apprentissage par requêtes

Nous nous focalisons dans cette partie sur deux algorithmes d'apprentissage par requêtes : l'algorithme de Koriche et Zanuttini [KZ10], et l'algorithme de Guerin *et al.* [GAG13].

Nous avons introduit dans la Section 1.2.3 page 15 l'apprentissage par requêtes de manière générique. Rappelons que l'apprentissage par requêtes se base sur un système de questions-réponses entre l'apprenant (l'algorithme d'apprentissage), et un oracle (typiquement, un utilisateur). Nous redéfinissons maintenant les deux types de requêtes utilisées dans le cadre des CP-nets :

- Requêtes d'appartenance $\texttt{MQ}(\mathbf{o}, \mathbf{o}')$: l'apprenant transmet à l'oracle un couple d'objets $(\mathbf{o}, \mathbf{o}')$, qui retourne vrai si $\mathbf{o} \succ \mathbf{o}'$, et faux sinon ;
- Requêtes d'équivalence $\texttt{EQ}(\mathcal{N}, \mathcal{N}')$: l'apprenant transmet à l'oracle un CP-net $\mathcal{N}$, qui retourne vrai si les deux CP-nets sont équivalents, *i.e.*, $\mathcal{N} \equiv \mathcal{N}'$, avec $\mathcal{N}'$ le CP-net cible. Dans le cas contraire, l'oracle retourne faux, ainsi qu'un contre-exemple $(\mathbf{o}, \mathbf{o}')$ tel que $\mathbf{o} \succ_{\mathcal{N}'} \mathbf{o}'$ et $\mathbf{o} \not\succ_{\mathcal{N}} \mathbf{o}'$.

Algorithme de Koriche et Zanuttini [KZ10]

Nous rappelons que l'idée de base derrière un algorithme d'apprentissage par requêtes est d'apprendre au fur et à mesure une structure via l'utilisation de requêtes formulées à un oracle.

Notons que l'équivalence entre deux CP-nets $\mathcal{N}$ et $\mathcal{N}'$ (voir la Définition 1.24 page 35) ne signifie pas que les deux CP-nets, s'il sont équivalents, sont également identiques (voir la Figure 1.12). Il est possible de représenter les mêmes préférences tout en ayant une structure différente. D'après [KZ10], il existe un unique CP-net acyclique de taille minimum (au sens du nombre de CP-règles) pouvant représenter un ensemble de préférences $\succ$ donné. La procédure présentée ci-dessous propose d'apprendre un tel CP-net acyclique de taille minimum.

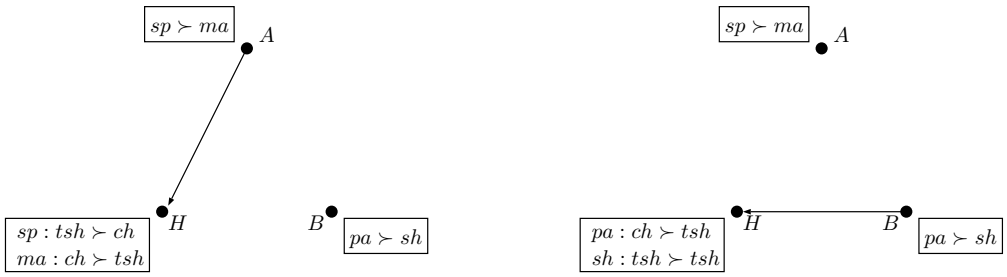


FIGURE 1.12 – Exemple d'un ensemble de préférences $\{(sp, tsh, sh \succ ma, tsh, sh), (sp, tsh, sh \succ sp, tsh, pa), (ma, ch, pa \succ ma, ch, sh)\}$ représenté par deux CP-nets.

Afin de faciliter la compréhension de l'algorithme, nous notons par $r_{\mathbf{u}}$ la CP-règle $(\mathbf{u} : v \succ v')$, et par $\bar{r}_{\mathbf{u}}$ la CP-règle inverse $(\mathbf{u} : v' \succ v)$.

Dans l'algorithme de [KZ10], chaque CP-règle $r_{\mathbf{u}} = (\mathbf{u} : v \succ v')$ (avec $\mathbf{u} \in Dom(Pa(V))$, $V \in \mathbf{V}$, et $Dom(V) = \{v, v'\}$) est associée à un objet $\mathbf{o}_{r_{\mathbf{u}}} = \mathbf{uz}v \in Dom(\mathbf{V})$ (avec $\mathbf{z} \in Dom(\mathbf{V} \setminus (Pa(V) \cup \{V\}))$ appelé **support** de la CP-règle $r_{\mathbf{u}}$. De plus, nous disons que le swap $(\mathbf{o}_{r_{\mathbf{u}}}[v], \mathbf{o}_{r_{\mathbf{u}}}[v'])_V$ est un **modèle** pour la CP-règle $r_{\mathbf{u}} = (\mathbf{u} : v \succ v')$. Cela permet de garder en mémoire l'objet ayant permis la création de la CP-règle.

Revenons aux requêtes d'équivalence décrites dans la Section 1.2.3 page 15. Lorsque l'équivalence $\texttt{EQ}(\mathcal{N}, \mathcal{N}')$ est invalidée, un contre-exemple $(\mathbf{o}, \mathbf{o}')$ est retourné. Nous distinguons deux types de contre-exemples :

- Contre-exemple positif signifiant que $\mathbf{o} \succ_{\mathcal{N}'} \mathbf{o}'$ mais que $\mathbf{o} \not\succ_{\mathcal{N}} \mathbf{o}'$ (signifiant que $\mathbf{o}$ n'est pas préféré à $\mathbf{o}'$ dans le CP-net $\mathcal{N}$, la CP-règle n'existant donc pas dans $\mathcal{N}$) ;
- Contre-exemple négatif signifiant que $\mathbf{o} \not\succ_{\mathcal{N}'} \mathbf{o}'$ tandis que $\mathbf{o} \succ_{\mathcal{N}} \mathbf{o}'$ (la CP-règle existe dans $\mathcal{N}$ mais n'est pas, ou plus, correcte).

Algorithme 1 : `apprentissage_acyclic_CPnet`($\mathbf{V}$) [KZ10]

Entrées :
$\mathbf{V}$: ensemble de variables ;
Un oracle ayant $\mathcal{N}_T$ comme CP-net cible, et capable de répondre aux requêtes `EQ` ;
Résultat :
$\mathcal{N}_L$: CP-net acyclique appris ;

1 **début**
2 Soit $\mathcal{N}_L$ un CP-net vide sur $\mathbf{V}$;
3 **tant que** $\neg\texttt{EQ}(\mathcal{N}_L, \mathcal{N}_T)$ **faire**
4 Soit $(\mathbf{o}, \mathbf{o}')_V$ le contre-exemple retourné par $\texttt{EQ}(\mathcal{N}_L, \mathcal{N}_T)$ et $r_{\mathbf{o}[Pa(V)]} = (\mathbf{o}[Pa(V)] : \mathbf{o}[V] \succ \mathbf{o}'[V])$ la règle induite par le contre-exemple;
5 **si** $\mathbf{o} \not\succ_{\mathcal{N}_T} \mathbf{o}'$ *mais que* $\mathbf{o} \succ_{\mathcal{N}_L} \mathbf{o}'$ **alors**
6 $(\mathbf{o}, \mathbf{o}')_V$ est alors un modèle de la CP-règle $r_{\mathbf{o}[Pa(V)]}$ ayant comme support un objet $\mathbf{o}_{r_{\mathbf{o}[Pa(V)]}}$;
7 $Q \leftarrow \texttt{recherche\_parent}(V, \mathbf{o}, \mathbf{o}_{r_{\mathbf{o}[Pa(V)]}}, 0, n)$;
8 $Pa(V) \leftarrow Pa(V) \cup \{Q\}$;
9 Mettre à jour chaque CP-règle $r'_{\mathbf{u}} \in CPT(V)$ $(\mathbf{u} \in Dom(Pa(V)))$ avec la valeur $\mathbf{o}_{r'_{\mathbf{u}}}[Q]$ de son support $\mathbf{o}_{r'_{\mathbf{u}}}$;
10 **sinon**
11 $CPT(V) \leftarrow CPT(V) \cup \{r_{\mathbf{o}[Pa(V)]}\}$;
12 **si** $\mathbf{o}' \succ_{\mathcal{N}_L} \mathbf{o}$ **alors**
13 $(\mathbf{o}', \mathbf{o})_V$ est alors un modèle de la CP-règle $r_{\mathbf{o}[Pa(V)]}$ ayant comme support un objet $\mathbf{o}_{r_{\mathbf{o}[Pa(V)]}}$;
14 $Q \leftarrow \texttt{recherche\_parent}(V, \mathbf{o}', \mathbf{o}_{r_{\mathbf{o}[Pa(V)]}}, 0, n)$;
15 $Pa(V) \leftarrow Pa(V) \cup \{Q\}$;
16 Mettre à jour chaque CP-règle $r'_{\mathbf{u}} \in CPT(V)$ $(\mathbf{u} \in Dom(Pa(V)))$ avec la valeur $\mathbf{o}_{r'_{\mathbf{u}}}[Q]$ de son support $\mathbf{o}_{r'_{\mathbf{u}}}$;
17 **Retourner** $\mathcal{N}_L$;

L'Algorithme 1 décrit la procédure générale d'apprentissage. Considérons le CP-net $\mathcal{N}_L$ comme étant le CP-net appris, et $\mathcal{N}_T$ le CP-net cible, celui servant de base à l'apprentissage. Des requêtes d'équivalence sont alors lancées tant que l'utilisateur renvoie des contre-exemples :

- Dans le cas d'un contre-exemple négatif (Ligne 5), la CP-règle existe mais est incorrecte, il est alors nécessaire de la raffiner avec l'ajout d'une variable parente Q ;
- Dans le cas d'un contre-exemple positif (Ligne 10), aucune CP-règle n'existe pour représenter cette préférence, et une CP-règle est alors créée. La CP-règle créée permet alors de déduire $\mathbf{o} \succ_{\mathcal{N}_L} \mathbf{o}'$. Cependant, il est nécessaire de tester

la préférence inverse $\mathbf{o}' \succ_{\mathcal{N}_L} \mathbf{o}$ (Ligne 12). Si cette préférence existe, alors nous sommes dans le cas d'une incohérence (voir la Section 1.4.1.3 page 32). Il devient nécessaire de trouver une nouvelle variable parente (Ligne 14) afin d'affiner la CP-règle. Cette variable parente est déterminée par l'Algorithme 2.

Algorithme 2 : `recherche_parent`$(V, \mathbf{o}, \mathbf{o}', a, b)$ [KZ10]

Entrées :
V_i : variable courante ;
$\mathbf{o}$ et $\mathbf{o}'$: deux objets ;
a et b : deux nombres ;
Un oracle capable de répondre à une requête `MQ` ;
Résultat :
P : nouvelle variable parente ;

```
1  début
2      si a = b − 1 alors
3          Retourner P correspondant à la b^e variable de V;
4      j ← ⌊(a+b)/2⌋;
5      o'' ← o[o'[J]], où o'[J] correspond à la projection des valeurs des j
         premières variables de V sur l'objet o';
6      si MQ(o''[o[V_i]], o''[ō[V_i]]) = VRAI alors
7          Retourner recherche_parent(V, o, o', a, j);
8      sinon
9          Retourner recherche_parent(V, o, o', j, b);
```

Supposons que nous avons numéroté chaque variable (dans l'ordre lexicographique par exemple), c'est-à-dire $\mathbf{V} = \{V_1, V_2, \ldots, V_n\}$. Une recherche dichotomique est alors effectuée sur les variables de $\mathbf{V}$ jusqu'à trouver la variable d'indice le plus faible respectant la préférence $\mathbf{o}[V_i] \succ \mathbf{o}'[V_i]$ (Ligne 6), où i correspond à l'indice de la variable du contre-exemple courant. Ceci est fait en créant un objet factice $\mathbf{o}''$ prenant les valeurs des premières variables de $\mathbf{o}'$, et les valeurs des dernières variables de $\mathbf{o}$ (Ligne 5). La recherche est alors bornée par a et b tel que $a < b$, et est relancée récursivement.

La complexité d'un algorithme d'apprentissage par requêtes se base sur le nombre de requêtes d'appartenance et d'équivalence nécessaires à l'apprentissage complet de la structure. Ces requêtes étant définies de façon purement théorique avec l'intervention d'un oracle, il est difficile de leur donner une complexité précise (l'oracle peut être une base de données ou un utilisateur). Notons par m la complexité d'une requête d'appartenance, et par e la complexité d'une requête d'équivalence. Il est nécessaire, dans l'Algorithme 1, de lancer $|CPT(\mathbf{V})| + 1$ requêtes d'équivalence et l'Algorithme 2 requiert $a\lceil \log n \rceil$ requêtes d'appartenance, où a correspond au nombre d'arcs du CP-net de taille minimum $|CPT(\mathbf{V})|$. L'Algorithme 1 est donc bien un algorithme d'apprentissage par requêtes au sens de la Définition 1.8 page 17 car le

nombre de requêtes d'appartenance et d'équivalence nécessaires à l'apprentissage du CP-net est bien borné par un polynôme fonction du nombre de variables n. Il admet de plus une complexité globale temporelle en $O(e(|CPT(\mathbf{V})|+1) \times m(a\lceil \log n \rceil))$.

Enfin, Koriche et Zanuttini proposent dans [KZ10, Section 5] un algorithme supplémentaire se focalisant sur l'apprentissage de CP-nets arborescents. Il s'agit de CP-nets dont le graphe orienté est une arborescence, *i.e.*, un CP-net acyclique au sein duquel chaque variable ne peut posséder qu'une seule variable parente.

Cette procédure, malgré une complexité intéressante, n'est cependant pas adaptée, en raison de ses requêtes d'équivalence, à une utilisation réelle (nous verrons dans la suite de ce chapitre une procédure en ligne également basée sur des requêtes). Elle ne tient, en outre, pas compte des potentielles préférences incohérentes, au contraire de la procédure de Liu *et al.* [LXW+14].

Algorithme de Guerin *et al.* [GAG13]

Nous présentons enfin un algorithme d'apprentissage en ligne proposé par Guerin *et al.* [GAG13]. Il s'agit du seul algorithme en ligne traitant de l'apprentissage des CP-nets acycliques connu à ce jour. Il se base, comme [KZ10], sur l'apprentissage par requêtes. Cependant, il n'apprend pas à partir de swaps (déduits des comparaisons entre deux objets *ceteris paribus*), mais à partir d'objets quelconques.

Cet algorithme se divise en deux étapes principales :

- L'apprentissage d'un CP-net séparable via des requêtes du type « entre les valeurs v_1 et v_2 de la variable V, laquelle préférez-vous ? » ;
- Le raffinage du CP-net avec l'ajout de liens de parentés entre variables, sous condition d'acyclicité du CP-net via des requêtes à l'utilisateur entre deux objets quelconques.

Procédure générale (Algorithme 3)

Après initialisation du CP-net appris $\mathcal{N}_L$, une requête est faite à l'utilisateur pour chaque variable $V \in \mathbf{V}$ afin de déterminer la préférence entre les valeurs de V, et d'obtenir un CP-net séparable.

Nous pouvons voir cette première partie de l'algorithme comme une vue d'ensemble des préférences qui semblent le plus importantes pour l'utilisateur dans la plupart des cas.

Le CP-net est ensuite affiné avec l'ajout de variables parentes via l'Algorithme 4. Le présent algorithme fonctionnant en ligne, nous pouvons noter la présence de deux ensembles de variables $\mathbf{U}$ et $\mathbf{C}$ représentant respectivement l'ensemble des variables de **méfiance** et de **confiance**. Une variable est considérée comme une variable de confiance dès lors que l'Algorithme 4 ne retourne pas une CP-table associée vide (elle est considérée comme une variable de méfiance dans le cas contraire).

Algorithme 3 : `apprentissage_CPnet`(k, c)

Entrées :
$\mathbf{V}$: ensemble de variables ;
k : nombre maximum de parents ;
c : seuil de confiance ;
Un utilisateur capable de répondre aux requêtes posées ;
Résultat :
$\mathcal{N}_L$: un CP-net acyclique ;

```
1  début
2      Soit N_L un CP-net vide sur V;
3      C ← ∅;
4      U ← V;
5      pour V ∈ V faire
6          Requête à l'utilisateur : « que préférez-vous entre v ∈ Dom(V) et
             v' ∈ Dom(V) ? »;
7          si v ≻ v' alors CPT(V) ← (∅ : v ≻ v');
8          si v' ≻ v alors CPT(V) ← (∅ : v' ≻ v);
9      tant que des parents sont ajoutés faire
10         pour r = 1 à k faire
11             pour V ∈ U faire
12                 (Q, CPT(V)) ← recherche_parents(V, r, c, C);
13                 si CPT(V) ≠ ∅ alors
14                     A ← A ∪ {(Q, V)}, ∀Q ∈ Q;
15                     U ← U \ {V};
16                     C ← C ∪ {V};
17     Retourner N_L;
```

Recherche de parents (Algorithme 4)

Guerin *et al.* commencent par parcourir l'ensemble des sous-ensembles de variables parentes candidates pour V. Cela permet notamment de prendre en compte les interactions pouvant s'établir entre plusieurs variables parentes, qui ne sont pas forcément observables lorsqu'elles sont prises séparément. Cependant, cela augmente de manière radicale la complexité de l'algorithme, car il est alors nécessaire de tester tous les sous-ensembles possibles parmi les variables de confiance $\mathbf{C}$.

Pour chacun de ces sous-ensembles $\mathbf{Q} \in 2^{\mathbf{C}}$, une CP-table est générée via un algorithme de construction d'instance `creation_CPtable`$(V, \mathbf{P})$ de 2-SAT [Ash07]. Cette CP-table, si elle existe, est accompagnée d'une valeur de confiance $valConf$, puis affinée via des requêtes utilisateurs jusqu'à ce qu'elle soit considérée comme assez pertinente. La variable associée devient alors une variable de confiance et pourra donc, ultérieurement, être une variable candidate à devenir parente.

Algorithme 4 : `recherche_parents`$(V, r, c, \mathbf{C})$

Entrées :
V : variable courante ;
r : nombre maximum de parents ;
c : seuil de confiance ;
$\mathbf{C}$: ensemble des variables de confiance ;
Un utilisateur capable de répondre aux requêtes posées ;
Résultat :
$\mathbf{Q}$: ensemble de nouveaux parents ;
$CPT(V)$: nouvelle CP-table de V ;

```
1 début
2     pour Q ∈ 2^C et |Q| ≤ r faire
3         (CPT(V), valConf) ← creation_CPtable(V, Q);
4         tant que CPT(V) ≠ ∅ et valConf < c faire
5             Récupérer (o, o') deux objets issus d'une requête aléatoire sur V;
6             Requête à l'utilisateur : « que préférez-vous entre o et o' ? »;
7             (CPT(V), valConf) ← creation_CPtable(V, Q) sachant le
               résultat de la comparaison;
8         si CPT(V) ≠ ∅ alors Retourner (Q, CPT(V));
9     Retourner (∅, ∅);
```

Guerin *et al.* garantissent l'arrêt de leur procédure et montrent qu'elle pourra toujours récupérer un CP-net en sortie. Ils prouvent également que l'Algorithme 3 admet une complexité en $O(n^k \times c)$, avec n le nombre de variables, k le nombre maximum de parents et c le nombre de requêtes à effectuer pour avoir confiance en la CP-table. Dans le pire des cas, l'Algorithme 3 admet alors une complexité en $O(n^{n-1} \times c)$ (lorsque le CP-net n'est pas borné).

Cette procédure propose donc un apprentissage beaucoup plus rapide, et moins dépendant des données, que les algorithmes hors lignes vus précédemment. Cependant, elle ne permet pas la gestion de préférences potentiellement bruitées, et n'est donc pas adaptée pour un usage réel.

1.5 Problème étudié dans ce manuscrit

Nous nous plaçons ici dans le cadre d'un apprentissage de CP-nets acycliques en milieu bruité. Parmi les algorithmes existants, seuls deux [LYX+13, LXW+14] permettent de traiter le bruit inhérent aux données. Cependant, à cause de leur aspect hors ligne, ils ne permettent pas un apprentissage efficace des CP-nets. Ainsi, nous voulons pouvoir visualiser à tout moment le CP-net appris afin de représenter au mieux les données déjà observées. Nous proposons alors, dans la suite de ce manuscrit, deux algorithmes d'apprentissage de CP-nets en milieu bruité :

- Le premier, hors ligne, s'inspirant des algorithmes d'apprentissage par requête ;
- Le deuxième, proposant deux versions : une hors ligne, et une en ligne. Ces deux versions s'appuient sur une mesure d'entropie inspirée des arbres des décisions, ainsi que sur des compteurs statistiques de règles.

Formalisation du problème

Nous définissons tout d'abord plusieurs éléments :

- Soit V un ensemble de n variables ;
- Posons $Dom(V) = \{v, v'\}$ les valeurs possibles pour une variable $V \in \mathbf{V}$. On définit alors un ensemble de classes $Y_V = \{1_V, 0_V\}$ tel que
  - Si un swap $s = (\mathbf{o}, \mathbf{o}')_V$ implique que $v \succ v'$, alors on associe à s la classe 1_V ;
  - Si un swap $s = (\mathbf{o}, \mathbf{o}')_V$ implique que $v' \succ v$, alors on associe à s la classe 0_V.
- Nous posons $Y = \bigcup_{V \in \mathbf{V}} Y_V$, ainsi que l'ensemble d'entraînement $S = \{(s_1, y_1), \ldots, (s_m, y_m)\}$, où un couple (s_i, y_i) est appelé exemple, avec $s_i = (\mathbf{o}, \mathbf{o}')^i_V$ un swap associé à une variable de swap $V \in \mathbf{V}$, ainsi qu'à une classe $y_i \in Y_V$;
- Il est alors possible de diviser l'ensemble S en n sous-ensembles $S_V = \{(s = (\mathbf{o}, \mathbf{o}')_{V'}, y_s) \in S \mid V' = V\}$;
- Notons de plus que $S = \bigcup_{V \in \mathbf{V}} S_V$, et que $\forall V, V' \in \mathbf{V}$ $(V \neq V')$, $S_V \cap S_{V'} = \emptyset$.

Exemple 1.14. *Considérons l'ensemble de variables* $\mathbf{V} = \{A, H, B\}$*, ainsi qu'un ensemble de swaps*

$$S = \begin{array}{l} \{((ma, ch, sh), (ma, tsh, sh))_H, \quad ((sp, tsh, sh), (sp, ch, sh))_H, \\ \quad ((sp, tsh, sh), (ma, tsh, sh))_A, \quad ((sp, tsh, sh), (sp, tsh, pa))_B\} \end{array}$$

Il est ainsi possible de diviser l'ensemble S *en trois sous-ensembles disjoints* $S_H = \{((ma, ch, sh), (ma, tsh, sh))_H, ((sp, tsh, sh), (sp, ch, sh))_H\}$*,* $S_A = \{((sp, tsh, sh), (ma, tsh, sh))_A\}$*, et* $S_B = \{((sp, tsh, sh), (sp, tsh, pa))_B\}$*.*

On rappelle qu'une **hypothèse** est une fonction de décision $h_S : Dom(\mathbf{V})^2 \to \{0, 1\}$ permettant d'étiqueter un swap, à partir d'un ensemble d'entraînement S. Cette hypothèse h_S représente la décision prise par un CP-net pour le classement du swap passé en paramètre. Ainsi, ce CP-net va recevoir un swap, puis retourne une valeur permettant de déduire envers quel objet va la préférence :

$$h_S((\mathbf{o}, \mathbf{o}')_V) = \begin{cases} 1 & \text{si } \mathbf{o} \succ \mathbf{o}', \\ 0 & \text{sinon.} \end{cases} \tag{1.17}$$

Nous rappelons également qu'une **classe d'hypothèses** $\mathcal{H}$ correspond à l'ensemble des hypothèses ayant une propriété donnée. Ainsi, la classe $\mathcal{H}^k_{\text{acy}}$ correspond à la classe de tous les CP-nets acycliques bornés par k variables parentes.

Notion de bruit dans les préférences

Nous supposons dans notre travail que l'ensemble des préférences de S proviennent d'un CP-net acyclique cible noté $\mathcal{N}^*$. Ces données étant potentiellement bruitées, nous nous plaçons alors dans un cadre d'**apprentissage irréalisable** (aussi appelé **apprentissage agnostique**) [KSS94, SSBD14]. Cela signifie qu'il n'est pas possible de modéliser l'ensemble des préférences de S avec le CP-net cible $\mathcal{N}^*$ à cause de la présence de swaps bruités.

Ce bruit est représenté par des préférences inversées, *i.e.*, si la « véritable » préférence (celle modélisée par le CP-net cible $\mathcal{N}^*$) est $\mathbf{o} \succ \mathbf{o}'$, l'ensemble S pourrait contenir la préférence inverse $\mathbf{o}' \succ \mathbf{o}$ en plus de la vraie préférence. Ce bruit est formalisé par une probabilité $p \in [0, 1]$ inversant une préférence $\mathbf{o} \succ \mathbf{o}'$ issue de l'hypothèse cible $h^*_S \in \mathcal{H}$, *i.e.*, $\mathbb{P}(y_i \neq h^*_S(s_i)|s_i) \leq p$, avec $s_i \in S$ un swap, et y_i sa classe associée.

Comme nous l'avons signifié dans l'introduction générale, les causes de ce bruit sont multiples :

1. Des préférences contradictoires au sein d'une base de données multi-utilisateurs, où chaque utilisateur exprime, sans se tromper, ses préférences. Le bruit intervient alors au niveau des différences de point de vue, engendrant alors des préférences contradictoires ;
2. Des erreurs lors de la réception, ou la transmission de données, dues par exemple à un mauvais réseau informatique ou à un disque endommagé. Les préférences reçues peuvent alors être inversées par rapport à ce qu'elles étaient au moment de leur envoi.

Ce bruit n'est pas destructif, car il préserve les objets reçus (aucune perte d'information) en ne modifiant que certaines préférences entre ces objets. Les données réelles utilisées dans les différents tests prendront en compte un mélange de ces deux causes : il s'agit de bases de données multi-utilisateurs (contenant de potentielles contradictions) dont les préférences ont pu être altérées à cause d'erreurs de transmissions.

Mesures de performances utilisées

Cette formalisation du problème permet d'introduire des mesures permettant de tester l'efficacité de l'apprentissage des algorithmes présentés dans les chapitres suivants. Comment peut-on savoir que la structure apprise représente au mieux nos préférences ? Soit $h_S \in \mathcal{H}^k_{\text{acy}}$ une hypothèse. Il existe plusieurs mesures de précision, se basant aussi bien sur le nombre de CP-règles du CP-net appris, que sur le nombre de préférences de S. Cette distinction entre les préférences « brutes » (comparaison par paires au moyen de swaps) et les CP-règles du CP-net est importante. En effet, dans le cas d'un CP-net séparable (*i.e.*, un CP-net ne possédant aucun arc dans son graphe), une CP-règle peut être représentée par un nombre très important de swaps différents (prenons l'exemple de 3 variables indépendantes, nous aurons alors $2^2 = 4$

swaps distincts pouvant représenter la même préférence, et de manière générale pour n variables indépendantes, nous aurons 2^{n-1} swaps distincts représentant la même préférence). La maximisation de CP-règles du CP-net appris n'est donc, de notre point de vue, pas un critère pertinent. De plus, l'apprentissage se faisant sur un ensemble de swaps, le CP-net cible n'est pas forcément connu. De ce fait, la maximisation du nombre de préférences de la base de données dans le CP-net semble être la mesure la plus pertinente dans notre cas[6]. Nous définissons ainsi deux mesures de performance :

1. La première mesure utilisée sera une mesure de précision $prec : Dom(\mathbf{V}) \times Y \rightarrow [0, 1]$ définie de la manière suivante :

$$prec(S, h_S) = \frac{1}{|S|} \sum_{s \in S} \mathbb{1}(y_s = h_S(s)) \in [0, 1], \tag{1.18}$$

   avec $|S|$ correspondant à la taille de l'ensemble S, c'est-à-dire le nombre de swaps contenus dans S. La fonction indicatrice $\mathbb{1}(.)$ prend alors la valeur 1 si la condition indiquée est vraie, et prend la valeur 0 dans le cas contraire. L'algorithme d'apprentissage aura donc pour but de maximiser la valeur de la fonction de précision (Équation (1.18)) :

$$\hat{h}_S = \underset{h_S \in \mathcal{H}^k}{\operatorname{argmax}} \ prec(S, h_S). \tag{1.19}$$

   Nous utiliserons cette mesure d'efficacité (Équation (1.18)) comme mesure de test dans le Chapitre 2, car il s'agit d'une mesure relativement intuitive, permettant de refléter au mieux l'efficacité de l'algorithme proposé, se basant sur des requêtes spécifiques effectuées sur un oracle. Notons également l'utilisation de l'Équation (1.18) dans le Chapitre 3 afin d'avoir un point de comparaison entre les algorithmes des deux chapitres.

2. La deuxième mesure utilisée sera une fonction de perte $\ell : Dom(\mathbf{V})^2 \times Y \times Y \rightarrow \{0, 1\}$, pour tout $s \in S$:

$$\ell(s, y_s, h_S(s)) = -y_s \log(h_S(s)) - (1 - y_s) \log(1 - h_S(s)). \tag{1.20}$$

   Cette fonction met en évidence le nombre de préférences correctement modélisées par rapport au nombre de préférences ne pouvant l'être, sur un CP-net donné, au travers d'une fonction d'entropie. Ainsi, plus cette fonction est proche de zéro, plus le nombre de préférences modélisées est important. Notre objectif est dans ce cas de trouver un CP-net minimisant la perte sur l'ensemble des swaps de chaque sous-ensemble de S, *i.e.*, pour un sous-ensemble $S_V \subseteq S$ donné,

$$\hat{h}_S = \underset{h_S \in \mathcal{H}^k}{\operatorname{argmin}} \ \frac{1}{|S_V|} \sum_{s \in S_V} \ell(s, y_s, h_S(s)). \tag{1.21}$$

6. Notons également l'utilisation, dans [LXW+14, LZLZ18], d'une mesure de similarité comparant le nombre d'arcs entre deux CP-nets. Cependant, comme cette mesure de tient pas compte des préférences, nous ne la prendrons pas en compte.

De manière plus générale, nous cherchons à minimiser la perte sur l'ensemble des swaps de S. Pour cela, nous commençons par définir la perte sur un ensemble S_V de swaps :

$$\ell(S_V, h_S) = -\frac{1}{2}p' \ln p' - \frac{1}{2}(1-p')\ln(1-p'), \tag{1.22}$$

avec p' correspondant à la proportion de swaps de S_V ayant la même classe prédite (donnée par h_S) que la classe réelle (donnée par y_s, pour tout $s \in S_V$). Nous pouvons alors déduire, en réécrivant l'équation précédente, une perte globale sur l'ensemble S :

$$L(S, h_S) = \sum_{V \in \mathbf{V}} \frac{|S_V|}{|S|} \ell(S_V, h_S). \tag{1.23}$$

Nous cherchons alors un CP-net minimisant cette perte globale sur l'ensemble S :

$$\hat{h}_S = \underset{h_S \in \mathcal{H}^k}{\operatorname{argmin}} \; L(S, h_S). \tag{1.24}$$

Cette deuxième mesure (Équation (1.23)) sera utilisée dans le Chapitre 3, car la perte logarithmique est naturellement associée au gain d'information et à l'entropie, qui sont les notions fondamentales de l'algorithme proposé dans ce dernier chapitre. Notons de plus que la notion de regret ne sera pas utilisée malgré la présence d'un algorithme en ligne, car les résultats énoncés sont essentiellement expérimentaux. Ainsi, nous nous restreignons à l'utilisation de cette perte pour les versions hors ligne et en ligne qui seront étudiées dans le Chapitre 3.

Entrée :	une base de données bruitée S à variables binaires.
Résultat :	une hypothèse $\hat{h}_S$.
Précision :	maximiser l'Équation (1.18).
Erreur :	minimiser l'Équation (1.23).

Tableau 1.4 – Le problème de l'apprentissage de CP-nets.

Nous faisons l'hypothèse, dans ce manuscrit, et ce pour des raisons de temps de calcul, que la base de données S ne contient que des swaps. Effectivement, bien que les CP-nets appris soient des CP-nets acycliques comportant, de ce fait, des préférences cohérentes, le fait de considérer des préférences entre deux objets $\mathbf{o}$ et $\mathbf{o}'$ quelconques implique une recherche, dans le CP-net, de la préférence $\mathbf{o} \succ \mathbf{o}'$ ou $\mathbf{o}' \succ \mathbf{o}$. Boutilier *et al.* dans [BBD+04, Section 4] font la distinction entre deux types de tests : le test de dominance, ainsi que le test de classement. Supposons une préférence $\mathbf{o} \succ \mathbf{o}' \in S$ entre deux objets non nécessairement *ceteris paribus*. Le test de dominance sera alors un test permettant, au moyen de séquences de swaps comme vu précédemment, de vérifier la préférence $\mathbf{o} \succ \mathbf{o}'$ dans le CP-net appris. Le test de

classement, quant à lui, s'attachera à vérifier que la préférence inverse $\mathbf{o}' \succ \mathbf{o}$ n'existe pas dans le CP-net appris. Ce dernier test est linéaire en le nombre de variables dans le cadre des CP-nets acycliques. Ceci reste cependant un calcul supplémentaire (dépendant du nombre de variables) à effectuer pour chaque préférence observée, ce qui explique notre choix de le mettre de côté, bien que cela soit envisageable de l'intégrer à l'ensemble de nos procédures. Nous argumentons de plus ce choix par les tests effectués dans la suite de ce manuscrit, sur des données réelles ayant des préférences entre objets *ceteris paribus*, nous permettant alors d'apprendre nos CP-nets.

1.6 Conclusion

Nous avons montré dans ce chapitre que l'apprentissage de préférences est un vaste domaine de recherche. Nous nous sommes ici concentrés sur une structure de préférences assez particulière qui fonctionne sur la base de préférences conditionnelles *ceteris paribus*. Malgré les nombreuses études, notamment algorithmiques, à son sujet, l'apprentissage d'une telle structure peut être amélioré afin de traiter de manière la plus fiable possible les contradictions et autres préférences bruitées, tout en proposant un meilleur passage à l'échelle.

Les deux prochains chapitres proposerons deux algorithmes bien distincts, dont le premier tentera de résoudre le problème des préférences contradictoires dans une base de données multi-utilisateurs, tandis que le deuxième traitera de préférences bruitées de manière plus générale, avec une déclinaison en ligne de cet algorithme.

CHAPITRE 2

ALGORITHME D'APPRENTISSAGE PAR REQUÊTES DE CP-NETS ACYCLIQUES

Sommaire

Résumé

Ce chapitre présente un algorithme d'apprentissage de CP-nets acycliques inspiré des algorithmes d'apprentissage par requête de Koriche et Zanuttini [KZ10] et de Guerin *et al.* [GAG13], au sein duquel l'oracle considéré et interrogé correspondra à une base de données. Cette base de données aura la particularité d'être constituée de préférences de **plusieurs** utilisateurs. L'algorithme proposé traitera de la façon

la plus efficace possible les préférences parfois **contradictoires** présentes dans les bases de données. Nous utiliserons pour cela une sélection fine de chaque variable conditionnée et parente en revenant sur les définitions litérales de telles variables. Nous utiliserons de plus une liste spéciale de swaps qu'il faudra minimiser, et au sein de laquelle les préférences contradictoires seront stockées. L'accent sera mis, dans ce chapitre, sur les expériences effectuées sur l'algorithme d'apprentissage proposé. Nous testerons ainsi cet algorithme sur des données synthétiques et réelles. Nous le comparerons également à un algorithme basé sur un apprentissage par requêtes de la littérature [GAG13], ainsi qu'avec un algorithme hors ligne prenant en compte les incohérences [LXW+14]. Enfin, nous étudierons sa convergence numérique.

Liste des contributions de ce chapitre :

- Nouvel algorithme d'apprentissage hors ligne de CP-nets acycliques basé sur des requêtes utilisant les propriétés des variables parentes afin de minimiser les préférences bruitées.
- Expérimentations sur des données synthétiques et réelles.
- Comparaisons avec des algorithmes de l'état de l'art.

Ces travaux ont donné lieu à la présentation d'un article à l'édition 2016 du workshop DA2PL (From Multiple Criteria Decision Aid to Preference Learning) [LYMA16], ainsi qu'à un article publié dans le journal EJDP (EURO Journal on Decision Processes) [LYMA18].

2.1 Introduction

Les CP-nets, comme la plupart des modèles de représentation des préférences, sont moins compréhensibles lorsqu'il s'agit de représenter des préférences contradictoires. En pratique, par exemple, construire un CP-net à partir des préférences fournies par plusieurs utilisateurs peut être délicat dès lors que deux utilisateurs expriment des préférences contradictoires sur deux objets. Générer un CP-net en présence de telles préférences demande donc une gestion différente des préférences reçues afin d'éviter l'apprentissage de préférences minoritaires. La Figure 2.1 ci-dessous résume la procédure générale expliquée dans ce chapitre.

La particularité des algorithmes d'apprentissage par requête (ou d'apprentissage exact), est de considérer qu'il existe un oracle ayant une connaissance sur sa structure induite, mais qui n'est pas capable d'énumérer l'ensemble de ses préférences. Ces algorithmes utilisent deux types de requêtes posées à l'oracle (voir la Section 1.2.3 page 15). Nous nous plaçons dans ce chapitre dans un cadre où cet oracle est considéré comme étant une base de données possiblement multi-utilisateurs. L'algorithme proposé, bien que reposant sur des requêtes, n'est pas un algorithme d'apprentissage exact au sens de Angluin [Ang87], car il se base sur des requêtes légèrement

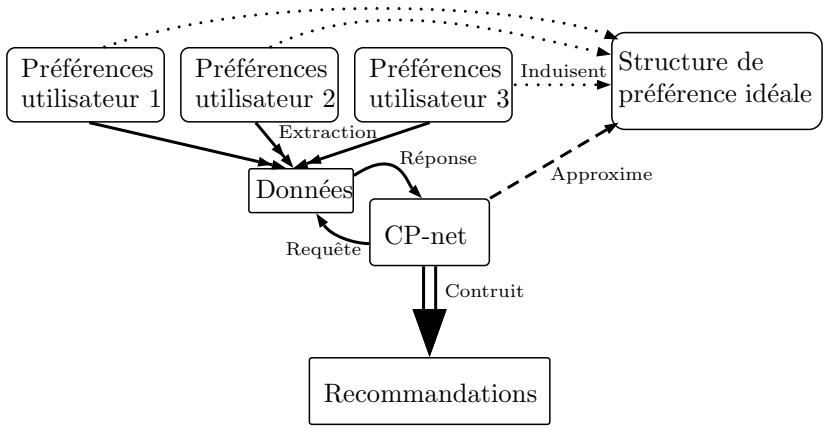


FIGURE 2.1 – Schéma global de l'apprentissage de CP-nets par requêtes.

différentes. Nous considérons de plus que cet oracle peut nous retourner des contre-exemples bruités, *i.e.*, des contre-exemples contradictoires à cause de conflits entre les préférences de plusieurs utilisateurs, ou bien à cause des erreurs d'un utilisateur (ce problème d'apprentissage sur des données bruitées est classiquement appelé apprentissage exact agnostique [BBL09]). Nous verrons dans la suite de ce chapitre comment il est possible d'apprendre, malgré ces préférences contradictoires, un CP-net acyclique. Nous nous focaliserons sur la description de l'algorithme d'apprentissage (s'inspirant de l'apprentissage par requêtes) proposé, ainsi que sur un certain nombre de tests empiriques effectués sur des données synthétiques bruitées, et une base de données réelle. Nous ferons également une analyse de sa complexité temporelle.

2.2 Requêtes étudiées

Nous faisons l'hypothèse, dans ces travaux, que l'oracle est en réalité une **base de données** S contenant les préférences d'un (ou de plusieurs) utilisateur(s). Ces préférences sont supposées refléter la vraie opinion de chaque utilisateur qui l'a émise [1]. De plus, nous supposons S capable de répondre en temps polynomial à

1. Une requête $\texttt{IM}(\mathcal{N}, S)$: l'oracle répond vrai si le CP-net $\mathcal{N}$ transmis par l'apprenant est capable de représenter toutes les préférences de l'oracle S. Dans le cas contraire, l'oracle fournit un contre-exemple sous la forme d'un swap [2] ne pouvant être représenté dans le CP-net $\mathcal{N}$;
2. Une requête $\texttt{SQ}(P, S)$: l'oracle est capable de retourner le nombre de paires de swaps vérifiant une propriété P donnée en paramètre.

1. Les CP-nets fonctionnant fondamentalement avec des préférences *ceteris paribus*, nous supposons ici que toutes les préférences présentes dans S sont des préférences *ceteris paribus*, *i.e.*, des swaps.

2. Cette requête est mise en pratique par le parcours des swaps de l'oracle S. Pour chaque swap $s = (\mathbf{o}, \mathbf{o}')_V \in S$, le test « $\mathbf{o} \succ_{\mathcal{N}} \mathbf{o}'$? » est alors réalisé.

Nous pouvons voir que l'oracle que nous manipulons diffère d'un oracle classique tel que celui défini dans la Section 1.2.3 page 15, car il ne possède ni de requêtes d'équivalences, ni de requêtes d'appartenance. Ceci est dû au fait que nous proposons ici un algorithme tentant de se rapprocher de problématiques pratiques faisant intervenir des bases de données sans connaissance réelle de la structure de préférences sous-jacente. Ajoutons de plus que le remplacement des requêtes d'équivalence en requêtes `IM` est dû au fait que les performances de l'algorithme proposé seront mises en évidence à l'aide de l'Équation (1.18) page 53, qui teste la présence de toutes les préférences de la base de données S au sein du CP-net appris $\mathcal{N}$ (sans faire le test inverse normalement nécessaire pour conclure à une équivalence).

La prise en compte des contradictions au sein de l'oracle est mise en pratique par le choix de chaque nouvelle variable parente, en revenant à la définition de base de telles variables. En reprenant l'Exemple 1.1 page 12, si l'activité à faire est la seule variable parente du choix vestimentaire, cela signifie que l'activité doit toujours être la même lorsque nous portons une chemise (aller à un mariage par exemple), et nous devons toujours effectuer une autre activité (faire du sport par exemple) lorsque nous portons un tee-shirt. Notre algorithme va ainsi parcourir la base de données grâce aux requêtes introduites précédemment afin de sélectionner la variable parente qui **minimise** le nombre de swaps ne respectant pas les conditions précédentes. La base de données contenant de potentielles préférences contradictoires, la probabilité d'obtenir des swaps « incorrects » est élevée. Les swaps générant une CP-règle ne pouvant être intégrée dans le CP-net seront alors placés au sein d'une **liste de CP-règles** L. Cette liste L doit être intégrée au sein des requêtes $\texttt{IM}(\mathcal{N}_L, S, L)$, car une couverture complète des swaps de S dans $\mathcal{N}_L$ est peu probable (ainsi, l'oracle S ne retournera que des contre-exemples n'étant pas présents dans L, permettant alors l'arrêt de notre algorithme) à cause des préférences contradictoires.

Nous allons séparer l'apprentissage du CP-net à partir des réponses de l'oracle S en deux phases : la première traite de l'apprentissage général, et la deuxième, de la recherche de la meilleure variable parente.

2.3 Algorithme d'apprentissage

2.3.1 Procédure générale

L'algorithme décrit ci-dessous est un algorithme tenant compte des contradictions entre préférences. Cela signifie qu'il est possible de rencontrer des préférences de type $\mathbf{o} \succ \mathbf{o}'$ et $\mathbf{o}' \succ \mathbf{o}$ au sein de la même base de données, qui vont alors impliquer la recherche d'une variable permettant de départager les deux préférences qui s'opposent. Il est donc nécessaire de pouvoir facilement distinguer les deux types de CP-règles représentant les deux préférences, notamment grâce à la Définition 1.14 introduite à la page 31. Il est ainsi possible de mettre facilement en évidence la coexistence d'une CP-règle $(\mathbf{u} : v \succ v')$ avec son inverse $(\mathbf{u} : v' \succ v)$, dont l'une

des deux sera considérée comme une **préférence contradictoire**[3], aussi appelée **préférence bruitée** (*i.e.*, la version minoritaire des deux types de CP-règles).

Algorithme 5 : `apprentissage_CPnet_requete`(S)

Entrées :
V : un ensemble de variables ;
S : oracle ;
k : nombre maximum autorisé de variables parentes par variable ;
Résultat :
$\mathcal{N}_L$: CP-net acyclique appris ;

1 **début**
2 L = $\emptyset$;
3 **Initialiser** $\mathcal{N}_L$;
4 **tant que** $\neg$`IM`$(\mathcal{N}_L, S, L)$ **faire**
5 Soit $(\mathbf{o}, \mathbf{o}')_V \in S$ un contre-exemple retourné par `IM` et $(\mathbf{o}[Pa(V)] : v \succ v')$ sa CP-règle associée;
6 **si** $(\mathbf{o}[Pa(V)] : v' \succ v) \in CPT(V)$ *et* $|CPT(V)| < k$ **alors**
7 $Q \leftarrow$ `rechercheParentR`$((\mathbf{o}, \mathbf{o}')_V, S)$;
8 **si** Q *existe, avec* $Dom(Q) = \{q, q'\}$ **alors**
9 $Pa(V) \leftarrow Pa(V) \cup \{(Q)\}$;
10 $CPT(V) \leftarrow \{(\mathbf{o}[Pa(V)]q : v \succ v'), (\mathbf{o}[Pa(V)]q' : v' \succ v)\}$, où $q = \mathbf{o}[\{Q\}]$;
11 Retirer de L toutes les CP-règles portant sur la variable V;
12 **sinon** $L \leftarrow L \cup \{(\mathbf{o}[Pa(V)] : v \succ v')\}$;
13 **sinon** $CPT(V) \leftarrow CPT(V) \cup \{(\mathbf{o}[Pa(V)] : v \succ v')\}$;
14 **retourner** $\mathcal{N}_L$;

L'Algorithme 5 va en premier lieu initialiser le CP-net appris $\mathcal{N}_L$ ainsi que la liste des CP-règles violées L. Il va ensuite effectuer une requête `IM` jusqu'à l'obtention d'un retour favorable sous contrainte des CP-règles de la liste L. Lorsqu'un contre-exemple $s = (\mathbf{o}, \mathbf{o}')_V$ est reçu, la CP-règle inverse $(\mathbf{o}[Pa(V)] : v' \succ v)$ de la CP-règle induite $(\mathbf{o}[Pa(V)] : v \succ v')$ est soit

- Absente du CP-net $\mathcal{N}_L$ (Ligne 13 de l'Algorithme 5), auquel cas il suffit d'ajouter $(\mathbf{o}[Pa(V)] : v \succ v')$ au CP-net ;
- Présente, il est alors nécessaire de raffiner les CP-règles de V en ajoutant une variable parente via l'algorithme `rechercheParentR` (voir la section suivante). Si ce parent existe, la CP-table de V est réinitialisée avec les deux nouvelles CP-règles et la liste L est mise à jour en supprimant les éventuelles CP-règles de V. Si la variable parente n'a pas été trouvée, alors $(\mathbf{o}[Pa(V)] : v \succ v')$ est ajoutée à L.

3. Présence d'un cycle de taille 2 dans l'ordre partiel des objets.

Il est important de noter la réinitialisation de la CP-table de la variable impliquée par $(\mathbf{o}[Pa(V)] : v \succ v')$ (contrairement aux algorithmes de [KZ10, GAG13]). Cette opération tient compte du fait qu'en pratique, un CP-net est rarement complet (*i.e.*, la CP-table d'une variable $V \in \mathbf{V}$ ne contient pas forcément $2^{|Pa(V)|}$ CP-règles). Par conséquent, il nous semble judicieux de repartir d'une CP-table vierge plutôt que de modifier chacune de ses CP-règles en tenant compte de la valeur de la nouvelle variable parente.

2.3.2 Recherche de la variable parente

La phase de recherche de la nouvelle variable parente est la partie la plus importante de l'apprentissage, car l'ajout d'une variable parente a pour conséquence de raffiner les CP-règles d'une variable (*i.e.*, ajouter de nouvelles variables parentes à cette variable), en les dédoublant dans le pire des cas. Une erreur dans le choix du parent peut alors être à l'origine d'importantes erreurs, qui ne peuvent être corrigées par la suppression d'une mauvaise variable parente à cause de la nature gloutonne [4] de la procédure proposée, et formellement décrite dans l'Algorithme 6.

Algorithme 6 : `rechercheParentR`$((\mathbf{o}, \mathbf{o}')_V, S)$

Entrées :
$(\mathbf{o}, \mathbf{o}')_V$: un swap ;
S : oracle ;
Résultat :
Une variable parente Q si elle existe, une erreur sinon ;

```
1 début
2     Q ← {Q ∈ V\({V}∪Pa(V)) | ¬cycle(N = (V, A∪{(Q,V)}, CPT(V)))
        et SQ(Équation (2.1)_{Q,V}, S) ≥ 1};
3     si Q ≠ ∅ alors
4         retourner argmin_{Q∈Q} (#SQ(Équation (2.2)_{Q,V}, S));
5     sinon retourner « parent non trouvé »;
```

Cette procédure de recherche d'une nouvelle variable parente a besoin du swap $s = (\mathbf{o}, \mathbf{o}')_V$ ayant permis le lancement de la présente procédure, ainsi que de l'oracle S. Soit $\mathbf{V}$ un ensemble de variables. Soient $Q \in \mathbf{V}$ la variable parente optimale de $V \in \mathbf{V}$, ainsi que $(\mathbf{o}, \mathbf{o}')_V$ et $(\mathbf{o}'', \mathbf{o}''')_V$ deux swaps. Q est une variable parente candidate pour V si

$$\begin{aligned} &\mathbf{o}[V] = \mathbf{o}'''[V] \neq \mathbf{o}'[V] = \mathbf{o}''[V], \\ \text{et} \quad &\mathbf{o}[Q] = \mathbf{o}'[Q] \neq \mathbf{o}''[Q] = \mathbf{o}'''[Q]. \end{aligned} \tag{2.1}$$

Cette équation met en évidence deux conditions nécessaires pour que Q soit la variable parente idéale :

4. Un algorithme glouton est un algorithme ne remettant pas en cause ce qui a été fait précédemment.

1. Pour tout couple d'objets *ceteris paribus* $\mathbf{o}''$, $\mathbf{o}'''$, l'oracle S est capable de donner sa préférence (soit $\mathbf{o}'' \succ \mathbf{o}'''$, soit $\mathbf{o}''' \succ \mathbf{o}''$) ;
2. Il existe au moins un autre swap $s' = (\mathbf{o}'', \mathbf{o}''')_V \in S$ tel que la préférence sur les valeurs de V est inversée sur la valeur q' de Q.

Ces contraintes sont modélisées dans la Ligne 2 de l'Algorithme 6 par l'application d'une requête `SQ` :

$$\texttt{SQ}(\text{Équation (2.1)}_{Q,V}, S).$$

Cette requête demande à S de retourner, pour une variable parente candidate Q de V, le nombre de paires de swaps vérifiant les contraintes de l'Équation (2.1). S'il existe au moins une paire les vérifiant, alors la variable Q est candidate pour être une variable parente de V.

Comme nous restreignons notre apprentissage aux CP-nets acycliques, il est également nécessaire de vérifier que l'ajout d'une nouvelle variable parente ne crée pas un cycle dans le graphe du CP-net. Pour cela, nous implémentons un algorithme basé sur le tri topologique, en supprimant tour à tour chaque sommet n'ayant aucun arc sortant. Si le graphe obtenu, à la fin de cette procédure, est vide, alors le CP-net est acyclique. Sinon, la variable parente choisie n'est pas prise en compte (voir l'Algorithme 7).

Algorithme 7 : `tri_topologique`$(\mathbf{V}, \mathcal{N})$

Entrées :
$\mathbf{V}$: ensemble de variables ;
$\mathcal{N}$: CP-net ;
Résultat :
`VRAI` si $\mathcal{N}$ est acyclique, `FAUX` sinon ;

```
1 début
2     V' ← V;
3     pour tout V ∈ V' faire
4         si Pa(V) = ∅ alors
5             V' ← V' \ {V};
6             pour tout V' ∈ V' tel que V ∈ Pa(V') faire
7                 Pa(V') ← Pa(V') \ {V};
8     si V' = ∅ alors  Retourner VRAI ;
9     sinon Retourner FAUX;
```

La deuxième partie de l'Algorithme 6 consiste en la recherche, parmi l'ensemble des variables parentes candidates $\mathbf{Q}$ trouvées à l'étape précédente (Ligne 2), de celle minimisant le nombre de swaps qui violent la CP-règle induite par le swap passé en paramètres de l'algorithme. En d'autres termes, nous recherchons la variable $Q \in \mathbf{V} \setminus \{V\}$ qui minimise le nombre de couples de swaps $(\mathbf{o}, \mathbf{o}')_V$ et $(\mathbf{o}'', \mathbf{o}''')_V$ ayant

la propriété suivante :

$$\begin{array}{rl} & (\mathbf{o}[\{Q\}] = \mathbf{o}''[\{Q\}] \text{ et } \mathbf{o}[\{V\}] \neq \mathbf{o}''[\{V\}]) \\ \text{ou} & (\mathbf{o}[\{Q\}] \neq \mathbf{o}''[\{Q\}] \text{ et } \mathbf{o}[\{V\}] = \mathbf{o}''[\{V\}]). \end{array} \tag{2.2}$$

En cas d'égalité, une sélection aléatoire uniforme est effectuée. Cette recherche est mise en œuvre dans l'Algorithme 6 (Ligne 4) par une requête `SQ` :

$$\texttt{SQ}(\text{Équation (2.2)}, S).$$

Cette requête demande à S de retourner le nombre de paires de swaps vérifiant les contraintes de l'Équation (2.2) en considérant la variable Q comme variable parente candidate à V. Ceci est fait pour l'ensemble des variables parentes candidates précédemment sélectionnées. Celle qui minimise ce nombre de couples de swaps est alors sélectionnée. Intuitivement, cela signifie que Q est, à ce moment précis, la plus adaptée pour être parente de V, car c'est celle qui minimise le nombre de swaps violant le contre-exemple fourni par S. Illustrons l'exécution de la procédure générale avec L'Exemple 2.1.

Exemple 2.1. *Reprenons l'Exemple 1.1 de la page 12 :*

Variable	Valeurs
A : Activité	sp : sport ma : mariage
H : haut de vêtement	tsh : t-shirt ch : chemise
B : bas de vêtement	sh : short pa : pantalon

Supposons que nous voulons déduire que la variable H ne peut pas être une variable parente de B (i.e., le fait de porter un tee-shirt ne conditionne pas le choix de mettre un pantalon ou un short). Pour cela, l'algorithme doit recevoir l'un de ces swaps :

- $\mathbf{o} = \{sp, tsh, sh\} \succ \mathbf{o}' = \{sp, tsh, pa\}$, *et* $\mathbf{o}'' = \{ma, ch, sh\} \succ \mathbf{o}''' = \{ma, ch, pa\}$. *Nous pouvons observer dans ce cas que les différents hauts portés n'influencent pas la préférence sur le bas porté ;*
- $\mathbf{o} = \{sp, tsh, sh\} \succ \mathbf{o}' = \{sp, tsh, pa\}$, *et* $\mathbf{o}'' = \{ma, tsh, pa\} \succ \mathbf{o}''' = \{ma, tsh, sh\}$. *Nous pouvons observer dans ce cas que la préférence sur le choix du bas à porter diffère alors que nous portons toujours un tee-shirt.*

La détection des variables parentes candidates (l'ensemble $\mathbf{Q}$) est conditionnée par l'existence du swap $(\mathbf{o}'', \mathbf{o}''')_V$ de l'Équation (2.1). De plus, la détection de contradictions (et leur minimisation) est réalisée par la recherche d'une variable $Q \in \mathbf{Q}$ qui minimise le nombre de swaps violant le contre-exemple (Équation (2.2)). Il est important de noter que cette partie de l'algorithme peut être parallélisée, en assignant une unité de calcul par variable parente candidate. L'Exemple 2.2 expose une exécution de la procédure générale (Algorithme 5 page 61).

Exemple 2.2. *Essayons maintenant, toujours à partir de l'Exemple 1.1 de la page 12, d'apprendre le CP-net binaire complet de la Figure 1.5 (sans la variable B). Commençons par demander à l'oracle si le CP-net de départ $\mathcal{N}_L$, vide, correspond à sa structure induite. Il nous répond naturellement non, et nous retourne le swap $((sp, tsh), (ma, tsh))_A$. L'algorithme en déduira la CP-règle $r = (\emptyset : sp \succ ma)$, qui est alors ajoutée à $\mathcal{N}_L$. Une requête* `IM` *est alors de nouveau exécutée, et nous obtenons le contre-exemple suivant $((sp, tsh), (sp, ch))_H$. La CP-règle $r = (\emptyset : tsh \succ ch)$ est alors déduite et ajoutée à $\mathcal{N}_L$.*
Le processus est alors répété une fois de plus, et la requête `IM` *retourne cette fois le contre-exemple $((ma, ch), (ma, tsh)_H)$ qui induit la CP-règle $r = (\emptyset : ch \succ tsh)$. Comme la règle inverse $(\emptyset : tsh \succ ch)$ existe déjà, l'Algorithme 6 va retourner la variable A comme seule variable parente de H. Les deux nouvelles règles suivantes sont alors ajoutées à $\mathcal{N}_L$: $(sp : tsh \succ ch)$ et $(ma : ch \succ tsh)$, et la prochaine requête* `IM` *nous renverra finalement vrai.*

Il est également important de noter que l'ordre des contre-exemples reçus peut perturber la sélection des bonnes variables parentes dans l'Algorithme 6 pendant la phase d'apprentissage (voir l'exemple ci-après).

Exemple 2.3. *Considérons l'ensemble de variables $\mathbf{V} = \{A, H, B\}$:*

Variable	Valeurs
A : Activité	sp : sport ma : mariage
H : haut de vêtement	tsh : t-shirt ch : chemise
B : bas de vêtement	sh : short pa : pantalon

Supposons que l'oracle S va énoncer ses contre-exemples dans l'ordre suivant :

- *Nous recevons en premier lieu $(sp, tsh, sh) \succ (sp, tsh, pa)$, ce qui nous permet de déduire la règle $(\emptyset : sh \succ pa)$;*
- *Puis nous recevons $(ma, ch, pa) \succ (ma, ch, sh)$. L'Algorithme 6 a ici le choix de la variable parente entre A et H. Supposons qu'il choisisse la variable H comme étant la variable parente de B ;*
- *Nous recevons finalement le contre-exemple $(ma, tsh, pa) \succ (ma, tsh, sh)$ qui signifie que la véritable variable parente de B n'est pas H mais A.*

Cet exemple montre que lorsque l'on a le choix entre plusieurs variables parentes candidates (l'algorithme avait ici le choix entre la A et H), il est possible que le choix se porte sur la mauvaise variable parente (*i.e.*, une variable n'étant pas parente dans le CP-net de l'oracle), ce qui ne serait pas arrivé avec l'ordre $(sp, tsh, sh) \succ (sp, tsh, pa)$, $(ma, tsh, pa) \succ (ma, tsh, sh)$, $(ma, ch, pa) \succ (ma, ch, sh)$. Pour corriger ce problème :

1. L'Algorithme 6 choisirait la variable parente qui **minimise** le nombre de contradictions (Équation (2.1)), en supposant que les contradictions restantes correspondent uniquement à des préférences bruitées ;
2. Le fait de considérer l'oracle S comme un ensemble de swaps rendrait possible la génération de l bases de données différentes en sélectionnant aléatoirement l' swaps différents de S, afin d'apprendre un CP-net par base de données. Il suffirait ensuite de sélectionner le CP-net qui minimise le nombre de contradictions sur la base de données globale S parmi l'ensemble des CP-nets appris [5].

Cette deuxième idée a été testée et a conduit à des résultats intéressants, où l'apprentissage d'un CP-net sur un dixième des swaps présents dans l'oracle S permet d'obtenir un CP-net appris respectant 63% des swaps présents dans S. Cependant, cette précision étant trop faible, nous avons décidé de laisser de côté cette idée pour de possibles futures améliorations. Cela reste néanmoins une méthode simple permettant d'apprendre très rapidement un CP-net relativement fiable comparativement aux préférences de l'oracle S.

2.3.3 Complexité

Dans cette partie, nous analysons la complexité globale de la procédure générale décrite par l'Algorithme 6. Nous supposons, pour des raisons de simplicité, que le modèle de préférences cible induit par l'oracle S est un CP-net noté $\mathcal{N}_T$.

Proposition 1. *Soit S un oracle, avec $m = |S|$ (i.e., le nombre de swaps contenus dans S), et soit n le nombre de variables décrivant chaque objet de S. L'Algorithme 6 possède une complexité temporelle en $O(n^3 + nm)$.*

Démonstration.
La Ligne 2 de l'Algorithme 6 a besoin de détecter les cycles existants dans le graphe du CP-net appris. Ceci est réalisé en $O(n^2)$. La recherche d'un swap qui, allié au swap donné en paramètre, respecte l'Équation (2.1), est effectuée en parcourant l'oracle S, donc en $O(m)$. Ces deux étapes sont alors répétées pour chaque variable candidate. La Ligne 2 a alors une complexité en $O((n-1)(m+n^2)) \subseteq O(n^3 + mn)$. Enfin, la Ligne 4 a besoin de compter le nombre de swaps dans S respectant, avec le swap passé en paramètre, l'Équation (2.2). Cette ligne admet donc une complexité en $O(mn)$. L'Algorithme 6 possède alors une complexité en $O(n^3 + mn)$. □

La prochaine proposition énonce un résultat de complexité sur l'apprentissage de CP-nets arborescents. Nous rappelons qu'un CP-net arborescent est un CP-net acyclique où chaque variable possède au plus une seule variable parente.

Proposition 2. *Soit $\mathcal{H}_{abr}$ la classe des CP-nets arborescents (i.e., $\forall V \in \mathbf{V}, |Pa(V)| \leq 1$ et $\mathcal{N}$ est acyclique, $\forall \mathcal{N} \in \mathcal{H}_{abr}$). Si le CP-net appris $\mathcal{N}_L \in \mathcal{H}_{abr}$, alors l'Algorithme 5 requiert l'utilisation de $2n$ requêtes* `IM` *au maximum et possède une complexité temporelle en $O(m(2n^4 + 2n^2 m))$, avec $m = |S|$.*

5. Cette solution possède, de plus, l'avantage de pouvoir être utilisée en parallèle en lançant chaque apprentissage sur une machine différente.

Démonstration.
Il est nécessaire d'effectuer n requêtes `IM` pour apprendre un CP-net séparable. Ces requêtes demandent d'effectuer l'opération « $\mathbf{o} \succ_{\mathcal{N}_L} \mathbf{o}'$? » pour chaque swap $s = (\mathbf{o}, \mathbf{o}')_V \in S$. La déduction d'une telle préférence étant immédiate dans le cadre des CP-nets grâce à la propriété de *ceteris paribus*, l'ensemble de ces requêtes `IM` admet une complexité en $O(mn)$. De plus, comme $\mathcal{N}_L$ possède une structure arborescente, il aura au plus $n-1$ variables parentes, donc l'Algorithme 6 est en $O(n^2+mn)$. En additionnant ces différentes complexités, la construction de $\mathcal{N}_L$ peut être effectuée en $O(n^2+m(n^2+n))$, avec n^2 requêtes `IM` (n pour apprendre la partie séparable de $\mathcal{N}_T$, et n pour apprendre les différents arcs des variables parentes). □

Proposition 3. *L'Algorithme 5 possède une complexité temporelle en* $O(2^k n^2 m(n^2+m))$ *pour générer le CP-net* $\mathcal{N}_L$*, avec* $k = \max_{V \in \mathbf{V}}\{|Pa(V)|\}$*, et* n *le nombre de variables,* $m = |S|$.

Démonstration.
Nous savons grâce à la Proposition 1 que l'Algorithme 6 possède une complexité temporelle en $O(n^3+nm)$. Nous considérons maintenant la boucle `tant que` de la Ligne 4. Supposons que le CP-net cible $\mathcal{N}_T$ est complet. Cela implique que l'ensemble des CP-tables sont complètes, et contiennent donc au maximum 2^k CP-règles. De plus, à chaque ajout d'une variable parente, il est nécessaire de supprimer l'ensemble des CP-règles de la CP-table concernée. Il est alors nécessaire, dans le pire des cas, d'effectuer $\sum_{i=1}^{k} 2^i = 2(2^k-1)$ requêtes `IM` pour apprendre une CP-table, donc 2^{k+1} requêtes `IM` au total. L'algorithme 5 possède donc une complexité en $O(2^{k+1}n^2m(n+m))$. □

Pour n le nombre de variables binaires, il ne peut y avoir plus de 2^n objets différents au sein de cette base de données. Il est donc possible de borner le nombre maximum de règles possibles par $h \leq 2^n$. De plus, chacune de ces règles peut être représentée par au moins un swap, ce qui rend le nombre de swaps exponentiel, et permet de borner $m \leq 2^n$ et $e \leq 2^n$. Cependant, dans le cas d'une application dans un environnement réel, nous supposons que $o < m = |S| \ll 2^n$ (avec $|S|$ la taille de la base de données, *i.e.*, son nombre de swaps, et o le nombre d'objets uniques présents dans S), et que dans la plupart des situations, le pire cas ne devrait pas être atteint, malgré une complexité exponentielle de l'algorithme [6].

2.4 Résultats expérimentaux

L'objectif de cette section est d'examiner expérimentalement l'efficacité de l'apprentissage de L'Algorithme 5. Nous allons pour cela utiliser deux types de protocoles : le premier consistera à comparer l'Algorithme 5 avec l'algorithme exposé

6. Nous pouvons noter que dans un cadre où les variables ne seraient pas binaires, alors le terme changeant dans la complexité donnée dans la Proposition 3 serait la base 2 de l'exponentiel. Ainsi, pour des variables à l valeurs, la complexité deviendrait $O(l^{k+1}n^2m(n+m))$.

dans [GAG13], et l'algorithme présenté dans [LXW+14], tous deux décrits dans les Sections 1.4.3.2 page 44 et 1.4.3.1 page 42, respectivement. Le second protocole examinera l'efficacité de notre algorithme sur des données réelles et des données simulées bruitées.

Les expériences ont été réalisées sur un ordinateur Windows ayant un processeur Intel core i7-4600U possédant 16Go de mémoire RAM (les algorithmes sont implémentés en Python). Le code source ainsi que les différents tests sont disponibles sur GitHub à l'adresse `https://github.com/FabienLab/CPnets-queries`.

Dans la suite des expérimentations, la précision de l'Algorithme 5 est calculée suivant l'Équation (1.18) page 53 :

$$prec(S, h_S) = \frac{1}{|S|} \sum_{s \in S} \mathbb{1}(y_s = h_S(s)) \in [0, 1],$$

avec S l'ensemble d'entraînement, et h_S l'hypothèse déterminée par l'Algorithme 5 à partir de S.

De plus, chaque point affiché sur les graphiques représente soit une mesure de performance, soit un temps d'exécution de l'algorithme d'apprentissage. La valeur de chacun de ces points correspond à une moyenne de l exécutions de l'algorithme d'apprentissage, *i.e.*, soit v_{ij} la j^{e} exécution du i^{e} point du graphique courant, alors $\tilde{\mathbf{v}}_i = \frac{1}{l}\sum_{j=1}^{l} v_{ij}$ correspond à la moyenne des exécutions du i^{e} point. Nous ajoutons de plus une barre d'erreur sur chaque point (illustrée par un segment vertical), représentant l'écart-type σ_i du i^{e} point, calculé par la formule suivante :

$$\sigma_i = \sqrt{\frac{1}{l-1} \sum_{j=1}^{l} (v_{ij} - \tilde{\mathbf{v}}_i)^2}, \tag{2.3}$$

où l représente le nombre d'exécutions de l'algorithme. La barre d'erreur présente sur chaque point $\tilde{\mathbf{v}}_i$ correspond alors à l'intervalle de valeurs $[\tilde{\mathbf{v}}_i - \sigma_i, \tilde{\mathbf{v}}_i + \sigma_i]$.

Les résultats de chaque expérience ont été générés suivant le protocole d'apprentissage sur des données spécifiques, puis testés sur des données différentes. Ainsi, chaque base de données S est séparée en deux bases : S_A contenant 75% des swaps de S et servant à l'apprentissage du CP-net, et S_T contenant les 25% de données restantes et servant au test d'efficacité du CP-net appris. Le Tableau 2.1 ci-dessous résume l'ensemble des expériences effectuées.

2.4.1 Données réelles et synthétiques

Dans cette section, nous allons tester les performances de l'Algorithme 5 sur différents jeux de données réelles et synthétiques :

Section	Expérience	Description	Page
2.4.1	1	Efficacité de l'apprentissage de l'Algorithme 5 sur des données synthétiques et réelles.	70
	2	Efficacité de l'apprentissage de l'Algorithme 5 sur des tailles de bases de données variables.	72
	3	Temps d'apprentissage de l'Algorithme 5 sur des données synthétiques.	73
2.4.2	4	Efficacité de l'apprentissage de l'Algorithme 5 sur des données synthétiques avec probabilité de bruitage.	74
	5	Efficacité de l'apprentissage de l'Algorithme 5 sur des données synthétiques avec probabilité de bruitage (sans séparation entraînement/test).	75
	6	Convergence empirique de l'Algorithme 5 sur des données synthétiques.	75
2.4.3	7	Comparaison de l'efficacité d'apprentissage entre l'Algorithme 5 et l'algorithme de [GAG13].	76
	8	Temps d'apprentissage de l'Algorithme 5 en fonction du nombre de variables parentes du CP-net appris.	78
	9	Efficacité de l'apprentissage de l'Algorithme 5 en fonction du nombre de variables parentes du CP-net cible.	79
	10	Comparaison de l'efficacité d'apprentissage entre l'Algorithme 5 et l'algorithme de [LXW$^+$14].	80

Tableau 2.1 – Résumé des différentes expériences.

1. La base de données réelle est une base de données issue du site internet TripAdvisor [7] [WLZ10, WLZ11] dont l'échelle a été modifiée afin de passer de valeurs non binaires à des valeurs binaires des différents attributs (considérés ici comme les variables d'un CP-net) ;
2. Une base de données synthétique générée aléatoirement (voir l'Algorithme 8) afin de tester la robustesse de notre algorithme au niveau du temps de calcul et de sa gestion des contradictions.

La base de données de TripAdvisor contient environ 240 000 critiques d'hôtels, dont seulement 60 000 critiques complètes ont été retenues (*i.e.*, ayant tous les champs renseignés). Les critiques ont été effectuées sur environ 6 000 hôtels anonymisés (aucune information sur leur emplacement), par 1 800 utilisateurs différents. Une critique est représentée par 7 notes comprises en 0 et 5 [8] (l'aspect global de l'hôtel, l'aspect des chambres, la localisation, la propreté, l'accueil, le service global, et le service spécialisé business) auxquelles s'ajoute une note globale sur l'ensemble de l'hôtel. Cette dernière est utilisée comme une relation de préférence afin de comparer entre elles deux critiques.

7. `http://times.cs.uiuc.edu/~wang296/Data/`.
8. 0 correspond à la note la plus basse, et 5 à la note la plus haute.

Exemple 2.4. *Prenons comme exemple un hôtel ayant obtenu les notes suivantes pour chacun des sept attributs décrits précédemment :* 3 *pour l'aspect global,* 4 *pour l'aspect des chambres,* 5 *pour la localisation,* 2 *pour la propreté,* 4 *pour l'accueil,* 3 *pour le service global, et* 3 *pour le service spécialisé business. On considérera donc cet hôtel par le vecteur suivant :* $(3, 4, 5, 2, 4, 3, 3)$.

Afin de permettre l'apprentissage d'un CP-net binaire, les différentes notes sont modifiées de la façon suivante : 1 si la note est strictement supérieure à 2, et 0 sinon (nous avons effectué ce choix en considérant qu'une note de 2 est plutôt considérée comme négative, et qu'une note de 3 est plutôt considérée comme positive, la note moyenne se plaçant à $2,5/5$).

Exemple 2.5. *En reprenant l'exemple précédent, l'hôtel caractérisé par les notes* $(3, 4, 5, 2, 4, 3, 3)$ *devient, après application de notre binarisation :* $(1, 1, 1, 0, 1, 1, 1)$.

Cette binarisation particulière des critiques conduit à des doublons (60 000 critiques pour $2^7 = 128$ critiques uniques). Nous remarquons qu'aucun hôtel n'a obtenu une note maximale partout, de même qu'aucun hôtel n'a obtenu que des notes minimales, les deux vecteurs $(1, 1, 1, 1, 1, 1, 1)$ et $(0, 0, 0, 0, 0, 0, 0)$ ne sont pas présents. Cela nous permet d'obtenir, à la fin de cette procédure, 126 hôtels différents qui n'induisent ici pas forcément un CP-net cible, mais plutôt un modèle de préférence inconnu noté $\mathcal{P}$ contenant de potentielles contradictions (dues soit aux propriétés du modèle $\mathcal{P}$ incompatibles avec le modèle CP-net, soit aux différences d'opinions entre les utilisateurs).

La construction des préférences *ceteris paribus* fut la deuxième étape dans le traitement de ces données réelles. En effet, la binarisation de la base de données TripAdvisor entraîne la création d'un nombre important de swaps. Notons cependant que cet ensemble de swaps, parmi l'ensemble total de préférences de la base d'origine, n'est pas nul : ainsi, nous avons pu observer environ 3 000 swaps sur l'ensemble des 60 000 critiques complètes (correspondant alors à $60\,000^2$ comparaisons par paires différentes).

L'autre base de données utilisée est une base de données synthétique générée aléatoirement de la façon suivante : pour un nombre de variables n donné, nous générons deux objets $\mathbf{o}$ et $\mathbf{o}'$ de façon à ce qu'ils forment un swap dont la préférence est arbitrairement fixée. L'étape est réitérée jusqu'à obtenir le nombre souhaité de swaps. Cette construction nous permet de simuler des contradictions venant d'un modèle de préférences différent d'un CP-net, et des différences d'opinions entre plusieurs utilisateurs. Afin de tester l'Algorithme 5, trois bases de données ont été créées : une base de 50 swaps à 7 variables, une base de 500 swaps à 10 variables, et une base de 10 000 swaps à 15 variables [9].

Expérience n° 1 : efficacité de l'apprentissage de l'Algorithme 5 sur des données synthétiques et réelles

9. Le nombre de variables n est en fait déduit directement en fonction du nombre de swaps en appliquant la formule $n = \lfloor \log_2 m \rfloor + 1$, avec m le nombre de swaps de la base de données.

Nous testons dans un premier temps l'importance d'avoir des préférences cohérentes durant la phase d'apprentissage. Le Tableau 2.2 et la Figure 2.2 exposent les résultats obtenus au niveau de la précision de l'apprentissage du CP-net.

Bases de données (nombre de swaps)	Précision	Erreur
	$k = 1$	
TripAdvisor (126)	73	27
hôtels synthétiques (50)	72	28
hôtels synthétiques (500)	57	43
hôtels synthétiques (10 000)	52	48
	$k = 5$	
TripAdvisor (126)	82	18
hôtels synthétiques (50)	82	18
hôtels synthétiques (500)	59	41
hôtels synthétiques (10 000)	51	49
	$k = \frac{n-1}{2}$	
TripAdvisor (126)	83	17
hôtels synthétiques (50)	79	21
hôtels synthétiques (500)	69	31
hôtels synthétiques (10 000)	66	34
	CP-net cyclique	
TripAdvisor (126)	100	0
hôtels synthétiques (50)	100	0
hôtels synthétiques (500)	100	0
hôtels synthétiques (10 000)	*calcul trop long (>24h)*	

TABLEAU 2.2 – Précision de l'Algorithme 5. La variable k correspond au nombre maximum de variables parentes par variable. La taille de chaque base est indiquée entre parenthèses. Les résultats sont moyennés sur 30 exécutions.

En mettant de côté les résultats relativement similaires entre TripAdvisor et la base à 50 swaps, nous pouvons observer au sein de la Figure 2.2 une importante différence entre d'un côté la base TripAdvisor, et de l'autre les deux bases synthétiques de 500 et de 10 000 swaps. Nous pouvons bien sûr voir une augmentation de la précision lorsque le nombre maximum de variables parentes augmente. Remarquons une baisse de la prédiction sur la Figure 2.2 lors du passage de $k = 1$ à $k = 2$. Cela peut s'expliquer par le fait que le modèle se complexifie trop lorsque $k = 2$, ce qui a pour effet de produire un sur-apprentissage de l'Algorithme 5. L'ajout de nouvelles variables parentes permet alors de corriger l'erreur commise.Nous observons de plus, grâce à la Figure 2.2 et à l'écart-type calculé sur chaque point des expérimentations, que les résultats de précision ne sont pas très stables. Ce phénomène est notamment dû au nombre peu élevé de swaps et à leur ordre d'apparition en tant que contre-exemples ne permettant pas de corriger les contradictions des bases de données (voir Exemple 2.3).

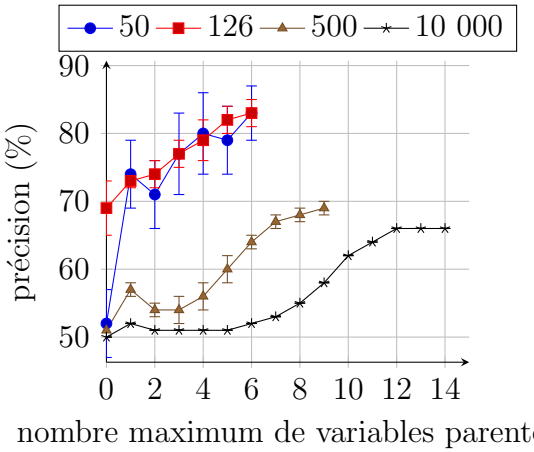


FIGURE 2.2 – Précision d'apprentissage en fonction du nombre maximum de parents par variable. Les bases de données sont générées aléatoirement sauf pour la base TripAdvisor. Chaque courbe correspond à une taille de base de données différente (la courbe rouge représente la base TripAdvisor). Les résultats sont moyennés sur 30 exécutions.

Expérience n° 2 : efficacité de l'apprentissage de l'Algorithme 5 sur des tailles de bases de données variables

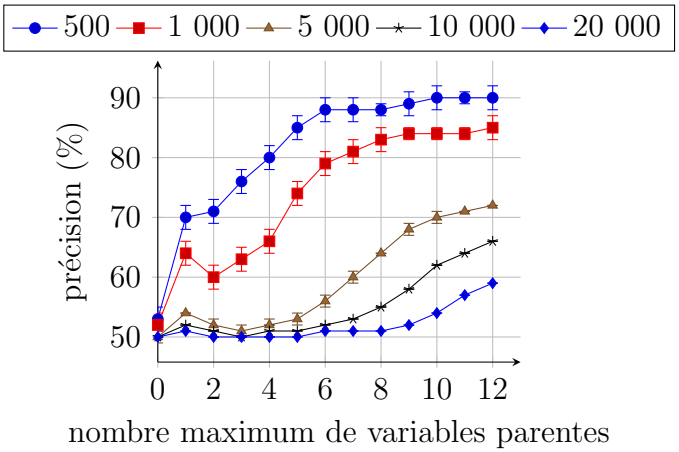


FIGURE 2.3 – Précision d'apprentissage sur des bases de données de 15 variables, où le nombre de swaps varie dans les différentes bases. Chaque courbe correspond à une taille de base différente. Les résultats sont moyennés sur 10 exécutions.

La Figure 2.3 montre l'influence du nombre de swaps m sur la précision de l'apprentissage pour un nombre fixe de 15 variables sur chaque base de données. Lorsque $m \ll 2^{15}$, nous obtenons une précision relativement bonne dépassant les 80% pour les bases de données de 500 et de 1 000 swaps. Cependant, lorsque m se rapproche de 2^{15}, cette précision descend alors jusqu'à 60%. Le phénomène de décroissance de la précision entre $k = 1$ et $k = 2$ apparait de nouveau sur cette expérience. Ce phénomène peut encore une fois s'expliquer par le choix de la première variable pa-

rente, crucial dans l'apprentissage de CP-nets, car cet ajout impliquera la division de CP-règles faisant intervenir un nombre très important de swaps (du fait de la généralité des CP-règles de départ). L'ajout d'une deuxième variable parente fera alors irrémédiablement intervenir un nombre plus restreint de swaps potentiellement bruités, amenant à un mauvais choix de variable parente, qui sera alors rectifié au fur et à mesure des nouveaux ajouts grâce à des CP-règles de plus en plus fines (cela explique donc la hausse de la précision pour des valeurs de $k > 2$).

Une remarque importante peut être faite sur l'ordre de réception des swaps, qui peut influer sur les choix de la nouvelle variable parente de l'Algorithme 6, comme illustré dans l'Exemple 2.3 page 65. L'écart-type observé est effectivement relativement faible ($\pm 2\%$). Cette expérience montre alors qu'en pratique, lorsqu'un nombre suffisant de swaps est présent dans la base de données, l'ordre de leur apparition en tant que contre-exemples n'a que peu d'impacts sur les résultats de l'apprentissage.

Expérience n° 3 : temps d'apprentissage de l'Algorithme 5 sur des données synthétiques

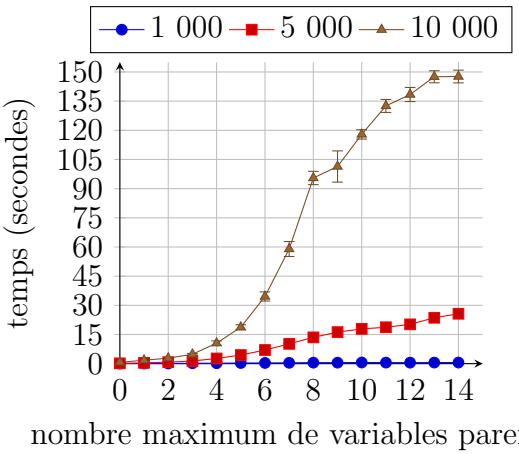


FIGURE 2.4 – Temps d'apprentissage pour 15 variables, où le nombre de swaps varie pour chaque base de données. Chaque courbe correspond à une taille de base différente. Les résultats sont moyennés sur 10 exécutions.

Nous terminons cette section avec un test de performances sur le temps d'apprentissage pour des bases de données contenant plus de swaps. Nous observons dans la Figure 2.4 que la procédure d'apprentissage est quasiment immédiate pour une base de données ayant un nombre de swaps $m \leq 1\,000$. Cependant, ce temps augmente jusqu'à 20 secondes pour $m = 5\,000$, et plus de 150 secondes pour $m = 10\,000$ (avec $k = 14$). Cela s'explique par le fait qu'à chaque nouvel ajout de variable parente, il est nécessaire de reparcourir la base. Nous observons par contre une croissance linéaire du temps d'apprentissage en fonction du nombre de variables parentes, alors que chaque ajout implique une multiplication du nombre de CP-règles à traiter. Ceci est dû au fait que le nombre de variables utilisées est relativement faible, ici 15

variables. De ce fait, la multiplication des CP-règles est ici négligeable d'un point de vue temps de calcul.

2.4.2 Données synthétiques avec probabilité de bruitage

Cette section est consacrée à la robustesse de notre Algorithme 5 au bruit en fixant une probabilité p de bruitage des swaps d'une base de données S.

Expérience n° 4 : efficacité de l'apprentissage de l'Algorithme 5 sur des données synthétiques avec probabilité de bruitage

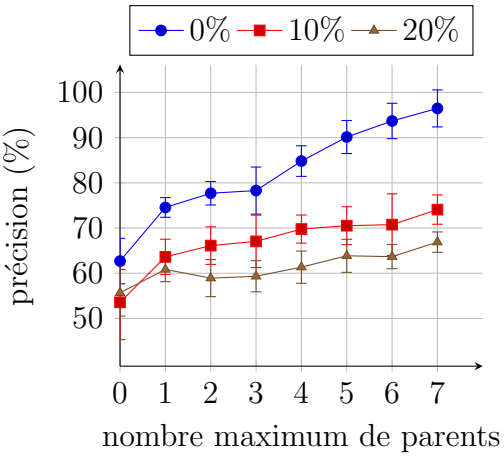


FIGURE 2.5 – Précision d'apprentissage en fonction du nombre maximum de parents sur le CP-net appris. La base de donnée est générée aléatoirement (5 000 comparaisons, 8 variables) avec une probabilité variable du bruit (une par courbe).

Le protocole de génération de la base de données est similaire à celui utilisé dans la section précédente : un CP-net est généré aléatoire, puis des swaps sont également générés de façon aléatoire. Nous introduisons ici une probabilité $p \in [0, 1]$ d'inverser la préférence induite par le swap. Nous lançons ensuite différentes exécutions de l'algorithme d'apprentissage sur cette base de données bruitée, en faisant varier la probabilité p de bruit. Les résultats sont résumés par la Figure 2.5. Nous pouvons observer une très bonne résistance de l'Algorithme 5 au bruit. Notons tout de même un écart relativement important entre la précision sur une base non bruitée et une base bruitée à 10%. Cela s'explique par le fait que le bruit est appliqué aux swaps, et non sur les CP-règles, impliquant donc un volume de CP-règles bruitées allant potentiellement bien au-delà de 10% CP-règles bruitées. L'algorithme reste cependant robuste lorsque le bruit augmente de 10% à 20%. Cette expérience est également à mettre en perspective avec la convergence empirique de l'algorithme (voir la Figure 2.7 page 76), qui montre qu'il est plus difficile de converger lorsque la base de données est bruitée, ce qui peut également expliquer l'écart de précision entre 0% et 10% de bruit sur les Figures 2.5 et 2.6.

Expérience n° 5 : efficacité de l'apprentissage de l'Algorithme 5 sur des données synthétiques avec probabilité de bruitage (sans séparation entraînement/test)

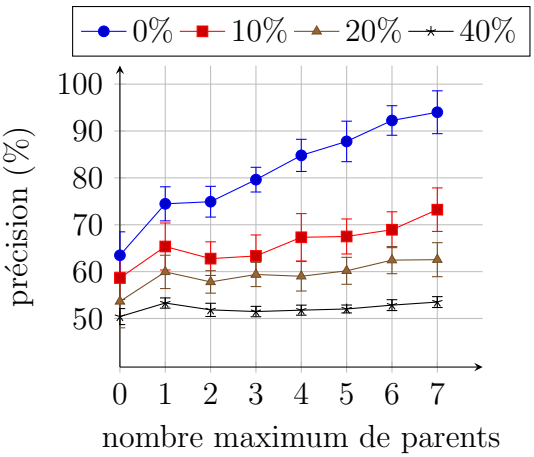


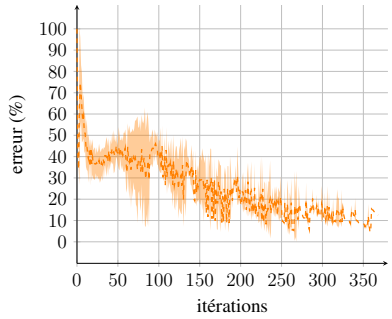
FIGURE 2.6 – Précision d'apprentissage en fonction du nombre maximum de parents sur le CP-net appris. La base de donnée est générée aléatoirement (5 000 comparaisons, 8 variables) avec une probabilité variable du bruit. Chaque courbe correspond à une probabilité de bruit différente (les données d'entraînement servent également de données de test). Les résultats sont lissés sur 30 exécutions.

La même expérience est de nouveau exécutée, en écartant cette fois la séparation de la base de données en ensemble d'apprentissage et en ensemble de test. Le CP-net appris est donc ici testé sur le même ensemble de données ayant servi à son apprentissage. Les résultats sont exposés dans la Figure 2.6 où nous pouvons observer que l'apprentissage, lorsque la base de données n'est pas bruitée, est relativement bon. Les erreurs d'apprentissage sont dues aux approximations commises par l'algorithme durant son exécution. Ces résultats sont finalement similaires à la Figure 2.5.

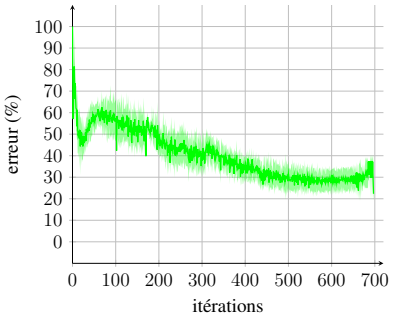
Nous pouvons également mettre en perspective ces résultats avec ceux donnés dans la Figure 2.3 au sein de laquelle le bruit était complètement aléatoire. Nous pouvons alors relever qu'à nombre de swap égal, le bruit de la Figure 2.3 page 72 se rapproche de 10%, modulo le nombre de variables du CP-net (12 contre 8).

Expérience n° 6 : convergence empirique de l'Algorithme 5 sur des données synthétiques

Nous souhaitons enfin observer la convergence empirique de l'Algorithme 5 en fonction du bruit. Pour cela, nous considérons le même protocole que précédemment, en vérifiant avant chaque nouvelle requête `IM` la précision du CP-net courant par rapport à la base de données S. La Figure 2.7 illustre cette expérience où nous remarquons de grandes variations dues au fait qu'à chaque ajout d'une variable parente, l'ensemble des CP-règles de la variable concernée est supprimé, puis de nouveau



(a) Convergence sur une base de données non bruitée.



(b) Convergence sur une base de données bruitée à 10%.

FIGURE 2.7 – Convergence empirique de notre algorithme. Chaque itération correspond à un appel à une requête `IM` sur une base de données dont le pourcentage de bruit varie (5 000 comparaisons, et 8 variables).

ajouté au fur et à mesure des contre-exemples retournés. L'algorithme converge néanmoins vers des valeurs de précision en accord avec l'expérience de la Figure 2.6. Notons que le nombre d'itérations est presque doublé entre une base de données non bruitée, et une base bruitée à 10% : le bruit entraîne l'observation de beaucoup plus de contre-exemples, dont certains ne peuvent être représentés. La convergence est donc plus nette et rapide sur une base de données cohérente.

2.4.3 Comparaisons avec l'état de l'art

Nous supposons dans cette section que le modèle de préférence induit par la base de données S est un CP-net, que nous notons $\mathcal{N}_T$. L'objectif est donc d'apprendre $\mathcal{N}_T$ à partir de la base de données S. Pour cela, nous allons générer un CP-net aléatoire grâce à l'Algorithme 8 page 77. Cet algorithme génère un CP-net acyclique complet à variables binaires. Il commence par construire l'ensemble des arcs du CP-net, puis l'ensemble des $2^{|Pa(V)|}$ CP-règles de chaque variable $V \in \mathbf{V}$ en créant aléatoirement une préférence $v \succ v'$ ou $v' \succ v$. La base de données S est alors remplie en sélectionnant aléatoirement des CP-règles du CP-net cible $\mathcal{N}_T$ et en générant des swaps induisant lesdites CP-règles.

Expérience n° 7 : comparaison de l'efficacité d'apprentissage entre l'Algorithme 5 et l'algorithme de [GAG13]

Il est important de garder à l'esprit que l'Algorithme 5 est un algorithme **hors ligne**, offrant donc des performances théoriquement meilleures qu'un algorithme **en ligne** tel que celui de Guerin *et al.* dans [GAG13]. Comme introduit dans la Section 1.4.3 page 41, les algorithmes en ligne n'ont qu'un accès restreint à la base de données, car celle-ci ne peut être stockée en raison de sa taille, cependant l'algorithme de [GAG13] est actuellement le seul algorithme basé sur un apprentissage par requêtes ayant réellement été implémenté. L'Algorithme 5, au contraire, parcourt plusieurs fois l'ensemble de la base de données (notamment pour la recherche

Algorithme 8 : `CPnet_aleatoire`(n, λ)

Entrées :
n : nombre de variables ;
λ : densité du CP-net ;
Résultat :
$\mathcal{N}$: CP-net aléatoire ;

début
  Initialiser $\mathcal{N}$ dans un ensemble $\mathbf{V}$ à n variables;
  `// |arcs| correspond au nombre d'arcs de` $\mathcal{N}$
  tant que $\frac{|arcs|}{n} < \lambda$ **faire**
    $\mathcal{N} \leftarrow \mathcal{N} \cup \{(V, V')\}$, où V et V' ont été sélectionnées aléatoirement et $\mathcal{N}$ reste acyclique;
  pour $V \in \mathbf{V}$ **faire**
    pour $\mathbf{u} \in 2^{Dom(Pa(V))}$ **faire**
      Ajouter aléatoirement $CPT(V) \leftarrow CPT(V) \cup \{(\mathbf{u} : v \succ v')\}$ ou $CPT(V) \leftarrow CPT(V) \cup \{(\mathbf{u} : v' \succ v)\}$;

de la meilleure variable parente candidate).

Afin d'avoir un protocole le plus proche possible de celui énoncé dans l'article de Guerin *et al.* dans [GAG13], nous fixons le nombre maximum de variables parentes par variable à $k = 5$ et nous ajoutons $\lambda = \frac{|\text{arcs}|}{n}$ un paramètre de **densité** du graphe du CP-net cible $\mathcal{N}_T$ (*i.e.*, le ratio moyen du nombre d'arcs par variable, avec $\lambda = \frac{n-1}{2}$ correspondant au graphe complet). Notons, dans cette expérience, l'utilisation de l'erreur err, correspondant aux swaps n'étant pas modélisés dans le CP-net appris, mais dont la préférence inverse l'est, ainsi que de l'indécision ind, correspondant aux swaps dont ni la préférence induite, ni son inverse, sont présentes. Soit h_S l'hypothèse apprise à partir de l'ensemble d'entraînement S, nous avons :

$$err(S, h_S) = \frac{1}{|S|} \sum_{s \in S} \mathbb{1}(h_S(s) = 1 - y_s) \in [0, 1]. \tag{2.4}$$

$$ind(S, h_S) = 1 - (prec(S, h_S) + err(S, h_S)) \in [0, 1]. \tag{2.5}$$

Les résultats de cette expérience sont synthétisés dans le Tableau 2.3 ci-dessous, et comme nous pouvions nous en douter, étant donnée la différence de nature entre les deux algorithmes (hors ligne *versus* en ligne), l'Algorithme 5 se montre plus performant dans tous les cas. Nous pouvons cependant observer que l'erreur de notre algorithme augmente plus rapidement que celui de [GAG13]. La différence s'effectue alors sur l'indécision, très élevée dans [GAG13] mais inexistante dans l'Algorithme 5. Cela est dû au fait que dans [GAG13], lorsqu'une CP-règle n'obtient pas assez d'informations valides pour être acceptée (voir la Section 1.4.3.2 page 44), elle n'est pas ajoutée. Le CP-net obtenu est alors incomplet, ce qui implique un nombre élevé de swaps de S ne pouvant pas être comparés dans [GAG13]. Cette indécision a l'avan-

n	Précision (%)	Erreur (%)	Indécision (%)
		$\lambda = 1$	
4	98	2	0
	100	**0**	0
8	77	2	21
	100	**0**	**0**
12	65	2	34
	99	**1**	**0**
		$\lambda = 3$	
4	98	2	0
	100	**0**	0
8	80	4	16
	97	**3**	**0**
12	62	4	34
	96	4	**0**
		$\lambda = \frac{n-1}{2}$	
4	98	2	0
	99	**1**	0
8	76	**2**	22
	96	4	**0**
12	54	4	42
	96	4	**0**

TABLEAU 2.3 – Résultats de comparaison entre l'Algorithme 5 (cellules grises) et l'algorithme de [GAG13] (cellules blanches). Les résultats sont issus d'une moyenne de 30 exécutions sur des bases de données de 1 000 comparaisons. Le meilleur résultat est indiqué en gras (n représente le nombre de variables).

tage de ne proposer que des CP-règles en lesquelles l'utilisateur peut avoir confiance (en fonction de la valeur qu'il aura lui-même fixé), au détriment, cependant, d'une indécision parfois assez élevée.

Expérience n° 8 : temps d'apprentissage de l'Algorithme 5 en fonction du nombre de variables parentes du CP-net appris

Nous avons également calculé le temps d'exécution de l'Algorithme 5 basé sur le protocole précédent. Nous pouvons observer sur la Figure 2.8 que le temps d'apprentissage augmente exponentiellement en fonction de la densité λ et du nombre de variables n. L'expérience similaire conduite dans [GAG13] montre qu'avec une densité $\lambda = 1$ sur 12 variables (sans contraintes sur le nombre maximum de variables parentes hors acyclicité), leur procédure s'exécute en moins d'une seconde sur une base de données de 1 000 comparaisons, alors que l'Algorithme 5 a besoin du double de temps pour les mêmes paramètres d'expériences. Lorsque la densité est poussée à la limite de l'acyclicité (*i.e.*, $\lambda = \frac{n-1}{2}$), l'Algorithme 5 a alors besoin de plus 10

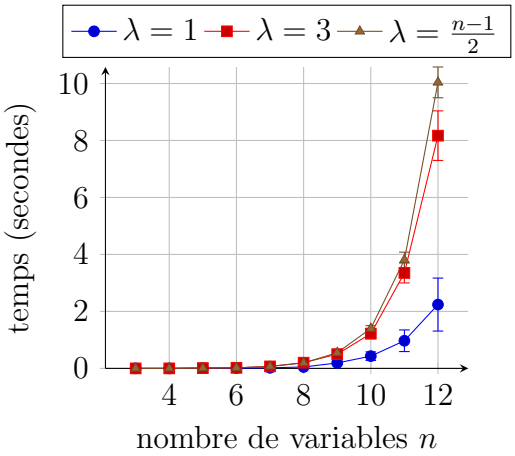


FIGURE 2.8 – Temps d'apprentissage en fonction du nombre de variables parentes du CP-net appris $\mathcal{N}_L$ (15 variables, sur une base de données de 20 000 comparaisons). Le nombre maximum de variables parentes est ici fixé à 5 pour le CP-net cible. Les résultats correspondent à une exécution de l'Algorithme 5 moyennée sur 30 exécutions. Chaque courbe correspond à une densité λ du CP-net cible et du CP-net appris différente.

secondes pour apprendre le CP-net. Ce phénomène trouve ses explications dans le fait que notre algorithme est hors ligne, alors que l'algorithme de Guerin *et al.* est en ligne, et ne requiert donc pas de reparcourt de la base de données.

Expérience n° 9 : efficacité de l'apprentissage de l'Algorithme 5 en fonction du nombre de variables parentes du CP-net cible

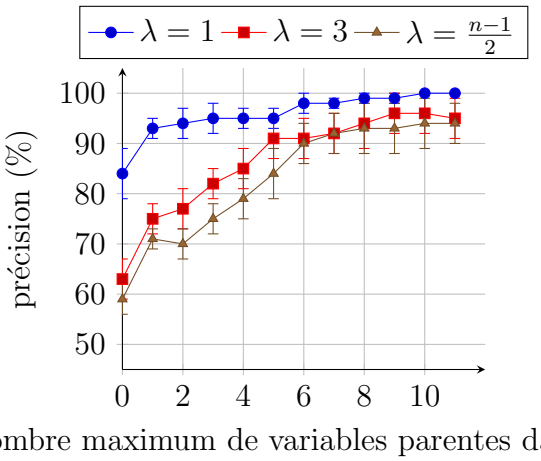


FIGURE 2.9 – Précision d'apprentissage en fonction du nombre maximum de variables parentes (12 variables, sur une base de données de 1 000 comparaisons). Les résultats sont moyennés sur 30 exécutions. Chaque courbe correspond à une densité λ du CP-net cible et du CP-net appris différente.

La Figure 2.9 présente les performances de l'Algorithme 5 lorsque le nombre

maximum k de variables parentes varie dans le CP-net appris $\mathcal{N}_L$. Nous observons ici une croissance très forte de la précision lorsque le nombre maximum de variables parentes est le même pour $\mathcal{N}_T$ et $\mathcal{N}_L$ ($k = 5$). Une fois cette limite franchie, nous pouvons voir une légère stagnation de la précision. Une remarque importante est que lorsque le CP-net cible $\mathcal{N}_T$ est « parcimonieux » (*i.e.*, son graphe associé est très peu dense), générer un CP-net $\mathcal{N}_L$ séparable permet d'obtenir une précision de 80%. Ce taux descend à environ 60% pour des densités plus élevées du CP-net cible. De plus, il est important de noter la différence de précision entre un CP-net séparable et un CP-net où chaque variable ne possède au maximum qu'un seul parent : cela peut être expliqué par le fait que l'ajout d'une variable parente double le nombre de CP-règles possibles, ce qui affine le modèle. L'ajout de variables parentes supplémentaires ne permet cependant pas d'obtenir un tel gain de précision, probablement à cause du CP-net qui devient alors trop complexe, où l'ajout de variables parentes apparait principalement pour corriger des erreurs dans le choix des précédentes variables parentes.

Expérience n° 10 : comparaison de l'efficacité d'apprentissage entre l'Algorithme 5 et l'algorithme de [LXW$^+$14]

Nous avons enfin comparé notre algorithme à celui de Liu *et al.* [LXW$^+$14] présenté dans la Section 1.4.3.1 page 42. Nous avons pour cela généré un CP-net de façon aléatoire, ayant 3 variables [10]. Nous avons ensuite généré 1 000 swaps à partir de ce CP-net, que nous avons bruité via une probabilité $p \in [0, 1]$ (pour chaque swap, un nombre entre 0 et 1 est aléatoirement généré, puis est mis en comparaison avec p afin de décider du bruitage ou non du swap). Le tableau 2.4 présente les résultats obtenus.

Bruit (%)	Liu *et al.* [LXW$^+$14] (%)	Algorithme 5 (%)
0	100	100
1	99	99
5	95	95
10	90	78
20	80	70
40	60	54

Tableau 2.4 – Précision d'apprentissage entre l'Algorithme 5 et l'algorithme de [LXW$^+$14], pour une base de données de 1 000 swaps sur des objets constitués de 3 variables.

Nous pouvons observer un jeu égal entre les deux algorithmes pour une valeur de bruit inférieure à 10%, puis une importante baisse de la précision de notre algorithme par rapport à celui de [LXW$^+$14]. Cette baisse peut s'expliquer par le fait que leur algorithme, grâce notamment à l'utilisation de méthodes de *branch-and-bound*,

10. Nous nous sommes limités à 3 variables, car l'algorithme de Liu *et al.* admettant une importante complexité, il n'était pas possible d'obtenir des résultats en des temps acceptables.

arrive à faire de meilleurs choix dans les variables parentes sélectionnées que notre algorithme.
Il est cependant à noter que notre algorithme propose une exécution plus rapide de son apprentissage comparativement à l'algorithme de Liu *et al.* [LXW+14]. Ceci n'est pas visible sur cette expérience à cause du nombre très restreint de variables, cependant, lors de l'apprentissage d'une base contenant 10 000 comparaisons d'objets sur 12 variables, l'algorithme de Liu *et al.* a besoin de plusieurs heures d'apprentissage [11] pour apprendre un CP-net ayant un graphe acyclique complet, contre moins de 3 minutes pour l'Algorithme 5 (voir la Figure 2.4 page 73).

2.5 Conclusion

Nous avons présenté dans ce chapitre un nouvel algorithme d'apprentissage de CP-nets s'inspirant des algorithmes d'apprentissage par requêtes hors ligne de CP-nets acycliques sur des bases de données contenant des préférences contradictoires. Nous avons pu montrer l'efficacité de notre algorithme à travers des expérimentations effectuées aussi bien sur un jeu de données réelles, que sur des bases de données synthétiques. Ces résultats montrent entre autres que malgré la complexité de l'Algorithme 5 (Proposition 3 page 67), il est capable d'apprendre en quelques secondes un CP-net sur des bases de données contenant quelques milliers d'exemples (swaps). De plus, la comparaison avec les algorithmes exposés dans [GAG13] et dans [LXW+14] a montré une amélioration de la précision sur l'apprentissage (malgré un temps de calcul plus important dans le cas de [GAG13]).

L'ingrédient principal de notre algorithme est sa gestion des contradictions, dont l'efficacité a été prouvée aussi bien sur des données réelles (base TripAdvisor), que sur des données synthétiques contenant un grand nombre de swaps. En particulier, le CP-net appris sur la base de données TripAdvisor souligne de bons résultats, malgré les contradictions sévissant à cause du grand nombre d'utilisateurs. Il a également été montré que l'algorithme converge empiriquement, et qu'il admet une bonne résistance au bruit.

Les points à améliorer dans le futur, pour cet algorithme, sont nombreux :

- Le premier concerne l'optimisation de certaines de ses parties (comme l'implémentation d'une parallélisation lors du calcul de la meilleure variable parente permettant de gagner un facteur n dans sa complexité générale) afin de réduire son temps d'apprentissage ;
- La recherche actuelle de la meilleure variable parente (Ligne 4 de l'Algorithme 6 page 62) est exhaustive. Cette recherche devient beaucoup trop couteuse en temps de calcul dès lors que la base de données S devient volumineuse ($>$ 20 000 swaps), même en parallélisant cette tache. Une solution serait alors de ne pas tout visiter et de chercher une approximation de ce nombre de swaps ;

11. Notons que pour ce test, l'algorithme sature la mémoire RAM avec plus de 12Go occupés, là où l'Algorithme 5 se limite à 1Go de mémoire.

- Enfin, la réception des contre-exemples pourrait également être revue. Cette réception est actuellement aléatoire, mais nous pourrions imaginer un contre-exemple maximisant les chances de ne pas commettre d'erreurs lors du choix de la variable parente.

Notre algorithme est hors ligne. Ainsi, il ne peut être une réponse au problème de passage à l'échelle. La contrainte de parcours de la base pour certaines requêtes ne permet notamment pas une adaptation simple de cet algorithme pour une utilisation en ligne. Le prochain chapitre introduit donc un nouvel algorithme proposant une version adaptée aux flux de données, et permettant également de gérer des contradictions au sein des préférences apprises.

CHAPITRE 3

ALGORITHME D'APPRENTISSAGE STATISTIQUE DE CP-NETS ACYCLIQUES

Sommaire

Résumé

Nous présentons dans ce chapitre un algorithme d'apprentissage de CP-nets comportant une version hors ligne et une version en ligne. Ces versions ont la particularité de se baser sur un critère entropique pour la détection et le choix des variables conditionnées et des variables parentes. La version en ligne utilise de plus la borne de McDiarmid afin d'obtenir des garanties sur la qualité finale de l'apprentissage. Dans cette version en ligne, chaque swap du flux de préférences sera représenté par un compteur particulier décrit dans ce chapitre, et servant au calcul de l'entropie. Chaque version de l'algorithme proposé sera analysée au niveau de sa complexité temporelle, et un nombre important d'expériences sera effectué afin de vérifier leur efficacité. Les tests seront appliqués sur des données synthétiques bruitées, et réelles. Nous comparerons également les résultats de notre algorithme à deux algorithmes de la littérature.

Liste des contributions de ce chapitre :

- Nouvelle version hors ligne d'apprentissage de CP-nets acycliques basée sur un apprentissage de plusieurs arbres de décision utilisant le gain d'information.
- Adaptation de la version hors ligne en une version en ligne utilisant la borne de McDiarmid.
- Généralisation à l'apprentissage de CP-nets cycliques.
- Nombreuses expérimentations sur des données synthétiques et réelles.
- Comparaisons avec quelques algorithmes de l'état de l'art.

Une version préliminaire en ligne de cet algorithme, basée sur la borne de Hoeffding [Hoe63], a été acceptée en tant que papier régulier dans l'édition 2017 de IEEE ICDM (International Conference of Data Mining) [LZM+17].

3.1 Introduction

Les données, récupérées par des outils de veille statistique, tels que les clics de souris sur des sites Web, permettent d'induire des préférences entre, par exemple, des articles sur des sites d'e-commerce (tels qu'Amazon). Cette récolte se déroule rarement sans accrocs, et deux types de problèmes peuvent alors intervenir :

- Des erreurs techniques lors de la récupération des préférences (coupure de réseau, ou réseau instable par exemple) ;
- Pour certaines raisons, la personne exposant ses préférences se trompe (contexte particulier entraînant un changement ponctuel de ses préférences, ou manque de réflexion).

De telles erreurs sont alors nommées **bruit** ou **préférences bruitées**, et sont souvent la cause d'incohérences au sein de l'ordre partiel des objets, entraînant alors un cycle de préférences de ces-dits objets (voir la Section 1.4.1.3 page 32). Il est alors nécessaire de pouvoir déterminer un maximum de ces préférences bruitées afin de ne pas les prendre en compte dans l'apprentissage.

Une autre contrainte, de plus en plus courante sur internet, est de recevoir tellement de données en temps réel qu'il n'est plus possible de les traiter dans leur totalité, mais au fur et à mesure de leur flux. Un algorithme d'apprentissage **en ligne**, ayant pour but de traiter le plus efficacement ce flux d'informations sans le stocker, devient alors nécessaire au bon apprentissage d'un système de recommandation (voir la Figure 3.1). Ce cadre streaming exige de plus un accès permanent à une structure à jour et modélisant le plus précisément possible l'ensemble des informations déjà observées.

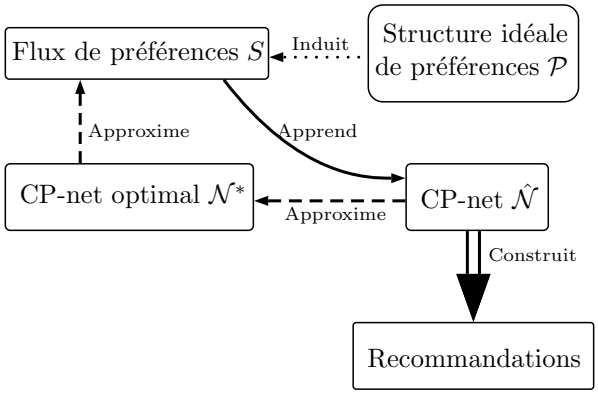


FIGURE 3.1 – Schéma global d'une procédure d'apprentissage en ligne de CP-nets.

Il existe actuellement deux algorithmes traitant du bruit dans les préférences [LYX+13, LXW+14] (voir la Section 1.4.3.1 page 42 pour une étude de la méthode proposée dans [LXW+14]), cependant il s'agit d'algorithmes hors ligne, et le seul algorithme en ligne recensé [GAG13] ne prend pas en compte les préférences bruitées. Nous proposons donc dans un premier temps une version hors ligne d'un nouvel algorithme traitant efficacement les préférences bruitées, puis nous dérivons de cet algorithme une version en ligne permettant d'apprendre un CP-net acyclique sans avoir à reparcourir les informations déjà aperçues.

Notre problème d'apprentissage peut être vu comme un apprentissage simultané de plusieurs arbres de décision, basé sur une mesure de séparation entropique permettant d'isoler les données bruitées, en considérant comme « vraies » les préférences majoritaires.

L'algorithme présenté dans ce chapitre est une amélioration d'un algorithme d'apprentissage en ligne de CP-nets acycliques [LZM+17], basé sur la borne de Hoeffding. Cette borne n'étant pas adaptée au problème d'apprentissage de CP-nets à cause de la non-linéarité du gain d'information, nous corrigeons ce point par l'utilisation

Variable	Valeurs	
A	a	a'
B	b	b'
C	c	c'
D	d	d'
E	e	e'
F	f	f'

TABLEAU 3.1 – Les variables utilisées et leurs valeurs.

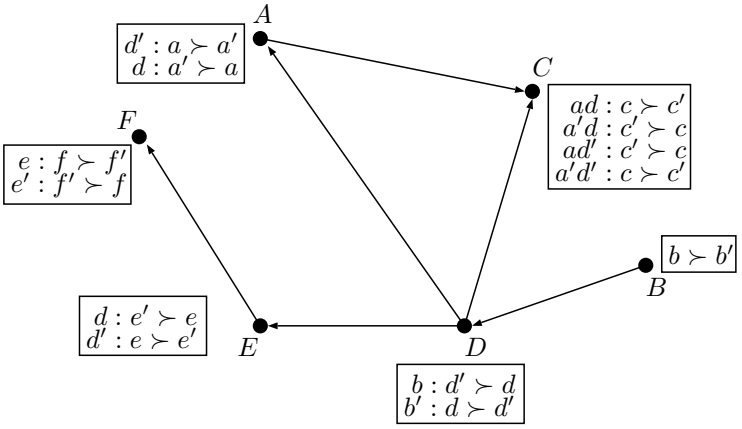


FIGURE 3.2 – CP-net ayant servi à générer les swaps exemples.

d'une autre borne, plus adaptée à notre problème, tout en améliorant le fonctionnement global de la procédure d'apprentissage, et en introduisant la version hors ligne de ce même algorithme.

Nous déroulerons les différentes méthodes sur un exemple de 6 variables (voir le Tableau 3.1) modélisé par le CP-net de la Figure 3.2. Pour ce CP-net, nous travaillerons avec la base de données de 50 swaps du Tableau 3.3 (pour plus de clarté, un objet type (a, b, c) sera par la suite noté directement abc), accompagnée du Tableau 3.2 résumant les différents ensembles de classes possibles.

Ensembles de classes	Classes	
Y_A	$1_A \Leftrightarrow a \succ a'$	$0_A \Leftrightarrow a' \succ a$
Y_B	$1_B \Leftrightarrow b \succ b'$	$0_B \Leftrightarrow b' \succ b$
Y_C	$1_C \Leftrightarrow c \succ c'$	$0_C \Leftrightarrow c' \succ c$
Y_D	$1_D \Leftrightarrow d \succ d'$	$0_D \Leftrightarrow d' \succ d$
Y_E	$1_E \Leftrightarrow e \succ e'$	$0_E \Leftrightarrow e' \succ e$
Y_F	$1_F \Leftrightarrow f \succ f'$	$0_F \Leftrightarrow f' \succ f$

TABLEAU 3.2 – Les différentes classes utilisées ainsi que leurs explications.

	Préférence	Classe		Préférence	Classe
1	$abcd'ef \succ a'bcd'ef$	1_A	26	$ab'c'def \succ ab'c'd'ef$	1_D
2	$abc'd'ef \succ a'bc'd'ef$	1_A	27	$a'b'cdef \succ a'b'cd'ef$	1_D
3	$ab'c'd'e'f' \succ a'b'c'd'e'f'$	1_A	28	$ab'cdef \succ ab'cd'ef$	1_D
4	$a'bcdef' \succ abcdef'$	0_A	29	$ab'cde'f \succ ab'cd'e'f$	1_D
5	$a'bc'de'f \succ abc'de'f$	0_A	30	$abcd'ef \succ abcdef$	0_D
6	$a'b'cdef \succ ab'cdef$	0_A	31	$abcd'e'f' \succ abcde'f'$	0_D
7	$a'bcde'f \succ abcde'f$	0_A	32	$a'bcd'ef \succ a'bcdef$	0_D
8	$a'b'c'def \succ ab'c'def$	0_A	33	$abcd'e'f \succ abcde'f$	0_D
9	$a'bcde'f' \succ abcde'f'$	0_A	34	$abcd'e'f \succ abcde'f$	0_D
10	$ab'c'def \succ abc'def$	0_B	35	$abcd'ef' \succ abcd'e'f'$	1_E
11	$ab'cdef \succ abcdef$	0_B	36	$ab'c'd'ef \succ ab'c'd'e'f$	1_E
12	$ab'cdef \succ abcdef$	0_B	37	$abcde'f' \succ abcdef'$	0_E
13	$a'b'cdef \succ a'bcdef$	0_B	38	$abcde'f \succ abcdef$	0_E
14	$abcdef \succ abc'def$	1_C	39	$a'bcde'f \succ a'bcdef$	0_E
15	$abcde'f \succ abc'de'f$	1_C	40	$a'b'cde'f \succ a'b'cdef$	0_E
16	$ab'cdef \succ ab'c'def$	1_C	41	$a'b'c'de'f \succ a'b'c'def$	0_E
17	$a'bcd'e'f \succ a'bc'd'e'f$	1_C	42	$abcdef \succ abcdef'$	1_F
18	$a'b'cd'ef \succ a'b'c'd'ef$	1_C	43	$a'bcdef \succ a'bcdef'$	1_F
19	$a'bc'def \succ a'bcdef$	0_C	44	$ab'cdef \succ ab'cdef'$	1_F
20	$abc'd'ef \succ abcd'ef$	0_C	45	$abc'def \succ abc'def'$	1_F
21	$a'bc'de'f \succ a'bcde'f$	0_C	46	$abcd'ef \succ abcd'ef'$	1_F
22	$abc'd'ef' \succ abcd'ef'$	0_C	47	$a'b'cdef \succ a'b'cdef'$	1_F
23	$ab'cdef \succ ab'cd'ef$	1_D	48	$a'b'cde'f' \succ a'b'cde'f$	0_F
24	$ab'cdef' \succ ab'cd'ef'$	1_D	49	$ab'c'de'f' \succ ab'c'de'f$	0_F
25	$a'b'c'def \succ a'b'c'd'ef$	1_D	50	$abcde'f' \succ abcde'f$	0_F

TABLEAU 3.3 – Base de 50 préférences *ceteris paribus* générées à partir du CP-net de la Figure 3.2.

3.2 Algorithme d'apprentissage hors ligne

Rappelons que nous travaillons à partir d'un ensemble $S = \{(s_1, y_1), \ldots, (s_m, y_m)\}$ correspondant à l'ensemble d'apprentissage contenant m swaps, chacun associé à une variable parmi l'ensemble $\mathbf{V}$ de n variables. Cet ensemble peut être scindé en n sous-ensembles d'apprentissage $S_V = \{(s = (\mathbf{o}, \mathbf{o}')_{V'}, y_s) \in S \mid V' = V\} \subseteq S$. Nous pouvons de plus observer qu'en présence de bruit, il est courant de rencontrer une CP-règle et son inverse. Nous introduisons donc le **schéma d'une CP-règle**, permettant d'abstraire cette CP-règle.

Définition 3.1 (Schéma d'une CP-règle)**.**
Soit $\mathbf{V}$ *un ensemble de variables, et soient* $(\mathbf{u} : v \succ v')$ *et* $(\mathbf{u} : v' \succ v)$ *une CP-règle et son inverse sur la variable* $V \in \mathbf{V}$ *(avec* $\mathbf{u} \in Dom(Pa(V))$*), respectivement. On appelle* ***schéma d'une CP-règle*** *une CP-règle dont la règle de préférence sur les valeurs de* V *n'a pas encore été établie. Ce schéma se note* $(\mathbf{u} :\succ_{Dom(V)})$ *(i.e., un ordre de préférence, encore inconnu, sur les valeurs de* V*, qui est conditionné par l'état* $\mathbf{u} \in Dom(Pa(V))$*), et permet ainsi d'abstraire les deux types de CP-règles* $(\mathbf{u} : v \succ v')$ *et* $(\mathbf{u} : v' \succ v)$*.*

Nous supposons, dans la suite de ce chapitre, que $CPT(V)$ contient désormais uniquement des schémas de CP-règles, pour toute variable V d'un ensemble $\mathbf{V}$.

3.2.1 Principe général

Le principe de notre algorithme est très grandement inspiré des algorithmes d'apprentissage d'arbres de décision basés sur le gain d'information tels que l'algorithme ID3 [Qui86] et l'algorithme C4.5 [Qui93]. Ce gain d'information utilise les mesures de pureté telles que l'indice de Gini [Gas72] ou l'entropie de Shannon [CT06], afin de déterminer les nouvelles variables parentes. Cette inspiration est également motivée par le fait que les arbres de décision sont, tout comme les CP-nets, des structures graphiques intuitives.

Nous pouvons voir que le problème d'apprentissage étudié (voir la Section 1.5 page 50) peut être vu comme la résolution de n arbres de décision, où chaque arbre permet de modéliser l'ensemble des variables parentes d'une variable $V \in \mathbf{V}$. Faisons l'hypothèse, dans un premier temps, de n'avoir qu'une seule variable $V \in \mathbf{V}$ pouvant être conditionnée. Ainsi, seuls les swaps de l'ensemble S_V seront utilisés. Nous cherchons alors à affecter les swaps de S_V dans l'une des deux classes possibles de Y_V : 1_V si le swap implique $v \succ v'$, et 0_V si le swap implique $v' \succ v$. Les différents nœuds internes de l'arbre de la variable V correspondent alors aux variables parentes de V, et les feuilles correspondent aux CP-règles de la variable courante (voir la Figure 3.3). Cet apprentissage s'effectue alors simultanément sur l'ensemble des n arbres.

Contrairement à l'apprentissage classique des arbres de décision, notre algorithme ajoutera une variable parente à chaque feuille de l'arbre courant. Cette variable

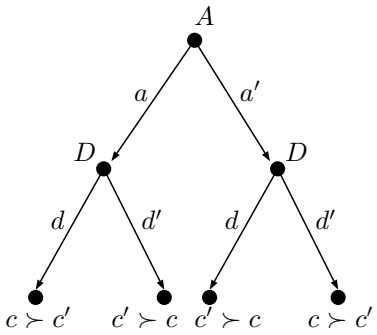


FIGURE 3.3 – Arbre de décision lié à une variable C possédant deux variables parentes A et D. Chaque feuille correspond à une préférence entre les valeurs de C, et chaque noeud interne correspond à une variable parente.

va être ajoutée sur chaque feuille de l'arbre courant (du fait que chaque feuille correspond à une seule CP-règle, et que chaque CP-règle sera dédoublée après l'ajout d'une nouvelle variable parente). Cela demande alors le calcul d'une entropie globale de l'arbre, là où seule l'entropie de la feuille sélectionnée est nécessaire pour un apprentissage classique d'un arbre de décision.

Définition 3.2 (Entropie d'une variable)**.**
Soient $\mathbf{V}$ *un ensemble de variables, et* $V \in \mathbf{V}$ *une variable de cet ensemble. Soit* $S_V = \{(s_1, y_1), \ldots, ((s_l, y_l)\}$ *un ensemble de couples composés d'un swap* $s_i \in Dom(\mathbf{V})^2$ *ayant* V *comme variable de swap, ainsi que d'une classe associée* $y_i \in Y_V$*. L'**entropie** de* V *associée à la classification des éléments de* S_V *est alors définie par*

$$\hat{H}_{1/2}(S_V) = \sum_{(\mathbf{u}:\succ_{Dom(V)}) \in CPT(V)} \frac{|S_V^{\mathbf{u}}|}{|S_V|} \hat{H}_{1/2}(S_V, (\mathbf{u} :\succ_{Dom(V)})), \tag{3.1}$$

avec

$$\hat{H}_{1/2}(S_V, (\mathbf{u} :\succ_{Dom(V)})) = -\sum_{y \in Y_V} \frac{|S_y^{\mathbf{u}}|}{2|S_V^{\mathbf{u}}|} \ln \frac{|S_y^{\mathbf{u}}|}{|S_V^{\mathbf{u}}|}, \tag{3.2}$$

où

- $S_V^{\mathbf{u}} = \{((\mathbf{o}, \mathbf{o}')_{V'}, y') \in S \mid V' = V, \mathbf{o}[Pa(V)] = \mathbf{o}'[Pa(V)] = \mathbf{u}\}$ *correspondant à l'ensemble des swaps ayant* V *comme variable de swap, et ayant l'état* $\mathbf{u}$ *pour l'ensemble des variables parentes de* V *;*
- $S_y^{\mathbf{u}} = \{((\mathbf{o}, \mathbf{o}')_{V'}, y') \in S \mid y' = y, \mathbf{o}[Pa(V)] = \mathbf{o}'[Pa(V)] = \mathbf{u}\}$ *correspondant à l'ensemble des swaps étant associés à la classe* y*, et ayant l'état* $\mathbf{u}$ *pour l'ensemble des variables parentes de* V*.*

Nous utilisons ici l'entropie normalisée [RC17] plutôt que l'entropie classique [CT06] afin de dériver plus facilement une borne d'erreur dans la suite de ce document. Cette entropie permet donc de déterminer la pureté globale d'une variable, en sommant l'entropie pondérée de chacun de ses schémas de CP-règles.

Exemple 3.1. *En reprenant les swaps donnés dans le Tableau 3.3 page 87, nous obtenons, pour le calcul de l'entropie de la variable A de l'ensemble S_A :*

$$\begin{aligned}\hat{H}_{1/2}(S_A) &= \sum_{(\mathbf{u}:\succ_{Dom(A)})\in CPT(A)} \frac{|S_A^{\mathbf{u}}|}{|S_A|}\hat{H}_{1/2}(S_A,(\mathbf{u}:\succ_{Dom(A)})) \\ &= -\sum_{(\mathbf{u}:\succ_{Dom(A)})\in CPT(A)} \frac{|S_A^{\mathbf{u}}|}{|S_A|}\sum_{y\in Y_A}\frac{|S_y^{\mathbf{u}}|}{2|S_A^{\mathbf{u}}|}\ln\frac{|S_y^{\mathbf{u}}|}{|S_A^{\mathbf{u}}|}.\end{aligned}$$

La variable A ne possède, à ce stade, aucune variable parente. Ainsi, $\mathbf{u} = \emptyset$, et seule la deuxième somme est utilisée (dans le cas présent, nous avons $S_A = S_A^{\emptyset}$). Il y a 9 swaps faisant intervenir la variable A, donc $|S_A^{\emptyset}| = 9$. Parmi ces 9 swaps, 3 impliquent que $a \succ a'$, et correspondent donc à la classe 1_A, et 6 impliquent que $a' \succ a$, et correspondent donc à la classe 0_A. Ainsi, $|S_{1_A}^{\emptyset}| = 3$, et $|S_{0_A}^{\emptyset}| = 6$. Nous obtenons alors

$$\hat{H}_{1/2}(S_A) = -\frac{3}{18}\ln\frac{3}{9} - \frac{6}{18}\ln\frac{6}{9} = 0.318.$$

Définition 3.3 (Pureté d'un schéma de CP-règle et d'une variable)**.**
Soient $V \in \mathbf{V}$ une variable, $(\mathbf{u}:\succ_{Dom(V)})$ un schéma de CP-règle, S_V un ensemble de swaps sur V, et $Y_V = \{1_V, 0_V\}$ un ensemble de classes. On dit que le schéma $(\mathbf{u}:\succ_{Dom(V)})$ est

- ***Pur** si $|S_{1_V}^{\mathbf{u}}| = 0$ ou $|S_{0_V}^{\mathbf{u}}| = 0$;*
- ***Impur** sinon.*

De plus, on dit qu'une variable V est

- ***Pure** si l'ensemble de ses schémas de CP-règles est pur ;*
- ***Impure** si au moins l'un de ses schémas est impur.*

Notre objectif étant d'arriver à un ensemble de variables les plus pures possible, nous cherchons la variable V maximisant son entropie $\hat{H}_{1/2}(S_V)$ sur l'ensemble S_V. Une fois l'entropie courante calculée, il est nécessaire de déterminer l'entropie obtenue si l'on ajoute une nouvelle variable parente. Soit $Q \in \overline{Pa}(V)$ la variable parente candidate [1]. Rappelons que l'ajout d'une variable parente aura pour conséquence la division en deux de chaque CP-règle existante. Pour calculer l'entropie après l'ajout de la variable parente, nous devons prendre en compte l'ensemble des feuilles de l'arbre.

Définition 3.4 (Entropie d'un couple de variables)**.**
Soient $\mathbf{V}$ un ensemble de variable, $V, Q \in \mathbf{V}$ deux variables, avec $Q \in \overline{Pa}(V)$. Soit $S_V = \{(s_1, y_1), \ldots, ((s_l, y_l)\}$ un ensemble de couples composés d'un swap $s_i \in Dom(\mathbf{V})^2$ ayant V comme variable de swap, ainsi qu'une classe associée

1. $\overline{Pa}(V)$ correspond à l'ensemble des variables non parentes de V (voir la Section 1.4.1.2 page 30).

$y_i \in Y_V$. *L'**entropie du couple de variables*** (Q, V), *étant donné un ensemble* S_V, *correspond alors à*

$$\hat{H}_{1/2}(S_V, Q) = \sum_{q \in Dom(Q)} \sum_{(\mathbf{u}:\succ_{Dom(V)}) \in CPT(V)} \frac{|S_V^{\mathbf{u}q}|}{|S_V|} \hat{H}_{1/2}(S_V, (\mathbf{u} :\succ_{Dom(V)}), q), \quad (3.3)$$

avec

$$\hat{H}_{1/2}(S_V, (\mathbf{u} :\succ_{Dom(V)}), q) = -\sum_{y \in Y_V} \frac{|S_y^{\mathbf{u}q}|}{2|S_V^{\mathbf{u}q}|} \ln \frac{|S_y^{\mathbf{u}q}|}{|S_V^{\mathbf{u}q}|}, \quad (3.4)$$

où

- $S_V^{\mathbf{u}q} = \{((\mathbf{o}, \mathbf{o}')_{V'}, y') \in S \mid V' = V, \mathbf{o}[Pa(V)] = \mathbf{o}'[Pa(V)] = \mathbf{u}, \mathbf{o}[Q] = \mathbf{o}'[Q] = q\}$ *correspondant à l'ensemble des swaps ayant* V *comme variable de swap, ayant l'état* $\mathbf{u}$ *comme valeur pour les variables parentes de* V, *et* q *comme valeur de la variable* Q ;
- $S_y^{\mathbf{u}q} = \{((\mathbf{o}, \mathbf{o}')_{V'}, y') \in S \mid \mathbf{o}[V'] = v, \mathbf{o}[Pa(V)] = \mathbf{o}'[Pa(V)] = \mathbf{u}, \mathbf{o}[Q] = \mathbf{o}'[Q] = q\}$ *correspondant à l'ensemble des swaps étant associés à la classe* y, *ayant l'état* $\mathbf{u}$ *comme valeur pour les variables parentes de* V, *et* q *comme valeur de la variable* Q.

L'entropie $\hat{H}_{1/2}(S_V, Q)$ représente donc l'entropie obtenue après l'ajout de la variable Q comme variable parente de V. L'objectif est donc de minimiser l'entropie obtenue, en cherchant à obtenir des variables les plus pures possibles.

Exemple 3.2. *Reprenons l'exemple précédent, en supposant que nous souhaitons ajouter la variable* B *comme variable parente de* A. *Nous calculons donc l'entropie* $\hat{H}_{1/2}(S_A, B)$, *correspondant à l'entropie attendue après l'ajout de l'arc* (B, A) *au CP-net, afin de la comparer à l'entropie initiale* $\hat{H}_{1/2}(S_A)$, *et voir si* B *ajoute de l'information à* A :

$$\begin{aligned}
\hat{H}_{1/2}(S_A, B) &= \sum_{b \in Dom(B)(\mathbf{u}:\succ_{Dom(A)}) \in CPT(A)} \sum \frac{|S_A^{\mathbf{u}b}|}{|S_A|} \hat{H}_{1/2}(S_A, (\mathbf{u} :\succ_{Dom(A)}), q) \\
&= \sum_{b \in Dom(B)(\mathbf{u}:\succ_{Dom(A)}) \in CPT(A)} \sum \frac{|S_A^{\mathbf{u}b}|}{|S_A|} \left(-\sum_{y \in Y_A} \frac{|S_y^{\mathbf{u}b}|}{2|S_A^{\mathbf{u}b}|} \ln \frac{|S_y^{\mathbf{u}b}|}{|S_A^{\mathbf{u}b}|} \right)
\end{aligned}$$

Comme pour l'Exemple 3.1 page 90, la variable A *ne possède toujours pas de variables parentes à ce stade. Ainsi, la deuxième somme n'entre pas en jeu car* $\mathbf{u} = \emptyset$. *La première somme permet de partager les swaps de* S_A *entre les valeurs* b *et* b' *de la variable* B : *sur les* 9 *swaps ayant* A *comme variable de swap,* 6 *possèdent la valeur* b, *et* 3 *possèdent la valeur* b'. *Les swaps sont ensuite de nouveau divisés en fonction de leurs classes. Ainsi, sur les* 9 *swaps faisant intervenir la valeur* b, 2 *sont associés à la classe* 1_A, *et* 4 *sont associés à la classe* 0_A. *La même opération est effectuée pour les swaps faisant intervenir la valeur* b', *ce qui permet d'obtenir l'entropie du couple de variables* (B, A) *suivante :*

$$\hat{H}_{1/2}(S_A, B) = \frac{6}{9}\left(-\frac{2}{12}\ln\frac{2}{6} - \frac{4}{12}\ln\frac{4}{6}\right) - \frac{3}{9}\left(-\frac{1}{6}\ln\frac{1}{3} - \frac{2}{6}\ln\frac{2}{3}\right) = 0.318.$$

Nous pouvons voir que l'information dégagée par l'ajout de la variable B est identique à l'information dégagée par A sans cet ajout.

Définition 3.5 (Gain d'information)**.**
Soient $\mathbf{V}$ un ensemble de variable, $V, Q \in \mathbf{V}$ deux variables, avec $Q \in \overline{Pa}(V)$. Soit $S_V = \{(s_1, y_1), \ldots, ((s_l, y_l)\}$ un ensemble de couples composés d'un swap $s_i \in Dom(\mathbf{V})^2$ ayant V comme variable de swap, ainsi qu'une classe associée $y_i \in Y_V$. Le ***gain d'information*** *de la variable Q par rapport à la variable V, correspondant à la différence entre l'entropie de V avant l'ajout de la variable Q comme variable parente, et l'entropie de V après son ajout, est donné par*

$$\hat{G}_{1/2}(S_V, Q) = \hat{H}_{1/2}(S_V) - \hat{H}_{1/2}(S_V, Q). \tag{3.5}$$

Nous cherchons donc le couple de variables $(\hat{Q}, \hat{V})$ maximisant $\hat{G}_{1/2}(S_V, Q)$:

$$(\hat{Q}, \hat{V}) = \underset{(Q,V),\ V \in \mathbf{V},\ Q \in \overline{Pa}(V)}{\operatorname{argmax}} \hat{G}_{1/2}(S_V, Q). \tag{3.6}$$

Intuitivement, le choix d'un tel couple, à chaque itération de l'algorithme, permet de minimiser le nombre de CP-règles non modélisées par le CP-net. Par conséquent, on minimise la perte introduite dans l'Équation (1.24) page 54.

Exemple 3.3. *Nous avons pu voir dans l'Exemple 3.2 page 92 que l'ajout de la variable B comme parente de A ne semble apporter aucune nouvelle information. Ceci est vérifié en calculant le gain d'information associé au couple (B, A) :*

$$\begin{aligned}\hat{G}_{1/2}(S_A, B) &= \hat{H}_{1/2}(S_A) - \hat{H}_{1/2}(S_A, B) \\ &= 0.318 - \tfrac{6}{9}\left(-\tfrac{2}{12}\ln\tfrac{2}{6} - \tfrac{4}{12}\ln\tfrac{4}{6}\right) - \tfrac{3}{9}\left(-\tfrac{1}{6}\ln\tfrac{1}{3} - \tfrac{2}{6}\ln\tfrac{2}{3}\right) = 0.\end{aligned}$$

La Figure 3.4 illustre plus clairement les conséquences de l'ajout de B : 3 swaps sur les 9 swaps ayant A comme variable de swap ne peuvent pas être modélisés sans l'ajout de variables parentes. Cependant, après ajout de B, 2 swaps sur les 6 faisant intervenir la valeur $b \in Dom(B)$ ne sont toujours pas modélisés, et 1 swap sur les 3 faisant intervenir l'autre valeur $b' \in Dom(B)$ n'est également pas modélisé, ce qui donne, de nouveau un total de 3 swaps non modélisés (les 3 swaps faisant intervenir la préférence $a \succ a'$).

3.2.2 Exemple déroulant

Soient $\mathbf{V}$ un ensemble de variables, et S un ensemble d'entraînement contenant des swaps pouvant être partitionné en n sous-ensembles S_V, $\forall V \in \mathbf{V}$. Le principe de l'algorithme proposé est le suivant :

1. Calculer l'entropie pour chaque variable $V \in \mathbf{V}$;
2. Calculer le gain d'information pour chaque couple de variables (Q, V), $\forall (Q, V) \in \mathbf{V}^2$;

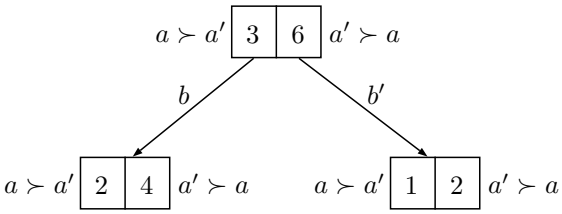


FIGURE 3.4 – Ajout de la variable B comme nouvelle variable parente de A dans l'exemple du Tableau 3.3 page 87. L'entropie du premier étage cet arbre de décision, et l'entropie du second étage, sont identiques.

3. Trier les différents gains d'information par ordre décroissant ;
4. Sélectionner le couple $(\hat{Q}, \hat{V})$ maximisant ce gain ;
5. Si l'ajout de ce couple ne crée pas de cycle, alors ajouter l'arc $(\hat{Q}, \hat{V})$ dans le graphe du CP-net. Sinon, recommencer cette étape en sélectionnant le couple de variables suivant dans la liste triée de l'étape 3 ;
6. Si aucun couple n'a été trouvé, alors retourner le CP-net obtenu, sinon recommencer à la première étape.

Appliquons ce processus sur la base de données S du Tableau 3.3 page 87.

Première étape

Le calcul de l'entropie de chaque variable (*i.e.*, l'entropie de l'arbre de décision associé à chaque variable V, composé pour l'instant d'une unique racine indiquant le nombre de swaps ayant $v \succ v'$ et $v' \succ v$) nous donne

- $\hat{H}_{1/2}(S_A) = -\frac{3}{18}\ln\frac{3}{9} - \frac{6}{18}\ln\frac{6}{9} = 0.318$;
- $\hat{H}_{1/2}(S_B) = 0$;
- $\hat{H}_{1/2}(S_C) = -\frac{4}{18}\ln\frac{4}{9} - \frac{5}{18}\ln\frac{5}{9} = 0.344$;
- $\hat{H}_{1/2}(S_D) = -\frac{7}{24}\ln\frac{7}{12} - \frac{5}{24}\ln\frac{5}{12} = 0.34$;
- $\hat{H}_{1/2}(S_E) = -\frac{2}{14}\ln\frac{2}{7} - \frac{5}{14}\ln\frac{5}{7} = 0.299$;
- $\hat{H}_{1/2}(S_F) = -\frac{3}{18}\ln\frac{3}{9} - \frac{6}{18}\ln\frac{6}{9} = 0.318$.

Ce premier calcul nous permet d'avoir une idée générale de la répartition des préférences de chaque variable. On note par exemple que la variable B est déjà pure puisqu'il n'existe que la préférence $b' \succ b$. À ce stade, le choix optimal devrait se porter sur la variable ayant l'entropie la plus importante (ici, la variable C, faisant intervenir le sous-ensemble de swaps S_C). Cependant, ce choix dépend également de l'entropie de la variable après ajout d'une nouvelle variable parente.

Deuxième étape

Nous rappelons que nous voulons obtenir un CP-net acyclique. Dans le cadre d'un apprentissage de CP-net cyclique, un apprentissage indépendant sur chaque variable (*i.e.*, sur chaque arbre) peut être envisagé. Cependant, dans le cadre présent, un choix dans les variables est nécessaire à cause de la limite que nous impose la contrainte d'acyclicité. Nous devons donc calculer les gains d'information de l'ensemble des couples de variables possibles :

- Pour la variable A :
  - $\hat{G}_{1/2}(S_A, B) = 0.318 - \frac{6}{9}\left(-\frac{2}{12}\ln\frac{2}{6} - \frac{4}{12}\ln\frac{4}{6}\right) - \frac{3}{9}\left(-\frac{1}{6}\ln\frac{1}{3} - \frac{2}{6}\ln\frac{2}{3}\right) = 0$;
  - $\hat{G}_{1/2}(S_A, C) = 0.318 - \frac{5}{9}\left(-\frac{1}{10}\ln\frac{1}{5} - \frac{4}{10}\ln\frac{4}{5}\right) - \frac{4}{9}\left(-\frac{2}{8}\ln\frac{2}{4} - \frac{2}{8}\ln\frac{2}{4}\right) = 0.025$;
  - $\hat{G}_{1/2}(S_A, D) = 0.318 - 0 = 0.318$;
  - $\hat{G}_{1/2}(S_A, E) = 0.318 - \frac{5}{9}\left(-\frac{2}{10}\ln\frac{2}{5} - \frac{3}{10}\ln\frac{3}{5}\right) - \frac{4}{9}\left(-\frac{1}{8}\ln\frac{1}{4} - \frac{3}{8}\ln\frac{3}{4}\right) = 0.006$;
  - $\hat{G}_{1/2}(S_A, F) = 0.318 - \frac{6}{9}\left(-\frac{2}{12}\ln\frac{2}{6} - \frac{4}{12}\ln\frac{4}{6}\right) - \frac{3}{9}\left(-\frac{1}{6}\ln\frac{1}{3} - \frac{2}{6}\ln\frac{2}{3}\right) = 0.$
- Pour la variable B : $\hat{G}_{1/2}(S_B, X) = 0,\ \forall X \in \mathbf{V} \setminus \{B\}$;
- Pour la variable C :
  - $\hat{G}_{1/2}(S_C, A) = 0.003$;
  - $\hat{G}_{1/2}(S_C, B) = 0.166$;
  - $\hat{G}_{1/2}(S_C, D) = 0.003$;
  - $\hat{G}_{1/2}(S_C, E) = 0.007$;
  - $\hat{G}_{1/2}(S_C, F) = 0.05$;
- Pour la variable D :
  - $\hat{G}_{1/2}(S_D, A) = 0.003$;
  - $\hat{G}_{1/2}(S_D, B) = 0.34$;
  - $\hat{G}_{1/2}(S_D, C) = 0.051$;
  - $\hat{G}_{1/2}(S_D, E) = 0.059$;
  - $\hat{G}_{1/2}(S_D, F) = 0.001$;
- Pour la variable E :
  - $\hat{G}_{1/2}(S_E, A) = 0.101$;
  - $\hat{G}_{1/2}(S_E, B) = 0.002$;
  - $\hat{G}_{1/2}(S_E, C) = 0.021$;
  - $\hat{G}_{1/2}(S_E, D) = 0.299$;
  - $\hat{G}_{1/2}(S_E, F) = 0.021$;
- Pour la variable F :
  - $\hat{G}_{1/2}(S_F, A) = 0$;

- $\hat{G}_{1/2}(S_F, B) = 0.054$;
- $\hat{G}_{1/2}(S_F, C) = 0.008$;
- $\hat{G}_{1/2}(S_F, D) = 0.024$;
- $\hat{G}_{1/2}(S_F, E) = 0.318$.

Nous pouvons observer, en nous concentrant sur la variable A, que cette variable maximise son gain d'information grâce à la variable D. Par conséquent, cette dernière semble être la variable la plus indiquée pour être parente de A. À l'inverse, les variables B et F donnent un gain nul, ce qui signifie qu'elles n'apportent aucune information à A (le nombre de swaps représentables dans l'arbre de A, après l'ajout de B ou F, est identique au nombre de swaps actuellement représentables).

Troisième, quatrième, et cinquième étape

Comme nous l'avons expliqué précédemment, il est nécessaire de considérer l'ensemble des gains d'information de toutes les variables afin de déterminer quel couple est à même d'être sélectionné. Ainsi, en triant l'ensemble des gains d'information de toutes les variables par ordre décroissant, le couple de variables finalement sélectionné est le couple (B, D) (la variable B devient la nouvelle variable parente de D). La Figure 3.5 illustre cet ajout. Nous pouvons voir que sans variable parente, sur les 12 swaps faisant intervenir la variable D, 7 modélisent $d \succ d'$, et 5 modélisent $d' \succ d$. Cependant, lorsque l'on prend en compte les valeurs de la variable B, nous pouvons voir qu'elle discrétise complètement les deux classes, avec la valeur b modélisant complètement $d' \succ d$, et la valeur b' modélisant complètement $d \succ d'$ (chaque feuille, donc chaque CP-règle, et donc la variable D, est pure).

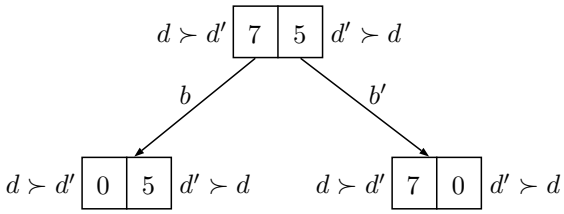


FIGURE 3.5 – Arbre de décision associé à la variable D ayant B comme variable parente.

Le processus est alors répété après l'ajout de l'arc (B, D). Remarquons que comme chaque variable fait intervenir un swap différent, les gains de chaque variable sont complètement dissociés, et les valeurs précédentes restent correctes (sauf pour la variable D, qui voit son gain descendre logiquement à 0 pour toutes les variables parentes candidates restantes). Dans les gains restants les plus élevés, nous avons le choix entre le couple (D, A) et le couple (E, F). Un choix aléatoire est donc fait et le couple (D, A) est alors sélectionné. Le couple (E, F) est ensuite choisi, puis le couple (D, E).

Le couple suivant permet de mettre en lumière une propriété très forte des CP-nets n'ayant pas encore été discutée : l'interaction entre groupes de variables parentes. Nous pouvons voir sur la Figure 3.2 page 86 que la variable C possède normalement deux variables parentes A et D. Cependant, les différents gains donnés précédemment montrent qu'aucun des deux couples (A, C) et (D, C) ne peut prétendre à être choisi. Le couple (B, C) sera, de loin, privilégié. Cela est dû au fait que les variables parentes A et D doivent être vues comme un ensemble insécable, car prises séparément, elles ne conditionnent pas C. Pour des raisons de calcul combinatoire trop important, nous imposons dans notre méthode un ajout incrémental des variables parentes. Le seul algorithme connu actuellement pouvant ajouter un ensemble de variables parentes est l'algorithme de Guerin *et al.* [GAG13], en bornant la taille maximum du sous-ensemble testé par un hyperparamètre.

Sixième étape

Notre méthode va ajouter, dans l'ordre, les variables B, E, D, puis A. Ainsi, en tenant compte des remarques précédentes, et en comparaison du CP-net cible de la Figure 3.2, des variables parentes inutiles sont ici ajoutées, nous permettant d'obtenir le CP-net illustré dans la Figure 3.6.

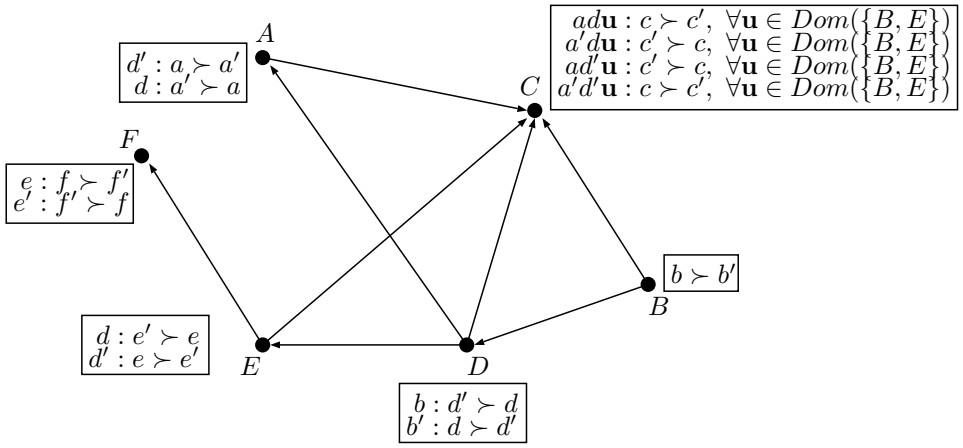


FIGURE 3.6 – CP-net obtenu après exécution de notre procédure sur la base de swaps du Tableau 3.3 page 87.

.

3.2.3 Algorithme formel

La procédure précédente est formellement décrite par l'Algorithme 9 ci-dessous. Il démarre par l'initialisation du CP-net (création des sommets du graphe du CP-net, correspondants aux variables), ainsi que de toutes les préférences des différentes variables. Nous décidons de garder en mémoire le moins d'informations possible, afin d'obtenir une adaptation facile aux flux de données comme nous le verrons dans la

Algorithme 9 : `appr_CPnets_HL`$(\mathbf{V}, S, k)$

Entrées :
$\mathbf{V}$: ensemble de variables ;
S : ensemble de swaps ;
k : nombre maximum de variables parentes ;
Résultat :
$\mathcal{N}_L$: CP-net appris ;

début
  Initialiser $\mathcal{N}_L$;
  pour *chaque variable* $V \in \mathbf{V}$ **faire**
    $CPT(V) \leftarrow CPT(V) \cup \{(\emptyset :\succ_{Dom(V)})\}$;
    si $|S_v| > |S_{v'}|$ **alors** $(\emptyset : v \succ v')$ devient la CP-règle courante;
    sinon $(\emptyset : v' \succ v)$ devient la CP-règle courante;
  $b \leftarrow$ `VRAI`;
  $\mathbf{G} \leftarrow \emptyset$;
  tant que b **faire**
    pour *chaque couple* $(Q, V) \in \mathbf{V}^2$, *avec* $Q \in \overline{Pa}(V)$ **faire**
      si $\hat{G}_{1/2}(S_V, Q) > 0$ **alors**
        $\mathbf{G} \leftarrow \hat{G}_{1/2}(S_V, Q)$ (Équation (3.5));
    si $\mathbf{G} \neq \emptyset$ **alors**
      $b2 \leftarrow$ `TRUE`;
      Trier $\mathbf{G}$ par ordre décroissant;
      tant que $b2$ *et* $\mathbf{G} \neq \emptyset$ **faire**
        Soit $\hat{G}_{1/2}(S_{\hat{V}}, \hat{Q})$ le plus grand gain de $\mathbf{G}$;
        $\mathbf{G} \leftarrow \mathbf{G} \setminus \{\hat{G}_{1/2}(S_{\hat{V}}, \hat{Q})\}$;
        si $\mathcal{N}_L \cup (\hat{Q}, \hat{V})$ *est acyclique et* $|Pa(\hat{V})| < k$ **alors**
          $Pa(\hat{V}) \leftarrow Pa(\hat{V}) \cup \{\hat{Q}\}$;
          $\overline{Pa}(\hat{V}) \leftarrow \overline{Pa}(\hat{V}) \setminus \{\hat{Q}\}$;
          pour *chaque état* $\mathbf{u} \in Pa(\hat{V}) \setminus \{\hat{Q}\}$ **faire**
            Considérons $Dom(\hat{V}) = \{\hat{v}, \hat{v}'\}$, et $Dom(\hat{Q}) = \{\hat{q}, \hat{q}'\}$;
            $CPT(\hat{V}) \leftarrow (CPT(\hat{V}) \setminus \{(\mathbf{u} :\succ_{Dom(\hat{V})})\}) \cup \{(\mathbf{u}\hat{q} :\succ_{Dom(\hat{V})}), (\mathbf{u}\hat{q}' :\succ_{Dom(\hat{V})})\}$;
            si $|S_{\hat{v}}^{\mathbf{u}\hat{q}}| > |S_{\hat{v}'}^{\mathbf{u}\hat{q}}|$ **alors** $(\mathbf{u}\hat{q} : \hat{v} \succ \hat{v}')$ devient la CP-règle courante;
            sinon $(\mathbf{u}\hat{q} : \hat{v}' \succ \hat{v})$ devient la CP-règle courante;
            si $|S_{\hat{v}}^{\mathbf{u}\hat{q}'}| > |S_{\hat{v}'}^{\mathbf{u}\hat{q}'}|$ **alors** $(\mathbf{u}\hat{q}' : \hat{v} \succ \hat{v}')$ devient la CP-règle courante;
            sinon $(\mathbf{u}\hat{q}' : \hat{v}' \succ \hat{v})$ devient la CP-règle courante;
          $b2 \leftarrow$ `FAUX`;
    sinon $b \leftarrow$ `FAUX`;
  Retourner $\mathcal{N}_L$;

section suivante. Ces informations sont directement liées à la structure du CP-net. Ainsi, pour un CP-net appris $\mathcal{N}_L$, et pour chaque schéma de CP-règle du CP-net $(\mathbf{u} :\succ_{Dom(V)})$ (avec $V \in \mathbf{V}$, et $\mathbf{u} \in Dom(Pa(V))$), les informations suivantes sont sélectionnées lors du parcours de l'ensemble de swaps S :

1. Le nombre de swaps qui vérifient la CP-règle $(\mathbf{u} : v \succ v')$, et son inverse $(\mathbf{u} : v' \succ v)$;
2. Le nombre de swaps qui vérifient, pour chaque CP-règle associée $(\mathbf{u} : v \succ v')$ et $(\mathbf{u} : v' \succ v)$, la valeur de chaque variable non parente de V.

Grâce à ces deux informations, nous arrivons à déterminer sans grandes difficultés l'entropie actuelle de chaque schéma de CP-règle (Équation (3.2) page 89). Ainsi :

- L'entropie de la variable est obtenue grâce à une somme pondérée de chacun de ses schémas de CP-règles (Équation (3.1) page 89) ;
- L'entropie d'une variable après l'ajout d'une variable parente fonctionne sur le même principe : pour chaque schéma de CP-règle, l'entropie, pour chacune des valeurs possibles de la variable parente candidate, est déterminée (Équation (3.4) page 91). Il suffit ensuite de sommer chaque entropie en les pondérant (Équation (3.3) page 91).

À chaque itération, l'Algorithme 9 parcourt l'ensemble des swaps de S afin de déterminer le gain d'information (Équation (3.5) page 92) de chaque couple de variable possible (Ligne 12). Ces couples sont ensuite triés et sélectionnés par ordre décroissant afin de privilégier celui qui maximise le gain d'information (Ligne 17). Si le couple sélectionné ne crée pas de cycle dans le CP-net $\mathcal{N}_L$ (Ligne 19), il est alors ajouté à celui-ci, et chaque schéma de CP-règle est divisé en deux nouveaux schémas (en fonction de chacune des deux valeurs de la nouvelle variable parente). La préférence estimée de chaque schéma est finalement déterminée en fonction du nombre de swaps (boucle de la Ligne 22).

Ces opérations sont répétées jusqu'à ce qu'aucune variable parente ne puisse être ajoutée (à cause, par exemple, de la contrainte d'acyclicité [2]), ou encore lorsque toutes les variables sont pures.

Proposition 4.
L'Algorithme 9 est correct, i.e., il retourne toujours un CP-net acyclique (possiblement incomplet), quelle que soit la base de données S.

Démonstration.
Par construction : chaque arc ajouté entre deux variables correspond à un lien de parenté, avec une duplication du nombre de CP-règles de la variable courante en fonction de l'état de l'ensemble de ses variables parentes. De plus, la condition d'acyclicité est vérifiée dès que l'Algorithme 9 ajoute une variable parente (Ligne 19). Enfin, les seuls objets ajoutés dans les CP-tables sont des CP-règles, ce qui vérifie la définition d'un CP-net acyclique, et prouve la correction de l'Algorithme 9. □

2. L'acyclicité du CP-net est testée via un tri topologique [CLRS09] des variables de $\mathbf{V}$ (voir l'Algorithme 7 du Chapitre 2 page 63).

La complexité temporelle de l'Algorithme 9 est donnée par le résultat suivant.

Proposition 5. *Soit S un ensemble de swaps sur n variables, et soit k le nombre maximum de variables parentes autorisées par variable. L'Algorithme 9 admet une complexité temporelle en $O(k^2|S|(2n^2 2^k + n^4))$.*

Démonstration. Pour chaque ajout d'une nouvelle variable parente, il est nécessaire de parcourir entièrement l'ensemble de swaps S. Ainsi, dans le pire des cas, il faut parcourir k^2 fois l'ensemble S. La Ligne 10 permet de calculer les gains d'information de chaque couple de variables, demandant alors de parcourir n^2 fois les 2^k différents schémas de CP-règles de la variable conditionnée candidate. La boucle de la Ligne 16 est effectuée n^2 fois dans le pire des cas, jusqu'à ce qu'un couple ne rendant pas le graphe du CP-net cyclique soit trouvé (l'acyclicité est testée en $O(n^2)$). De plus, la mise à jour de chaque CP-net demande de parcourir les 2^k CP-nets de la variable conditionnée sélectionnée. Nous avons donc $O(k^2|S|(n^2 2^k + n^2(n^2 + 2^k))) = O(k^2|S|(2n^2 2^k + n^4))$. □

3.3 Adaptation aux flux de données

Nous souhaitons, dans cette section, adapter l'algorithme précédent dans le cadre de flux de données. Imaginons par exemple l'utilisation de notre algorithme d'apprentissage sur un site d'e-commerce, où un flux de données constant et potentiellement infini est reçu. À cause de la taille du flux, et du changement permanent des swaps rencontrés, il peut être difficile de parcourir plusieurs fois ces swaps. Il faudra également pouvoir proposer, à tout moment, un CP-net cohérent avec les données déjà observées. Ces deux contraintes soulèvent deux points majeurs dans l'algorithme précédent :

1. **Observation unique de chaque swap** : la version hors ligne présentée parcourt, à chaque itération, l'ensemble de swaps S_V afin de déterminer les nouvelles entropies de la variable V ayant reçue une nouvelle variable parente (avec une multiplication de ses schémas de CP-règles). Il n'est cependant pas possible, dans cette version en ligne, de reparcourir les swaps observés. Il sera donc nécessaire de les estimer ;
2. **Délai d'attente avant l'ajout d'une nouvelle variable parente** : le choix du meilleur couple de variables tenait compte de toute l'information disponible dans la base de données qui demeurait inchangée. Sachant que cette dernière peut être modifiée à tout instant, il faut donc pouvoir prendre une décision sur le moment adéquat pour l'ajout d'une nouvelle variable parente (ni trop tôt pour ne pas avoir à subir une trop grande influence du bruit, ni trop tard pour ne pas modéliser un CP-net ne contenant aucune variable parente). Ce problème sera résolu grâce à l'utilisation d'une borne donnant un intervalle de confiance sur l'ajout d'une variable parente.

3.3.1 Compteurs et leurs estimations

Considérons un flux de données noté $S^{(t)}$ désignant l'ensemble des swaps observés jusqu'à l'instant t. La première étape de l'adaptation de notre version hors ligne aux flux de données concerne l'estimation des différentes valeurs nécessaires au calcul des entropies (Équations (3.1) page 89 et (3.3) page 91). C'est le cas de l'ensemble $S_V^{\mathbf{u}}$ correspondant à l'ensemble des swaps ayant V comme variable de swap, et ayant l'état $\mathbf{u}$ comme valeurs des variables parentes de V. Le cardinal de cet ensemble est recalculé à partir de l'ensemble S_V dès l'ajout d'une nouvelle variable parente. Un moyen d'estimer ce nouvel ensemble à partir des données gardées s'avère nécessaire, car il n'est plus possible de reparcourir les $t-1$ swaps de l'ensemble $S^{(t)}$. Pour effectuer ces différents calculs, nous introduisons des compteurs désignés, comme dans la Figure 3.7, par la lettre m.

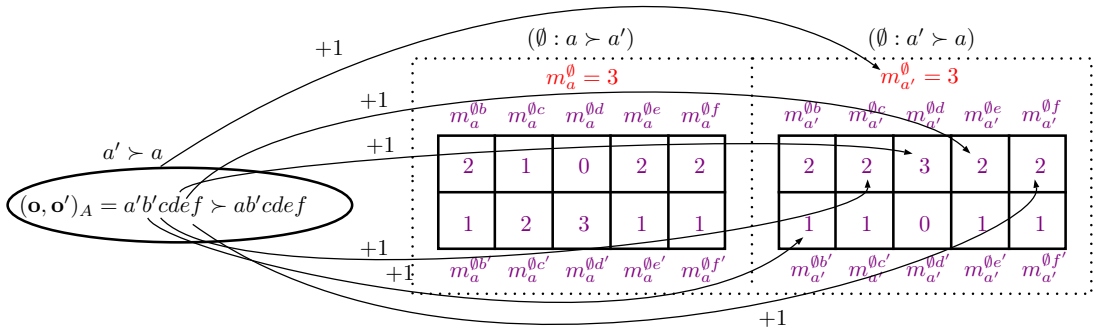


FIGURE 3.7 – Exemple de mise à jour incrémentale des compteurs m basée sur le sixième swap du Tableau 3.3 page 87. Les cinq premiers swaps ont déjà été observés.

Dans cette adaptation en ligne de l'Algorithme 9, deux types d'informations seront extraits de chaque swap observé : la préférence sur les valeurs de la variable de swap [3], ainsi que les valeurs de chaque variable non parente de la variable swap. Ainsi, après avoir observé les t swaps de $S^{(t)}$, nous récupérons, pour chaque schéma de CP-règle de chaque variable :

1. Le nombre de fois où chaque type de CP-règle a été observé, symbolisé par les compteurs rouges de la Figure 3.7 ;
2. Le nombre de fois où chaque variable non parente a été observée, pour chaque type de CP-règle, symbolisé par les compteurs violets de la Figure 3.7.

Cette sauvegarde de l'information sous la forme de compteurs est nécessaire car lors de l'ajout d'une variable parente, il ne sera plus possible de préserver l'ensemble

3. Soit s un swap. La variable de swap correspond à la variable dont la valeur change entre deux objets d'un swap (voir la Définition 1.19 page 33).

de l'information observée jusqu'alors. Les compteurs présentés dans la Figure 3.7 peuvent être généralisés dans le cas où la variable courante V possède des variables parentes. Soit $\mathbf{u} \in Dom(Pa(V))$, et un schéma de CP-règle $(\mathbf{u} :\succ_{Dom(V)})$:

1. Le compteur de la CP-règle $(\mathbf{u} : v \succ v')$ (resp. $(\mathbf{u} : v' \succ v)$) est noté $m_v^{\mathbf{u}}$ (resp. $m_{v'}^{\mathbf{u}}$), et correspond au nombre de swaps observés ayant la variable de swap V avec la préférence $v \succ v'$ (resp. $v' \succ v$), et l'état $\mathbf{u}$ pour l'ensemble de ses variables parentes ;
2. Le compteur de chaque valeur $b \in Dom(B)$ de chaque variable non parente $B \in \overline{Pa}(V)$ de V est noté $m_v^{\mathbf{u}b}$ pour la CP-règle $(\mathbf{u} : v \succ v')$, et correspond au nombre de swaps observés ayant la variable de swap V avec la préférence $v \succ v'$, l'état $\mathbf{u}$ pour l'ensemble de ses variables parentes, et la valeur b pour la variable non parente B.

Lors de l'ajout d'une nouvelle variable parente, il est donc nécessaire d'estimer les nouveaux compteurs avec l'unique information donnée par les compteurs courants. Considérons une nouvelle variable parente Q de V. Les compteurs $m_v^{\mathbf{u}}$ de chaque CP-règle $(\mathbf{u} : v \succ v')$ sont par définition directement remplacés par les compteurs respectifs de la nouvelle variable parente : pour la valeur $q \in Dom(Q)$ de la nouvelle variable parente $Q \in Pa(V)$, le compteur $m_v^{\mathbf{u}}$ de l'ancienne CP-règle $(\mathbf{u} : v \succ v')$ est remplacé par l'ancien compteur $m_v^{\mathbf{u}q}$. Ainsi :

$$\begin{aligned}
&\bullet \text{ Le nouveau compteur de la CP-règle } (\mathbf{u}q : v \succ v') \text{ est } m_v^{\mathbf{u}q},\\
&\bullet \text{ Le nouveau compteur de la CP-règle } (\mathbf{u}q' : v \succ v') \text{ est } m_v^{\mathbf{u}q'},\\
&\bullet \text{ Le nouveau compteur de la CP-règle } (\mathbf{u}q : v' \succ v) \text{ est } m_{v'}^{\mathbf{u}q},\\
&\bullet \text{ Le nouveau compteur de la CP-règle } (\mathbf{u}q' : v' \succ v) \text{ est } m_{v'}^{\mathbf{u}q'}.
\end{aligned} \tag{3.7}$$

L'estimation des nouveaux compteurs des variables non parentes est cependant beaucoup plus difficile à déterminer, car cela demande le calcul du nombre de swaps ayant les mêmes valeurs pour un ensemble de variables données. Plus précisément, leur estimation peut être vue de la manière suivante : considérons un ensemble d'objets (connus) $\mathbf{E}$ et une taille (connue) pour deux sous-ensembles $\mathbf{E_1} \subseteq \mathbf{E}$ et $\mathbf{E_2} \subseteq \mathbf{E}$. Comment estimer la taille de l'intersection de ces deux sous-ensembles $|\mathbf{E_1} \cap \mathbf{E_2}|$? Une telle intersection peut être bornée entre la t-norme de Łukasiewicz [4] [LS08] (borne inférieure) notée $\underline{m}$, et le minimum nilpotent (borne supérieure) noté $\overline{m}$. Dans notre cas, si nous considérons une CP-règle $(\mathbf{u} : v \succ v')$, une nouvelle variable parente Q, et une variable non parente B, alors pour une valeur $q \in Dom(Q)$ de la nouvelle variable parente, et une valeur $b \in Dom(B)$ de la variable non parente B, nous avons :

$$\underline{m}_v^{\mathbf{u}qb} = \max\left(m_v^{\mathbf{u}b} - m_v^{\mathbf{u}} + m_v^{\mathbf{u}q}, 0\right), \tag{3.8}$$

$$\overline{m}_v^{\mathbf{u}qb} = \min\left(m_v^{\mathbf{u}b}, m_v^{\mathbf{u}q}\right). \tag{3.9}$$

4. La t-norme de Łukasiewicz entre deux réels $a, b \in [0, 1]$ est habituellement décrite par $t_L(a, b) = \max(0, a + b - 1)$.

Étant donné que nous souhaitons préserver au maximum les valeurs réelles des compteurs, notre choix s'est donc porté sur la borne inférieure $\underline{m}$ de l'Équation (3.8). Nous obtenons ainsi, pour tout schéma de CP-règle $(\mathbf{u} : v \succ v')$, pour toute nouvelle variable parente Q, et pour toute variable non parente B, l'estimation suivante :

$$\begin{aligned}
m_v^{\mathbf{u}qb} &= \max\left(m_v^{\mathbf{u}b} - m_v^{\mathbf{u}} + m_v^{\mathbf{u}q}, 0\right), \\
m_v^{\mathbf{u}qb'} &= \max\left(m_v^{\mathbf{u}b'} - m_v^{\mathbf{u}} + m_v^{\mathbf{u}q}, 0\right), \\
m_v^{\mathbf{u}q'b} &= \max\left(m_v^{\mathbf{u}b} - m_v^{\mathbf{u}} + m_v^{\mathbf{u}q'}, 0\right), \\
m_v^{\mathbf{u}q'b'} &= \max\left(m_v^{\mathbf{u}b'} - m_v^{\mathbf{u}} + m_v^{\mathbf{u}q'}, 0\right), \\
m_{v'}^{\mathbf{u}qb} &= \max\left(m_{v'}^{\mathbf{u}b} - m_{v'}^{\mathbf{u}} + m_{v'}^{\mathbf{u}q}, 0\right), \\
m_{v'}^{\mathbf{u}qb'} &= \max\left(m_{v'}^{\mathbf{u}b'} - m_{v'}^{\mathbf{u}} + m_{v'}^{\mathbf{u}q}, 0\right), \\
m_{v'}^{\mathbf{u}q'b} &= \max\left(m_{v'}^{\mathbf{u}b} - m_{v'}^{\mathbf{u}} + m_{v'}^{\mathbf{u}q'}, 0\right), \\
m_{v'}^{\mathbf{u}q'b'} &= \max\left(m_{v'}^{\mathbf{u}b'} - m_{v'}^{\mathbf{u}} + m_{v'}^{\mathbf{u}q'}, 0\right).
\end{aligned} \tag{3.10}$$

La Figure 3.8 illustre cette borne.

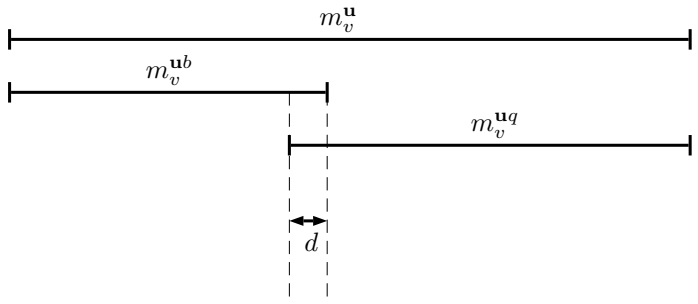


FIGURE 3.8 – Soit $(\mathbf{u} : v \succ v')$ une CP-règle, et soient q et b les valeur d'une nouvelle variable parente et d'une variable non parente, respectivement. La valeur d correspond au chevauchement des compteurs des valeurs q et b dans les swaps observés (Équation (3.10)). Si $m_v^{\mathbf{u}b} + m_v^{\mathbf{u}q} \leq m_v^{\mathbf{u}}$, alors $d = 0$ car il n'est pas possible de déterminer de manière certaine le nombre de valeurs q et de valeurs b simultanées dans la CP-règle $(\mathbf{u} : v \succ v')$.

3.3.2 Redéfinition des différentes entropies

L'estimation des nouveaux compteurs introduite dans la section précédente ne permet plus l'utilisation sous forme ensembliste des différentes valeurs utilisées pour le calcul de l'entropie présentée dans la Section 3.2.1 page 88. Soient $\mathbf{V}$ un ensemble de variable, et $V \in \mathbf{V}$ une variable. Soit $S_V = \{(s_1, y_1), \ldots, ((s_l, y_l)\}$ un ensemble de couples composés d'un swap $s_i \in Dom(\mathbf{V})^2$ ayant V comme variable de swap, ainsi qu'une classe associée $y_i \in Y_V$. On note par m_V le nombre de swaps observés faisant intervenir la variable V, et par $S_V^{(m_V)}$ l'ensemble composé des m_V premiers couples de l'ensemble S_V. Nous redéfinissons alors les différentes entropies, ainsi que le gain d'information à partir de ces compteurs :

1. L'entropie $\hat{H}_{1/2}\left(S_V^{(m_V)}\right)$ associée à une variable $V \in \mathbf{V}$ sera donnée par l'équation suivante

$$\hat{H}_{1/2}\left(S_V^{(m_V)}\right) = \sum_{(\mathbf{u}:\succ_{Dom(V)})\in CPT(V)} \frac{m_V^{\mathbf{u}}}{m_V} \hat{H}_{1/2}\left(S_V^{(m_V)}, (\mathbf{u}:\succ_{Dom(V)})\right), \quad (3.11)$$

avec

$$\hat{H}_{1/2}\left(S_V^{(m_V)}, (\mathbf{u}:\succ_{Dom(V)})\right) = -\sum_{v\in Dom(V)} \frac{m_v^{\mathbf{u}}}{2m_V^{\mathbf{u}}} \ln \frac{m_v^{\mathbf{u}}}{m_V^{\mathbf{u}}}, \quad (3.12)$$

où

- $m_v^{\mathbf{u}}$ correspond au nombre d'observations de la CP-règle $(\mathbf{u} : v \succ v')$;
- $m_V^{\mathbf{u}} = \sum_{v\in Dom(V)} m_v^{\mathbf{u}}$;
- $m_V = \sum_{(\mathbf{u}:\succ_{Dom(V)})\in CPT(V)} m_V^{\mathbf{u}}$.

2. Le calcul de l'entropie $\hat{H}_{1/2}\left(S_V^{(m_V)}, Q\right)$ associée à une variable $V \in \mathbf{V}$, et à une variable parente candidate $Q \in \overline{Pa}(V)$ sera donné par

$$\hat{H}_{1/2}\left(S_V^{(m_V)}, Q\right) = \sum_{q\in Dom(Q)} \sum_{(\mathbf{u}:\succ_{Dom(V)})\in CPT(V)} \frac{m_V^{\mathbf{u}q}}{m_V} \hat{H}_{1/2}\left(S_V^{(m_V)}, (\mathbf{u}:\succ_{Dom(V)}), q\right), \quad (3.13)$$

avec

$$\hat{H}_{1/2}\left(S_V^{(m_V)}, (\mathbf{u}:\succ_{Dom(V)}), q\right) = -\sum_{v\in Dom(V)} \frac{m_v^{\mathbf{u}q}}{2m_V^{\mathbf{u}q}} \ln \frac{m_v^{\mathbf{u}q}}{m_V^{\mathbf{u}q}}, \quad (3.14)$$

où

- $m_v^{\mathbf{u}q}$ correspond au nombre d'observations de la valeur q de la variable non parente Q pour la CP-règle $(\mathbf{u} : v \succ v')$;
- $m_V^{\mathbf{u}q} = \sum_{v\in Dom(V)} m_v^{\mathbf{u}}$.

3. Enfin, le calcul du gain d'information $\hat{G}_{1/2}\left(S_V^{(m_V)}, Q\right)$ associé à une variable $V \in \mathbf{V}$, et à une variable parente candidate $Q \in \overline{Pa}(V)$ correspondra à

$$\hat{G}_{1/2}\left(S_V^{(m_V)}, Q\right) = \hat{H}_{1/2}\left(S_V^{(m_V)}\right) - \hat{H}_{1/2}\left(S_V^{(m_V)}, Q\right). \quad (3.15)$$

3.3.3 Borne de McDiarmid

Rappelons que cette version en ligne fonctionnera sur un flux potentiellement infini de données, dont les swaps futurs ne sont pas connus. Il n'est donc pas facile de savoir à l'avance à quel moment il est judicieux d'ajouter une nouvelle variable parente. Ce problème est résolu pour notre algorithme, par l'utilisation d'une borne portant sur le nombre de swaps nécessaires avant toute prise de décision. Nous nous

sommes d'abord basés sur la borne de Hoeffding [Hoe63] dans [LZM+17], car elle a longtemps été utilisée dans les arbres de décision [DH00], ainsi que dans les algorithmes de bandits manchots [ACF02]. Cependant, à cause de la non linéarité de l'entropie, elle s'est avérée inadaptée pour notre algorithme. Nous lui avons donc préféré la borne de McDiarmid [McD89], qui est présentée comme beaucoup plus adaptée aux problèmes faisant intervenir des mesures non linéaires telles que l'entropie.

Théorème 1 (Inégalité de McDiarmid [McD89])**.** *Si $X_1, \ldots, X_m$ sont m variables aléatoires indépendantes prenant leurs valeurs dans un espace mesurable A, et si $f : A^m \to \mathbb{R}$ est une fonction de $X_1, \ldots, X_m$ satisfaisant, $\forall i,\ \forall x_1, \ldots, x_m, x'_i \in A$,*

$$\sup_{\{x_1,\ldots,x_m,x'_i\}} |f(x_1, \ldots, x_i, \ldots, x_m) - f(x_1, \ldots, x_{i-1}, x'_i, x_{i+1}, \ldots, x_m)| \leq c_i, \quad (3.16)$$

alors, pour tout $\epsilon > 0$,

$$\mathbb{P}(|f(x_1, \ldots, x_m) - \mathbb{E}(f(x_1, \ldots, x_m))| > \epsilon) \leq \exp\left(\frac{-2\epsilon^2}{\sum_{i=1}^{m} c_i^2}\right). \quad (3.17)$$

Notons que dans notre cas, les variables aléatoires correspondent aux m premiers swaps du flux de données S, et f correspond à l'entropie normalisée $H_{1/2}$. Cette borne signifie que si l'entropie, lors de la modification d'un swap, n'excède pas une constante (*i.e.*, elle est bornée), alors il est possible de borner la différence entre l'entropie et son espérance par une valeur ϵ avec une probabilité. Nous obtenons alors le théorème suivant :

Théorème 2. *Soient $V \in \mathbf{V}$ une variable, $X \in \overline{Pa}(V)$ une variable non parente de V, et $S_V^{(m_V)}$ un ensemble composé des m_V premiers swaps d'un ensemble S_V (les swaps sont tirés i.i.d). Considérons $\hat{H}_{1/2}$ une entropie normalisée issue des m_V swaps, et $\hat{G}_{1/2}\left(S_V^{(m_V)}, Q\right) = \left|\hat{H}_{1/2}\left(S_V^{(m_V)}\right) - \hat{H}_{1/2}\left(S_V^{(m_V)}, Q\right)\right|$. Alors pour tout $\delta \in (0, 1]$,*

$$\left|\hat{G}_{1/2}\left(S_V^{(m_V)}, Q\right) - G_{1/2}\left(S_V^{(m_V)}, Q\right)\right| \leq \epsilon(m_V, \delta), \quad (3.18)$$

$$\text{où } \epsilon(m_V, \delta) = \ln(m_V)\sqrt{\frac{2}{m_V}\ln\frac{4}{\delta}} + \frac{2}{m_V}, \quad (3.19)$$

avec une probabilité d'au moins $1 - \delta$ sur l'ensemble des m swaps.

Démonstration. Soit $\delta = \exp\left(\frac{-2\epsilon^2}{\sum_{i=1}^{m_V} c_i^2}\right)$, alors il a été montré dans [AK01] (remarque item (iii), Corollaire 1) et [RC17] (preuve Théorème 1) que pour une entropie normalisée et des variables binaires :

$$\left|\hat{H}_{1/2}\left(S_V^{(m_V)}\right) - \mathbb{E}\left(\hat{H}_{1/2}\left(S_V^{(m_V)}\right)\right)\right| \leq \frac{\ln(m_V)}{2}\sqrt{\frac{2}{m_V}\ln\frac{4}{\delta}}, \quad (3.20)$$

$$\left|\hat{H}_{1/2}\left(S_V^{(m_V)}, Q\right) - \mathbb{E}\left(\hat{H}_{1/2}\left(S_V^{(m_V)}, Q\right)\right)\right| \leq \frac{\ln(m_V)}{2}\sqrt{\frac{2}{m_V}\ln\frac{4}{\delta}}. \quad (3.21)$$

Ce résultat permet de borner la déviation $|\hat{H}_{1/2} - \mathbb{E}(\hat{H}_{1/2})|$. Cette déviation n'est cependant pas suffisant : il existe également un biais $|\mathbb{E}(\hat{H}_{1/2}) - H_{1/2}|$ entre l'entropie théorique réelle $H_{1/2}$, et l'espérance de l'entropie empirique $\mathbb{E}(\hat{H}_{1/2})$. De Rosa *et al.* proposent alors dans [RC17] (preuve Théorème 1) de prendre en compte ce biais en se basant sur les résultats de [Pan03] (Proposition 1) :

$$-\ln\left(1+\frac{N-1}{m_V}\right) \leq \mathbb{E}(\hat{H}_{1/2}) - H_{1/2} \leq 0. \tag{3.22}$$

N correspondant aux différentes valeurs que peuvent prendre les variables, nous avons $N = 2$ pour l'entropie $H_{1/2}\left(S_V^{(m_V)}\right)$, et $N = 4$ pour l'entropie $H_{1/2}\left(S_V^{(m_V)}, Q\right)$. En utilisant la propriété $-a \leq -\ln(1+a)$, nous obtenons :

$$\left|H_{1/2}\left(S_V^{(m_V)}\right) - \mathbb{E}\left(\hat{H}_{1/2}\left(S_V^{(m_V)}\right)\right)\right| \leq \frac{1}{2m_V}, \tag{3.23}$$

$$\left|H_{1/2}\left(S_V^{(m_V)}, Q\right) - \mathbb{E}\left(\hat{H}_{1/2}\left(S_V^{(m_V)}, Q\right)\right)\right| \leq \frac{3}{2m_V}. \tag{3.24}$$

□

Ce théorème nous donne alors un seuil, représenté par ϵ, autorisant l'ajout d'une variable parente si ce seuil est dépassé par $\hat{G}_{1/2}$. Une conséquence immédiate de ce théorème est le fait que si l'on sélectionne le couple de variables $(\hat{Q}, \hat{V})$ qui maximise la fonction $\hat{G}_{1/2}$, ainsi que le couple (Q', V') ayant la deuxième plus grande valeur de $\hat{G}_{1/2}$, et que $\hat{G}_{1/2}\left(S_{\hat{V}}^{(m_{\hat{V}})}, \hat{Q}\right) - \hat{G}_{1/2}\left(S_{V'}^{(m_{V'})}, Q'\right) > (\epsilon(m_{\hat{V}}, \delta) + \epsilon(m_{V'}, \delta))$ (avec $m_{\hat{V}}$ et $m_{V'}$ le nombre d'observations des variables $\hat{V}$ et V', respectivement), alors le couple $(\hat{Q}, \hat{V})$ constitue le meilleur choix avec une probabilité $1 - \delta$ (voir la Figure 3.9) [5].

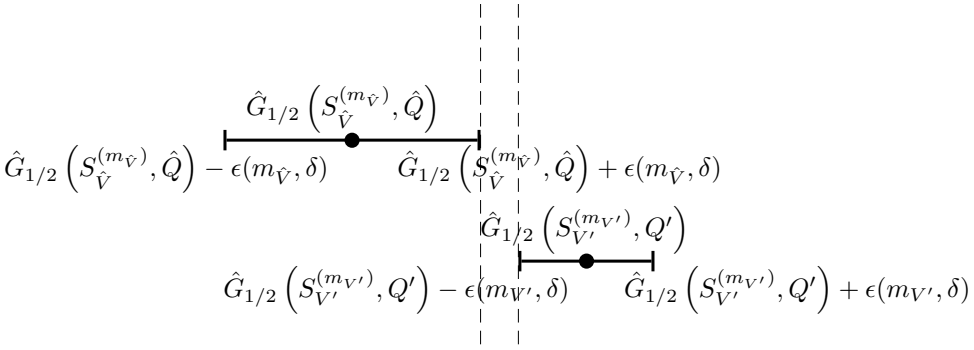


FIGURE 3.9 – Illustration de la borne de McDiarmid. La condition $\hat{G}_{1/2}\left(S_{\hat{V}}^{(m_{\hat{V}})}, \hat{Q}\right) - \hat{G}_{1/2}\left(S_{V'}^{(m_{V'})}, Q'\right) > (\epsilon(m_V, \delta) + \epsilon(m_{V'}, \delta))$ garantit que les intervalles de confiance pour les vrais gains $G_{1/2}\left(S_{\hat{V}}^{(m_{\hat{V}})}, \hat{Q}\right)$ et $G_{1/2}\left(S_{V'}^{(m_{V'})}, Q'\right)$ ne se chevauchent pas.

5. Dans le cas où la variable $m_{V'} = 0$, alors $\epsilon(m_{V'}, \delta) = 0$ par convention.

Notons que ce résultat diffère des résultats énoncés dans [RC15, RC17] à cause du fait qu'ici, nous travaillons simultanément sur plusieurs arbres de décision, en comparant leurs différentes entropies. Deux arbres de décision induits par deux variables V et V' font intervenir deux sous-ensembles distincts de S : S_V et $S_{V'}$, dont les cardinaux, décrits dans le Théorème 2 page 104 par la valeur m, ne correspondent pas. Cela implique donc deux valeurs distinctes de ϵ.

L'utilisation de la borne de McDiarmid a l'avantage de pouvoir, d'une certaine manière, « corriger » l'estimation effectuée sur les compteurs lors de l'ajout de nouvelles variables parentes (voir la Section 3.3.1 page 100). En pratique, le paramètre δ du Théorème 2 est fixé par l'utilisateur (nous appelons un tel paramètre un **hyperparamètre**). Cela lui permet de faire varier la borne de McDiarmid en fonction du degré de confiance qu'il souhaite accorder aux choix de la procédure d'apprentissage. Cela signifie que la seule variable restante est le nombre de swaps m de la variable conditionnée candidate. Ainsi, plus le nombre de swaps observés est important, plus la valeur de ϵ décroît, jusqu'au moment la différence des deux gains d'information dépassera celle de ϵ. Considérons $\hat{m}$ comme étant le nombre de swaps minimum requis pour autoriser l'ajout d'une nouvelle variable parente, pour une valeur δ donnée. Alors, l'erreur, engendrée par l'estimation faite après chaque ajout d'une variable parente, est corrigée grâce à l'observation des (au minimum) $\hat{m}$ swaps séparant les différents ajouts de variables parentes (voir la Figure 3.10).

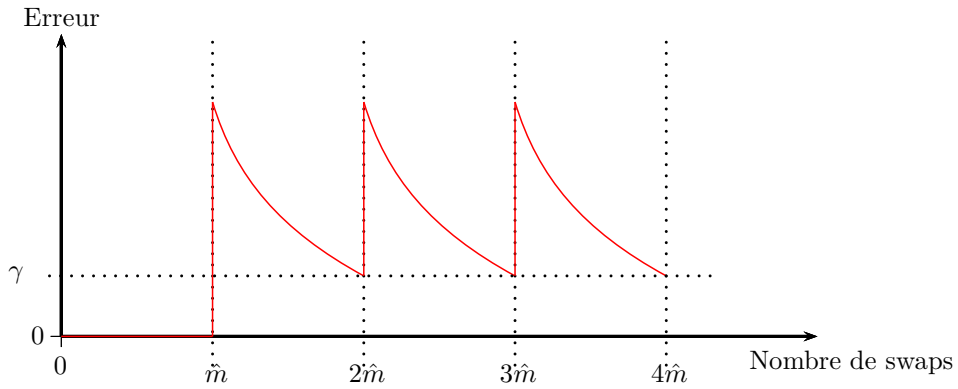


FIGURE 3.10 – Illustration de l'erreur entre les compteurs estimés et les compteurs réels. $\hat{m}$ représente le nombre de swaps minimum requis par la borne de McDiarmid pour ajouter une nouvelle variable parente. Nous supposons ici un ajout immédiat d'une variable parente après l'observation de $\hat{m}$ swaps, nous permettant de réajuster l'estimation des swaps avec une erreur par rapport aux compteurs réels au plus égale à γ.

Nous verrons dans les sections suivantes qu'en pratique, cette valeur minimale n'est pas la contrainte bloquante à l'ajout d'une nouvelle variable parente, car il est assez rare, expérimentalement, d'obtenir une différence entre gains d'information suffisamment élevée pour ajouter une nouvelle variable parente dès que $\hat{m}$ swaps ont été observés.

3.3.4 Exemple déroulant

Soient $\mathbf{V}$ un ensemble de variables, et S un flux de données. L'adaptation en ligne de l'Algorithme 9 ayant un fonctionnement légèrement différent, nous le déroulons sur l'exemple donné par le Tableau 3.3 page 87. Le principe de la version en ligne proposée est le suivant :

1. Observer un nouveau swap $(\mathbf{o}, \mathbf{o}')_V$ et mettre à jour les différents compteurs ;
2. Calculer l'entropie de la variable V ;
3. Calculer le gain d'information des couples $(Q, V),\ \forall Q \in \overline{Pa}(V)$;
4. Trier les gains d'information de tous les couples $(Q, V),\ \forall V \in \mathbf{V},\ \forall Q \in \overline{Pa}(V)$;
5. Sélectionner le couple $(\hat{Q}, \hat{V})$ qui maximise ce gain, et le couple (Q', V') ayant le deuxième meilleur gain ;
6. Si la différence entre les gains des deux couples $(\hat{Q}, \hat{V})$ et (Q', V') dépasse le seuil de la borne de McDiarmid, et que le couple $(\hat{Q}, \hat{V})$ ne crée pas de cycle, alors ajouter l'arc $(\hat{Q}, \hat{V})$ dans le graphe du CP-net ;
7. Si l'arc $(\hat{Q}, \hat{V})$ est ajouté au CP-net, alors mettre à jour les différents compteurs de la variable V. Sinon, ne rien faire ;
8. Recommencer à la première étape.

Comme nous avons pu le voir précédemment, la borne de McDiarmid oblige l'algorithme à observer un certain nombre d'exemples avant d'autoriser l'ajout de variables parentes. Ce nombre d'exemples varie en fonction du degré de confiance que l'on souhaite accorder, et qui est symbolisé par l'hyperparamètre δ. L'exemple du Tableau 3.3 page 87 n'ayant pas assez d'exemples pour dépasser le seuil de cette borne, nous décidons de multiplier par 1 300 chaque exemple (les 1 300 premières préférences seront les mêmes $abcd'ef \succ a'bcd'ef$, puis les 1 300 suivantes seront $abc'd'ef \succ a'bc'd'ef$, et ainsi de suite, jusqu'à obtenir un total de 65 000 swaps), et de fixer $\delta = 1$. Nous supposons également avoir déjà observé les 11 699 premiers swaps, tous associés à la variable A.

Première étape

Nous observons un nouveau swap impliquant la préférence $a'bcde'f' \succ abcde'f'$. Il s'agit du dernier swap de la variable A présent dans le flux de données. Nous obtenons alors une valeur sur chaque compteur décrite dans la Figure 3.11.

Deuxième étape

On note par $S_A^{(11700)}$ l'ensemble contenant les 11 700 premiers swaps de l'ensemble S_A. L'entropie de la variable A est alors calculée à partir des valeurs de ses différents compteurs :

$$\hat{H}_{1/2}\left(S_A^{(11700)}\right) = -\frac{3900}{23400}\ln\frac{3900}{11700} - \frac{7800}{23400}\ln\frac{7800}{11700} = 0.318.$$

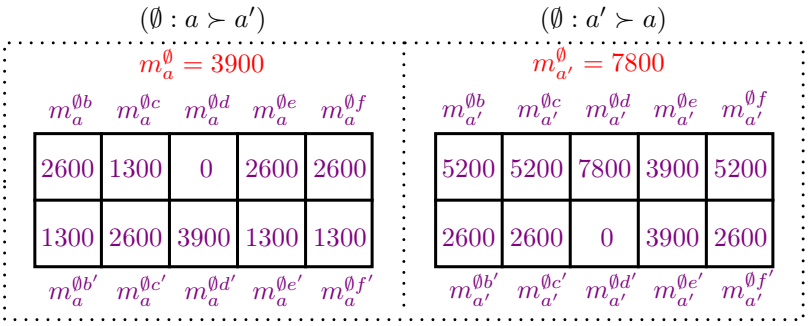


FIGURE 3.11 – Valeurs des différents compteurs de la variable A après l'observation des 11 700 premiers swaps du Tableau 3.3 page 87 (chaque swap ayant été observé 1 300 fois).

Notons, pour informations, que l'entropie des variables B, C, D, E, et F est égale à 0, car seuls les swaps faisant intervenir la variable A ont, pour le moment, été observés.

Troisième étape

Le gain d'information de chaque couple faisant intervenir la variable A est alors calculé :

- $\hat{G}_{1/2}\left(S_A^{(11700)}, B\right) = 0.318 - \frac{7800}{11700}\left(-\frac{2600}{15600}\ln\frac{2600}{7800} - \frac{5200}{15600}\ln\frac{5200}{7800}\right) - \frac{3900}{11700}\left(-\frac{1300}{7800}\ln\frac{1300}{3900} - \frac{2600}{7800}\ln\frac{2600}{3900}\right) = 0\,;$
- $\hat{G}_{1/2}\left(S_A^{(11700)}, C\right) = 0.318 - \frac{6500}{11700}\left(-\frac{1300}{13000}\ln\frac{1300}{6500} - \frac{5200}{13000}\ln\frac{5200}{6500}\right) - \frac{5200}{11700}\left(-\frac{2600}{10400}\ln\frac{2600}{5200} - \frac{2600}{10400}\ln\frac{2600}{5200}\right) = 0.025\,;$
- $\hat{G}_{1/2}\left(S_A^{(11700)}, D\right) = 0.318 - 0 = 0.318\,;$
- $\hat{G}_{1/2}\left(S_A^{(11700)}, E\right) = 0.318 - \frac{6500}{11700}\left(-\frac{2600}{13000}\ln\frac{2600}{6500} - \frac{3900}{13000}\ln\frac{3900}{6500}\right) - \frac{5200}{11700}\left(-\frac{1300}{10400}\ln\frac{1300}{5200} - \frac{3900}{10400}\ln\frac{3900}{5200}\right) = 0.006\,;$
- $\hat{G}_{1/2}\left(S_A^{(11700)}, F\right) = 0.318 - \frac{7800}{11700}\left(-\frac{2600}{15600}\ln\frac{2600}{7800} - \frac{5200}{15600}\ln\frac{5200}{7800}\right) - \frac{3900}{11700}\left(-\frac{1300}{7800}\ln\frac{1300}{3900} - \frac{2600}{7800}\ln\frac{2600}{3900}\right) = 0.$

Les gains de tous les autres couples sont égaux à 0 car aucun swap faisant intervenir les autres variables n'a encore été observé, *i.e.*, $\hat{G}_{1/2}\left(S_V^{(11700)}, Q)\right) = 0,\ \forall V \in \{B, C, D, E, F\},\ \forall Q \in \mathbf{V} \setminus \{V\}$.

Quatrième, cinquième, et sixième étape

Les gains d'information sont alors triés par ordre décroissant, et les deux premiers sont sélectionnés. Seuls certains gains de la variable A sont non nuls. Ainsi, le premier gain sélectionné est $\hat{G}_{1/2}\left(S_A^{(11700)}, D\right)$, et le deuxième gain est $\hat{G}_{1/2}\left(S_A^{(11700)}, C\right)$. Les deux gains sont ensuite comparés dans le but de savoir si l'arc (D, A) peut être

ajouté au CP-net. Nous utilisons pour cela le Théorème 2, en calculant la valeur de $\epsilon(m_A, \delta)$:

$$\epsilon(11700, 1) = \ln(11700)\sqrt{\frac{2}{11700}\ln\frac{4}{1}} + \frac{2}{11700} \approx 0.144.$$

La valeur de ϵ est ensuite comparée avec les gains d'information sélectionnés :

$$\begin{array}{ll} & \hat{G}_{1/2}\left(S_A^{(11700)}, D\right) - \hat{G}_{1/2}\left(S_A^{(11700)}, C\right) > 2\epsilon(m_A, \delta) \\ \Leftrightarrow & 0.318 - 0.025 > 0.288 \\ \Leftrightarrow & 0.293 > 0.288. \end{array}$$

Le seuil modélisé par ϵ est donc dépassé. De plus, l'ajout de l'arc (D, A) ne crée pas de cycles dans le CP-net. La variable D devient donc la nouvelle variable parente de A.

Septième étape

La variable D étant une nouvelle variable parente de A, il faut maintenant mettre à jour les différents compteurs de cette dernière variable. Deux types de compteurs doivent être mis à jour : les compteurs sur chaque CP-règle, et les compteurs sur chaque variable non parente de chaque CP-règle.
La mise à jour du premier type de compteur est la suivante :

$$\begin{array}{rcl} m_a^d & = & m_a^{\emptyset d}, \\ m_a^{d'} & = & m_a^{\emptyset d'}, \\ m_{a'}^d & = & m_{a'}^{\emptyset d}, \\ m_{a'}^{d'} & = & m_{a'}^{\emptyset d'}. \end{array}$$

La mise à jour du deuxième type de compteur est effectuée à l'aide de l'Équation (3.10) (voir la Section 3.3.1 page 100). Nous détaillons ici les résultats de cette équation sur la variable B :

$$\begin{array}{rcl} m_a^{db} & = & \max\left(m_a^{\emptyset b} - m_a^{\emptyset} + m_a^{\emptyset d}, 0\right) = \max(2600 - 3900 + 0, 0) = 0, \\ m_a^{db'} & = & \max\left(m_a^{\emptyset b'} - m_a^{\emptyset} + m_a^{\emptyset d}, 0\right) = \max(1300 - 3900 + 0, 0) = 0, \\ m_a^{d'b} & = & \max\left(m_a^{\emptyset b} - m_a^{\emptyset} + m_a^{\emptyset d'}, 0\right) = \max(2600 - 3900 + 3900, 0) = 2600, \\ m_a^{d'b'} & = & \max\left(m_a^{\emptyset b'} - m_a^{\emptyset} + m_a^{\emptyset d'}, 0\right) = \max(1300 - 3900 + 3900, 0) = 1300, \\ m_{a'}^{db} & = & \max\left(m_{a'}^{\emptyset b} - m_{a'}^{\emptyset} + m_{a'}^{\emptyset d}, 0\right) = \max(5200 - 7800 + 7800, 0) = 5200, \\ m_{a'}^{db'} & = & \max\left(m_{a'}^{\emptyset b'} - m_{a'}^{\emptyset} + m_{a'}^{\emptyset d}, 0\right) = \max(2600 - 7800 + 7800, 0) = 2600, \\ m_{a'}^{d'b} & = & \max\left(m_{a'}^{\emptyset b} - m_{a'}^{\emptyset} + m_{a'}^{\emptyset d'}, 0\right) = \max(5200 - 7800 + 0, 0) = 0, \\ m_{a'}^{d'b'} & = & \max\left(m_{a'}^{\emptyset b'} - m_{a'}^{\emptyset} + m_{a'}^{\emptyset d'}, 0\right) = \max(2600 - 7800 + 0, 0) = 0. \end{array}$$

La Figure 3.12 présente les résultats des mises à jours de l'ensemble des compteurs de la variable A.

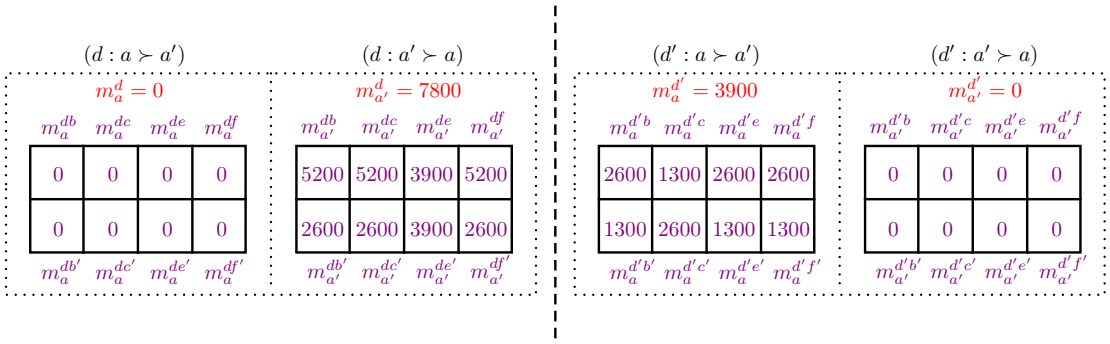


FIGURE 3.12 – Résultats de la mise à jour de tous les compteurs de la variable A après l'ajout de D comme variable parente de A.

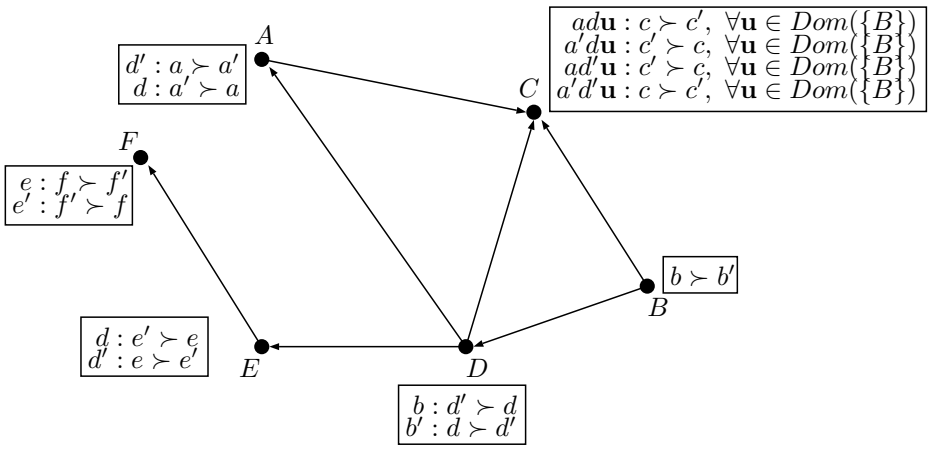


FIGURE 3.13 – CP-net obtenu après exécution de notre version en ligne sur la base de swaps du Tableau 3.3 page 87 (chaque swap apparait 1 300 fois).

.

Huitième étape

Le processus est répété jusqu'à avoir observé les 65 000 swaps disponibles, nous permettant d'obtenir le CP-net exposé dans la Figure 3.13. Les arcs ont été rajoutés dans l'ordre suivant : (D, A), (A, C), (B, C), (D, C), (B, D), (D, E), (E, F). Ceci est dû au fait que l'algorithme observe des swaps faisant intervenir les mêmes variables successives. Ainsi, un ordre d'apparition des swaps différent aurait entrainé la génération d'un CP-net n'ayant pas forcément les mêmes arcs. La borne de McDiarmid nous assure cependant une stabilité sur l'ajout des différents arcs, car elle oblige l'algorithme à observer un grand nombre de swaps avant de l'autoriser à prendre une décision. Nous observons par ailleurs un CP-net très proche de celui obtenu avec la version hors ligne sur le même ensemble de préférences (voir la Figure 3.6 page 96). Ce CP-net ne possède effectivement pas l'arc (E, C) le rendant ainsi plus proche du CP-net cible décrit dans la Figure 3.2 page 86. Cela peut constituer un avantage, car l'attente causée par la borne de McDiarmid permet, dans une certaine mesure,

d'éviter l'ajout de variables parentes inutiles dans le CP-net.

3.3.5 Algorithme formel

L'Algorithme 10 ci-dessous formalise la procédure d'apprentissage en ligne complète. Rappelons que dans cette version, S est un flux de préférences potentiellement infini. L'algorithme initialise d'abord le CP-net $\mathcal{N}_L$ (création des différentes variables de l'ensemble $\mathbf{V}$, et des compteurs pour chaque CP-règle de base $v \succ v'$ et $v' \succ v$, $\forall V \in \mathbf{V}$). Une itération sur cette version en ligne correspond à la prise en compte d'un unique swap observé (boucle de la Ligne 3), ce qui permet de récupérer, entre chaque itération, le CP-net courant. Ainsi, pour chaque swap observé, l'Algorithme 10 déduit la CP-règle correspondante dans le CP-net courant $\mathcal{N}_L$, puis met à jour les compteurs appropriés par rapport au swap observé (Figure 3.7 page 100). Le gain d'information de chaque couple de variables possible est ensuite calculé (Ligne 9) en utilisant les entropies définies dans la Section 3.3.2 page 102. Les deux couples maximisant ce gain sont alors récupérés, et la borne ϵ est calculée. Soit (Q_1, V_1) le couple de variables maximisant le gain d'information, la Ligne 12 expose les différentes conditions à respecter afin d'ajouter une nouvelle variable parente (les trois points suivants doivent être vérifiés) :

1. $|Pa(V_1)| < k$: la variable V ne peut pas avoir plus de k variables parentes ;
2. $\mathcal{N}_L \cup (Q_1, V_1)$ est acyclique[6] : le graphe du CP-net $\mathcal{N}_L$ doit rester acyclique après l'ajout du couple (Q_1, V_1) ;
3. L'une de ces deux inéquations doit être correcte :
   - $\hat{G}_{1/2}\left(S_{V_1}^{(m_{V_1})}, Q_1\right) - \hat{G}_{1/2}\left(S_{V_2}^{(m_{V_2})}, Q_2\right) > (\epsilon(m_{V_1}, \delta) + \epsilon(m_{V_2}, \delta))$: les intervalles de confiance de chaque couple (Q_1, V_1) et (Q_2, V_2) ne doivent pas s'intercaler (voir la Figure 3.9 page 105),
   - $\epsilon(m_{V_1}, \delta) + \epsilon(m_{V_2}, \delta) < \tau$: dans le cas où ils se chevauchent, un nombre suffisant de swaps exemples doit être observé pour assurer un ajout.

Cette dernière condition, représentée par l'inéquation $\epsilon(m_{V_1}, \delta) + \epsilon(m_{V_2}, \delta) < \tau$ (Ligne 12), rend possible la prise en compte de deux couples ayant un gain d'information important, et qui croient parallèlement. Cela permet alors une prise de décision bloquée par un écart trop faible du gain entre ces deux couples.

Si les trois conditions précédentes sont respectées, alors la variable Q_1 du couple (Q_1, V_1) devient la nouvelle variable parente de V_1, chaque schéma de CP-règles de V_1 est divisé, et les valeurs des nouveaux compteurs sont recalculées (Ligne 18), permettant la détermination des nouvelles CP-règles courantes. Dans le cas où les conditions d'ajout d'une variable parente ne sont pas réunies, alors seules les CP-règles actives du schéma courant sont déterminées grâce à la mise à jour des compteurs (Ligne 23).

6. Tout comme la version hors ligne, l'acyclicité du CP-net est testée via un tri topologique [CLRS09] des variables de $\mathbf{V}$ avec l'Algorithme 7 du Chapitre 2 page 63.

Algorithme 10 : `appr_CPnets_EL`$(\mathbf{V}, S, k, \delta, \tau)$

Entrées :
$\mathbf{V}$: ensemble de variables ;
S : flux de swaps ;
k : nombre maximum de variables parentes ;
δ : intervalle de confiance ;
τ : seuil sur la borne de McDiarmid ;
Résultat :
$\mathcal{N}_L$: CP-net appris ;

début
  Initialiser $\mathcal{N}_L$ et $CPT(V)$, $\forall V \in \mathbf{V}$;
  pour *chaque* $(\mathbf{o}, \mathbf{o}')_V \in S$ **faire**
    $\mathbf{u} \leftarrow \mathbf{o}[Pa(V)]$;
    Mettre à jour les différents compteurs du schéma $(\mathbf{u} :\succ_{Dom(V)})$;
    $\mathbf{G} \leftarrow \emptyset$;
    pour *chaque couple* $(Q', V') \in \mathbf{V}^2$, *avec* $Q' \in \overline{Pa}(V')$ **faire**
      si $\hat{G}_{1/2}\left(S_{V'}^{(m_{V'})}, Q'\right) > 0$ **alors**
        $\mathbf{G} \leftarrow \hat{G}_{1/2}\left(S_{V'}^{(m_{V'})}, Q'\right)$ (Équation (3.5));
    Soient $\hat{G}\left(S_{V_1}^{(m_{V_1})}, Q_1\right) \in \mathbf{G}$ et $\hat{G}\left(S_{V_2}^{(m_{V_2})}, Q_2\right) \in \mathbf{G}$ le plus grand gain d'information et celui arrivant en deuxième position, respectivement;
    Calculer $\epsilon(m_{V_1}, \delta)$ et $\epsilon(m_{V_2}, \delta)$ grâce au Théorème 2 page 104;
    si *(*$|Pa(V_1)| < k$*) et (*$\mathcal{N}_L \cup (Q_1, V_1)$ *est acyclique) et (*$\hat{G}_{1/2}\left(S_{V_1}^{(m_{V_1})}, Q_1\right) - \hat{G}_{1/2}\left(S_{V_2}^{(m_{V_2})}, Q_2\right) > (\epsilon(m_{V_1}, \delta) + \epsilon(m_{V_2}, \delta))$ *ou* $\epsilon(m_{V_1}, \delta) + \epsilon(m_{V_2}, \delta) < \tau$*)* **alors**
      $Pa(V_1) \leftarrow Pa(V_1) \cup \{Q_1\}$;
      $\overline{Pa}(V_1) \leftarrow \overline{Pa}(V_1) \setminus \{Q_1\}$;
      pour *chaque état* $\mathbf{u}' \in Pa(V_1) \setminus \{Q_1\}$ **faire**
        Considérons $Dom(V_1) = \{v_1, v_1'\}$, et $Dom(Q_1) = \{q_1, q_1'\}$;
        $CPT(V_1) \leftarrow (CPT(V_1) \setminus \{(\mathbf{u} :\succ_{Dom(V_1)})\}) \cup \{(\mathbf{u}q_1 :\succ_{Dom(V_1)}), (\mathbf{u}q_1' :\succ_{Dom(V_1)})\}$;
        Mettre à jour chaque compteur à l'aide des Équations (3.7) et (3.10);
        si $m_{v_1}^{\mathbf{u}'q_1} > m_{v_1'}^{\mathbf{u}'q_1}$ **alors** $(\mathbf{u}'q_1 : v_1 \succ v_1')$ devient la CP-règle courante;
        sinon $(\mathbf{u}'q_1 : v_1' \succ v_1)$ devient la CP-règle courante;
        si $m_{v_1}^{\mathbf{u}'q_1'} > m_{v_1'}^{\mathbf{u}'q_1'}$ **alors** $(\mathbf{u}'q_1' : v_1 \succ v_1')$ devient la CP-règle courante;
        sinon $(\mathbf{u}'q_1' : v_1' \succ v_1)$ devient la CP-règle courante;
    sinon
      si $m_{v_1}^{\mathbf{u}} > m_{v_1'}^{\mathbf{u}}$ **alors** $(\mathbf{u} : v_1 \succ v_1')$ devient la CP-règle courante;
      sinon $(\mathbf{u} : v_1' \succ v_1)$ devient la CP-règle courante;

De façon similaire à la Proposition 4 de l'Algorithme 9, l'Algorithme 10 génère à tout moment un CP-net acyclique. La proposition suivante énonce la complexité temporelle de cette version en ligne.

Proposition 6. *Soit S un ensemble de swaps sur n variables, et soit k le nombre maximum de variables parentes autorisées par variable. On appelle* ***itération*** *une exécution de la boucle* ***pour*** *de la Ligne 3. L'Algorithme 10 admet une complexité temporelle en $O(2^k(n^2+4(n-k)+2)+n^2)$ pour chacune de ses itérations, et une complexité temporelle totale en $O(|S|(2^k(n^2+4(n-k)+2)+n^2))$.*

Démonstration. L'ensemble de swaps S étant ici considéré comme un flux de données potentiellement infini, il parait plus cohérent d'analyser la complexité d'une seule itération de l'Algorithme 10. Ainsi, la première étape consiste à calculer le gain d'information de chaque couple de variables possible (Ligne 7). Il est pour cela nécessaire de parcourir, pour chacun des n^2 couples possibles, les 2^k schémas de CP-règles de la variable conditionnée candidate. Le calcul de ϵ pour la borne de McDiarmid est immédiat (Ligne 12), il suffit de vérifier l'acyclicité du CP-net, ce qui est effectué en n^2, avec n le nombre de sommets (*i.e.*, le nombre de variables). La deuxième et dernière étape importante consiste à mettre à jour chaque schéma. Rappelons qu'il est nécessaire, dans la version en ligne, de mettre à jour des compteurs. Ces compteurs sont de deux types : les compteurs propres aux CP-nets, et les compteurs propres aux variables non parentes de chaque schéma de CP-règle. Ainsi, pour chacun des 2^k schémas de CP-règles, il est nécessaire de mettre à jour $4(n-k)$ compteurs de variables non parentes, et 2 compteurs pour chaque type de schéma. Cela donne une complexité totale, pour une itération, en $O(2^k(n^2+4(n-k)+2)+n^2)$. Lorsque cette itération est exécutée sur l'ensemble des swaps de S, L'Algorithme 10 admet alors une complexité en $O(|S|(2^k(n^2+4(n-k)+2)+n^2))$. □

3.4 Apprentissage de CP-nets cycliques

Rappelons que l'objectif des versions hors ligne et en ligne de notre algorithme est d'apprendre un CP-net acyclique minimisant la perte logarithmique sur l'ensemble des n arbres de décision (un par variable), comme décrit par l'Équation (1.24) page 54. La contrainte d'acyclicité du CP-net nous oblige alors à faire des choix, en privilégiant, par exemple, à chaque itération, l'arbre (*i.e.*, la variable) maximisant le gain d'information, au détriment des autres arbres (*i.e.*, des autres variables). Notre choix se porte donc sur l'arbre ayant le gain d'information maximum, sans normalisation de ce gain afin de tenir compte du nombre d'observations de chaque variable. Ce choix est motivé par le fait que nous supposons une distribution uniforme des swaps de la base de données S.

Nous pouvons remarquer que ce choix n'a plus lieu d'être dans le cas où la contrainte d'acyclicité du CP-net est relaxée, car il n'est alors plus nécessaire de privilégier un arbre en particulier par rapport à un autre. Rappelons que pour l'apprentissage, l'ensemble de swaps S est partitionné en n sous-ensembles S_V, où chaque

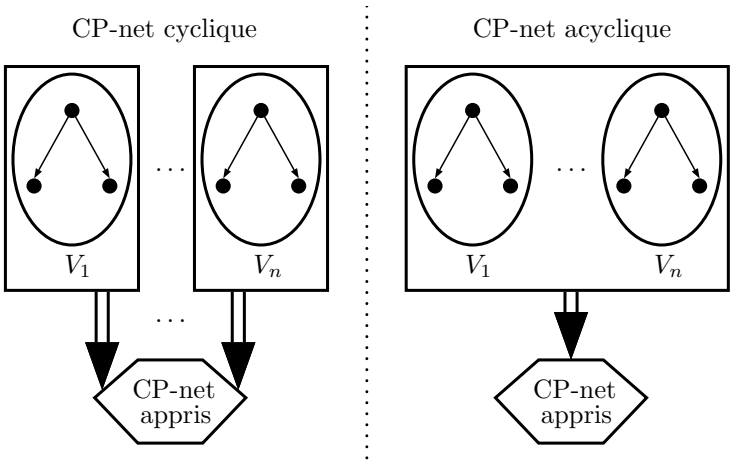


FIGURE 3.14 – Différence entre l'apprentissage d'un CP-net cyclique, et l'apprentissage d'un CP-net acyclique.

swap $s \in S_V$ possède la variable V comme variable de swap. Ainsi, l'apprentissage de l'arbre associé à la variable $V \in \mathbf{V}$ exploitera uniquement le sous-ensemble S_V, et peut donc être effectué indépendamment des autres arbres. *A contrario*, l'apprentissage d'un CP-net acyclique requiert une comparaison entre les différents arbres afin de s'assurer que l'arc qui sera rajouté au CP-net est bien celui maximisant la mesure de perte, car son ajout rendra impossible l'ajout de certaines autres variables parentes. De ce fait, l'apprentissage d'un CP-net cyclique revient à apprendre, de manière indépendante, et en parallèle, n arbres de décisions, en cherchant à minimiser, pour chaque arbre, l'Équation (1.21) page 53. Nous illustrons ces différences dans la Figure 3.14.

Nous avons vu dans le Chapitre 1 qu'un CP-net cyclique peut entraîner des cycles au sein de son ordre partiel des objets (voir la Section 1.4.1.3 page 32), créant alors des incohérences au sein des préférences, et rendant un certain nombre de problèmes difficiles à résoudre (comme la détermination de la préférence entre deux objets qui ne sont pas *ceteris paribus*). D'un point de vue calculatoire cependant, du fait de la parallélisation d'un tel apprentissage, il devient plus rapide d'apprendre un CP-net cyclique qu'un CP-net acyclique, du fait qu'aucune comparaison entre les différents arbres ne soit nécessaire.

3.5 Résultats expérimentaux

Nous étudions dans cette section l'efficacité des deux versions, hors ligne et en ligne, de l'algorithme proposé à travers des expérimentations sur des données synthétiques et réelles. Nous commencerons par une série de tests issus de données synthétiques où nous pouvons insérer du bruit, ce qui permettra de vérifier la résistance des algorithmes à un tel bruit, leur convergence, ainsi que leur temps de calcul. La deuxième partie des tests s'attachera à valider nos algorithmes sur des

données réelles. Nous comparerons notre algorithme avec celui, hors ligne, de Liu *et al.* [LXW+14].

Chacune des expérimentations est **cross-validée**[7] via un 10-fold cross-validation. Cette validation consiste à séparer une base S contenant m swaps en 10 sous-ensembles disjoints et de même taille S_i, $i = 1, \ldots, 10$. On effectue alors 10 apprentissages avec une répartition spécifique de chaque sous-ensemble S_i pour chaque apprentissage. Le i^{e} apprentissage utilisera les sous-ensembles $S_1, \ldots, S_{i-1}, S_{i+1}, \ldots, S_k$ afin d'apprendre le CP-net $\mathcal{N}_L^i$, et testera le CP-net sur l'ensemble S_i (voir la Figure 3.15). Cela permettra de tester la capacité des algorithmes d'apprentissage à généraliser le modèle sur des données n'étant pas forcément présentes dans l'ensemble d'apprentissage.

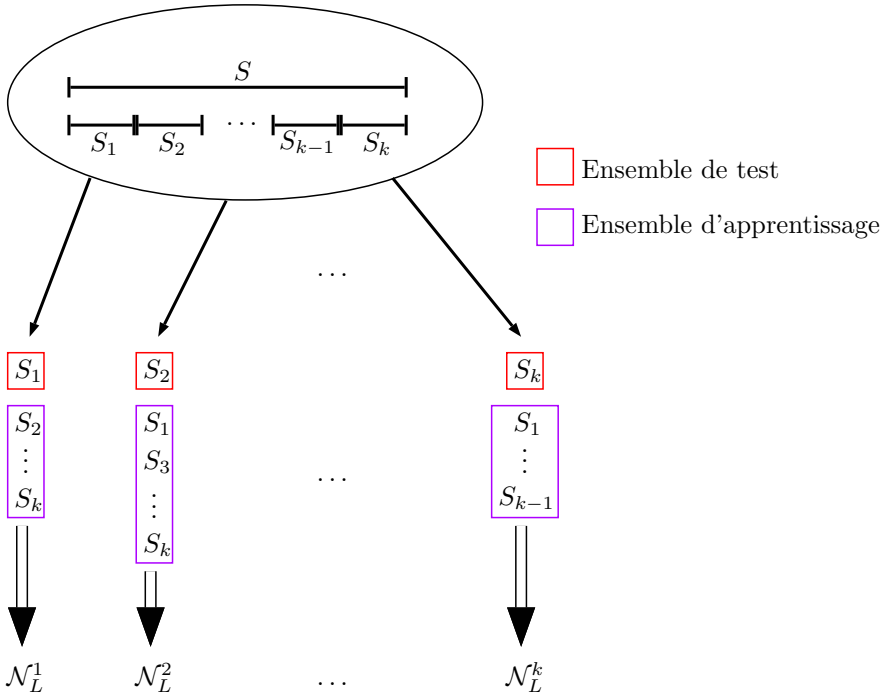


FIGURE 3.15 – Exemple d'une 10-fold cross-validation.

Dans la suite des expérimentations, la précision des différents algorithmes sera calculée par la perte globale logarithmique $L(S, h_S)$ introduite dans l'Équation (1.23) page 54, et la précision $prec(S, h_S)$ introduite par l'Équation (1.18) page 53, où S représente un ensemble d'apprentissage, et h_S l'hypothèse déterminée par les Algorithmes 9 et 10 à partir de S.

Enfin, pour chaque point (correspondant à la moyenne des 10 apprentissages), l'écart-type σ_i du i^{e} point est calculé suivant la formule de l'Équation (2.3) page 68. Les barres d'erreurs présentes dans la suite de cette section modélisent alors, pour chaque point $\tilde{\mathbf{v}}_i$, l'intervalle $[\tilde{\mathbf{v}}_i - \sigma_i, \tilde{\mathbf{v}}_i + \sigma_i]$. Nous utiliserons également le terme

7. On utilisera également l'appellation française **validation croisée** dans la suite de ce chapitre.

CP-net non borné pour un CP-net n'ayant aucune borne sur le nombre possible de variables parentes (voir la Définition 1.18 page 32).

Nous précisons que chaque expérience a été effectuée sur un ordinateur personnel doté d'un Intel(R) core(TM) i7-4600U et de 16Go de mémoire RAM. Ce code n'est pas parallélisé, et l'ensemble du code source, des expériences, et des résultats est disponible sur GitHub à l'adresse `https://github.com/FabienLab/CPnets-McDiarmid`. Le Tableau 3.4 ci-dessous résume l'ensemble des expériences effectuées.

3.5.1 Données synthétiques

Nous considérons ici un bruit aléatoire obtenu à partir d'une probabilité $p \in [0, 1]$ suivant une loi uniforme. La valeur p correspond à la probabilité qu'une vraie préférence s'inverse pour créer une préférence bruitée.

Pour générer la base de données synthétique S, nous commençons par construire un CP-net de manière aléatoire suivant l'Algorithme 8 page 77. Des swaps sont ensuite générés à partir de ce CP-net, puis bruités suivant une probabilité de bruitage p. L'Algorithme 11 ci-dessous résume la procédure de génération d'une base de données bruitée.

Algorithme 11 : `gen_BDD`$(\mathbf{V}, m, p)$

Entrées :
$\mathbf{V}$: ensemble de variables ;
m : nombre de swaps ;
p : probabilité de bruit ;
k : nombre maximum de variables parentes du CP-net
Résultat :
S : base de données bruitée ;

1 **début**
2 Générer un CP-net $\mathcal{N}$, borné par k variables parentes par variable, aléatoirement sur l'ensemble $\mathbf{V}$ suivant l'Algorithme 8;
3 $S \leftarrow \emptyset$;
4 **pour** $i = 1$ *à* m **faire**
5 Générer un swap $(\mathbf{o}, \mathbf{o}')_V$ (avec $V \in \mathbf{V}$) aléatoirement. On suppose que $\mathbf{o} \succ_{\mathcal{N}} \mathbf{o}'$;
6 Générer $x \in [0, 1]$;
7 **si** $x > p$ **alors**
8 $S \leftarrow S \cup \{\mathbf{o} \succ_{\mathcal{N}} \mathbf{o}'\}$;
9 **else**
10 $S \leftarrow S \cup \{\mathbf{o}' \succ_{\mathcal{N}} \mathbf{o}\}$;
11 **Retourner** S;

Section	Expérience	Description	Page
3.5.1	1	Efficacité de l'apprentissage hors ligne sur des données synthétiques.	118
	2	Efficacité de l'apprentissage hors ligne sur des données synthétiques en faisant varier les variables parentes du CP-net cible.	120
	3	Efficacité de l'apprentissage en ligne sur des données synthétiques	120
	4	Efficacité de l'apprentissage en ligne sur des données synthétiques en faisant varier les variables parentes du CP-net cible.	123
	5	Résistance au bruit de la version hors ligne.	124
	6	Résistance au bruit de la version en ligne.	124
	7	Efficacité de l'apprentissage hors ligne sur des tailles de bases de données variables.	126
	8	Efficacité de l'apprentissage hors ligne sur des tailles de bases de données variables sans validation croisée.	126
	9	Efficacité de l'apprentissage en ligne sur des tailles de bases de données variables.	128
	10	Efficacité de l'apprentissage en ligne sur des tailles de bases de données variables sans validation croisée.	130
	11	Convergence empirique de la version hors ligne.	131
	12	Convergence empirique de la version en ligne.	132
	13	Influence de l'hyperparamètre δ de la version en ligne sur l'efficacité de l'apprentissage.	132
	14	Influence de l'hyperparamètre τ de la version en ligne sur l'efficacité de l'apprentissage.	134
	15	Temps d'apprentissage des versions hors ligne et en ligne.	135
	16	Temps d'apprentissage des versions hors ligne et en ligne avec variation du nombre de variables.	136
3.5.2.1	17	Efficacité de l'apprentissage de la version hors ligne sur les jeux de données TripAdvisor et SUSHI.	139
	18	Efficacité de l'apprentissage de la version en ligne sur les jeux de données TripAdvisor et SUSHI.	140
3.5.2.2	19	Efficacité de l'apprentissage de la version hors ligne sur le jeu de données MovieLens.	142
	20	Efficacité de l'apprentissage de la version en ligne sur le jeu de données MovieLens.	142
	21	Efficacité de l'apprentissage de la version hors ligne sur le jeu de données MovieLens ayant des préférences uniques.	142
3.5.3	22	Comparaison de l'efficacité d'apprentissage entre les Algorithmes 9 et 10 et l'algorithme de [LXW+14].	145
	23	Comparaison de l'efficacité d'apprentissage entre l'Algorithme 10, et l'algorithme de [GAG13].	145

Tableau 3.4 – Résumé des différentes expériences.

Afin de tester plusieurs probabilités de bruit sur la même base de données d'origine, les mêmes swaps sont générés sans bruit à partir du CP-net cible construit, puis plusieurs probabilités de bruit sont appliquées sur ce swap comme expliqué dans la Figure 3.16. Ainsi, la génération et l'apprentissage des différentes bases de données bruitées s'effectue de la façon suivante :

1. Générer un CP-net aléatoire ;
2. Générer un swap à partir de ce CP-net, et le placer dans la base non bruitée S ;
3. Pour chaque probabilité de bruit p_i souhaitée, appliquer p sur le swap afin de, potentiellement, le bruiter, puis placer le résultat dans la base S^i ;
4. Répéter m fois l'étape 2 ;
5. Exécuter les algorithmes d'apprentissage sur chaque base de données S^i en utilisant la méthode de test 10-fold cross-validation (voir la Figure 3.15 ci-dessous).

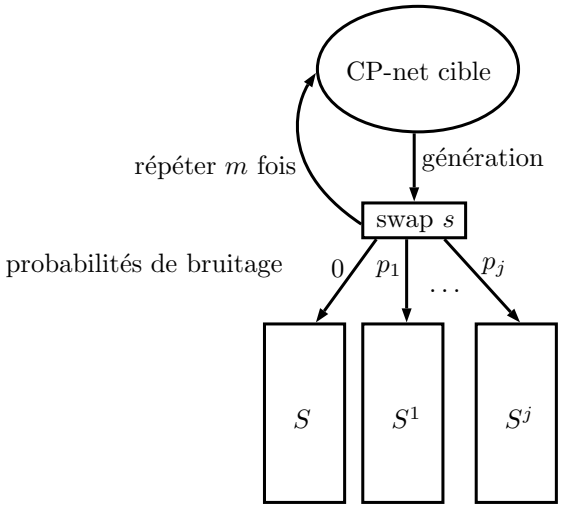
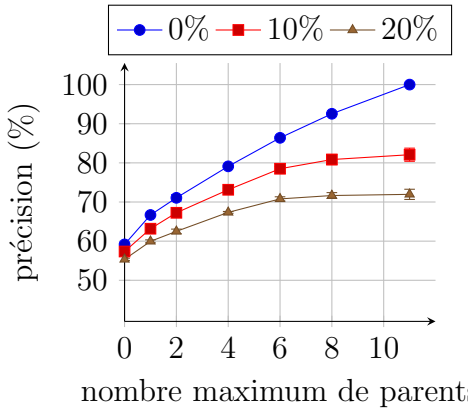


FIGURE 3.16 – Génération des différentes bases de données bruitées.

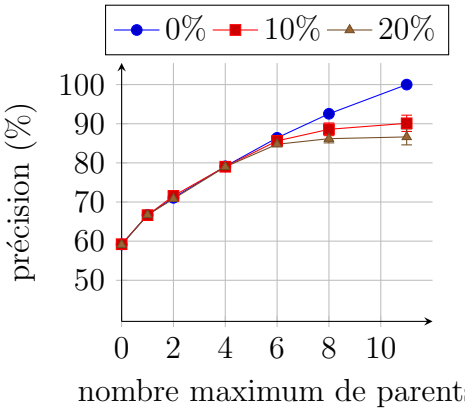
Expérience n° 1 : efficacité de l'apprentissage hors ligne sur des données synthétiques

Nous commençons nos expérimentations par l'analyse de l'efficacité de l'apprentissage de la version hors ligne de notre algorithme avec différentes probabilités de bruitage appliquées sur S. La Figure 3.17 résume cette expérience, découpée en 2 parties :

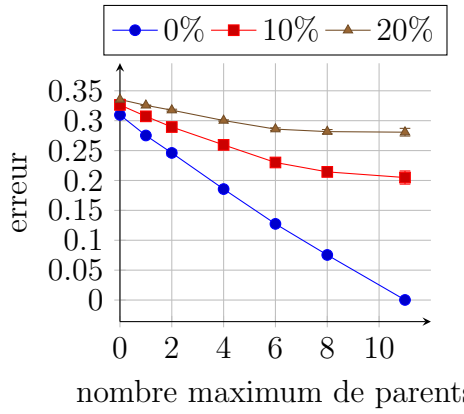
- La première partie, visible sur les Figures 3.17a et 3.17c, expose les résultats d'un apprentissage sur des bases à bruits variables, suivi du test de chaque CP-net appris sur les bases correspondantes (*e.g.*, apprentissage sur la base bruitée à 10%, puis test sur cette même base à 10% de bruit, en cross-validation).



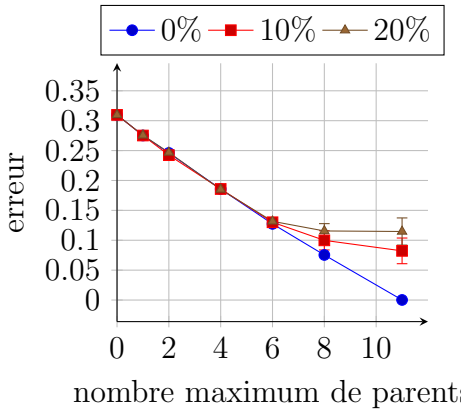
(a) Précision *prec* (Équation (1.18)) sur les bases bruitées.



(b) Précision *prec* (Équation (1.18)) sur la base non bruitée.



(c) Perte logarithmique (Équation (1.23)) sur les bases bruitées.



(d) Perte logarithmique (Équation (1.23)) sur la base non bruitée.

FIGURE 3.17 – Résultats d'apprentissage de la version hors ligne en fonction du nombre maximum de parents sur le CP-net appris (le CP-net cible est non borné). La base de donnée est générée aléatoirement (200 000 swaps, 12 variables). Chaque courbe correspond à un bruitage spécifique de la base de données.

Cela permet de juger l'efficacité de l'apprentissage de manière « naïve », en considérant le bruit inhérent aux données ;

- La deuxième partie, illustrée par les Figures 3.17b et 3.17d, expose les résultats d'un apprentissage sur des bases à bruits variables, suivi du test de chaque CP-net appris, sur la base non bruitée (*e.g.*, apprentissage sur la base bruitée à 10%, puis test sur la base non bruitée, en cross-validation), ce qui nous permet alors de juger la « résistance au bruit » de l'apprentissage.

Nous pouvons observer, dans les deux parties de cette expérience, une croissance constante de la précision lorsque le nombre de variables parentes augmente. Notons l'important écart de précision entre un CP-net séparable (*i.e.*, ne contenant aucune variable parente), et un CP-net comportant au maximum une variable parente par

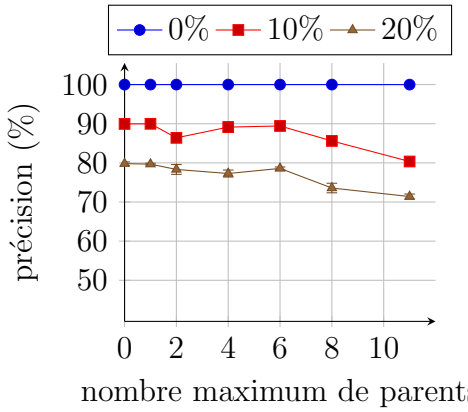
variable. Cet écart s'accroît avec la mesure de perte logarithmique des Figures 3.17c et 3.17d. Ceci s'explique par le fait que plus une CP-règle est simple (*i.e.*, une CP-règle avec peu de variables parentes), plus elle regroupe de swaps. Ainsi, les premières variables parentes ajoutées sont celles regroupant le plus grand nombre de swap, comparativement aux variables parentes suivantes. Ceci se vérifie dans cette expérience, où la pente faiblit au fur et à mesure des ajouts de variables parentes. Soulignons un important impact du bruit, avec une perte d'environ 20% de la précision lors du passage de l'apprentissage d'une base de données non bruitée à une base bruitée à 10% dans la Figure 3.17a. Ce phénomène s'explique par le fait que le bruitage est effectué sur les swaps, et non sur les CP-règles du CP-net cible. Ainsi, 10% de bruit n'implique pas forcément un bruitage de 10% des CP-règles d'origine. Ce point précis est à mettre en comparaison avec la Figure 3.17b montrant, pour les deux mêmes points, une perte de seulement 10% de la précision sur la base non bruitée, nous pouvons donc supposer que le bruit présent dans la base d'apprentissage influe directement sur les entropies des swaps non bruités modélisés par le CP-net appris, au profit de swaps bruités. Au-delà de 10% de bruit, nous observons une très bonne résistance au bruit, avec un très faible écart de précision entre la courbe à 10% de bruit, et celle à 20% de bruit (Figure 3.17b).

Enfin, cette expérience permet également de mettre en lumière la bonne généralisation de l'apprentissage pour cette version hors ligne, où aucun sur-apprentissage (modélisé par une décroissance de l'efficacité de l'apprentissage) est à noter, malgré une forte densité des CP-nets obtenus (pour k le nombre maximum de variables parentes autorisées par variable, nous obtenons $\frac{k(k-1)}{2}$ liens de parentés sur chaque CP-net appris).

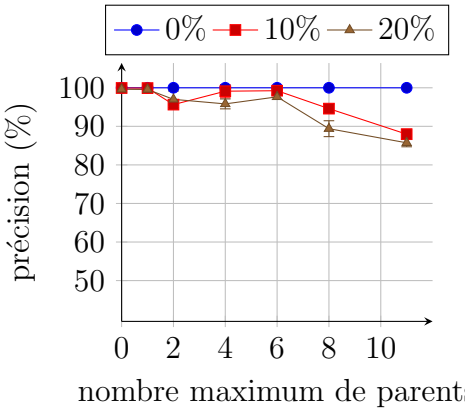
Expérience n° 2 : efficacité de l'apprentissage hors ligne sur des données synthétiques en faisant varier les variables parentes du CP-net cible

Nous effectuons la même expérience que précédemment, en faisant cette fois varier le nombre maximum de variables parentes par variable du CP-net cible (les résultats sont disponibles dans la Figure 3.18). L'objectif de cette expérience est d'observer les variations des performances d'apprentissage lorsque la densité du CP-net cible évolue. Nous observons ainsi une généralisation parfaite de la version hors ligne lorsque la base de données est non bruitée, et un très bon apprentissage, semblant suivre le bruit, jusqu'à ce que le nombre de parents du CP-net cible atteint 6. La décroissance qui suit peut alors être expliquée par le bruit des données ne permettant plus l'ajout des variables parentes optimales. De plus, la croissance du nombre de variables parentes dans le CP-net cible rend de plus en plus difficile la correction des erreurs d'apprentissage dues au bruit, grâce à l'ajout de nouvelles variables parentes du CP-net appris.

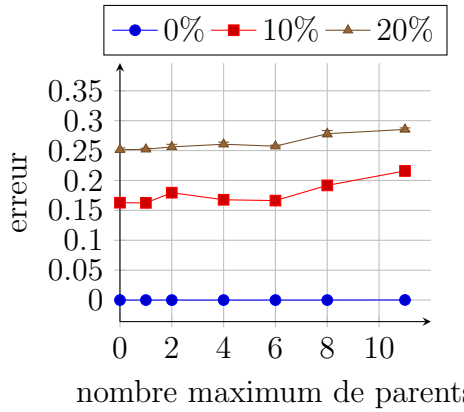
Expérience n° 3 : efficacité de l'apprentissage en ligne sur des données synthétiques



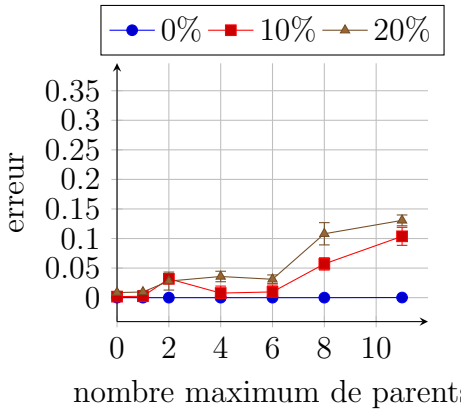
(a) Précision *prec* (Équation (1.18)) sur les bases bruitées.



(b) Précision *prec* (Équation (1.18)) sur la base non bruitée.



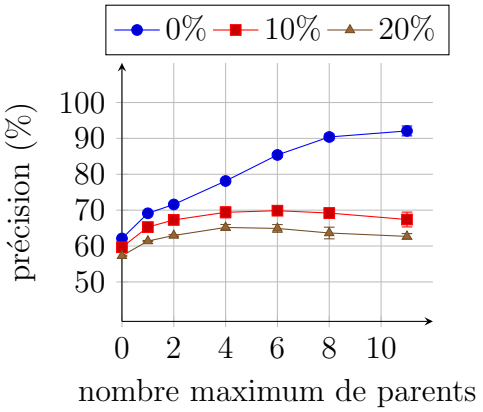
(c) Perte logarithmique (Équation (1.23)) sur les bases bruitées.



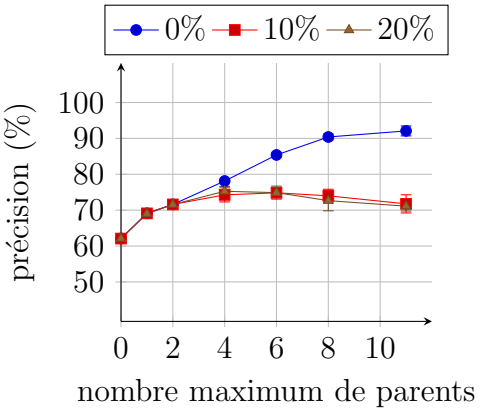
(d) Perte logarithmique (Équation (1.23)) sur la base non bruitée.

FIGURE 3.18 – Résultats d'apprentissage de la version hors ligne en fonction du nombre maximum de parents sur le CP-net cible (le CP-net appris est non borné). La base de donnée est générée aléatoirement (200 000 swaps, 12 variables). Chaque courbe correspond à un bruitage spécifique de la base de données.

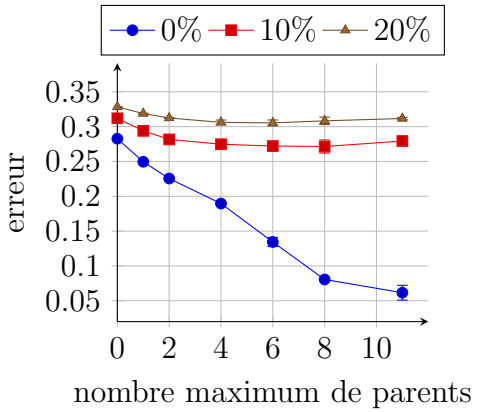
La Figure 3.19 présente les résultats de l'Expérience n° 1 (variation du nombre de variables parentes du CP-net appris) effectuée sur la version en ligne de notre procédure. En comparant les deux versions de notre algorithme, on peut observer en tout logique que l'efficacité est moindre pour la version en ligne par rapport à celle hors ligne, du fait que cette dernière dispose de beaucoup plus d'informations pour apprendre le CP-net. Cette différence d'efficacité entre les deux versions est présente aussi bien sur la Figure 3.19a que sur la Figure 3.19b, et est plus nette avec la perte logarithmique (Figures 3.19c et 3.19d). En effet, lorsque chaque variable peut avoir 11 variables parentes, l'entropie associée à l'apprentissage est extrêmement élevée dès que l'Algorithme 10 apprend sur une base bruitée. Notons que les CP-nets obtenus avec cette méthode en ligne sont beaucoup moins denses que ceux de la méthode hors ligne (pour 11 variables parentes au maximum par variable, nous observons



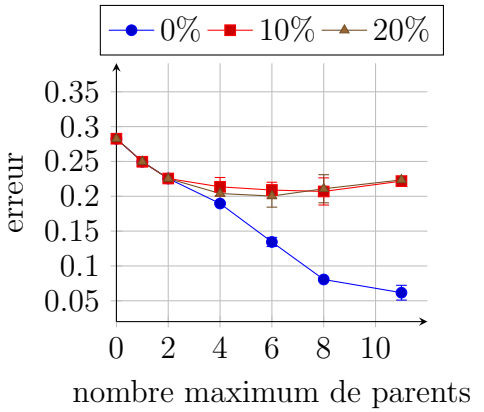
(a) Précision *prec* (Équation (1.18)) sur les bases bruitées.



(b) Précision *prec* (Équation (1.18)) sur la base non bruitée.



(c) Perte logarithmique (Équation (1.23)) sur les bases bruitées.



(d) Perte logarithmique (Équation (1.23)) sur la base non bruitée.

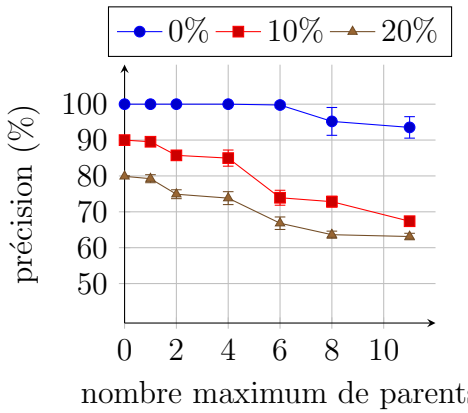
FIGURE 3.19 – Résultats d'apprentissage de la version en ligne en fonction du nombre maximum de parents sur le CP-net appris (le CP-net cible est non borné). La base de donnée est générée aléatoirement (1 500 000 swaps, 12 variables, $\delta = 0.1$, $\tau = 0.05$). Chaque courbe correspond à un bruitage spécifique de la base de données.

une moyenne de 6 variables parentes). Cela est dû à la borne de McDiarmid, symbolisée par l'hyperparamètre δ, demandant un grand nombre d'observations avant d'autoriser l'ajout d'une nouvelle variable parente, sous réserve d'une différence significative entre les deux premiers gains d'information. Le deuxième hyperparamètre τ permet de contre-balancer cette dernière contrainte, en ne basant la décision que sur le nombre d'exemples observés. Nous avons ainsi relevé une moyenne de 13 000 swaps nécessaires avant l'ajout d'une nouvelle variable parente, lorsque nous fixons $\tau = 0.05$.

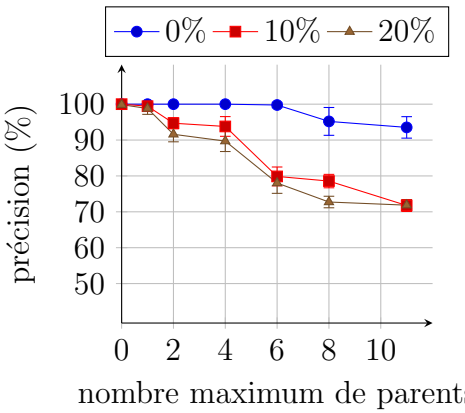
La deuxième observation principale concerne l'effet de sur-apprentissage apparaissant lorsque le nombre maximum de variables parentes par variable est supérieur ou égal à 8, visible par une perte de précision au sein des Figures 3.19a et 3.19b (et une augmentation de la perte logarithmique sur les Figures 3.19c et 3.19d). Ce sur-

apprentissage peut être expliqué par le fait que les dernières variables parentes ont été ajoutées après plusieurs estimations des nouveaux compteurs, qui s'éloignent, au fur et à mesure, des valeurs réelles de ceux-ci, et ne permettent pas de suffisamment généraliser les nouvelles CP-règles apprises.

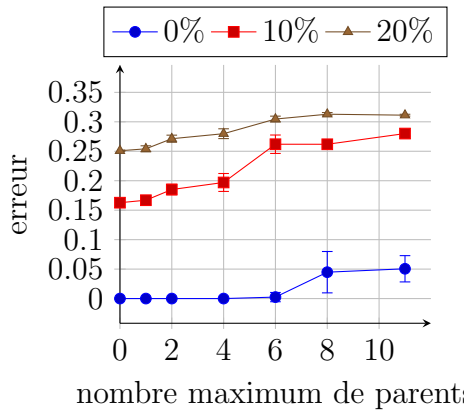
Expérience n° 4 : efficacité de l'apprentissage en ligne sur des données synthétiques en faisant varier les variables parentes du CP-net cible



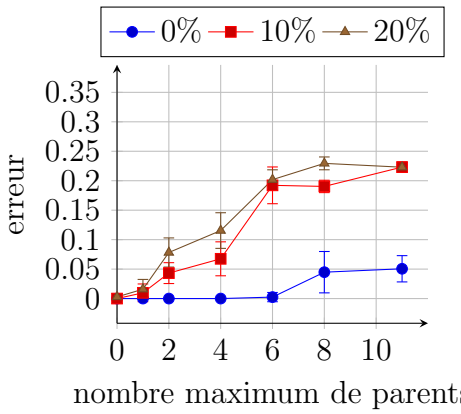
(a) Précision *prec* (Équation (1.18)) sur les bases bruitées.



(b) Précision *prec* (Équation (1.18)) sur la base non bruitée.



(c) Perte logarithmique (Équation (1.23)) sur les bases bruitées.



(d) Perte logarithmique (Équation (1.23)) sur la base non bruitée.

FIGURE 3.20 – Résultats d'apprentissage de la version en ligne en fonction du nombre maximum de parents sur le CP-net cible (le CP-net appris est non borné). La base de donnée est générée aléatoirement (1 500 000 swaps, 12 variables, $\delta = 0.1$, $\tau = 0.05$). Chaque courbe correspond à un bruitage spécifique de la base de données.

Nous testons maintenant la capacité d'apprentissage de la version en ligne lorsque le nombre de variables parentes du CP-net cible varie. Les résultats sont exposés dans la Figure 3.20. Nous observons des résultats similaires à ceux obtenus avec la version hors ligne représentés par la Figure 3.18 page 121, avec une baisse logique

de la précision, due à la nature en ligne de l'algorithme. Remarquons par ailleurs un taux d'apprentissage parfait sur la base de données non bruitée (courbes bleues) lorsque le nombre maximum de parents du CP-net cible est inférieur à 6.

Expérience n° 5 : résistance au bruit de la version hors ligne

La prochaine expérience porte sur la résistance au bruit de la version hors ligne. Elle a été conçue de la manière suivante : dix bases de données, issues du même CP-net, sont générées. Chaque base S^i correspond à une base où chaque swap a été bruité avec une probabilité p_i (voir la Figure 3.16 page 118). Un CP-net est alors appris sur chacune des bases de données S^i, puis testé uniquement sur la base de données non bruitée S^0. Afin d'effectuer une validation croisée des résultats, chaque base est séparée suivant le même protocole, *i.e.*, chaque swap d'origine généré à partir du CP-net aléatoire est bruité en fonction des différentes probabilités de bruit p_i, puis placé dans la même partie de chaque base. Ainsi, il devient cohérent d'effectuer un apprentissage sur les neuf premières parties $S^i_1, \ldots, S^i_9$ d'une base S^i, et de tester sur la dixième partie S^0_{10} de la base non bruitée S^0. Soit h_S l'hypothèse apprise à partir de l'ensemble d'entraînement S, nous définissons alors l'erreur err comme étant

$$err(S, h_S) = \frac{1}{|S|} \sum_{s \in S} \mathbb{1}(h_S(s) = 1 - y_s) \in [0, 1]. \tag{3.25}$$

Une illustration est donnée dans la Figure 3.21, avec un exemple d'exécution des différents apprentissages sur le premier des dix apprentissages de la cross-validation. Les résultats sont ensuite moyennés sur les dix apprentissages.

Nous souhaitons, dans l'idéal, obtenir une courbe qui tend vers une erreur nulle sur la base de données d'origine (non bruitée) lorsque le bruit est compris dans l'intervalle $[0\%, 50\%)$, obtenir une erreur de 50% lorsque le bruit est de 50%, puis une erreur de 100% lorsque le bruit est compris dans l'intervalle $(50\%, 100\%]$ (les préférences bruitées étant alors considérées comme les vraies préférences du fait de leur dominance dans la base de données).

Les résultats de la version hors ligne sont exposés dans la Figure 3.22, où nous pouvons observer une bonne résistance au bruit avec, par exemple, une différence d'erreur d'environ 20% entre la base de données non bruitée et la base de données bruitée à 30%. Cette différence est bien symétrique, tout en s'« annulant » à 50%, où il n'est alors plus possible de distinguer le bruit des vraies données, l'Algorithme 9 page 97 n'apprend que la moitié des règles. Cela signifie que l'algorithme arrive à minimiser le bruit appris afin de modéliser un maximum de swaps de la base de données non bruitée.

Expérience n° 6 : résistance au bruit de la version en ligne

Les résultats de la version en ligne sont illustrés par la Figure 3.23. Nous pouvons observer une courbe très différente de celle obtenue avec la version hors ligne dans la Figure 3.22 page 125. Cela vient du fait du grand écart de 25% existant entre

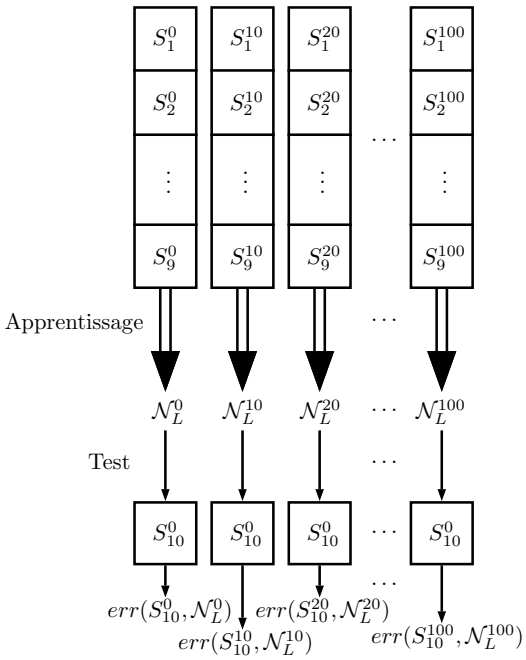


FIGURE 3.21 – Exemple de test de robustesse au bruit sur le premier apprentissage de la cross-validation. S_j^i correspond à j^e partie de la base de donnée S^i, avec i correspondant à la probabilité p_i de bruiter chaque swap de la base. Chaque S^i permet alors de générer un CP-net appris $\mathcal{N}_L^i$. Les erreurs sont ensuite moyennées par la cross-validation.

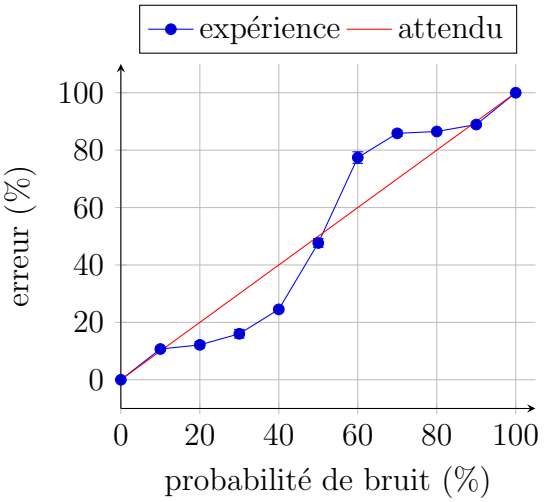


FIGURE 3.22 – Résistance au bruit de la version hors ligne (les CP-nets cible et appris sont non bornés). L'erreur d'apprentissage donnée correspond à la différence de précision entre une base de donnée sans bruit et une base de données à bruit variable. La base de donnée est générée aléatoirement (200 000 swaps, 12 variables). Chaque courbe correspond à un bruitage spécifique de la base de données.

l'apprentissage sur la base non bruitée et la base bruitée à 10%. Cet écart est dû en partie, comme expliqué précédemment, aux swaps bruités empêchant l'algorithme

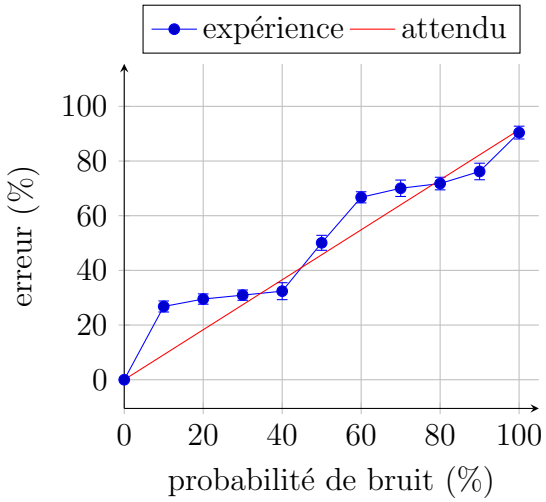


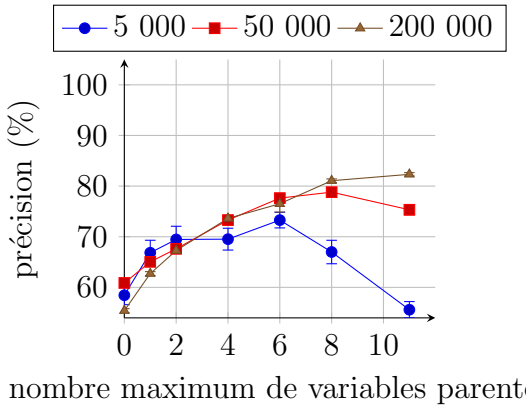
FIGURE 3.23 – Résistance au bruit de la version en ligne (les CP-nets cible et appris sont non bornés). L'erreur d'apprentissage donnée correspond à la différence de précision entre une base de donnée sans bruit et une base de données à bruit variable. La base de donnée est générée aléatoirement (1 500 000 swaps, 12 variables, $\delta = 0.1$, $\tau = 0.05$). Chaque courbe correspond à un bruitage spécifique de la base de données.

d'obtenir une entropie pure sur chaque feuille de chaque arbre, ainsi qu'à un accès plus restreint aux informations de la version en ligne par rapport à son homologue hors ligne. Notons cependant une bonne résistance au bruit lorsque ce bruit est compris entre 10% et 40%, et entre 60% et 90% (où l'objectif est alors d'apprendre le bruit, qui se retrouve prédominant).

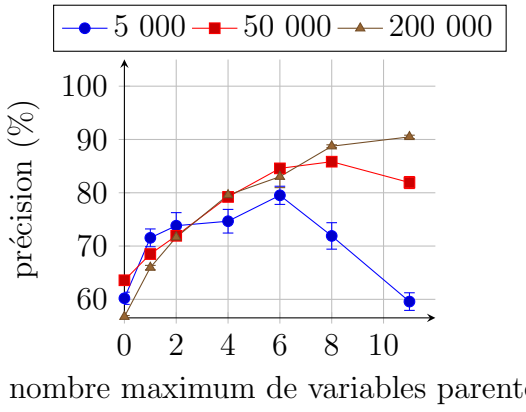
Expérience n° 7 : efficacité de l'apprentissage hors ligne sur des tailles de bases de données variables

La prochaine expérience permet de visionner la capacité d'apprentissage de la version hors ligne sur des bases de données de taille variable. Les résultats sont exposés dans la Figure 3.24. Nous observons sur les différentes sous-figures une rapide décroissance (resp. croissance) de la courbe de précision (resp. d'erreur) à partir de 6 parents par variable. Cette décroissance s'estompe au fur et à mesure de l'augmentation de la taille de la base de données. Ce phénomène est dû, comme nous le verrons dans la prochaine expérience, à un sur-apprentissage de la version hors ligne, n'apparaissant pas lorsque nous testons le CP-net appris sur les mêmes données ayant servies à son apprentissage. Ce sur-apprentissage intervient à cause d'un manque de données ne permettant pas à l'algorithme d'effectuer une bonne généralisation : nous pouvons effectivement observer que pour des bases de données de « grande taille » (200 000 comparaisons), ce sur-apprentissage n'apparait plus.

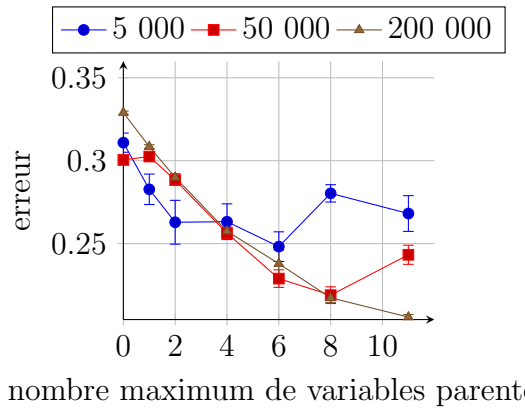
Expérience n° 8 : efficacité de l'apprentissage hors ligne sur des tailles de bases de données variables sans validation croisée



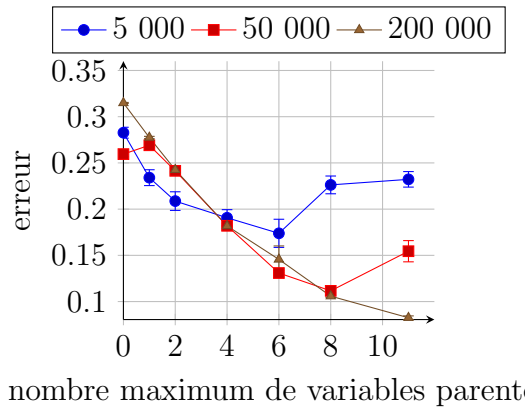
(a) Précision *prec* (Équation (1.18)) sur les bases bruitées.



(b) Précision *prec* (Équation (1.18)) sur les bases non bruitées.



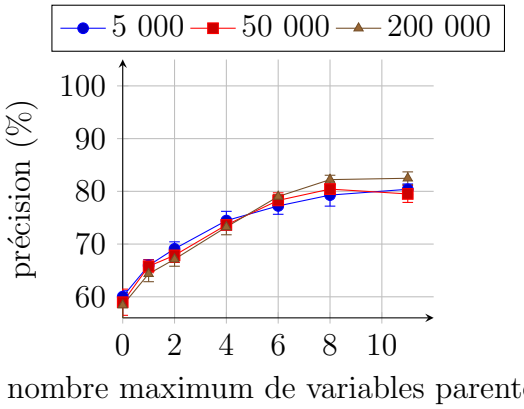
(c) Perte logarithmique (Équation (1.23)) sur les bases bruitées.



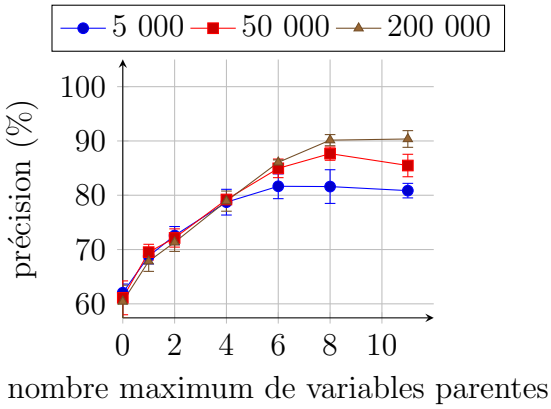
(d) Perte logarithmique (Équation (1.23)) sur les bases non bruitées.

FIGURE 3.24 – Résultats d'apprentissage de la version hors ligne en fonction du nombre maximum de parents sur le CP-net appris (le CP-net cible est non borné). Chaque courbe représente la génération d'une base de données bruitée à 10% de taille différente (12 variables).

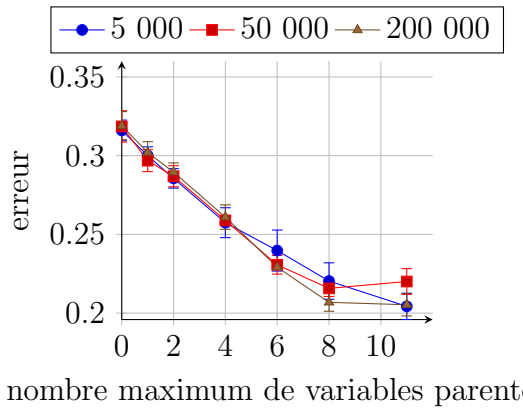
Nous souhaitons vérifier ce sur-apprentissage en effectuant la même expérience que précédemment, mais en testant cette fois le CP-net appris sur les mêmes données ayant servies à son apprentissage (*i.e.*, sans validation croisée). Les résultats sont disponibles dans la Figure 3.25, où nous pouvons voir sur les différentes sous-figures un comportement très différent de celui figurant des sous-figures de la Figure 3.24. Nous rappelons qu'ici, les expériences ont été menées sans validation croisée, avec un apprentissage, puis un test, sur le même ensemble de préférences (bruitées à 10%). Ainsi, les Figures 3.25a et 3.25c montrent que peu importe la taille de la base de données, la version hors ligne arrive à reconstituer environ 80% de cette base bruitée. Notons cependant que si les tests sont effectués sur la version non bruitée des swaps (Figures 3.25b et 3.25d), alors la taille de ces mêmes bases influence la qualité de l'apprentissage (plus la base est grande, mieux est son approximation par



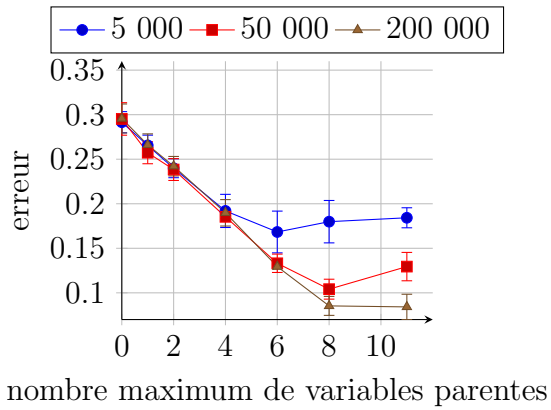
(a) Précision *prec* (Équation (1.18)) sur les bases bruitées.



(b) Précision *prec* (Équation (1.18)) sur les bases non bruitées.



(c) Perte logarithmique (Équation (1.23)) sur les bases bruitées.



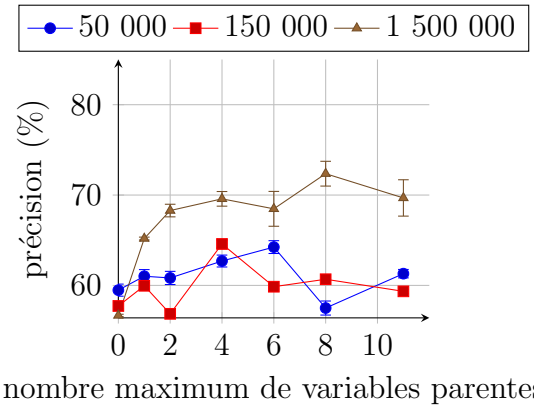
(d) Perte logarithmique (Équation (1.23)) sur les bases non bruitées.

FIGURE 3.25 – Résultats d'apprentissage de la version hors ligne en fonction du nombre maximum de parents sur le CP-net appris (le CP-net cible est non borné) sans validation croisée. Chaque courbe représente la génération d'une base de données bruitée à 10% de taille différente (12 variables).

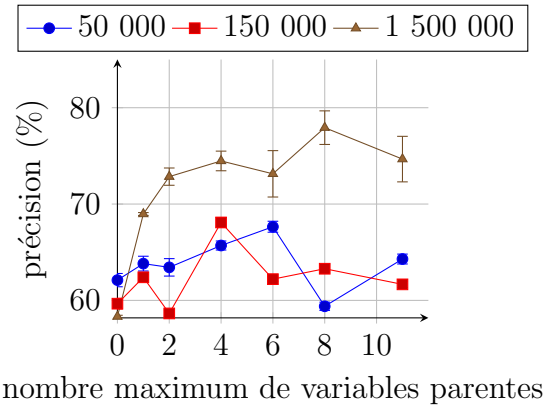
l'apprentissage). Cela s'explique par le fait que le nombre accru de données permet de gérer plus facilement le bruit. Ainsi, un bruit de 10% sur les données ne peut pas être corrigé avec 5 000 swaps, cependant, lorsque ce chiffre augmente jusqu'à 200 000, le nombre de swaps non bruités permet de prendre en compte de manière efficace le bruit, en atteignant 90% de swaps modélisés.

Expérience n° 9 : efficacité de l'apprentissage en ligne sur des tailles de bases de données variables

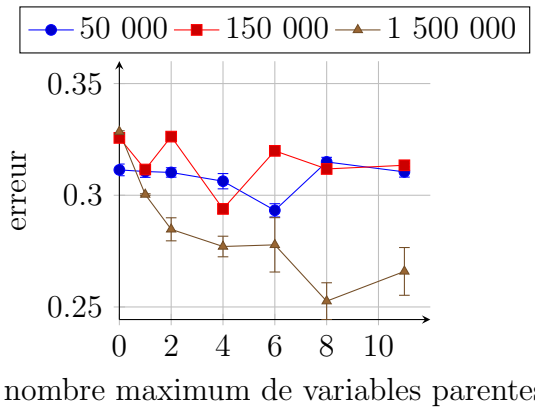
Vérifions maintenant l'impact de la taille des bases de données sur l'apprentissage d'un CP-net via la version en ligne de notre algorithme. Les résultats en validation croisée sont disponibles sur la Figure 3.26. Nous observons qu'ils sont en dents de



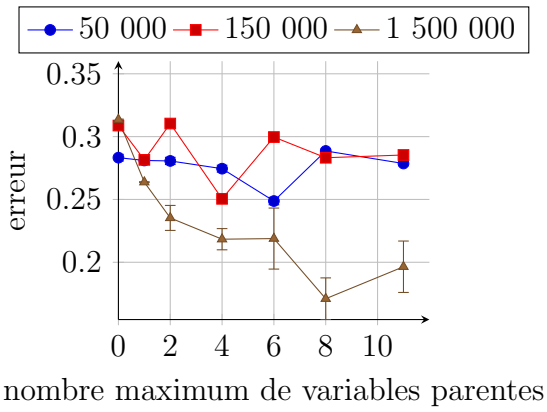
(a) Précision *prec* (Équation (1.18)) sur les bases bruitées.



(b) Précision *prec* (Équation (1.18)) sur les bases non bruitées.



(c) Perte logarithmique (Équation (1.23)) sur les bases bruitées.



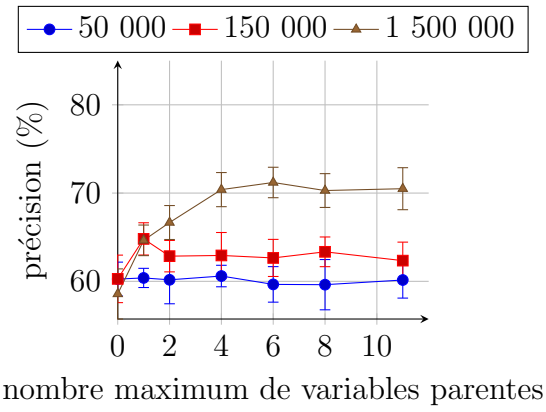
(d) Perte logarithmique (Équation (1.23)) sur les bases non bruitées.

FIGURE 3.26 – Résultats d'apprentissage de la version en ligne en fonction du nombre maximum de parents sur le CP-net appris (le CP-net cible est non borné). Chaque courbe représente la génération d'une base de données bruitée à 10% de taille spécifique (12 variables, $\delta = 0.1$, $\tau = 0.05$).

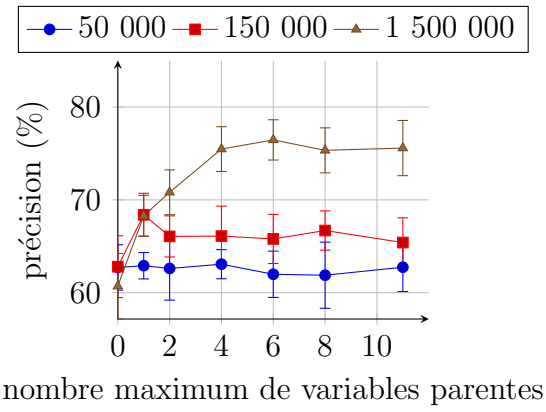
scie, aussi bien lorsque les tests sont effectués sur la base bruitée à 10%, que sur la base non bruitée. Ces résultats se stabilisent au fur et à mesure de l'augmentation de la base de donnée. La raison est simple : la borne de McDiarmid contraint la version en ligne à ajouter de nouvelles variables parentes relativement tardivement durant l'apprentissage (il est nécessaire d'observer environ 13 000 swaps d'une même variable avant d'ajouter une nouvelle variable parente, ce qui signifie que pour 12 variables et des swaps répartis de manière uniforme, la première variable parente ajoutée intervient après l'observation d'environ 150 000 swaps). Ainsi, sur les bases de données testées, seule la base contenant 1 500 000 swaps est assez grande pour autoriser l'ajout de plusieurs variables parentes.

Expérience n° 10 : efficacité de l'apprentissage en ligne sur des tailles

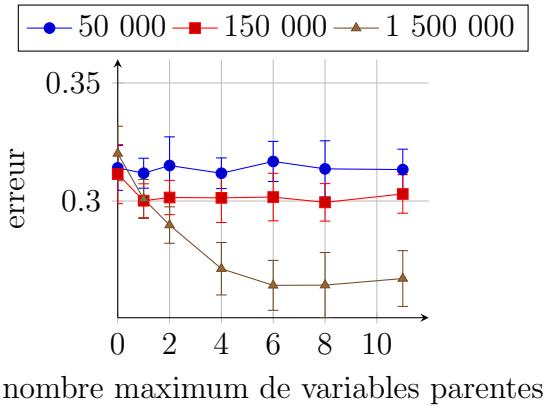
de bases de données variables sans validation croisée



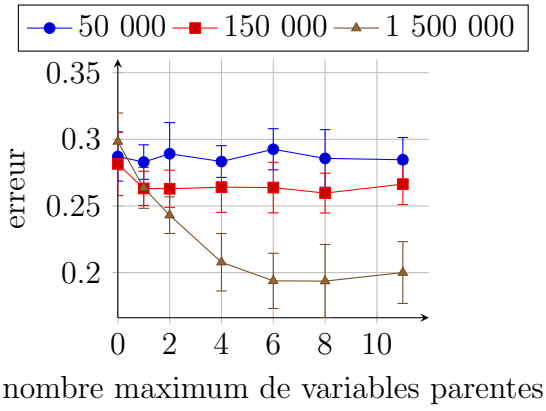
(a) Précision *prec* (Équation (1.18)) sur les bases bruitées.



(b) Précision *prec* (Équation (1.18)) sur les bases non bruitées.



(c) Perte logarithmique (Équation (1.23)) sur les bases bruitées.



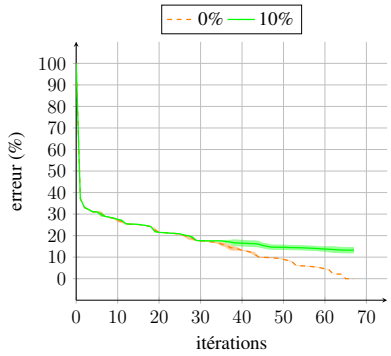
(d) Perte logarithmique (Équation (1.23)) sur les bases non bruitées.

FIGURE 3.27 – Résultats d'apprentissage de la version en ligne en fonction du nombre maximum de parents sur le CP-net appris (le CP-net cible est non borné) sans validation croisée. Chaque courbe représente la génération d'une base de données bruitée à 10% de taille spécifique (12 variables, $\delta = 0.1$, $\tau = 0.05$).

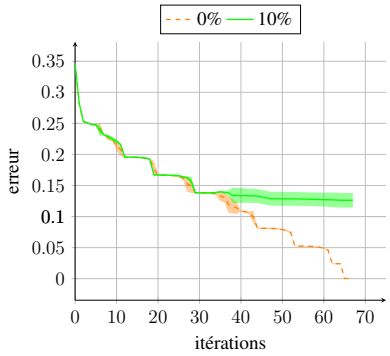
Nous répétons l'expérience en testant le CP-net appris sur les mêmes swaps ayant servis à l'apprentissage. Les résultats sont illustrés dans la Figure 3.27. Nous pouvons voir que les résultats de cette expérience corroborent ceux de la version hors ligne, avec une stabilité n'apparaissant pas lorsque les apprentissages sont cross-validés. Nous observons de plus que la taille des petites bases de données reste un problème pour une version en ligne destinée à manipuler de grands flux. Ainsi, les bases contenant 50 000 et 150 000 swaps stagnent à 60% et 62% de précision, respectivement (voir la Figure 3.27a). Nous remarquons une augmentation de la précision de plus en plus nette lorsque le test est effectué sur la version non bruitée de l'ensemble de préférences d'apprentissage (Figure 3.27b), malgré une allure identique

des courbes. La plus grande stabilité des courbes des Figures 3.27c et 3.27d s'explique par le fait que l'algorithme ait été élaboré autour de la mesure de perte empirique, et fonctionne en sélectionnant les couples de variables permettant d'améliorer l'entropie globale de chaque variable.

Expérience n° 11 : convergence empirique de la version hors ligne



(a) Erreur *err* (Équation (3.25)).



(b) Perte logarithmique (Équation (1.23)).

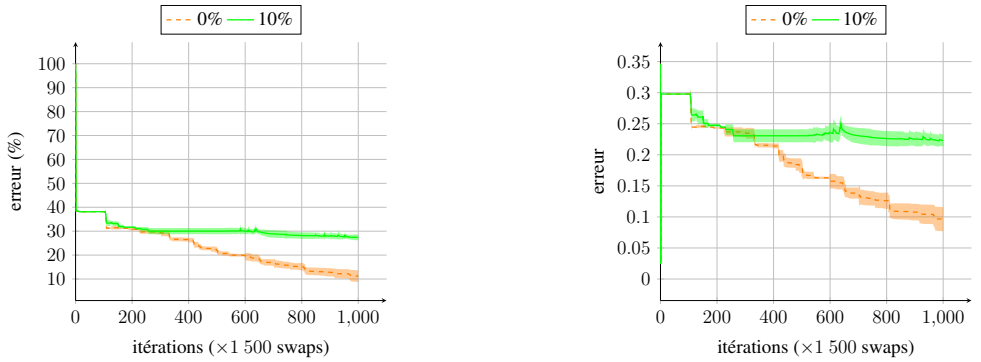
FIGURE 3.28 – Convergence empirique de la version hors ligne (les CP-nets cibles et appris sont non bornés). Chaque itération correspond à un parcours de la base de données. Cette base de données est générée aléatoirement (200 000 swaps, 12 variables). Chaque courbe correspond à un bruitage spécifique de la base de données.

La prochaine expérience concerne la convergence empirique des deux versions de notre algorithme. Nous commençons par la version hors ligne, dont les résultats sont résumés au sein de la Figure 3.28. Une itération correspond à un nouveau parcours complet de la base de données (permettant de mettre en lumière l'ensemble des CP-règles actives). Ce parcours n'ayant lieu que dans le cas où une modification intervient dans le CP-net appris (ajout d'une variable parente), il est possible de borner ce nombre d'itérations au nombre maximum d'arcs présents dans le CP-net. Notons que dans cette expérience, les deux mesures d'erreur des Figures 3.28a et 3.28b n'évoluent pas de la même manière, avec une forte décroissance dès la première itération pour l'erreur en pourcentage de la Figure 3.28a, tandis que la perte logarithmique de la Figure 3.28b décroît par paliers. Ceci s'explique par le fait que dès la première itération, lors de l'ajout de la première variable parente, le nombre de swaps nouvellement modélisés est important. Or, seule l'entropie des douze arbres de décision diminue. De plus, la nouvelle entropie calculée peut continuer à être importante, mais fait intervenir un nombre moins élevé de swaps modélisés (d'où une décroissance plus faible de l'erreur de la Figure 3.28a alors que la perte de la Figure 3.28b décroît de façon constante tout au long de l'apprentissage.

Remarquons en outre que, lorsque la base de données est bruitée, la convergence est plus rapide, pour atteindre son minimum aux alentours de la 35^{e} itération. Cette observation peut être due au fait qu'en présence de bruit, il n'est pas possible de terminer un apprentissage en n'ayant que des variables pures (du fait que certains swaps

ne pourront jamais être modélisés). Ainsi, plus l'apprentissage progresse, moins de nouveaux swaps pouvant être modélisés sont trouvés lors des ajouts de nouvelles variables parentes.

Expérience n° 12 : convergence empirique de la version en ligne



(a) Erreur *err* (Équation (3.25)).

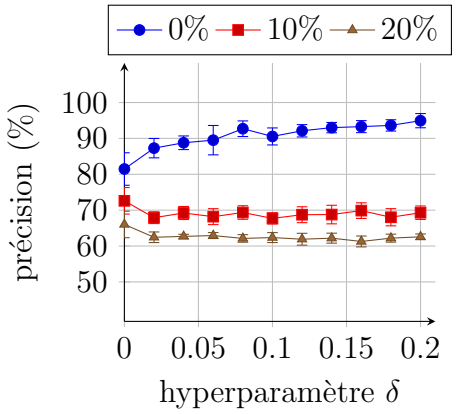
(b) Perte logarithmique (Équation (1.23)).

FIGURE 3.29 – Convergence empirique de la version en ligne (les CP-nets cible et appris sont non bornés). Chaque itération correspond à un parcours de la base de données. Cette base de données est générée aléatoirement (1 500 000 swaps, 12 variables, $\delta = 0.1$, $\tau = 0.05$). Chaque courbe correspond à un bruitage spécifique de la base de données.

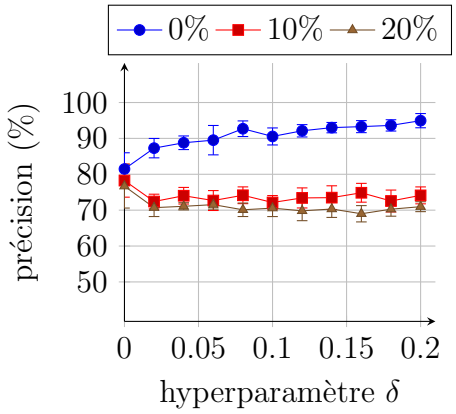
La même expérience est reproduite sur la version en ligne dans la Figure 3.29. Chaque itération de la version en ligne correspond à l'observation d'un unique swap. Cependant, comme il n'était pas possible d'afficher l'erreur obtenue à l'issue de l'observation de chaque swap sur une base en contenant 1 500 000, chaque point de la figure correspond à l'erreur obtenue après l'observation d'environ 1 500 swaps. Étant donné que la borne de McDiarmid, ayant comme hyperparamètres $\delta = 0.1$ et $\tau = 0.05$, n'autorise l'ajout d'une nouvelle variable parente qu'après avoir vu environ 13 000 swaps faisant intervenir la même variable, cela signifie que la première variable parente est ajoutée dans le meilleur des cas après la 8^{e} itération. Cela se vérifie dans la Figure 3.29a qui, tout comme son homologue hors ligne, montre une forte décroissance de son erreur. Notons cependant une faible décroissance de la perte logarithmique dans la Figure 3.29b, cela étant dû, comme dit auparavant, à l'absence de corrélation entre le nombre de swaps modélisés par le CP-net appris, et l'entropie sous-jacente des différents arbres de décision. Notons enfin, tout comme la version hors ligne, une rapide stagnation de la convergence lorsque l'apprentissage est effectué sur une base bruitée, pour les raisons évoquées précédemment.

Expérience n° 13 : influence de l'hyperparamètre δ de la version en ligne sur l'efficacité de l'apprentissage

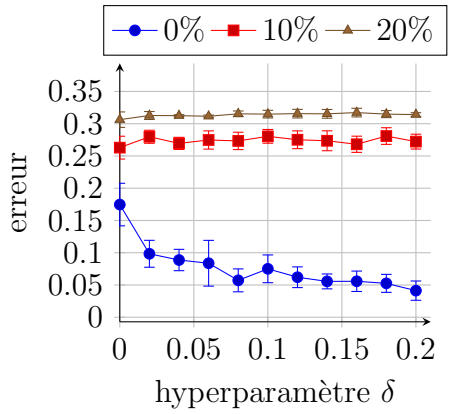
Analysons à présent l'influence des hyperparamètres δ et τ sur la qualité de l'apprentissage de la version en ligne. Nous commençons par l'influence de δ, dont



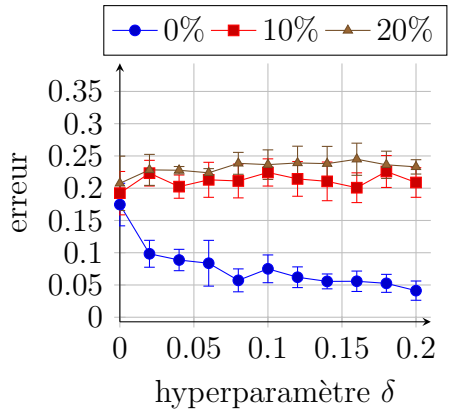
(a) Précision *prec* (Équation (1.18)) sur les bases bruitées.



(b) Précision *prec* (Équation (1.18)) sur la base non bruitée.



(c) Perte logarithmique (Équation (1.23)) sur les bases bruitées.



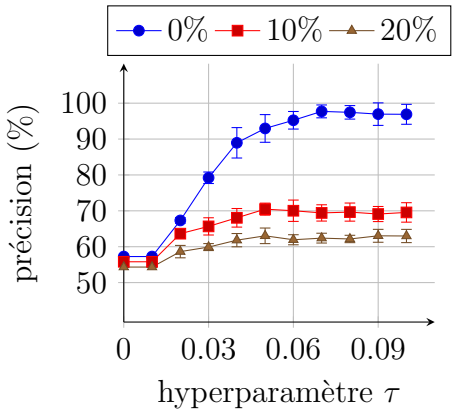
(d) Perte logarithmique (Équation (1.23)) sur la base non bruitée.

FIGURE 3.30 – Résultats de l'apprentissage de la version en ligne (les CP-nets cible et appris sont non bornés) en fonction de l'hyperparamètre δ (1 500 000 swaps, 12 variables, $\tau = 0.05$). Chaque courbe correspond à un bruitage spécifique de la base de données.

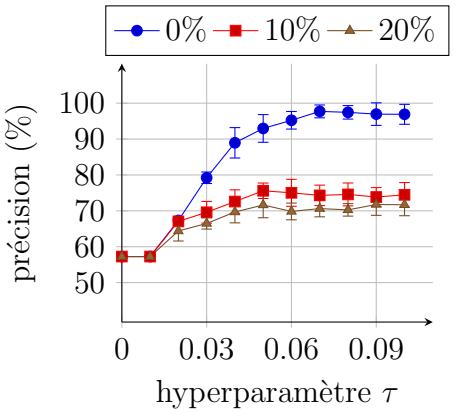
les résultats sont illustrés dans la Figure 3.30, où nous remarquons une certaine constance, aussi bien sur la précision que sur la perte, lorsque l'hyperparamètre δ varie. Rappelons que plus δ est grand, moins la confiance portée sur les choix de l'algorithme est importante. Cela signifie que plus δ est grand, plus l'algorithme est susceptible d'ajouter de nouvelles variables parentes, mais qu'un nombre restreint d'exemples observés a été nécessaire pour leur ajout. Cela explique l'augmentation (resp. la baisse) de la précision (resp. de l'erreur) sur les Figures 3.30a et 3.30b (resp. les Figures 3.30c et 3.30d) lorsque l'apprentissage est effectué sur la base non bruitée (courbes bleues), car les seules incohérences de l'apprentissage sont dues à des variables parentes manquantes. Il n'y a pas de bruit spécifique pouvant alors induire en erreur l'algorithme, et il n'est donc pas nécessaire d'attendre longtemps avant d'ajouter une variable parente de confiance (rappelons que le CP-net appris est peu dense, chaque variable n'ayant au maximum que 6 variables parentes, et

que plus δ augmente, plus la densité du CP-net appris devient importante). Avoir $\delta = 0.2$ signifie obtenir une confiance de 80%, ce qui reste assez faible. Cependant, plus δ est petit, plus le nombre de swaps nécessaires à un nouvel ajout est important. Nous avons donc décidé de fixer $\delta = 0.1$ pour l'ensemble de nos expériences, celui-ci correspondant à une confiance de 90% sur les différents ajouts de variables parentes, et permettant un ajout de nouvelles variables parentes sur des bases de données de taille raisonnable.

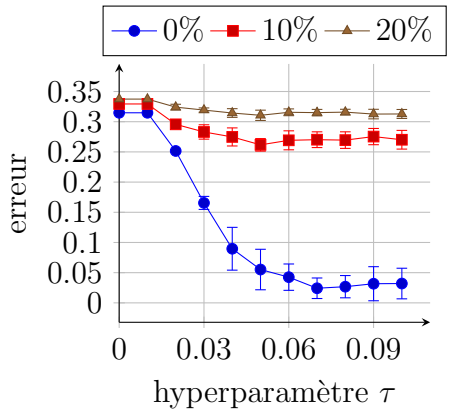
Expérience n° 14 : influence de l'hyperparamètre τ de la version en ligne sur l'efficacité de l'apprentissage



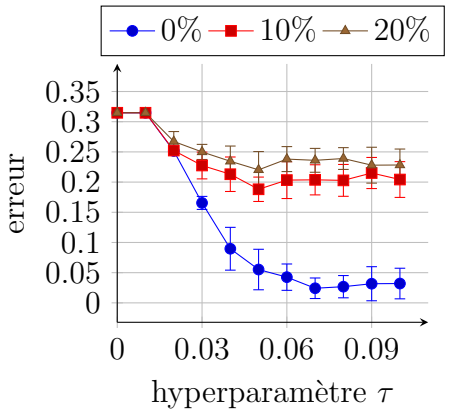
(a) Précision *prec* (Équation (1.18)) sur les bases bruitées.



(b) Précision *prec* (Équation (1.18)) sur la base non bruitée.



(c) Perte logarithmique (Équation (1.23)) sur les bases bruitées.



(d) Perte logarithmique (Équation (1.23)) sur la base non bruitée.

FIGURE 3.31 – Résultats de l'apprentissage de la version en ligne (les CP-nets cible et appris sont non bornés) en fonction de l'hyperparamètre τ (1 500 000 swaps, 12 variables, $\delta = 0.1$). Chaque courbe correspond à un bruitage spécifique de la base de données.

Analysons maintenant l'influence du deuxième paramètre τ sur l'efficacité de l'apprentissage. Les résultats sont disponibles dans la Figure 3.31. Nous rappelons

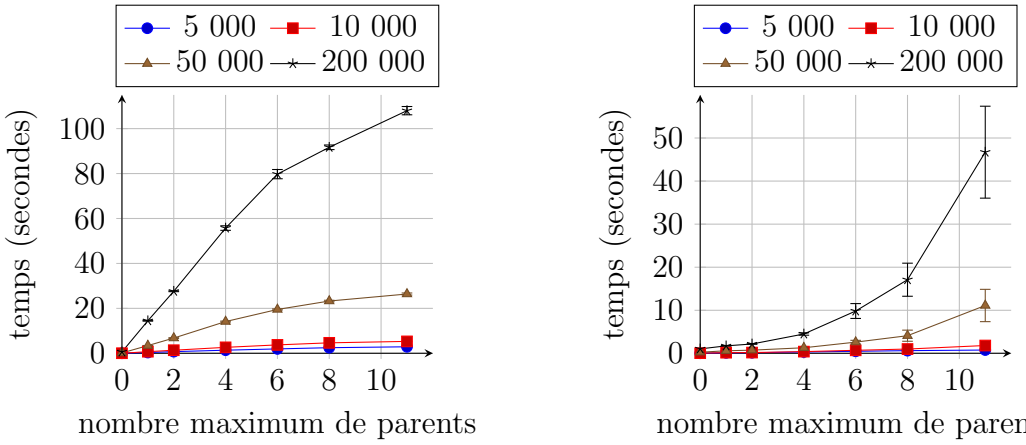
que cet hyperparamètre est présent pour faire face à la montée parallèle de deux couples de variables maximisant le gain d'information, ne permettant alors pas un écart suffisamment grand pour autoriser l'ajout d'une nouvelle variable parente malgré un nombre important de swaps observés. Ce paramètre permet donc de ne se focaliser que sur le nombre de swaps observés afin de prendre une décision (lorsque δ est fixé, seul le nombre de swaps observés fait varier la valeur de la borne ϵ). Ainsi, plus τ est petit, plus les variables parentes seront ajoutées tardivement (*i.e.*, plus il sera nécessaire d'avoir observé un nombre important de swaps avant de se décider). Ce constat se vérifie sur toutes les courbes de la Figure 3.31, avec une augmentation (resp. une baisse) de la précision (resp. de l'erreur) au fur et à mesure de l'augmentation de la valeur de τ. Remarquons que les précisions et les erreurs la base de données non bruitée (courbes bleues) suivent la même évolution que les courbes bleues des différentes sous-figures de la Figure 3.30. En effet, pour les mêmes raisons que précédemment, plus τ est grand, plus il est possible d'ajouter rapidement de nouvelles variables parentes. De plus, le fait que la base de données ne soit pas bruitée permet un ajout correct de variables parentes auxquelles nous avons moins confiance. Ainsi, lorsque $\tau = 0$, aucune variable parente n'est ajoutée, d'où une faible (resp. importante) précision (resp. erreur) d'apprentissage. *A contrario*, $\tau = 0.1$ autorise l'ajout rapide de variables parentes, nous obtenons alors un CP-net de densité maximum.

Lorsque la base de données est bruitée, l'importance d'une grande confiance envers chaque variable parente ajoutée est vérifiée (courbes rouges et marrons), avec une stagnation de la précision et de l'erreur à partir de $\tau = 0.05$. Nous choisissons pour cette raison, dans l'ensemble de nos expériences, une valeur de l'hyperparamètre $\tau = 0.05$.

Expérience n° 15 : temps d'apprentissage des versions hors ligne et en ligne

Enfin, les dernières expériences effectuées sur ces données synthétiques concernent le temps mis par les deux versions de notre algorithme pour apprendre un CP-net. Deux expériences sont menées : la première permet de mettre en exergue l'impact du nombre de variables parentes sur la durée de l'apprentissage, alors que la deuxième montre l'impact du nombre de variables sur la durée de l'apprentissage.

Les résultats de la première expérience sont synthétisés dans la Figure 3.32. La Figure 3.32a, à gauche, représente le temps pris par la version hors ligne pour apprendre un CP-net, alors que la Figure 3.32b, à droite, représente le temps pris par la version en ligne pour apprendre un CP-net sur la même base de données. Nous remarquons logiquement que la version en ligne a besoin de moitié moins de temps pour apprendre un CP-net par rapport à la version hors ligne (nous avons fixé de grandes valeurs pour les hyperparamètres δ et τ de la version en ligne afin d'apprendre un CP-net aussi dense que celui de la version hors ligne). Un point important à noter concerne la croissance, différente, du temps d'apprentissage en



(a) Version hors ligne.

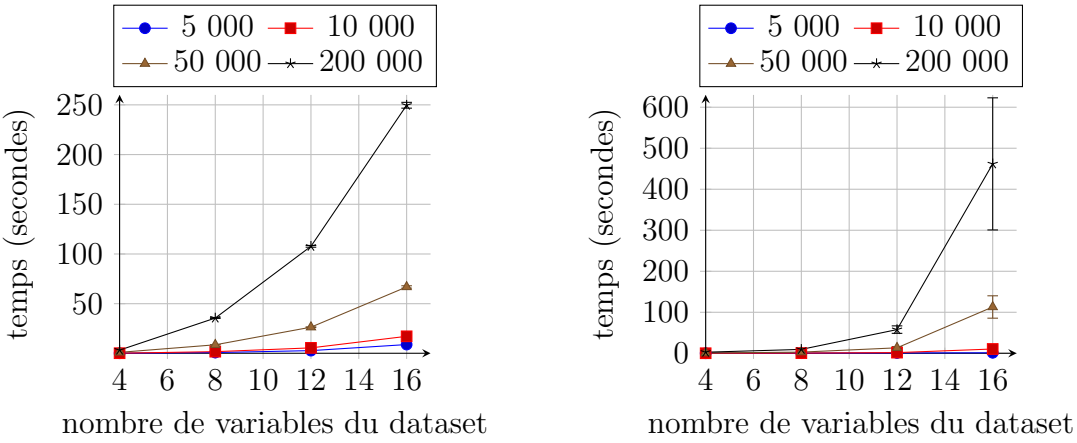
(b) Version en ligne ($\delta = 1$, $\tau = 0.5$).

FIGURE 3.32 – Temps d'apprentissage des deux versions hors ligne et en ligne en fonction du nombre maximum de parents sur le CP-net appris (le CP-net cible est non borné). Chaque courbe représente la génération d'une base de données non bruitée de taille spécifique (12 variables).

fonction du nombre de variables parentes entre les deux versions : la version en ligne croît bien plus rapidement que la version hors ligne. La raison de ce phénomène concerne le calcul du gain d'information. Dans la version hors ligne, ce gain est calculé après chaque parcours de la base, et chaque ajout d'une nouvelle variable parente, c'est-à-dire environ $\frac{12\times11}{2} = 66$ fois. Ce même calcul est effectué, dans la version en ligne, pour chaque swap nouvellement observé, c'est-à-dire 200 000 fois (dans le cas d'une base contenant 200 000 swaps). Ce calcul faisant intervenir les CP-règles de chaque variable (puisque ce nombre est doublé à chaque ajout d'une variable parente), il explique une croissance beaucoup plus forte du temps de calcul dans la version en ligne.

Expérience n° 16 : temps d'apprentissage des versions hors ligne et en ligne avec variation du nombre de variables

Nous souhaitons maintenant visionner le temps nécessaire à l'algorithme pour apprendre un CP-net lorsque le nombre de variables de chaque objet varie. Pour cela, nous générons des CP-nets à partir de 4, 8, 12, puis 16 variables, et pour chacun de ces CP-nets, nous générons un nombre défini de swaps. Cela permet ainsi, par exemple, de comparer le temps d'apprentissage d'un CP-net à 4 variables sur une base de données de 5 000 comparaisons, par rapport au temps d'apprentissage d'un CP-net à 16 variables sur 5 000 comparaisons. Les résultats sont illustrés dans la Figure 3.33. La Figure 3.33a de la version hors ligne, sur la base de données contenant 50 000 comparaisons, permet d'observer un temps d'apprentissage qui double toutes les 4 variables, ceci étant dû à la gestion des variables supplémentaires, avec le calcul du gain d'information pour chaque couple de variables possibles qui implique également la prise en compte d'un nombre exponentiel de CP-règles (en le nombre



(a) Version hors ligne.

(b) Version en ligne ($\delta = 1$, $\tau = 0.5$).

FIGURE 3.33 – Temps d'apprentissage des deux versions hors ligne et en ligne en fonction du nombre de variables parentes dans chaque jeu de données (les CP-nets cible et appris ne sont pas bornés). Chaque courbe représente la génération d'une base de données non bruitée de taille spécifique.

de variables). Ce phénomène est généralisable à toutes les tailles de jeux de données, et est particulièrement visible sur le jeu contenant 200 000 swaps. La version en ligne, quant à elle, parait beaucoup plus lente pour apprendre le CP-net lors de l'augmentation du nombre de variables sur de grandes bases de données. Comme nous l'avions exprimé précédemment, cela est dû au fait que la version en ligne a besoin de recalculer, à chaque nouvelle observation, le gain d'information de tous les couples possible, alors que la version hors ligne ne recalcule cela qu'après un parcours complet de la base de données, d'où l'explication du temps d'apprentissage presque 5 fois plus élevé pour une base de 200 000 comparaisons sur 16 variables, par rapport à la même taille de base de données sur 12 variables.

Sachant que la version en ligne doit être capable de retourner, à tout moment, le CP-net appris, le temps moyen d'une itération (*i.e.*, de la gestion d'un nouveau swap observé) est, en ce sens, un indicateur déterminant pour une gestion en flux de l'apprentissage de CP-nets. Il faut ainsi, en moyenne, moins d'une milliseconde à l'Algorithme 10 page 112 pour mettre à jour les compteurs, calculer les différents gains d'information, ajouter une éventuelle nouvelle variable parente, et mettre à jour les nouvelles CP-règles. Cela permet donc une utilisation « temps réel » acceptable de notre version en ligne, où chaque nouveau swap est traité de manière quasiment instantanée. Nous pensons également que de nombreuses optimisations du code pourraient être effectuées afin de gagner du temps de calcul, notamment en mettant à jour dynamiquement chaque gain d'information, ce qui permettrait, en théorie, de retrouver une croissance du temps de calcul pour la version en ligne équivalente à celle de la version hors ligne.

3.5.2 Données réelles

Analysons à présent le comportement de notre algorithme sur des données réelles. Trois bases de données différentes ont été utilisées : deux bases de taille modeste (TripAdvisor et SUSHI), ainsi qu'une base plus volumineuse (MovieLens). Nous divisons cette section en deux parties, la première étant réservée aux deux bases de données les plus petites, tandis que la deuxième est réservée à une analyse plus poussée de la troisième base de données.

3.5.2.1 Bases TripAdvisor et SUSHI

Nous utilisons dans cette section les bases de données suivantes :

- TripAdvisor [8] Dataset [WLZ10, WLZ11], contenant environ 240 000 évaluations [9], dont 60 000 complètes (*i.e.*, ayant tous les champs de remplis) d'utilisateurs réparties entre environ 1 800 hôtels anonymisés (ce qui donne une moyenne de 34 évaluations par hôtel). Ces évaluations sont constituées de 8 notes entre 0 et 5, dont 7 sur différents critères (l'aspect global, l'aspect des chambres, la localisation, la propreté, l'accueil, le service global, et le service spécialisé business), ainsi qu'une note globale sur l'évaluation. Nous utiliserons cette note globale comme indicateur de préférence entre deux évaluations.
- SUSHI [10] Dataset [Kam03], contenant 5 000 classements d'utilisateurs sur 100 différents types de sushis (représentés par 9 attributs qualitatifs et quantitatifs). Ce classement servira de base pour la comparaison des différents types de sushis.

Comme nous devons travailler sur des bases de données à variables binaires, des transformations sont nécessaires afin de rendre ces bases compatibles avec notre algorithme.

TripAdvisor 7 variables sont disponibles (la 8^{e} étant réservée à la comparaison des évaluations), et doivent être binarisées. Nous appliquons pour cela la transformation suivante : pour chacune des 7 variables V_i, nous calculons la moyenne de ses différentes valeurs (*i.e.*, la note moyenne obtenue), notée $\tilde{V}_i$. Nous attribuons ensuite la valeur 1 à la note courante v_{ij} de la j^{e} critique si $v_{ij} \geq \tilde{V}_i$, 0 sinon. Cela nous permet d'obtenir environ 60 000 évaluations associées à une note de 0 à 5. Nous pouvons remarquer qu'après cette binarisation, seules $2^7 = 128$ évaluations sont identifiables, nous obtenons alors beaucoup de redondance, avec de possibles incohérences où des évaluations identiques peuvent avoir des notes différentes. Nous simulons de cette manière le bruit induit par notre base de données. La dernière étape consiste à remplir notre base de données par un nombre variable de comparaisons, en prenant aléatoirement des évaluations *ceteris paribus*, puis en déterminant leur préférence via leur note globale. Cette transformation de la base de données est

8. `http://times.cs.uiuc.edu/~wang296/Data/`.
9. On emploiera également le terme de critique pour désigner une évaluation.
10. `http://www.kamishima.net/sushi/`.

différente de celle effectuée dans le Chapitre 2, afin de se rapprocher du bruit réel dans les bases de données, où les mêmes préférences peuvent apparaitre plusieurs fois. Nous verrons dans la suite de cette section que cela a un impact non négligeable sur l'apprentissage du CP-net.

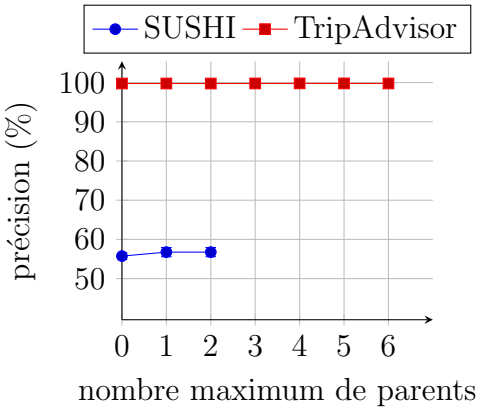
SUSHI 9 variables sont disponibles. Nous avons, pour ce jeu de données, reproduit la transformation énoncée par Liu *et al.* dans [LYX+13, Section 5.2]. Nous sélectionnons alors seulement 3 variables parmi celles disponibles : le style de sushi, son nombre de calories, ainsi que son prix normalisé. Ces trois variables sont considérées comme les variables affectant le plus les clients. Nous procédons alors à la même transformation effectuée sur la base de données de TripAdvisor : pour chacune de ces trois variables, la moyenne des valeurs est calculée. Si la valeur courante est plus grande ou égale à la moyenne, alors nous remplaçons sa valeur par 1, et 0 sinon. Enfin, nous sélectionnons deux sushis *ceteris paribus* issus d'un classement, puis nous déterminons la préférence en fonction de leurs positions dans ce classement. Nous recommençons alors la procédure jusqu'à posséder un nombre suffisant de comparaisons.

Notons que l'une des faiblesses des CP-nets est leur impossibilité à représenter les cycles directs (*i.e.*, cycles de taille 2) entre deux objets. De tels cycles rentrent en contradiction avec la notion de *ceteris paribus* : considérons l'exemple $abc \succ abc' \succ abc$. Nous avons bien un cycle direct entre les deux objets abc et abc', et du fait de la contrainte *ceteris paribus*, il ne peut pas exister une variable parente permettant leur représentation au sein d'un CP-net. De tels cycles sont naturels dans des bases de données bruitées, et n'ont en règle générale qu'un impact limité dans l'apprentissage des CP-nets (voir les expériences précédentes). Cependant, à cause du protocole de binarisation des données réelles, et du nombre restreint de variables, ces cycles directs entre deux objets sont relativement nombreux, de l'ordre de 30% à 40% aussi bien sur le jeu de données TripAdvisor que sur le jeu de données SUSHI. Pour cette raison, nous avons décidé de supprimer de tels cycles de TripAdvisor, et de les préserver dans SUSHI, afin de comparer l'impact de leur présence dans l'apprentissage.

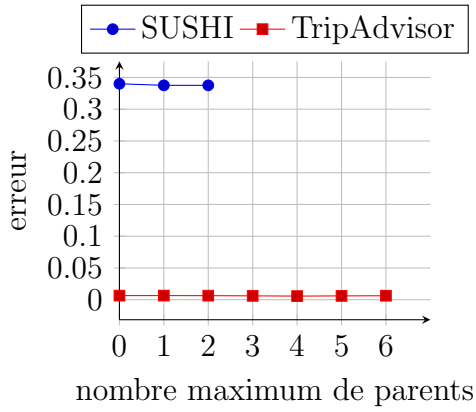
La suppression des cycles de taille 2 se fait de la manière suivante : pour chaque préférence repérée, nous comptons son nombre d'occurrences, puis nous comptons également le nombre d'occurrences de la préférence inverse. Nous gardons alors la préférence majoritaire, en supprimant l'autre version.

Expérience n° 17 : efficacité de l'apprentissage de la version hors ligne sur les jeux de données TripAdvisor et SUSHI

Les résultats de la version hors ligne sont donnés par la Figure 3.34. Contrairement aux résultats de l'expérience du chapitre précédent résumés dans le Tableau 2.2 page 71, nous pouvons observer une stagnation de la précision d'apprentissage, aussi bien sur TripAdvisor que sur SUSHI. L'ajout de variables parentes n'influence également presque pas cette précision, cela signifie que l'information apportée par un



(a) Précision *prec* (Équation (1.18)).



(b) Perte logarithmique (Équation (1.23)).

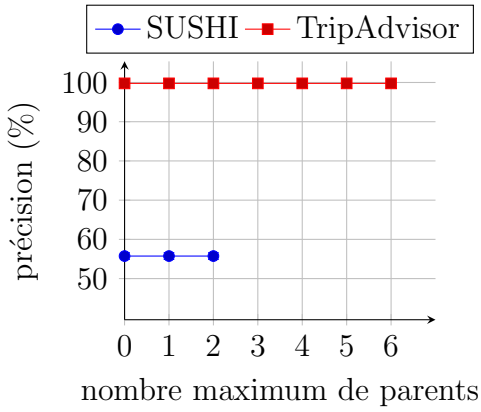
FIGURE 3.34 – Résultats de l'apprentissage de la version hors ligne en fonction du nombre maximum de parents. Chaque courbe représente une base de données réelle contenant 20 000 swaps (7 variables pour TripAdvisor, 3 variables pour SUSHI).

CP-net très dense n'apporte pratiquement pas d'informations supplémentaires par rapport à un CP-net séparable (*i.e.*, un CP-net ne contenant aucune variable parente, Définition 1.23 page 35). Nous expliquons cette différence de résultats par le fait que la base TripAdvisor ait été transformée différemment, avec, dans ce chapitre, une duplication des préférences afin d'obtenir une base plus conséquente (de 20 000 swaps). Le seuil de binarisation de chaque note a également été déterminé de manière différente, ce qui a eu comme effet d'avoir des critiques de la base d'origine ayant des représentations binaires différentes au sein des bases modifiées. Nous pensons cependant que cette expérience est plus représentative de la réalité que celle du Chapitre 2 grâce au seuil binarisation calculé par une moyenne (et non manuellement), et à la duplication des préférences. Notons en outre une forte influence, dans ces bases de données réelles, des cycles directs entre objets : des tests préliminaires ont montré une précision d'environ 67% sur TripAdvisor, là où nous obtenons, sur la Figure 3.34a, 100% de précision. Cela explique également la faible efficacité de l'apprentissage sur la base SUSHI, qui contient environ 40% de cycles directs.

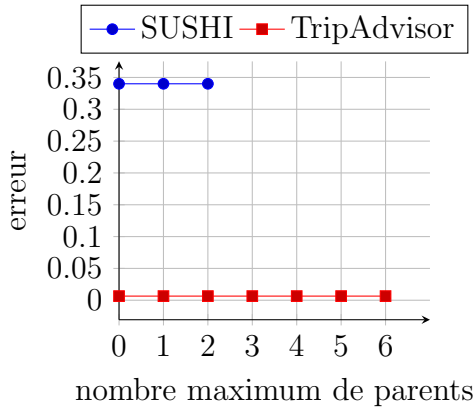
Expérience n° 18 : efficacité de l'apprentissage de la version en ligne sur les jeux de données TripAdvisor et SUSHI

Analysons maintenant les résultats de la version en ligne dans la Figure 3.35. Notons que cette version n'est que peu adaptée à cette taille de base de données, car elle ne dispose pas d'informations suffisantes pour autoriser une prise de décision sur la mise à jour du CP-net. Ainsi, la Figure 3.35 montre une précision et une erreur constantes, car aucune variable parente n'est ajoutée. Les autres résultats sont similaires à ceux obtenus dans la Figure 3.34 de la version hors ligne.

Nous avons également répété l'expérience de la version en ligne en augmentant la valeur de l'hyperparamètre τ (permettant de réguler le nombre de swaps nécessaires



(a) Précision *prec* (Équation (1.18)).



(b) Perte logarithmique (Équation (1.23)).

FIGURE 3.35 – Résultats de l'apprentissage de la version en ligne en fonction du nombre maximum de parents. Chaque courbe représente une base de données réelle contenant 20 000 swaps (7 variables pour TripAdvisor, 3 variables pour SUSHI, $\delta = 0.1$, $\tau = 0.05$).

à l'ajout de nouvelles variables parentes), afin d'autoriser l'ajout de parentes avec un nombre réduit de données. Les résultats étant cependant similaires à ceux de la version hors ligne (Figure 3.34), nous n'avons pas tracé les résultats.

3.5.2.2 Base MovieLens

Le troisième et dernier jeu de données réelles ayant servi à tester les Algorithmes 9 et 10 est une base de données issue du site Internet "MovieLens" (`https://movielens.org/`). Cette base a été récupérée sur le site `https://grouplens.org/datasets/movielens/`, et correspond au jeu de données stable contenant plus de 20 millions de notes données à environ 27 000 films par plus de 138 000 utilisateurs. Cela représente une moyenne de 144 notes par utilisateur. Chaque note est comprise entre 0.5 et 5, et varie par pas de 0.5 points. Les objets considérés dans notre étude sont les films, car il devient simple, grâce à leurs notes, de les comparer entre-eux. Chaque film est étiqueté par des genres, parmi un ensemble de 19 genres ('Adventure', 'Animation', 'Children', 'Comedy', 'Fantasy', 'Romance', 'Drama', 'Action', 'Crime', 'Thriller', 'Horror', 'Mystery', 'Sci-Fi', 'IMAX', 'Documentary', 'War', 'Musical', 'Western', et 'Film-Noir'). Rappelons que nous travaillons sur un ensemble de variables binaires. Ainsi, chaque film sera, dans notre cas, représenté par un vecteur de 19 composantes binaires, représentant chacune un genre particulier, et étant codée de la façon suivante : soit v_i^j la i^{e} composante du j^{e} film, alors

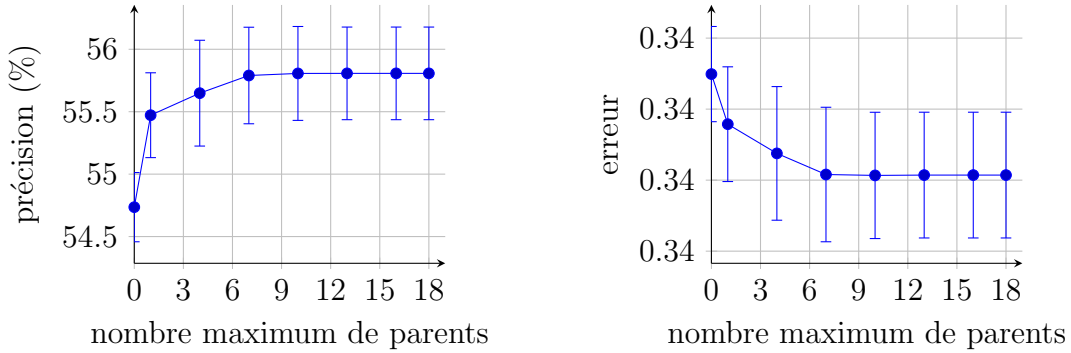
$$v_i = \begin{cases} 1 & \text{si le } j^{\text{e}} \text{ film est étiqueté par le } i^{\text{e}} \text{ genre,} \\ 0 & \text{sinon.} \end{cases}$$

Cette transformation implique que deux films sont *ceteris paribus* si l'un des deux possède un genre en moins, ou en plus, de l'autre.

La taille de ce jeu de données étant trop importante pour permettre une génération exhaustive de toutes les comparaisons *ceteris paribus* possibles en un temps

raisonnable, nous avons décidé de sélectionner aléatoirement deux films *ceteris paribus* parmi l'ensemble des 20 millions de notes, puis d'ajouter la préférence résultante dans une base de données. L'opération est répétée jusqu'à obtenir le nombre de comparaisons désiré.

Expérience n° 19 : efficacité de l'apprentissage de la version hors ligne sur le jeu de données MovieLens



(a) Précision *prec* (Équation (1.18)).

(b) Perte logarithmique (Équation (1.23)).

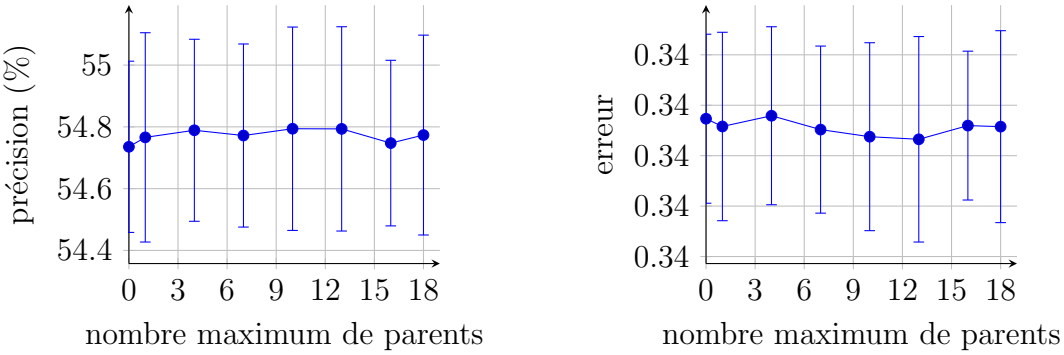
FIGURE 3.36 – Résultats de l'apprentissage de la version hors ligne en fonction du nombre maximum de parents sur une partie de la base de données MovieLens contenant 200 000 swaps définis sur 19 variables.

Les résultats d'apprentissage de la version hors ligne sur une telle base de données sont illustrés dans la Figure 3.36. Nous observons, sur ce premier test, des résultats équivalents à ceux des bases TripAdvisor et SUSHI obtenus précédemment. Notons que ce jeu de données contient environ 44% de cycles de taille 2, et que le fait de supprimer de tels cycles entraine une augmentation de la précision d'apprentissage, qui atteint alors 100% de précision.

Expérience n° 20 : efficacité de l'apprentissage de la version en ligne sur le jeu de données MovieLens

Nous effectuons des tests équivalents sur la version en ligne de notre algorithme. Ils sont exposés dans la Figure 3.37. Nous observons également le même phénomène de stagnation de la précision d'apprentissage, dont la cause principale, pour la version en ligne, est le manque de données nécessaires à l'ajout de nouvelles variables parentes.

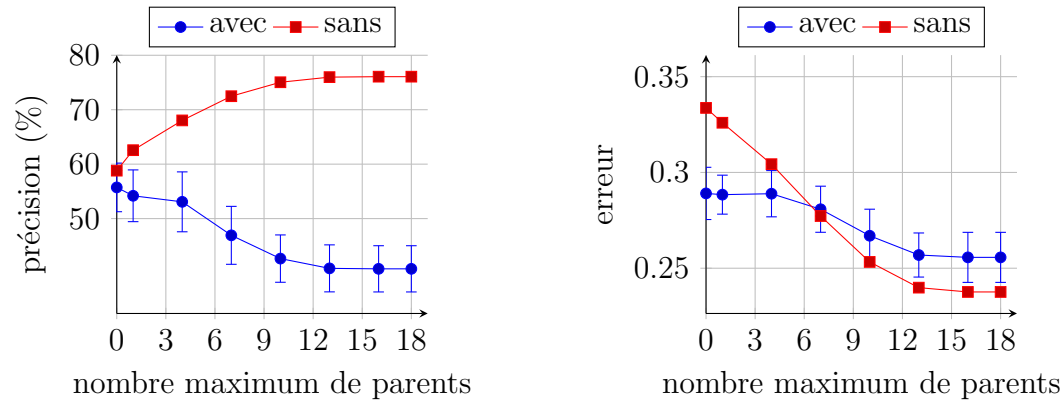
Expérience n° 21 : efficacité de l'apprentissage de la version hors ligne sur le jeu de données MovieLens ayant des préférences uniques



(a) Précision *prec* (Équation (1.18)).

(b) Perte logarithmique (Équation (1.23)).

FIGURE 3.37 – Résultats de l'apprentissage de la version en ligne en fonction du nombre maximum de parents sur une partie de la base de données MovieLens contenant 200 000 swaps définis sur 19 variables ($\delta = 0.1$, $\tau = 0.05$).



(a) Précision *prec* (Équation (1.18)).

(b) Perte logarithmique (Équation (1.23)).

FIGURE 3.38 – Résultats d'un apprentissage de la version hors ligne en fonction du nombre maximum de parents appliqué à un utilisateur fixe de la base de données MovieLens. Le jeu de données contient 1 170 swaps définis sur 19 variables. Chaque courbe correspond à la présence ou l'absence de séparation du jeu de données en données entraînement et de test lors de l'apprentissage.

Les résultats sur les données réelles étant différents des résultats observés sur des données synthétiques, nous avons cherché à connaitre les raisons d'une telle stagnation. La base de données MovieLens est constituée de plusieurs centaines de milliers d'utilisateurs ayant noté un certain nombre de films. Cette base de données multi-utilisateurs implique des préférences incohérentes causées par des notes données à un même film qui diffèrent d'un utilisateur à l'autre, créant alors des cycles au sein des préférences entre les films. Nous avons donc créé une base de données par utilisateur, où seules ses propres notes y figurent [11]. Les différentes bases générées étant relativement petites en taille, il fut alors possible de générer, de façon exhaustive,

11. Nous rappelons que les CP-nets ont été initialement pensés pour refléter les préférences d'un unique utilisateur.

toutes les préférences *ceteris paribus* (*i.e.*, les swaps) de chaque utilisateur. Notons que les préférences générées sont uniques, elles n'apparaissent qu'une et une seule fois dans la base de l'utilisateur sélectionné. De plus, les cycles directs sont proscrits. Nous obtenons, ainsi, des bases de données ayant une moyenne de 77 swaps uniques. Nous avons sélectionné la base la plus grande possible parmi l'ensemble des bases générées : il s'agit d'une base contenant 1 170 swaps uniques. Nous avons ensuite lancé un apprentissage hors ligne sur cette base, avec et sans validation croisée.

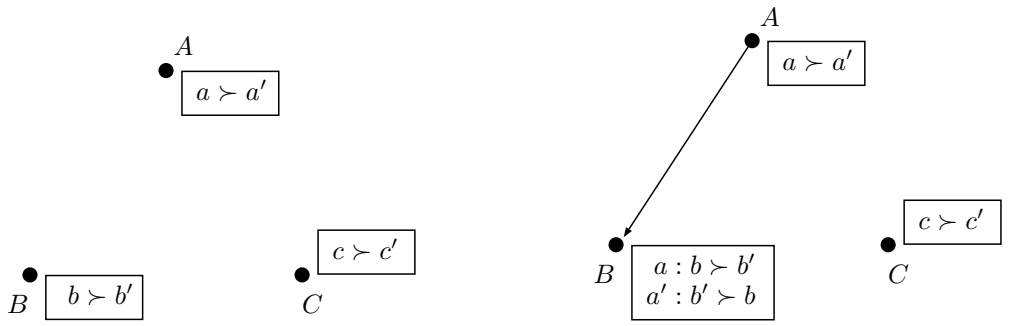
Les résultats sont disponibles dans la Figure 3.38, et montrent un fort sur-apprentissage de la version hors ligne lorsque l'apprentissage est cross-validé (courbes bleues), alors que la précision augmente de façon constante lorsque les tests sont effectués sur les données d'apprentissage (courbes rouges). Ce phénomène s'explique par le nombre très restreint de préférences disponibles alors même que le nombre de variables, donc de CP-règles potentielles, est élevé. Cela implique que lors de la validation croisée, les préférences testées n'ont jamais été aperçues précédemment, et ne peuvent donc pas être représentées. Ce phénomène est, de plus, accentué par la complexification du CP-net lors de l'ajout de variables parentes. Ce CP-net tente alors de correspondre au plus près à l'ensemble d'apprentissage, et ne parvient pas à généraliser les résultats.

Exemple 3.4. *Nous pouvons illustrer ce phénomène de sur-apprentissage sur la base de données S suivante : soient A, B, C trois variables avec $Dom(A) = \{a, a'\}$, $Dom(B) = \{b, b'\}$, et $Dom(C) = \{c, c'\}$. Considérons l'ensemble de swaps S suivant :*

s_1	$abc \succ a'bc$	s_5	$abc' \succ abc$
s_2	$abc' \succ ab'c'$	s_6	$abc' \succ ab'c'$
s_3	$abc \succ ab'c$	s_7	$a'bc \succ a'b'c$
s_4	$a'b'c \succ a'bc$		

Nous partitionnons S en deux ensembles $S_{app} = \{s_1, s_2, s_3, s_4, s_5\}$ et $S_{test} = \{s_6, s_7\}$, et nous lançons la procédure d'apprentissage de l'Algorithme 9 sur S_{app} en fixant le nombre de variables parentes à $k = 0$ et à $k = 1$. Les CP-nets obtenus sont illustrés dans la Figure 3.39. Nous pouvons ainsi voir que dans le CP-net de la Figure 3.39a, 80% *des swaps d'apprentissage sont représentés (il n'est pas possible de modéliser la préférence $abc' \succ abc$ du swap $(abc', abc)_C$ sans l'ajout d'une variable parente), contre* 100% *pour le CP-net de la Figure 3.39b. Cependant, lors du test avec S_{test},* 100% *des swaps sont représentés dans le CP-net de la Figure 3.39a, contre* 50% *dans le CP-net de la Figure 3.39b (la préférence $a'bc \succ a'b'c$ implique de fixer la valeur de A à a', qui implique dans le CP-net la préférence $b' \succ b$).*

Ce constat nous a donc amené à retenter l'expérience de la Figure 3.36 page 142 sans validation croisée. Cela n'a cependant rien changé aux résultats déjà obtenus, l'explication la plus probable étant que le jeu de données utilisé contient beaucoup de préférences doublons, présentes à la fois dans l'ensemble d'apprentissage, et dans l'ensemble de test.



(a) CP-net obtenu avec $k = 0$.

(b) CP-net obtenu avec $k = 1$.

FIGURE 3.39 – Résultats de l'apprentissage de l'Algorithme 9 sur $S_{\text{app}} = \{abc \succ a'bc, abc' \succ ab'c', abc \succ ab'c, a'b'c \succ a'bc, abc' \succ abc\}$ en fixant une borne k sur le nombre maximum de variables parentes.

3.5.3 Comparaison avec l'état de l'art

Cette section compare la version hors ligne de notre algorithme avec un autre algorithme hors ligne promettant une gestion efficace des préférences bruitées [LXW+14] (voir la Section 1.4.3.1 page 42 pour plus de détails). Pour cela, nous générons trois bases de données de taille différente (50, 500 et 1 000 swaps), sur 3 variables (à partir d'un CP-net généré aléatoirement) que nous bruitons avec différents niveaux de bruit.

Expérience n° 22 : comparaison de l'efficacité d'apprentissage entre les Algorithmes 9 et 10 et l'algorithme de [LXW+14]

Les résultats sont énoncés dans le Tableau 3.5. Ils ne font pas intervenir de validation croisée afin de correspondre au mieux au protocole décrit par Liu *et al.* dans [LXW+14] (les données de tests sont les mêmes que celles ayant servies à l'apprentissage). Nous pouvons observer une précision d'apprentissage similaire entre l'algorithme de [LXW+14] et l'Algorithme 9. La version en ligne (Algorithme 10) est également testée, mais elle n'est pas adaptée à un si petit jeu de données, ce qui explique des résultats moindres par rapport aux deux autres algorithmes (la méthode de [LXW+14] étant très lente à s'exécuter, il n'était pas possible de la lancer sur des jeux de données assez grands pour permettre à l'Algorithme 10 d'être performant). Sans pouvoir l'affirmer de manière certaine, nous pouvons émettre l'hypothèse, sur l'ensemble des trois algorithmes, d'une bonne gestion du bruit, car les précisions d'apprentissage suivent la courbe de bruit. On peut noter des résultats en hausse pour l'algorithme de Liu *et al.* pour les bases de données de 50 et de 500 swaps, ce que l'on peut expliquer par une gestion un peu moins fine du bruit.

Expérience n° 23 : comparaison de l'efficacité d'apprentissage entre l'Algorithme 10, et l'algorithme de [GAG13]

Bruit (%)	Liu *et al.* [LXW$^+$14] (%)	Algorithme 9 (%)	Algorithme 10 (%)
50 swaps			
0	100	100	86
1	99	99	85
5	96	94	82
10	92	89	79
20	85	80	72
40	77	68	62
500 swaps			
0	100	100	83
1	99	98	83
5	95	94	80
10	91	89	77
20	81	79	70
40	68	60	57
1 000 swaps			
0	100	100	82
1	99	98	81
5	95	94	79
10	90	89	75
20	80	80	69
40	60	60	57

TABLEAU 3.5 – Résultats d'apprentissage par rapport à la mesure de précision *prec* (Équation (1.18)) entre les Algorithme 9 et 10 et l'algorithme de [LXW$^+$14]. Les résultats sont lissés sur 100 exécutions (les hyperparamètres de la version en ligne sont $\delta = 0.1$ et $\tau = 0.05$).

Nous avons étudié dans le Chapitre 1 l'algorithme en ligne proposé par Guerin *et al.* [GAG13] (voir la Section 1.4.3.2 page 44). Nous proposons dans cette dernière partie une comparaison des précisions entre cet algorithme en ligne et notre Algorithme 10, sur des données synthétiques générées comme expliqué dans la Section 3.5.1 page 116. Nous fixons dans cette expérience le nombre maximum de variables parentes par variable à 5 et nous ajoutons un paramètre de **densité** du graphe du CP-net cible (le ratio moyen du nombre d'arcs par variable), noté $\lambda = \frac{|\text{arcs}|}{n}$ ($\lambda = \frac{n-1}{2}$ correspondant au graphe complet).

Nous rappelons que l'algorithme de Guerin *et al.* est basé sur les **requêtes** :

- Construire un CP-net séparable via des requêtes données à l'utilisateur ;
- Affiner ce CP-net en ajoutant des sous-ensembles de variables parentes, puis en construisant les CP-tables associées sous contrainte d'une valeur de confiance sur ces constructions.

Cet algorithme prend en compte la notion d'indécision, du fait de la création des CP-tables qui peuvent alors être incomplètes. Nous intégrons alors cette indécision

dans nos résultats, basés sur la mesure de précision des Équations (1.18) page 53 et (3.25) page 124 [12].

n	Précision (%)	Erreur (%)	Indécision (%)
		$\lambda = 1$	
4	98	2	0
	99	**1**	0
8	77	2	21
	100	**0**	**0**
12	65	2	34
	99	**1**	**0**
		$\lambda = 3$	
4	**98**	**2**	0
	94	6	0
8	80	4	16
	98	**2**	**0**
12	62	4	34
	99	**1**	**0**
		$\lambda = \frac{n-1}{2}$	
4	**98**	**2**	0
	92	8	0
8	76	2	22
	99	**1**	**0**
12	54	4	42
	98	**2**	**0**

TABLEAU 3.6 – Résultats d'apprentissage par rapport aux mesures des les Équations (1.18) et (3.25) entre l'Algorithme 10 (cellules grises) et l'algorithme de [GAG13] (cellules blanches), avec 1 500 000 comparaisons, $\delta = 0.1$, et $\tau = 0.5$. Les meilleurs résultats sont écrits en gras (n représente le nombre de variables).

Les résultats sont résumés dans le Tableau 3.6. Nous observons des valeurs de précision tournant presque toutes en faveur de notre version en ligne. Nous pouvons cependant noter que les valeurs d'erreur de l'algorithme de [GAG13] restent toujours très basses, avec une prise en compte importante de l'indécision, due au fait que lorsque l'algorithme n'accorde pas assez confiance à une nouvelle CP-règle, il ne l'ajoute pas. À l'inverse, notre algorithme va placer son seuil de confiance dans le choix et l'ajout de nouvelles variables parentes, en utilisant un vote majoritaire afin de décider quel type de CP-règle est sélectionné (en cas d'égalité entre $v \succ v'$ et $v' \succ v$, le choix est fait aléatoirement sur l'une des deux préférences). Nous pouvons, de cette manière, toujours modéliser n'importe quelle préférence, ce qui explique donc l'absence d'indécisions dans notre algorithme.

12. Notons que dans le cas de la prise en compte de l'indécision, nous notons l'indécision par $ind(S, \mathcal{N}) = 1 - (prec(S, \mathcal{N}) + err(S, \mathcal{N}))$ avec $\mathcal{N}$ un CP-net et S une base de données.

3.6 Conclusion

Nous avons présenté dans ce chapitre un nouvel algorithme d'apprentissage de CP-nets acycliques binaires. Nous proposons deux versions de cet algorithme : une version hors ligne, dans laquelle les informations présentes dans la base de données sont disponibles à tout moment, et une version en ligne, au sein de laquelle il est nécessaire d'estimer certaines informations à cause d'un flux de préférences potentiellement infini, qu'il n'est pas possible de conserver. Les deux versions de l'algorithme montrent une très bonne résistance au bruit au travers d'expériences réalisées sur des données synthétiques bruitées, et réelles. Nous avons également pu constater son efficacité lors d'une mise en concurrence avec l'algorithme de [GAG13], et où les résultats présentés sont très majoritairement de meilleure qualité.

Les résultats empiriques méritent néanmoins d'être nuancés, car nous avons pu observer un comportement très différent des deux versions de notre algorithme. Les différents tests permettent d'envisager que les CP-nets sont peu adaptés à une situation réelle à cause de deux raisons :

- Le nombre très restreint de comparaisons *ceteris paribus* dans la vraie vie, ne permettant pas une vision assez générale de toutes les CP-règles possibles. Cela entraîne un apprentissage biaisé, et donc une faible capacité de généralisation ;
- La présence de cycles de taille 2, ne pouvant être représentés par les CP-nets.

Malgré les bons résultats présentés de la version en ligne, il est nécessaire d'évoquer de nouveau la remarque décrite dans la Section 3.3.4 page 107 concernant l'interaction entre plusieurs variables parentes. Nous avons vu, dans l'exemple déroulant de la section précitée, qu'il existe une interaction très forte entre les variables parentes d'une variable, si bien que lorsque ces variables sont prises séparément, le gain d'information associé peut se retrouver très faible. L'exemple numérique, ainsi que la plupart des expériences, montrent que l'algorithme arrive à retrouver cette interaction au bout de plusieurs essais. Cela signifie cependant qu'un certain nombre de variables parentes inutiles est préservé, ce qui a pour effet

(i) de densifier inutilement le CP-net,

(ii) d'interdire l'ajout d'autres variables parentes à cause de la contrainte d'acyclicité.

Pour contourner cette difficulté, on pourrait ajouter une procédure s'exécutant après chaque ajout d'une nouvelle variable parente, et permettant de supprimer les variables parentes inutiles.
Notons de plus, dans le cas de la version en ligne, l'impossibilité de modifier en profondeur la structure du graphe appris. Cette version étant effectivement destinée à apprendre de façon constante un CP-net, nous pouvons penser que les conditionnements peuvent changer au cours du temps, et qu'une telle procédure permettrait de garder un CP-net dont la structure suit l'évolution des données d'apprentissage.

La vision du problème d'apprentissage de CP-nets prise dans ce chapitre, avec l'apprentissage de plusieurs arbres de décision, permettrait théoriquement de résoudre facilement ce problème de variables parentes devenant inutiles au cours de

l'apprentissage, en utilisant les techniques, déjà existantes, d'élagage des différents arbres (*i.e.*, de suppression de certains nœuds de l'arbre), ce qui correspond à la suppression de certaines variables parentes.

CONCLUSION

Sommaire

Travail réalisé

L'objectif de cette thèse de doctorat fut de proposer un apprentissage efficace, dans un cadre réel, d'une structure de préférences intuitive permettant une représentation compacte de celles-ci.

Nous avons ainsi introduit dans le Chapitre 1 l'utilisation des réseaux de préférences conditionnelles (CP-nets), qui possèdent une structure basée sur les préférences *ceteris paribus* et conditionnelles. Les CP-nets ont l'avantage d'être relativement intuitifs et de factoriser les préférences au sein de règles, malgré quelques limitations sur la représentation de certaines préférences, et un apprentissage difficile. Nous avons pu voir des algorithmes assez variés, dont trois ont été présentés : deux apprentissages par requêtes, et un apprentissage se basant sur l'ordre partiel des objets. Malgré la diversité des travaux sur les CP-nets, peu d'algorithmes tentaient de prendre en compte l'aspect **bruité** des préférences, essentiel dans une utilisation réelle d'une telle structure. Ce bruitage se manifestait par l'inversion de certaines préférences (à cause d'opinions divergentes, ou d'erreurs de transmissions), entrainant alors des contradictions et des cycles.

Le Chapitre 2 a présenté un nouvel algorithme hors ligne basé sur des requêtes. Nous avons montré que cet algorithme était robuste aux contradictions, sans renier sur la précision ou la vitesse d'apprentissage. Nous avons prouvé son efficacité et sa convergence grâce à une série d'expérimentations, aussi bien sur des données synthétiques que réelles. Nous avons également pu le confronter à l'algorithme de requêtes

en ligne de [GAG13], ainsi qu'à l'algorithme hors ligne de [LXW+14] traitant des incohérences dans les préférences, dont les résultats sont pour la majorité meilleurs.

Le Chapitre 3 a proposé deux versions d'un nouvel algorithme :

- Une version hors ligne, basée sur une mesure entropique, permettant d'apprendre un arbre de décision par variable, chaque arbre étant représentatif des différentes variables parentes associées à la variable courante. La variable conditionnée, ainsi que la variable parente sont choisies simultanément, en fonction de leur gain d'information ;
- Une version en ligne, permettant notamment de répondre à la dernière question posée en introduction concernant la recherche d'un algorithme supportant le passage à l'échelle. Cette version reprend comme base le fonctionnement de la version hors ligne via un apprentissage simultané de plusieurs arbres de décision en se basant sur la mesure de gain d'information à maximiser, tout en ajoutant une notion de compteur permettant d'estimer les informations précédemment observées. La contrainte de temps de décision relative aux versions en ligne est ici résolue grâce à l'utilisation de la borne de McDiarmid. La fonction de décision associée à cette borne se base également sur la même mesure d'information des variables, permettant alors d'avoir une gestion plus ou moins efficace du bruit en fonction de la valeur d'un intervalle de confiance fixé manuellement par l'utilisateur de l'algorithme.

Nous avons montré, au travers de plusieurs expériences et de comparaisons avec les algorithmes de [GAG13] et de [LXW+14], que ces deux versions se montrent efficaces, aussi bien sur la qualité de l'apprentissage que sur le temps pris par les algorithmes pour s'exécuter. Ces expériences ont de plus montré de façon empirique leur convergence, et ont mis en exergue le fait que certaines préférences ne semblent pas conditionnelles (une modélisation des préférences par des CP-nets séparables semble alors suffisante).

Bilan

Les différents tests effectués semblent montrer que l'utilisation de CP-nets comme structure permettant la représentation de préférences, dans la vie réelle, ne s'avère pas toujours efficace (voir les expériences effectuées sur les critiques d'hôtels de TripAdvisor au sein des Figures 3.34 page 140 et 3.35 page 141, dont l'ajout de variables parentes, donc de conditions, n'apporte aucune amélioration de la précision d'apprentissage). De plus, la présence de nombreux cycles de taille 2 ne permet pas une utilisation pertinente des CP-nets du fait que ces cycles rentrent en contradiction avec la notion *ceteris paribus*. De telles préférences sont, par ailleurs, relativement rares dans la vie réelle, comme les expériences sur le jeu de données MovieLens l'ont montré. Les utilisateurs préfèrent effectivement exprimer des préférences plus naturelles, n'ayant aucune propriété de condition *a priori* (et encore moins de structures

sous-jacentes). Un travail sur les bases de données de préférences, est alors actuellement nécessaire afin d'obtenir des préférences compatibles avec de tels réseaux. Cette faible proportion de préférences *ceteris paribus* entraîne de plus un manque de généralisation des algorithmes d'apprentissage, car les préférences d'apprentissage constituent une très faible part de l'ensemble des préférences possibles.

Les CP-nets peuvent cependant s'avérer très utiles dans certains contextes particuliers tels que la représentation de préférences au sein de restaurants (exemple classiquement utilisé pour expliquer de telles structures), ou bien dans les films. Plus généralement, l'utilisation de ces structures semble plutôt pertinent pour tout contexte faisant intervenir des variables qualitatives sans notion d'ordre *a priori* dans les valeurs de chaque variable. Le nombre de ces valeurs ne doit en outre pas être trop important, sous peine d'avoir une explosion de la complexité d'apprentissage des CP-nets.

Les Chapitres 2 et 3 montrent que l'apprentissage de telles structures, pour une utilisation en milieu réel, peut également être envisagé : il est possible d'apprendre un CP-net en ligne, et d'éviter de manière assez robuste le bruit inhérent aux bases de données traitées. Cependant, cet apprentissage s'accompagne de contraintes non négligeables telles que le fait d'apprendre des CP-nets

(i) acycliques,

(ii) reposants sur des variables binaires.

Ces deux contraintes impliquent une recherche encore plus poussée dans l'étude algorithmique de telles structures.

Perspectives

En considérant le bilan précédemment exposé, il apparait que les recherches dans le domaine de l'apprentissage de structures conditionnelles sont encore nombreuses. La recherche d'algorithmes efficaces permettant un apprentissage de CP-nets plus généraux est bien sûr à considérer, tels que l'utilisation :

- De préférences n'ayant pas forcément la propriété de *ceteris paribus* : comment, alors, permettre un apprentissage efficace de CP-nets avec de telles structures ?
  Nous pouvons ainsi imaginer une procédure permettant un apprentissage sur des données incomplètes, via plusieurs apprentissages parallèles éludant, pour chacun d'entre-eux, les variables non *ceteris paribus* ;
- De variables n'étant pas forcément binaires : comment, alors, adapter nos algorithmes pour préserver un apprentissage rapide malgré l'explosion combinatoire des potentielles préférences conditionnelles du CP-net ?
  Les versions hors ligne et en ligne présentées dans le Chapitre 3 peuvent facilement être adaptées pour des variables non binaires. De plus, la rapidité de l'apprentissage doit pouvoir être, dans une moindre mesure, préservée, grâce à une parallélisation de certaines tâches, notamment du calcul des gains d'information. Notons par ailleurs que la relaxation de la contrainte de binarité

des variables influera sur la complexité temporelle des algorithmes par l'augmentation de la base exponentielle qui est, dans le cas binaire, de 2^k (avec k le nombre de variables parentes) ;

- De CP-nets cycliques, permettant alors une représentation plus complète des différentes préférences, tout en assurant une cohérence au sein de l'ordre partiel des objets associé, comme peuvent le faire Liu *et al.* dans [LXW+14], mais de manière en ligne.

  Notons que pour ce point précis, une généralisation des versions de l'algorithme du Chapitre 3 a rapidement été introduite dans la Section 3.4 page 113, permettant alors l'apprentissage de CP-nets cycliques n'assurant cependant pas une cohérence de l'ordre partiel des objets ;

- De CP-nets agrégés, via la recherche de fonctions de distance.

  Des travaux sur l'agrégation de CP-nets ont été initiés depuis quelques années [LMX12, GLMR14]. Nous souhaiterions cependant, au travers de cette question, agréger des CP-nets en fonction de la structure de leur graphe dirigé. Nous pourrions alors envisagé un **bagging**, c'est-à-dire une agrégation de CP-nets, où chacun de ces CP-nets serait issu de l'apprentissage d'une partie d'un ensemble de données. Cette piste serait relativement nouvelle dans le domaine des CP-nets grâce au fait qu'elle mette en œuvre l'aspect structurel du graphe du CP-net, comme initié dans l'Annexe A.

La définition du problème de l'apprentissage de CP-nets introduite dans la Section 1.5 page 50 présente également l'intérêt fort de permettre l'utilisation d'algorithmes d'apprentissage plus génériques, et ouvre ainsi la voie à des techniques d'apprentissages profonds par exemple. Cela aurait notamment l'avantage d'intégrer de façon forte la parallélisation de certaines tâches au sein de l'apprentissage de CP-nets.

Cependant, nous pensons que la démocratisation de telles structures conditionnelles, implémentées au sein de systèmes réels, viendra de l'apprentissage (encore inexistant) des extensions des CP-nets, à l'image des TCP-nets, qui permettent d'étendre les propriétés initiales des CP-nets avec une représentation bien plus complète des préférences, malgré une structure plus complexe mais toujours graphique et relativement intuitive.

L'ensemble de ces améliorations permettrait alors une utilisation acceptable des structures de préférences conditionnelles dans certains contextes de recommandation, avec un système complet partant de l'apprentissage de telles structures, jusqu'à leur utilisation en tant que système de recommandation via une analyse de leur structure. Des efforts ont par ailleurs été initiés dans ce sens, avec une tentative de fouille des préférences au sein du CP-net nouvellement appris, afin de recommander des objets de façon intuitive en se basant sur les propriétés de cette structure (voir l'Annexe A).

ANNEXE A

ÉTUDE STRUCTURELLE DES CP-NETS ACYCLIQUES

Sommaire

Résumé

Cette annexe présente une étude structurelle des CP-nets, en introduisant des opérations possibles à effectuer sur le graphe acyclique des CP-nets (suppression de variables, paramétrisation des CP-nets). Nous introduirons également l'analyse formelle des concepts, ainsi que les contextes de préférences, en proposant un lien entre ces contextes et les CP-nets.

Liste des contributions de cette annexe :

- Étude du graphe du CP-net avec des propositions de suppression de variables et de paramétrisation du CP-net.
- Liens entre les CP-nets et les concepts de préférences.
- Conjecture sur la structure de treillis de l'ordre des objets associé aux CP-nets ayant une structure d'arbre.

Cette étude constitue un travail commun avec Miguel Conceiro, Quentin Brabant, et Amedeo Napoli de l'INRIA Nancy.

A.1 Introduction

Les réseaux de préférences conditionnelles (CP-nets) acycliques ont été, au sein de ce manuscrit, principalement étudiés d'un point de vue algorithmique, afin d'en proposer un apprentissage efficace (voir les Chapitres 2 et 3). Une fois apprises, de telles structures peuvent s'avérer très utiles à des fins de recommandation, via l'utilisation de leur graphe orienté, ou de leur ordre partiel des objets associé. La première partie de cette annexe présentera donc une analyse des CP-nets (via la suppression de variables et l'incidence d'une telle opération sur le graphe du CP-net, ou la fixation de certaines valeurs des variables afin de paramétrer le CP-net).

L'analyse formelle des concepts [GW99, Wil09] est une méthode d'analyse de données permettant la recherche et l'extraction de concepts au sein d'un ensemble de données. Cette méthode constitue donc un moyen efficace de fouiller des données au travers d'opérateurs qui extraient de l'information sans avoir besoin de parcourir l'ensemble des données, qui admettent une structure bien spécifique. Les CP-nets constituant une structure qui factorise les différentes préférences au sein d'objets combinatoires, nous avoir pu voir que la taille de l'ordre partiel des objets associé à un CP-net possède une taille exponentielle par rapport au nombre de variables du CP-net. L'analyse formelle des concepts s'avère alors être une méthode idéale dans le cadre d'une fouille des données de l'ordre partiel des objets obtenu après l'apprentissage d'un CP-net par les algorithmes des Chapitres 2 et 3. La deuxième partie de cette annexe traitera ainsi de l'analyse formelle des concepts, où nous nous concentrerons notamment sur un lien existant entre les CP-nets et la fouille de préférences [Obi12, OIKZ12, Obi13].

A.2 Suppression de variables

Il peut être intéressant d'analyser ce qu'il se passe au sein d'un CP-net lorsque nous tentons de supprimer une (ou plusieurs) variable(s). Cette transformation sera

notamment utile dans la suite de ce rapport, lors de l'étude structurelle des CP-nets.

Soit $\mathcal{N} = (\mathbf{V}, \mathbf{A}, CPT(\mathbf{V}))$ un CP-net. Nous avons vu dans la Section 1.4.1.3 page 32 l'ordre partiel des objets associé au CP-net $\mathcal{N}$. Nous représentons dans ce chapitre cet ordre partiel par un graphe orienté, noté $G_{\mathcal{N}} = (Dom(\mathbf{V}), \mathbf{A}')$, où $Dom(\mathbf{V})$ correspond à l'ensemble des objets de l'ordre partiel, et $\mathbf{A}'$ l'ensemble des arcs du graphe tel que $(\mathbf{o}, \mathbf{o}')_V \in \mathbf{A}'$ (avec $V \in \mathbf{V}$) si et seulement si $\mathbf{o} \succ_{\mathcal{N}} \mathbf{o}'$ et $\nexists \mathbf{o}'' \in Dom(\mathbf{V})$ tel que $\mathbf{o} \succ_{\mathcal{N}} \mathbf{o}'' \succ_{\mathcal{N}} \mathbf{o}'$ (réduction transitive du graphe).

Soient $\mathbf{o} \in Dom(\mathbf{V})$ un objet et $Q \in \mathbf{V}$ une variable. Nous notons $\mathbf{o}_{\setminus\{Q\}}$ l'état au sein duquel la valeur de la variable Q a été supprimée, *i.e.*, $\mathbf{o}_{\setminus\{Q\}} \in Dom(\mathbf{V}'), \mathbf{V}' = \mathbf{V} \setminus \{Q\}$. Nous généralisons cette notation à un ensemble $\mathbf{Q}$ de variables : soient $\mathbf{o} \in Dom(\mathbf{V})$ un objet et $\mathbf{Q} \subseteq \mathbf{V}$ un ensemble de variables. Nous notons $\mathbf{o}_{\setminus\mathbf{Q}}$ l'état au sein duquel la valeur de chaque variable de $\mathbf{Q}$ a été supprimée, *i.e.*, $\mathbf{o}_{\setminus\mathbf{Q}} \in Dom(\mathbf{V}_{\mathbf{Q}}), \mathbf{V}_{\setminus\mathbf{Q}} = \mathbf{V} \setminus \mathbf{Q}$.

Notons $\mathbf{V}_{\setminus\mathbf{Q}}, \mathcal{N}_{\setminus\mathbf{Q}} = (\mathbf{V}_{\setminus\mathbf{Q}}, \mathbf{A}, CPT(\mathbf{V}_{\setminus\mathbf{Q}}))$ et $G_{\mathcal{N}_{\setminus\mathbf{Q}}} = (Dom(\mathbf{V}_{\setminus\mathbf{Q}}), \mathbf{A}')$ respectivement un ensemble de variables, un CP-net associé à cet ensemble et son ordre partiel des objets privés de l'ensemble des variables $\mathbf{Q}$. Cette suppression entraîne une fusion de plusieurs objets : $\forall Q \in \mathbf{Q}$, si $\mathbf{o} = q\mathbf{x}$ et $\mathbf{o}' = q'\mathbf{x}$, $q, q' \in Dom(Q)$, alors $\mathbf{o}_{\setminus Q} = \mathbf{o}'_{\setminus Q}$. La suppression d'une variable va diviser par deux le nombre d'objets du graphe $G_{\mathcal{N}}$. Plus généralement, pour un ensemble $\mathbf{Q}$ de variables à supprimer, le nombre d'objets de $G_{\mathcal{N}}$ va être divisé par $2^{|\mathbf{Q}|}$, dans le cas où $|Dom(Q)| = 2$, $\forall Q \in \mathbf{Q}$.

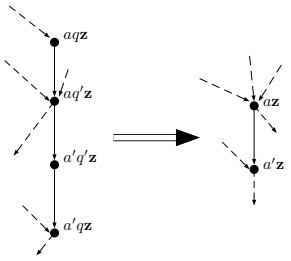
A.2.1 Suppression de variables non parentes

Soit $\mathbf{Q} \subseteq \mathbf{V}$ tel que $\forall V \in \mathbf{V}, \forall Q \in \mathbf{Q}, Q \notin Pa(Y)$ (*i.e.*, aucune variable de $\mathbf{Q}$ n'est parente). Cela implique que $\forall (\mathbf{o}, \mathbf{o}') \in Dom(\mathbf{V})^2$ et $\forall Q \in \mathbf{Q}$ avec Q la variable de swap de $\mathbf{o}$ et $\mathbf{o}'$, si $\mathbf{o} \succ \mathbf{o}'$, $\nexists \mathbf{o}'' \in Dom(\mathbf{V})$ tel que $\mathbf{o} \succ \mathbf{o}'' \succ \mathbf{o}'$ (*i.e.*, $\mathbf{o}$ et $\mathbf{o}'$ sont forcément adjacents dans $G_{\mathcal{N}}$).

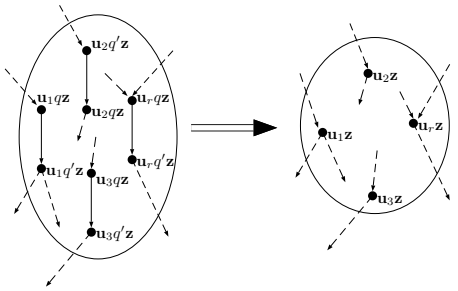
En supposant que Q n'est pas parente, trois types de suppressions sont alors distinguables dans $G_{\mathcal{N}}$:

- Variable parente de Q lorsque $|Pa(Q)| = 1$: soit $Y \in Pa(Q)$, alors il existe une séquence de swaps au sein de $G_{\mathcal{N}}$ ayant la forme $qy\mathbf{z} \succ q'y\mathbf{z} \succ q'y'\mathbf{z} \succ qy'\mathbf{z}$ avec $q, q' \in Dom(Q), y, y' \in Dom(Pa(Q)), \mathbf{z} \in Dom(\mathbf{Z}), \mathbf{Z} = \mathbf{V} \setminus \{Q, Y\}$. Après suppression de Q, cette séquence devient $y\mathbf{z} \succ y'\mathbf{z}$ (voir la Figure A.1a).
- Variables parentes de Q lorsque $|Pa(Q)| > 1$: $\forall \mathbf{u} \in Dom(Pa(Q))$, il existe une séquence de swaps au sein de $G_{\mathcal{N}}$ ayant la forme $\mathbf{u}q\mathbf{z} \succ \mathbf{u}q'\mathbf{z}$ avec $q, q' \in Dom(Q)$ et $\mathbf{z} \in Dom(\mathbf{Z}), \mathbf{Z} = \mathbf{V} \setminus (Pa(Q) \cup \{Q\})$. Après suppression de Q, cette séquence se réduit à un seul état $\mathbf{uz}$ (voir la Figure A.1b).
- Variables indépendantes de Q : $\forall Y \notin Pa(Q)$, il existe deux séquences de swaps au sein de $G_{\mathcal{N}}$ ayant la forme $qy\mathbf{z} \succ q'y\mathbf{z} \succ q'y'\mathbf{z}$ et $qy\mathbf{z} \succ qy'\mathbf{z} \succ q'y'\mathbf{z}$ avec $q, q' \in Dom(Q), y, y' \in Dom(Y), \mathbf{z} \in Dom(\mathbf{Z}), \mathbf{Z} = \mathbf{V} \setminus \{Q, Y\}$. Après suppression de Q, ces séquences se réduisent à $y\mathbf{z} \succ y'\mathbf{z}$ (voir la Figure A.1c).

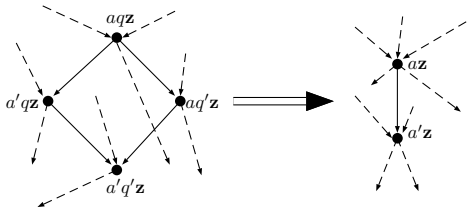
Cela signifie que la suppression d'une variable va raccourcir toutes les chaines de préférences de l'ensemble de ses variables parentes et supprimer l'incomparabilité de toutes les autres variables par rapport à elle (variables indépendantes). Un exemple de suppression est donné dans la Figure A.2.



(a) Nous supposons ici que $A \in \mathbf{V}$ est la seule variable parente de Q avec $a \succ a'$ ($Dom(A) = \{a, a'\}$), $a : q \succ q'$ et $a' : q' \succ q$. $\mathbf{z} \in Dom(\mathbf{Z}), \mathbf{Z} = \mathbf{V} \setminus \{A, Q\}$.



(b) Nous supposons ici que $|Pa(Q)| > 1$ et que $Dom(Pa(Q)) = \{\mathbf{u}_1, \ldots, \mathbf{u}_r\}$ ($r = 2^{|Pa(Q)|}$) avec des préférences quelconques sur les valeurs de Q en fonction des différents $\mathbf{u}_i, i = 1, \ldots, r$. $\mathbf{z} \in Dom(\mathbf{Z}), \mathbf{Z} = \mathbf{V} \setminus (Pa(Q) \cup \{Q\})$.



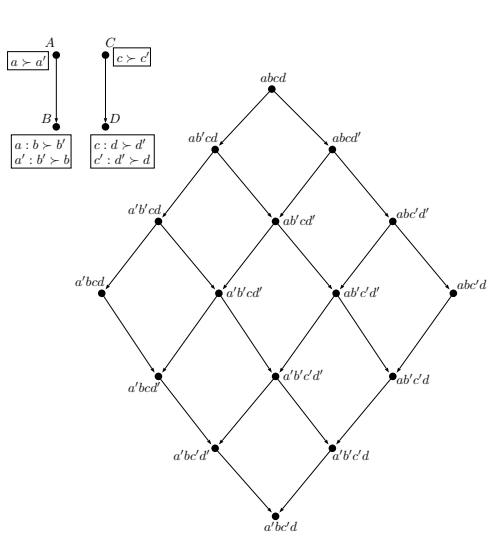
(c) Nous supposons ici que $A \notin Pa(Q)$ ($A \in \mathbf{V}$) avec $a \succ a'$ ($Dom(A) = \{a, a'\}$) et $q \succ q'$. $\mathbf{z} \in Dom(\mathbf{Z}), \mathbf{Z} = \mathbf{V} \setminus \{A, Q\}$.

FIGURE A.1 – Suppression de la variable $Q \in \mathbf{V}$ d'un CP-net $\mathcal{N}$ sur un ensemble de variables $\mathbf{V}$ tel que Q n'est pas parente et $Dom(Q) = \{q, q'\}$.

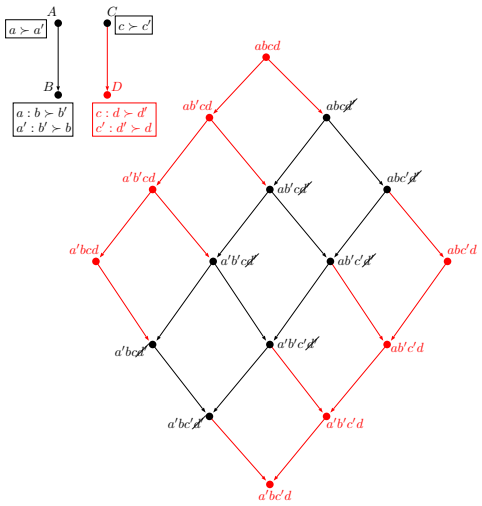
A.2.2 Suppression de variables parentes

Supposons $\mathbf{o} \in Dom(\mathbf{V})$ un objet dont nous voulons supprimer une variable $Q \in \mathbf{V}$. Cela entraîne la réunion de deux objets : $\mathbf{o} = q\mathbf{z}$ et $\mathbf{o}' = q'\mathbf{z}$ avec $q \succ q', q, q' \in Dom(Q), \mathbf{z} \in Dom(\mathbf{V} \setminus \{Q\})$. Supposons ces objets adjacents dans $G_{\mathcal{N}}$, par exemple les objets $a'b'cd$ et $ab'cd$ dans l'exemple de la Figure A.2a. Nous pouvons voir que dans ce cas-ci, la variable à supprimer est A qui est parente. Il est alors nécessaire de supprimer, en plus de A, l'ensemble de ses descendants et des parents de ses descendants, afin d'éviter des problèmes lors de la réunion de swaps non adjacents. Deux cas de suppression groupée peuvent être identifiés :

- Plusieurs variables non parentes possèdent Q comme unique parent commun. Plus formellement $\exists \mathbf{X} \subseteq \mathbf{V} \setminus \{Q\}$ tel que $Q \in Pa(X), |Pa(X)| = 1$ et $\forall Y \in \mathbf{V}, X \notin Pa(Y), \forall X \in \mathbf{X}$ (voir la Figure A.3a).
- Il existe une chaine partant d'une variable non parente et remontant jusqu'à Q. Plus formellement, on définit une fonction ch pour $Q \notin \mathbf{X}$, et $\mathbf{X} \subseteq \mathbf{V}$ de la



(a) CP-net d'origine $\mathcal{N}$.



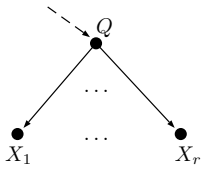
(b) CP-net $\mathcal{N}_{\setminus\{D\}}$ sur lequel nous avons retiré la variable D.

FIGURE A.2 – Exemple de CP-net $\mathcal{N}$ sur $\mathbf{V} = \{A, B, C, D\}$.

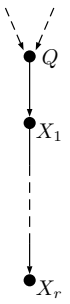
façon suivante :

$$ch(Q) = \begin{cases} \{Y \cup Ch(Y) & | \ Q \in Pa(Y), |Pa(Y)| = 1, Y \in \mathbf{V} \\ & \text{et} \ \nexists W \in \mathbf{V} \setminus \{Y\} \text{ tel que } Q \in Pa(W)\} \\ & \text{avec } Y \in Pa(X), \forall X \in \mathbf{V}, \\ \{Y & | \ Q \in Pa(Y), |Pa(Y)| = 1, Y \in \mathbf{V} \\ & \text{et} \ \nexists W \in \mathbf{V} \setminus \{Y\} \text{ tel que } Q \in Pa(W)\} \\ & \text{avec } Y \notin Pa(X), \forall X \in \mathbf{V}. \end{cases} \tag{A.1}$$

Il faut donc enlever l'ensemble $\mathbf{X} = ch(Q) \cup \{Q\}$ (voir la Figure A.3b).



(a) Q possède r variables filles. Par conséquent, il n'est pas possible de déterminer une préférence des valeurs d'une variable $X_i, i = 1, \ldots, r$ sans Q.



(b) Une chaine existe entre X_r et Q. D'après les cas précédents, il n'est pas possible de supprimer Q sans supprimer la variable fille, ce qui s'applique récursivement jusqu'à X_r qui n'est pas parente.

FIGURE A.3 – Suppression de la variable Q d'un CP-net $\mathcal{N}$ sur $\mathbf{V}$ tel que Q est parente.

De manière générale, la suppression d'une variable parente Q est une combinaison

des deux cas de base présentés dans la Figure A.3. Un exemple de suppression de variable parente est donné dans la Figure A.4.

Remarque : soit $\mathcal{N} = (\mathbf{V}, \mathbf{A}, CPT(\mathbf{V}))$ un CP-net, et soit $\mathbf{V}_1, \mathbf{V}_2 \subseteq \mathbf{V}$ une partition de $\mathbf{V}$ ($\mathbf{V}_1 \cup \mathbf{V}_2 = \mathbf{V}, \mathbf{V}_1 \cap \mathbf{V}_2 = \emptyset$) tel que l'ensemble $\mathbf{V}_1$ provient de la formule précédente. Notons $G_{\mathcal{N}_{\setminus \mathbf{V}_2}} = (Dom(\mathbf{V}_2), \mathbf{A}')$ l'ordre partiel des objets engendré par le sous-CP-net $\mathcal{N}_{\setminus \mathbf{V}_2} = (\mathbf{V}_2, \mathbf{A}, CPT(\mathbf{V}_2))$. Le graphe $G_{\mathcal{N}_{\setminus \mathbf{V}_2}}$ va apparaitre $2^{|\mathbf{V}_1|}$ fois. Formellement, $\forall \mathbf{o}_1 \in Dom(\mathbf{V}_1), \exists G_{\mathcal{N}_{\setminus \mathbf{V}_2}}$ car $\mathcal{N}_{\setminus \mathbf{V}_2}$ est indépendant de $\mathcal{N}_{\setminus \mathbf{V}_1}$ (voir la Figure A.5 ci-dessous).

La suppression de variables dans un CP-net, qu'elles soient parentes ou non, a donc pour incidence de supprimer des sous-graphes des objets identiques au sein de l'ordre partiel global des objets. Notre objectif est de « fouiller » cet ordre partiel des objets. La suppression peut avoir un effet relativement radical, aussi bien d'un point de vue structurel qu'en complexité.

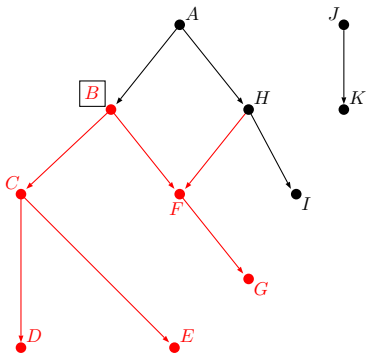


FIGURE A.4 – Exemple de suppression d'une variable parente. Nous supprimons la variable B du CP-net, ce qui implique de supprimer également les variables C, D, E, F, G.

A.3 CP-net paramétré

Soit $\mathcal{N} = (\mathbf{V}, \mathbf{A}, CPT(\mathbf{V}))$ un CP-net et soit $\mathbf{V}_1, \mathbf{V}_2 \subseteq \mathbf{V}$ une partition de $\mathbf{V}$ ($\mathbf{V}_1 \cup \mathbf{V}_2 = \mathbf{V}, \mathbf{V}_1 \cap \mathbf{V}_2 = \emptyset$). On appelle **CP-net paramétré** par $\mathbf{o} \in Dom(\mathbf{V}_1)$ le sous-CP-net noté $\mathcal{N}^{\mathbf{o}}$ engendré par les variables de $\mathbf{V}_2$ lorsque les valeurs des variables de $\mathbf{V}_1$ sont fixées.

A.3.1 Notations

Définition A.1 (Conditionnement existentiel et universel entre deux groupes de variables)**.**
Soient $\mathcal{N} = (\mathbf{V}, \mathbf{A}, CPT(\mathbf{V}))$ *un CP-net,* $\mathbf{V}_1 \subseteq \mathbf{V}$ *et* $\mathbf{V}_2 = \mathbf{V} \setminus \mathbf{V}_1$*. On dit que* $\mathbf{V}_1$ ***conditionne existentiellement*** *(resp.* ***conditionne universellement****)* $\mathbf{V}_2$ *ssi*

- $\exists V_1 \in \mathbf{V}_1$ *(resp.* $\forall V_1 \in \mathbf{V}_1$*),* $\exists V_2 \in \mathbf{V}_2$ *tel que* $V_1 \in Pa(V_2)$*,*
- $\nexists V_2 \in \mathbf{V}_2$ *tel que* $V_2 \in Pa(V_1), \forall V_1 \in \mathbf{V}_1$*.*

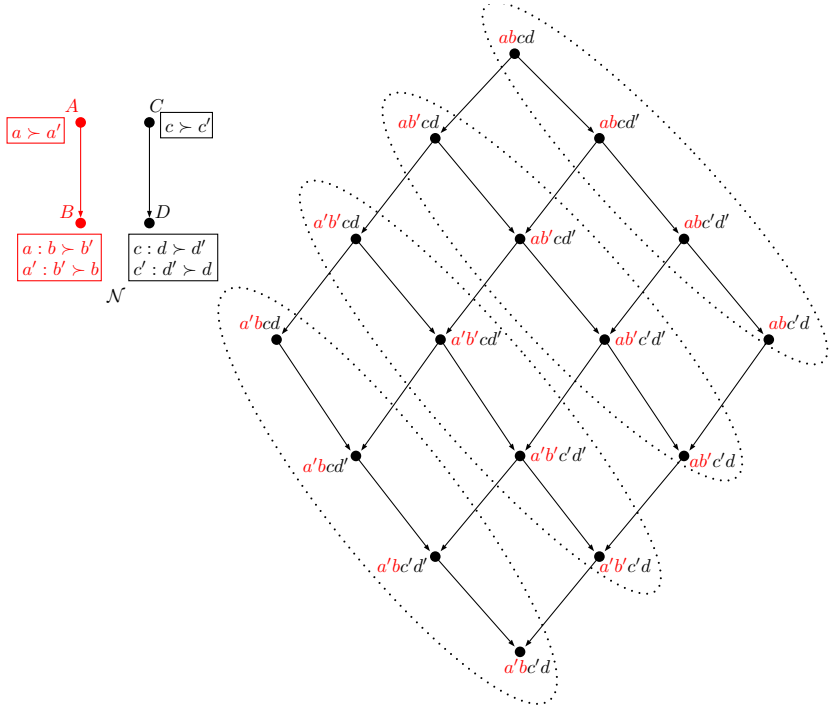


FIGURE A.5 – Nous supprimons les variables A et B sur l'exemple de la Figure A.2a page 159. Nous pouvons voir l'ordre partiel des objets $G_{\mathcal{N}_{\setminus\{A,B\}}}$ du sous-CP-net $\mathcal{N}_{\setminus\{A,B\}}$ (une chaine de quatre éléments) apparaitre $2^{|\{A,B\}|} = 2^2 = 4$ fois (les éléments en rouge ne sont plus considérés).

Cette réduction implique alors un ordre partiel des objets de la forme $G_{\mathcal{N}^{\mathbf{o}}} = (\{\mathbf{z} = \mathbf{ou} \mid \mathbf{o} \in Dom(\mathbf{V}_1), \forall \mathbf{u} \in Dom(\mathbf{V}_2)\}, \mathbf{A}')$. Un exemple d'une telle paramétrisation est donné dans la Figure A.6. Il est important de remarquer que plusieurs CP-nets paramétrés identiques existent. Ainsi, dans l'exemple de la Figure A.2a page 159, nous pouvons voir que $G_{\mathcal{N}^{ab}} = G_{\mathcal{N}^{a'b}} = G_{\mathcal{N}^{ab'}} = G_{\mathcal{N}^{a'b'}}$. Cela s'explique par la Définition A.1 : si l'état $\mathbf{o}_1 \in Dom(\mathbf{V}_1)$ qui paramétrise le CP-net n'est constitué que de variables $\mathbf{V}_1$ qui ne conditionnent pas les autres variables $\mathbf{V}_2$, alors quelque soit l'état appartenant à $Dom(\mathbf{V}_1)$, nous aurons $2^{|\mathbf{V}_1|}$ CP-nets paramétrés identiques. Au contraire, si $\mathbf{V}_1$ conditionne universellement les variables de $\mathbf{V}_2$, alors tous les CP-nets paramétrés par un état de $Dom(\mathbf{V}_1)$ seront différents, quelque soit l'état. Nous pouvons ainsi voir que $G_{\mathcal{N}^{ac}} \neq G_{\mathcal{N}^{a'c}} \neq G_{\mathcal{N}^{ac'}} \neq G_{\mathcal{N}^{a'c'}}$. De manière générale, $2^{|\mathbf{V}'_1|}$ CP-nets paramétrés seront identiques (resp. différents) lorsque $\mathbf{V}'_1 \subseteq \mathbf{V}_1$ conditionne (resp. ne conditionne pas) universellement $\mathbf{V}_2$.

Nous pouvons également remarquer que la paramétrisation d'un CP-net permet d'obtenir un nouveau CP-net plus restreint au sein duquel les variables fixées ont disparues. Ainsi, dans l'exemple de la Figure A.6, la variable A étant fixée, nous ne gardons que les préférences « compatibles » avec la valeur de A. Les valeurs de la variable B ne possèdent alors plus qu'une seule préférence relative à a'.

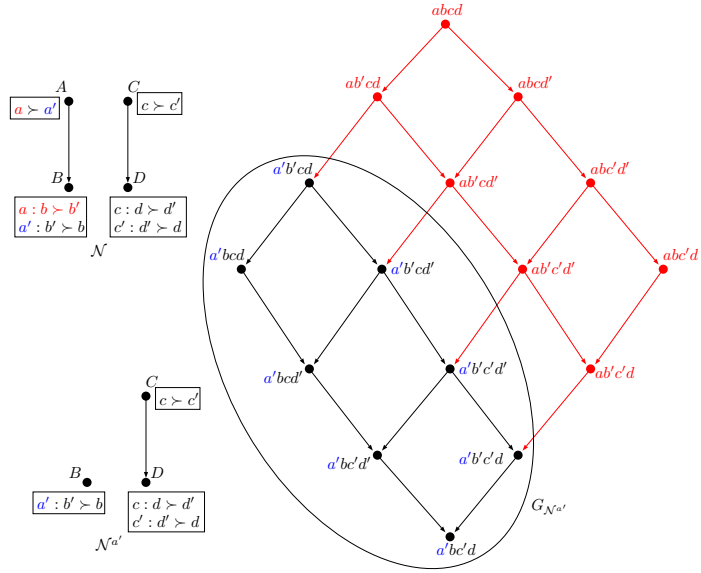


FIGURE A.6 – Exemple de paramétrisation du CP-net $\mathcal{N}$ donné dans la Figure A.2a page 159 en fixant la variable A sur la valeur a' (en bleu). Le nouveau CP-net est alors noté $\mathcal{N}^{a'}$ et son ordre partiel des objets $G_{\mathcal{N}^{a'}}$ (les éléments en rouge ne sont plus considérés).

A.3.2 Relation d'ordre entre CP-nets paramétrés

Nous avons précédemment vu qu'il était possible, via la paramétrisation d'un CP-net sur $\mathbf{V}$, de fixer un sous-CP-net grâce à un état. Est-il alors possible, à partir de la fixation des valeurs d'un ensemble de variables $\mathbf{X} \subset \mathbf{V}$, d'obtenir un ordre partiel sur les différents CP-nets paramétrés par des états $\mathbf{x} \in Dom(\mathbf{X})$?

Définition A.2 (Importance universelle)**.**
Soit $\mathbf{V}$ *un ensemble de variables. On dit qu'un ensemble de variables* $\mathbf{X} \subseteq \mathbf{V}$ ***est universellement plus important*** *qu'un autre ensemble de variables* $\mathbf{Y} \subseteq \mathbf{V}$ $(\mathbf{X} \cap \mathbf{Y} = \emptyset)$ *sachant* $\mathbf{w} \in Dom(\mathbf{W}), \mathbf{W} = \mathbf{V} \setminus (\mathbf{X} \cup \mathbf{Y})$*, noté* $\mathbf{X} \rhd_\forall \mathbf{Y}$*, si* $\forall \mathbf{x}, \mathbf{x}' \in Dom(\mathbf{X}), \forall \mathbf{y}, \mathbf{y}' \in Dom(\mathbf{Y})$ *avec* $\mathbf{x} \succ \mathbf{x}'$ *et* $\mathbf{y} \succ \mathbf{y}'$*, on a*

$$\mathbf{xy'w} \succ \mathbf{x'yw}.$$

Définition A.3 (Importance existentielle)**.**
Soit $\mathbf{V}$ *un ensemble de variables. On dit qu'un ensemble de variables* $\mathbf{X} \subseteq \mathbf{V}$ ***est existentiellement plus important*** *qu'un ensemble de variables* $\mathbf{Y} \subseteq \mathbf{V}$ $(\mathbf{X} \cap \mathbf{Y} = \emptyset)$ *sachant* $\mathbf{w} \in Dom(\mathbf{W}), \mathbf{W} = \mathbf{V} \setminus (\mathbf{X} \cup \mathbf{Y})$*, noté* $\mathbf{X} \rhd_\exists \mathbf{Y}$*, si* $\exists \mathbf{x}, \mathbf{x}' \in Dom(\mathbf{X}), \mathbf{x} \succ \mathbf{x}'$ *tel que*

$$\mathbf{xy'w} \succ \mathbf{x'yw}, \ \forall \mathbf{y}, \mathbf{y}' \in Dom(\mathbf{Y}) \text{ avec } \mathbf{y} \succ \mathbf{y}'.$$

La Définition A.3 fait partie des limitations connues des CP-nets et est à l'origine d'extensions de ce modèle tels que les TCP-nets [BDS06]. Nous verrons par la suite comment déduire cette importance grâce à l'analyse formelle des concepts. Nous nous concentrons pour l'instant sur l'étude de l'*importance universelle*, décrite par la Définition A.2.

Supposons qu'une telle importance existe et qu'il existe une variable $A \in \mathbf{V}$ tel que $A \rhd_\forall \mathbf{V} \setminus \{A\}$. Cela signifie, d'après la Définition A.2, que $\forall a, a' \in Dom(A)$ tel que $a \succ a'$, on a $a\mathbf{o} \succ a'\mathbf{o}', \forall \mathbf{o}, \mathbf{o}' \in Dom(\mathbf{V} \setminus \{A\})$. Nous pouvons alors dire que l'ordre partiel des objets $G_{\mathcal{N}^a}$ du CP-net paramétré $\mathcal{N}^a$ **est préféré** à l'ordre partiel des objets $G_{\mathcal{N}^{a'}}$ du CP-net paramétré $\mathcal{N}^{a'}$, que l'on simplifie en notant $\mathcal{N}^a \succ \mathcal{N}^{a'}$ (*i.e.*, le CP-net paramétré $\mathcal{N}^a$ est préféré au CP-net paramétré $\mathcal{N}^{a'}$). Un exemple d'ordre partiel de CP-nets paramétrés est donné dans la Figure A.7.

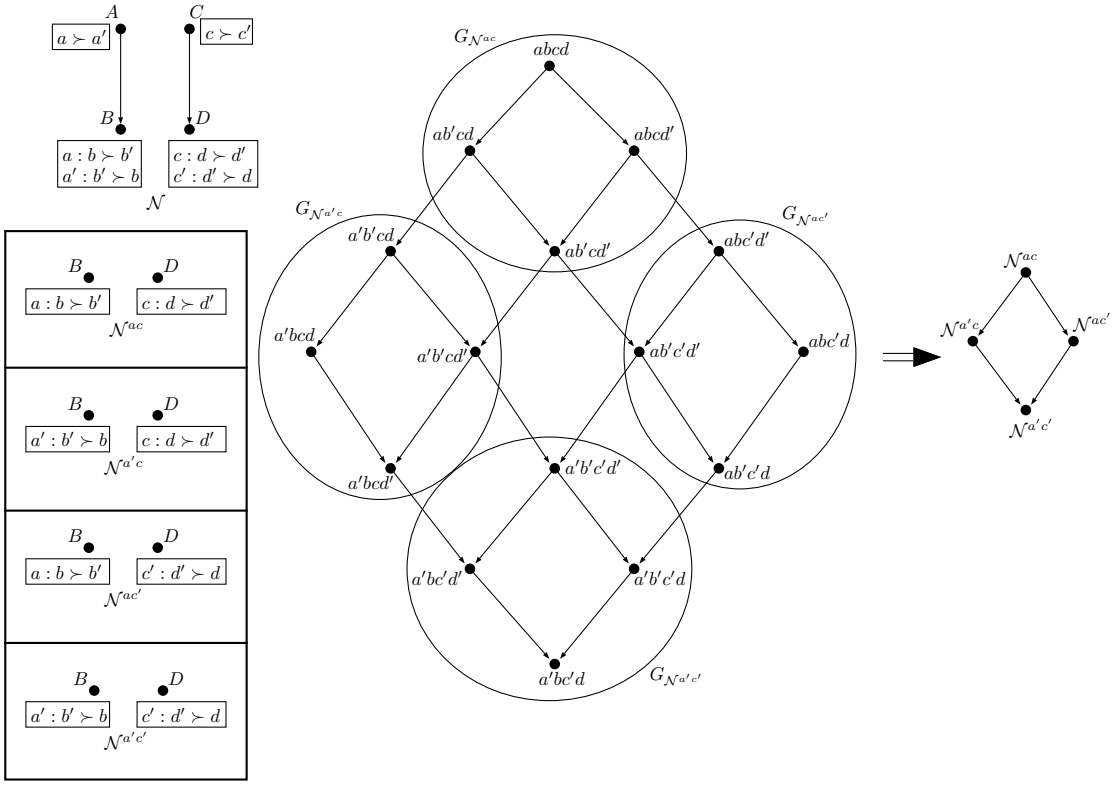


FIGURE A.7 – Exemple d'ordre de CP-nets paramétrés lorsque l'on fixe les valeurs des variables A et C qui sont plus importantes que les variables B et D ($A \rhd_\forall B$ et $C \rhd_\forall D$).

Nous allons maintenant nous intéresser à des structures algébriques particulières appelées **treillis**. Nous verrons notamment que des liens avec les CP-nets peuvent être établis.

A.4 Analyse formelle des concepts

Les treillis ont été officiellement introduits par Birkhoff en 1948 [Bir48]. Ils sont initialement définis comme étant une structure algébrique munie de deux opérateurs nommés borne inférieure et borne supérieure.

A.4.1 Préliminaires

Les treillis peuvent être vus de multiples manières, nous allons ici les étudier du point de vue de l'**analyse formelle des concepts** (AFC) [GW99]. Dans le domaine de l'analyse de données, un **treillis des concepts**, ou **treillis de Galois**, est un espace de recherche d'une table binaire composée des **objets** en ligne et des **attributs** de ces objets en colonne. Nous pouvons prendre l'exemple du panier de la ménagère, qui correspond à la liste des produits (attributs) de différentes personnes (objets). Les achats sont alors représentés par des croix à l'intersection d'une personne et d'un produit.

	Pain	Fromage	Banane	Tomate
c_1	x	x		
c_2		x	x	
c_3		x		x
c_4	x	x		x
c_5			x	x
c_6	x	x	x	

Tableau A.1 – Achats de plusieurs clients dans un magasin.

En suivant la modélisation du Tableau A.1, nous définissons un **contexte formel** comme étant un triplet $\mathbb{K} = (G, M, I)$, où G représente l'ensemble des objets, M représente l'ensemble des attributs et $I \subseteq G \times M$ est une relation entre les objets et les attributs. Une paire $(g, m) \in I$ signifie que l'objet g possède l'attribut m. Les **concepts formels** d'un contexte $\mathbb{K}$ sont toutes les paires (X, Y), avec $X \subseteq G$ et $Y \subseteq M$ tels que (X, Y) soit maximale sur la propriété $X \times Y \subseteq I$ (*i.e.*, $\nexists g \in G \setminus X$ tel que $(X \cup \{g\}, Y) \in I$ et $\nexists m \in M \setminus Y$ tel que $(X, Y \cup \{m\}) \in I$). L'ensemble X est appelé l'**extension** et l'ensemble Y est appelé l'**intension** d'un concept formel (X, Y). On note alors, pour tout concept formel a, son extension par $e(a)$ et son intension par $i(a)$ (donc $a = (e(a), i(a))$).

A.4.2 Treillis des concepts

La théorie des treillis fait intervenir l'opérateur $\succeq$ ayant une signification différente de l'apprentissage de préférences : il s'agit ici d'une relation de **couverture** entre deux ensembles. Ainsi, pour deux ensembles A et B, on note $A \succeq B$ (resp. $A \preceq B$) pour dire que A **couvre** B (resp. A est couvert par B), ou B **est couvert par** A (resp. B couvre A), et signifiant que $B \subseteq A$ (resp. $A \subseteq B$). L'ensemble des

concepts formels d'un contexte donné peut être ordonné hiérarchiquement par inclusion de leurs extensions : $(X_2, Y_2) \succeq (X_1, Y_1) \Leftrightarrow X_1 \subseteq X_2 (\Leftrightarrow Y_2 \subseteq Y_1)$. Cet ordre, appelé **relation sousconcept-superconcept**, induit toujours un treillis complet appelé **treillis des concepts** du contexte $\mathbb{K} = (G, M, I)$, noté $\mathfrak{C}(\mathbb{K})$.

Définition A.4 (Treillis des concepts)**.**
Soit $\mathbb{K} = (G, M, I)$ un contexte formel. Le ***treillis des concepts*** $\mathfrak{C}(\mathbb{K})$ *associé au contexte $\mathbb{K}$ est une relation d'ordre $\mathfrak{C}(\mathbb{K}) = (\mathcal{C}, \preceq)$ où $\mathcal{C}$ est l'ensemble des concepts associé au contexte $\mathbb{K}$: $(X, Y) \in \mathcal{C} \Leftrightarrow X \subseteq G, Y \subseteq M, Y = \alpha(X)$ et $X = \beta(Y)$ avec :*

$$\alpha(X) = \{m \in M \mid \forall x \in X, (x, m) \in I\}, \tag{A.2}$$
$$\beta(Y) = \{g \in G \mid \forall y \in Y, (g, y) \in I\}. \tag{A.3}$$

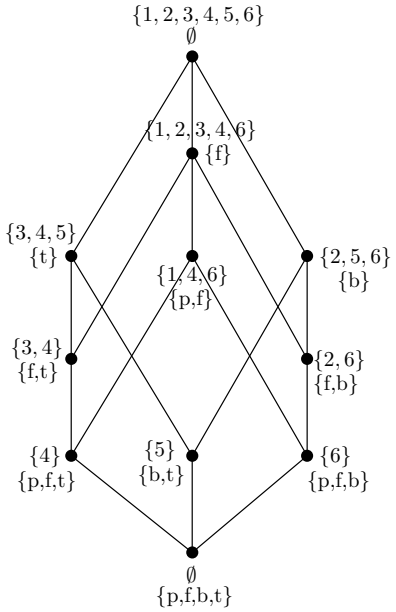


FIGURE A.8 – Treillis des concepts associé au contexte du Tableau A.1 (les c ont été omis pour plus de clarté).

La Figure A.8 donne un exemple de treillis des concepts associé au contexte du Tableau A.1. D'un point de vu algébrique, un treillis se définit comme une relation d'ordre sur des ensembles tel que toute famille d'ensembles possède une borne supérieure et une borne inférieure. Ces bornes sont déterminées par les deux opérateurs $\vee$ pour la borne supérieure et $\wedge$ pour la borne inférieure. La définition suivante introduit les bornes supérieures et inférieures d'un treillis des concepts.

Définition A.5 (Bornes supérieures et inférieures)**.**
Soit $\mathfrak{C}(\mathbb{K})$ le treillis des concepts d'un contexte formel $\mathbb{K} = (G, M, I)$. Soit $\mathcal{T} \subseteq \mathfrak{C}(\mathbb{K})$

une famille de concepts formels $(X_t, Y_t) \in \mathcal{T}$. *Les opérateurs de* ***borne supérieure*** $\vee$ *et de* ***borne inférieure*** $\wedge$ *se définissent de la manière suivante :*

$$\bigvee_{t\in\mathcal{T}} (X_t, Y_t) = \left(\beta(\alpha(\bigcup_{t\in\mathcal{T}} X_t)), \bigcap_{t\in\mathcal{T}} Y_t\right), \tag{A.4}$$

$$\bigwedge_{t\in\mathcal{T}} (X_t, Y_t) = \left(\bigcap_{t\in\mathcal{T}} X_t, \alpha(\beta(\bigcup_{t\in\mathcal{T}} Y_t))\right). \tag{A.5}$$

Définition A.6 (Connexion de Galois)**.**
Soit $\mathbb{K} = (G, M, I)$ *un contexte formel. Les opérateurs* $\alpha : G \to M$ *et* $\beta : M \to G$, *définis dans les Équations* (A.2) *et* (A.3) *respectivement, sont appelés* ***connexions de Galois*** *si,* $\forall X \in G$ *et* $\forall Y \in M$, $Y \subseteq \alpha(X) \Longleftrightarrow X \subseteq \beta(Y)$.

On dit que la paire (α, β) est une **connexion de Galois** entre deux ensembles d'ordres partiels $(P(G), \subseteq)$ et $(P(M), \subseteq)$ d'un contexte $\mathbb{K} = (G, M, I)$.

A.4.3 Contexte de préférence

Les travaux d'Obiedkov [Obi12, OIKZ12, Obi13] permettent d'établir un lien entre la notion de préférences et l'analyse formelle des concepts.

Définition A.7 (Contexte de préférence)**.**
On appelle ***contexte de préférence*** *un quadruplet* $\mathbb{P} = (G, M, I, \geq)$ *où* G *correspond à l'ensemble des objets,* M *correspond à l'ensemble des attributs,* I *est une relation entre* G *et* M, *et* $\geq$ *est une relation de préférence entre les objets de* G *représentée par un préordre. Ce contexte de préférence peut être vu comme la* ***composition*** *de deux contextes formels* $\mathbb{K} = (G, M, I)$ *et* $\mathbb{K}' = (G, G, \geq)$.

Prenons le contexte donné dans le Tableau A.1 et dans la Figure A.8 comme exemple de contexte $\mathbb{K}$, et la Figure A.9 comme exemple de contexte associé $\mathbb{K}'$.

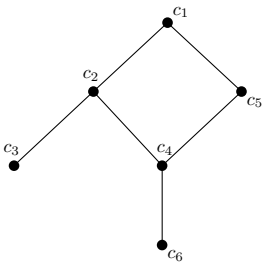


FIGURE A.9 – Exemple de relation d'ordre des objets issus du Tableau A.1.

Définition A.8 (Préférence sur les attributs)**.**
On dit qu'un ensemble d'attributs $A \subseteq M$ ***est préféré ceteris paribus*** *à un autre*

ensemble $B \subseteq M$ par rapport à un ensemble d'attributs $C \subseteq M$ dans un contexte de préférence $\mathbb{P} = (G, M, I, \geq)$, si

$$\forall g \in \beta(A) \forall h \in \beta(B),\ (\alpha(g) \cap C = \alpha(h) \cap C \rightarrow g \geq h). \tag{A.6}$$

*On dit alors que la préférence ceteris paribus $A \succsim_C B$ est **valide** dans $\mathbb{P}$, que l'on note $\mathbb{P} \models A \succsim_C B$.*

A.5 Étude structurelle des CP-nets

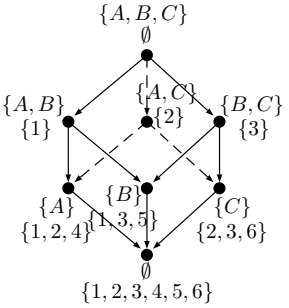
Les contextes de préférence définis dans la Section A.4.3 peuvent facilement être adaptés aux CP-nets complets à variables binaires : soit $\mathcal{N}$ un CP-net sur $\mathbf{V}$. Le contexte de préférence associé sera alors $\mathbb{P} = (Dom(\mathbf{V}) \setminus \{\top, \bot\}, \mathbf{V}, I, \succ)$, où $Dom(\mathbf{V}) \setminus \{\top, \bot\}$ correspond à l'ensemble des objets du contexte formel ($\top$ et $\bot$ correspondent respectivement au meilleur et au pire objet dans $G_{\mathcal{N}}$ [1]), et $\mathbf{V}$ correspond à l'ensemble des attributs du contexte formel. Le préordre est ici un ordre partiel $\succ$ correspondant à l'ordre partiel des objets $G_{\mathcal{N}}$, et $I \subseteq Dom(\mathbf{V}) \setminus \{\top, \bot\} \times \mathbf{V}$ est une relation binaire spécifiant quels attributs possède chaque objet. Le symbole $'$ sur une valeur x' d'une variable X signifie que la variable X n'est pas présente (ainsi, $a'bc$ doit être interprété comme un objet ayant les attributs B et C, mais pas l'attribut A). Un exemple d'un tel contexte est donné dans le Tableau A.2 et dans la Figure A.10.

	Objet	A	B	C
($\top$)	abc	x	x	x
(1)	abc'	x	x	
(2)	$ab'c$	x		x
(3)	$a'bc$		x	x
(4)	$ab'c'$	x		
(5)	$a'bc'$		x	
(6)	$a'b'c$			x
($\bot$)	$a'b'c'$			

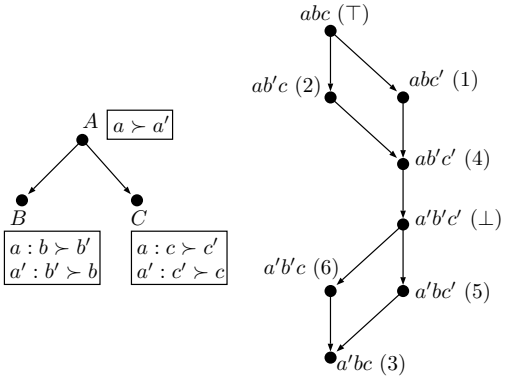
TABLEAU A.2 – Exemple de contexte sur un CP-net à trois variables A, B et C. Nous renommons les objets pour plus de clarté pour la suite.

Il est important de remarquer que le contexte formel $\mathbb{K}$ sera identique pour tous les CP-nets ayant le même nombre de variables. Seul le second contexte formel $\mathbb{K}'$ impliquant l'ordre des objets de $\mathbb{K}$ variera en fonction des conditions et des préférences induises par chaque CP-net. Cette remarque est évidente lorsque nous regardons les exemples illustrés dans la Figure A.10, où le contexte formel de la Figure A.10a sera toujours identique pour tous les CP-nets à 3 variables, car seule

1. $\top$ est appelé *top*, et $\bot$ est appelé *bottom*.



(a) Treillis des concepts associé au Tableau A.2.



(b) Ordre d'un contexte de préférence (à droite) induit par un CP-net (à gauche). Chaque sortie est associée à son numéro dans le contexte du Tableau A.2. ⊤ et ⊥ correspondent respectivement au *top* et au *bottom* du treillis des concepts de la Figure A.10a.

FIGURE A.10 – Contexte de préférence (contexte formel et ordre partiel) associés à l'exemple du Tableau A.2.

la préférence entre les objets du CP-net (Figure A.10b) sera modifiée d'un CP-net à l'autre. Ainsi, le contexte de préférence est dépendant de la structure du CP-net.

Il devient alors possible d'utiliser les méthodes existantes de l'AFC pour déduire des propriétés des CP-nets. Cependant, ces propriétés se limitent actuellement à la recherche d'importances entre variables, chose qui, comme nous l'avons déjà vu, est relativement rare dans les CP-nets à cause de la difficulté structurelle de représenter une telle propriété.

L'ordre partiel des objets est, de plus, de taille exponentielle par rapport au nombre de variables. Pouvoir se déplacer de façon rapide en évitant certains objets au sein de cet ordre est donc une propriété très intéressante, qui peut être facilement effectuée au moyen de l'utilisation des opérateurs de borne supérieure et inférieure. Ceci requiert cependant d'avoir une structure de treillis sur cet ordre partiel des objets obtenus après application des CP-règles du CP-net.

Conjecture 1. *Tout CP-net acyclique complet à variables binaires ayant une structure d'arbre possède un ordre partiel des objets admettant une structure de treillis.*

A.6 Conclusion

Nous avons pu, au sein de cette étude, entrevoir certaines propriétés structurelles des CP-nets acycliques, ainsi que des liens existants entre les CP-nets et la fouille de préférences.

Nous souhaitons poursuivre sur cette étude afin de trouver, notamment, des opérateurs de bornes inférieures et supérieures basés sur les CP-règles du CP-net permettant alors une recherche simple d'objets préférés au sein de l'ordre partiel des objets, sans avoir à le parcourir entièrement (à cause du nombre exponentiel de ses objets), sous réserve que la Conjecture 1 soit vérifiée.

Les applications d'une telle étude sont nombreuses, et permettent une recommandation directe des objets :

1. Apprentissage du CP-net d'un utilisateur sur un sujet quelconque (par exemple, le CP-net de ses préférences sur les objets vus sur Amazon), puis construction de l'ordre partiel des objets de celui-ci pour ensuite, à l'aide de la Définition A.8 page 167, déterminer l'ordre partiel de l'importance des attributs, et enfin utiliser cet ordre et les propriétés du CP-net pour placer rapidement un nouvel objet vu dans l'ordre partiel des objets ;
2. Agrégation de CP-nets ;
3. Définition d'une métrique au sein des CP-nets permettant de les comparer structurellement.

RÉFÉRENCES

[ACF02] Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2-3) :235–256, 2002. 104

[Aga05] Shivani Agarwal. A study of the bipartite ranking problem in machine learning. Technical report, Department of Computer and Information Science, University of Pennsylvania, 2005. 23

[AGJ+17] Thomas E. Allen, Judy Goldsmith, Hayden Elizabeth Justice, Nicholas Mattei, and Kayla Raines. Uniform random generation and dominance testing for cp-nets. *Journal of Artificial Intelligence Research*, 59 :771–813, 2017. 11, 35

[AK01] András Antos and Ioannis Kontoyiannis. Convergence properties of functional estimates for discrete distributions. *Random Structures & Algorithms*, 19(3-4) :163–193, 2001. 104

[AMZ16] Eisa Alanazi, Malek Mouhoub, and Sandra Zilles. The complexity of learning acyclic cp-nets. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016*, pages 1361–1367, 2016. 11, 35, 36

[Ang87] Dana Angluin. Queries and concept learning. *Machine Learning*, 2(4) :319–342, 1987. 58

[Arr12] Kenneth J Arrow. *Social choice and individual values*, volume 12. Yale university press, 2012. 34

[Ash07] Charles Ashbacher. The art of computer programming, volume 4 : Generating all trees, history of combinatorial generation. *Journal of Object Technology*, 6(1) :65, 2007. 49

[BBD+04] Craig Boutilier, Ronen I. Brafman, Carmel Domshlak, Holger H. Hoos, and David Poole. Cp-nets : A tool for representing and reasoning with conditional ceteris paribus preference statements. *Journal of Artificial Intelligence Research*, 21 :135–191, 2004. 11, 28, 29, 31, 32, 33, 34, 35, 54

[BBL09] Maria-Florina Balcan, Alina Beygelzimer, and John Langford. Agnostic active learning. *Journal of Computer and System Sciences*, 75(1) :78–89, 2009. 59

[BD02] Ronen I. Brafman and Carmel Domshlak. Introducing variable importance tradeoffs into cp-nets. In *UAI '02, Proceedings of the 18th Conference in Uncertainty in Artificial Intelligence, University of Alberta, Edmonton, Alberta, Canada, August 1-4, 2002*, pages 69–76, 2002. 38, 39

[BDS06] Ronen I. Brafman, Carmel Domshlak, and Solomon Eyal Shimony. On graphical modeling of preference and importance. *Journal of Artificial Intelligence Research*, 25 :389–424, 2006. 38, 163

[BEL09] Sylvain Bouveret, Ulle Endriss, and Jérôme Lang. Conditional importance networks : A graphical language for representing ordinal, monotonic preferences over sets of goods. In *IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, July 11-17, 2009*, pages 67–72. AAAI Press, 2009. 39, 40

[BFH06] Klaus Brinker, Johannes Fürnkranz, and Eyke Hüllermeier. A unified model for multilabel classification and ranking. In *ECAI 2006, 17th European Conference on Artificial Intelligence, August 29 - September 1, 2006, Riva del Garda, Italy, Including Prestigious Applications of Intelligent Systems (PAIS 2006), Proceedings*, pages 489–493, 2006. 22

[BH06] Klaus Brinker and Eyke Hüllermeier. Case-based label ranking. In *Machine Learning : ECML 2006, 17th European Conference on Machine Learning, Berlin, Germany, September 18-22, 2006, Proceedings*, pages 566–573, 2006. 22

[Big15] Damien Bigot. *Représentation et apprentissage de préférences*. PhD thesis, Université de Toulouse, Université Toulouse III-Paul Sabatier, 2015. 37

[Bir48] Garrett Birkhoff. *Lattice Theory*, volume 25. American Mathematical Society New York, 1948. 164

[Bis07] Christopher M. Bishop. *Pattern recognition and machine learning, 5th Edition*. Information science and statistics. Springer, 2007. 21

[BL04] Ronald J. Brachman and Hector J. Levesque. *Knowledge Representation and Reasoning*. Elsevier, 2004. 11

[BZFM13] Damien Bigot, Bruno Zanuttini, Hélène Fargier, and Jérôme Mengin. Probabilistic conditional preference networks. In *Proceedings of the Twenty-Ninth Conference on Uncertainty in Artificial Intelligence, UAI 2013, Bellevue, WA, USA, August 11-15, 2013*, 2013. 37, 38

[CGG+15] Cristina Cornelio, Umberto Grandi, Judy Goldsmith, Nicholas Mattei, Francesca Rossi, and Kristen Brent Venable. Reasoning with pcp-nets in a multi-agent context. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2015, Istanbul, Turkey, May 4-8, 2015*, pages 969–977, 2015. 37

[CGM+12] Cristina Cornelio, Judy Goldsmith, Nicholas Mattei, Francesca Rossi, and K Brent Venable. Dynamic and probabilistic cp-nets. *Mémoire de DEA, University of Padova.(Cité en pages 103 et 127.)*, 2012. 37

[CGM+13] Cristina Cornelio, Judy Goldsmith, Nicholas Mattei, Francesca Rossi, and Kristen Brent Venable. Updates and uncertainty in cp-nets. In *AI 2013 : Advances in Artificial Intelligence - 26th Australasian Joint Conference, Dunedin, New Zealand, December 1-6, 2013. Proceedings*, pages 301–312, 2013. 37, 38

[CK05] Wei Chu and S. Sathiya Keerthi. New approaches to support vector ordinal regression. In *Machine Learning, Proceedings of the Twenty-Second International Conference (ICML 2005), Bonn, Germany, August 7-11, 2005*, pages 145–152, 2005. 24, 25

[CK07] Wei Chu and S. Sathiya Keerthi. Support vector ordinal regression. *Neural Computation*, 19(3) :792–815, 2007. 25

[CKL+10] Yann Chevaleyre, Frédéric Koriche, Jérôme Lang, Jérôme Mengin, and Bruno Zanuttini. Learning ordinal preferences on multiattribute domains : The case of cp-nets. In *Preference Learning.*, pages 273–296. Springer, 2010. 11, 35, 36, 41

[CLRS09] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, 3rd Edition*. MIT Press, 2009. 98, 111

[CMM12] Olivier Cailloux, Patrick Meyer, and Vincent Mousseau. Eliciting electre tri category limits for a group of decision makers. *European Journal of Operational Research*, 223(1) :133–140, 2012. 25

[Cor16] Cristina Cornelio. *Preference reasoning and aggregation over combinatorial domains in uncertain and multi-agent scenarios*. PhD thesis, University of Padova, Italy, 2016. 37, 38

[CQL+07] Zhe Cao, Tao Qin, Tie-Yan Liu, Ming-Feng Tsai, and Hang Li. Learning to rank : from pairwise approach to listwise approach. In *Machine Learning, Proceedings of the Twenty-Fourth International Conference (ICML 2007), Corvallis, Oregon, USA, June 20-24, 2007*, pages 129–136, 2007. 20

[CSS99] William W. Cohen, Robert E. Schapire, and Yoram Singer. Learning to order things. *Journal of Artificial Intelligence Research*, 10 :243–270, 1999. 26, 27, 28

[CT06] Thomas M. Cover and Joy A. Thomas. *Elements of information theory (2. ed.)*. Wiley, 2006. 88, 89

[DB02] Carmel Domshlak and Ronen I. Brafman. Cp-nets : Reasoning and consistency testing. In *Proceedings of the Eights International Conference on Principles and Knowledge Representation and Reasoning (KR-02), Toulouse, France, April 22-25, 2002*, pages 121–132. Citeseer, 2002. 35

[DF03] Camil Demetrescu and Irene Finocchi. Combinatorial algorithms for feedback problems in directed graphs. *Information Processing Letters*, 86(3) :129–136, 2003. 22

[DH00] Pedro M. Domingos and Geoff Hulten. Mining high-speed data streams. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining, Boston, MA, USA, August 20-23, 2000*, pages 71–80, 2000. 104

[DHKP11] Carmel Domshlak, Eyke Hüllermeier, Souhila Kaci, and Henri Prade. Preferences in AI : an overview. *Artificial Intelligence*, 175(7-8) :1037–1052, 2011. 28

[DMA09] Yannis Dimopoulos, Loizos Michael, and Fani Athienitou. Ceteris paribus preference elicitation with predictive guarantees. In *IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, July 11-17, 2009*, pages 1890–1895, 2009. 41

[DRVW09] Carmel Domshlak, Francesca Rossi, Kristen Brent Venable, and Toby Walsh. Reasoning about soft constraints and conditional preferences : complexity results and approximation techniques. *Computing Research Repository (CoRR)*, abs/0905.3766, 2009. 35

[EW01] André Elisseeff and Jason Weston. A kernel method for multi-labelled classification. In *Advances in Neural Information Processing Systems 14 [Neural Information Processing Systems : Natural and Synthetic, NIPS 2001, December 3-8, 2001, Vancouver, British Columbia, Canada]*, pages 681–687, 2001. 22

[FH01] Eibe Frank and Mark A. Hall. A simple approach to ordinal classification. In *Machine Learning : EMCL 2001, 12th European Conference on Machine Learning, Freiburg, Germany, September 5-7, 2001, Proceedings*, pages 145–156, 2001. 23, 24

[FH03] Johannes Fürnkranz and Eyke Hüllermeier. Pairwise preference learning and ranking. In *Machine Learning : ECML 2003, 14th European Conference on Machine Learning, Cavtat-Dubrovnik, Croatia, September 22-26, 2003, Proceedings*, pages 145–156, 2003. 18, 19, 20

[FH10a] Johannes Fürnkranz and Eyke Hüllermeier, editors. *Preference Learning*. Springer, 2010. 10, 28

[FH10b] Johannes Fürnkranz and Eyke Hüllermeier. Preference learning and ranking by pairwise comparison. In *Preference Learning*, pages 65–82. Springer, 2010. 19, 20, 24

[FHV09] Johannes Fürnkranz, Eyke Hüllermeier, and Stijn Vanderlooy. Binary decomposition methods for multipartite ranking. In *Machine Learning and Knowledge Discovery in Databases, European Conference, ECML PKDD 2009, Bled, Slovenia, September 7-11, 2009, Proceedings, Part I*, pages 359–374, 2009. 23, 24

[FMR05] José Figueira, Vincent Mousseau, and Bernard Roy. ELECTRE methods. In *Multiple criteria decision analysis : State of the art surveys*, pages 133–153. Springer, 2005. 25

[FS97] Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, 55(1) :119–139, 1997. 27

[Für01] Johannes Fürnkranz. Round robin rule learning. In *Proceedings of the Eighteenth International Conference on Machine Learning (ICML 2001), Williams College, Williamstown, MA, USA, June 28 - July 1, 2001*, pages 146–153, 2001. 20

[Für02] Johannes Fürnkranz. Round robin classification. *Journal of Machine Learning Research*, 2 :721–747, 2002. 20

[GAG13] Joshua T. Guerin, Thomas E. Allen, and Judy Goldsmith. Learning cp-net preferences online from user queries. In *Late-Breaking Developments in the Field of Artificial Intelligence, Bellevue, Washington, USA, July 14-18, 2013*, 2013. xvii, xviii, 11, 42, 44, 48, 57, 58, 62, 68, 69, 76, 77, 78, 81, 85, 96, 117, 145, 146, 147, 148, 152

[Gas72] Joseph L Gastwirth. The estimation of the lorenz curve and gini index. *The Review of Economics and Statistics*, pages 306–316, 1972. 88

[GH05] Mithat Gönen and Glenn Heller. Concordance probability and discriminatory power in proportional hazards regression. *Biometrika*, 92(4) :965–970, 2005. 22

[GLMR14] Umberto Grandi, Hang Luo, Nicolas Maudet, and Francesca Rossi. Aggregating cp-nets with unfeasible outcomes. In *Principles and Practice of Constraint Programming - 20th International Conference, CP 2014, Lyon, France, September 8-12, 2014. Proceedings*, pages 366–381, 2014. 154

[GLTW08] Judy Goldsmith, Jérôme Lang, Miroslaw Truszczynski, and Nic Wilson. The computational complexity of dominance and consistency in cp-nets. *Journal of Artificial Intelligence Research*, 33 :403–432, 2008. 35

[GST+15] Bin Gu, Victor S. Sheng, KengYeow Tay, Walter Romano, and Shuo Li. Incremental support vector learning for ordinal regression. *IEEE Transactions on Neural networks and learning systems*, 26(7) :1403–1416, 2015. 24

[GW99] Bernhard Ganter and Rudolf Wille. *Formal concept analysis - mathematical foundations*. Springer, 1999. 156, 164

[HF04] Eyke Hüllermeier and Johannes Fürnkranz. Comparison of ranking procedures in pairwise preference learning. In *Proceedings of the 10th International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems (IPMU-04), Perugia, Italy*, 2004. 21

[HFCB08] Eyke Hüllermeier, Johannes Fürnkranz, Weiwei Cheng, and Klaus Brinker. Label ranking by learning pairwise preferences. *Artificial Intelligence*, 172(16-17) :1897–1916, 2008. 20

[HLY08] Yang Hu, Mingjing Li, and Nenghai Yu. Multiple-instance ranking : Learning to rank images for image retrieval. In *2008 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2008), 24-26 June 2008, Anchorage, Alaska, USA*. IEEE, 2008. 11, 18

[Hoe63] Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American statistical association*, 58(301) :13–30, 1963. 84, 104

[IBS08] Ihab F. Ilyas, George Beskales, and Mohamed A. Soliman. A survey of top-k query processing techniques in relational database systems. *ACM Comput. Surv.*, 40(4) :11 :1–11 :58, 2008. 26

[JLC00] Jeffrey D. Johnson, Jinghong Li, and Zengshi Chen. Reinforcement learning : An introduction : R.S. sutton, A.G. barto, MIT press, cambridge, MA 1998, 322 pp. ISBN 0-262-19398-1. *Neurocomputing*, 35(1-4) :205–206, 2000. 26

[Kam03] Toshihiro Kamishima. Nantonac collaborative filtering : recommendation based on order responses. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, August 24 - 27, 2003*, pages 583–588. ACM, 2003. 138

[Kee88] Ralph L Keeney. Building models of values. *European Journal of Operational Research*, 37(2) :149–157, 1988. 18

[KKA05] Toshihiro Kamishima, Hideto Kazawa, and Shotaro Akaho. Supervised ordering - an empirical survey. In *Proceedings of the 5th IEEE International Conference on Data Mining (ICDM 2005), 27-30 November 2005, Houston, Texas, USA*, pages 673–676, 2005. 26

[KKA10] Toshihiro Kamishima, Hideto Kazawa, and Shotaro Akaho. A survey and empirical comparison of object ranking methods. In *Preference Learning*, pages 181–201. Springer, 2010. 11, 18, 26

[KSS94] Michael J. Kearns, Robert E. Schapire, and Linda Sellie. Toward efficient agnostic learning. *Machine Learning*, 17(2-3) :115–141, 1994. 52

[KZ10] Frédéric Koriche and Bruno Zanuttini. Learning conditional preference networks. *Artificial Intelligence*, 174(11) :685–703, 2010. xix, 11, 17, 41, 42, 44, 45, 46, 47, 48, 57, 62

[LL06] Ling Li and Hsuan-Tien Lin. Ordinal regression by extended binary classification. In *Advances in Neural Information Processing Systems 19, Proceedings of the Twentieth Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 4-7, 2006*, pages 865–872, 2006. 24

[LM08] Jérôme Lang and Jérôme Mengin. Learning preference relations over combinatorial domains. *Proceedings of the Twelfth International Workshop on Non-Monotonic Reasoning, Sydney, Australia*, pages 207–214, 2008. 35

[LM16] Fabien Labernia and Brice Mayag. Élicitation des paramètres d'electre tri : Apprentissage par réduction. In *17ième Conférence ROADEF (Société Française de Recherche Opérationnelle et d'Aide à la Décision), Compiègne, UTC, France, Février 10-12, 2016*, pages 1–2, 2016. 25

[LMX12] Jérôme Lang, Jérôme Mengin, and Lirong Xia. Aggregating conditionally lexicographic preferences on multi-issue domains. In *Principles and Practice of Constraint Programming - 18th International Conference, CP 2012, Québec City, QC, Canada, October 8-12, 2012. Proceedings*, pages 973–987, 2012. 154

[LS08] Thomas Lukasiewicz and Umberto Straccia. Managing uncertainty and vagueness in description logics for the semantic web. *Web Semantics : Science, Services and Agents on the World Wide Web*, 6(4) :291–308, 2008. 101

[LXW+14] Juntao Liu, Yi Xiong, Caihua Wu, Zhijun Yao, and Wenyu Liu. Learning conditional preference networks from inconsistent examples. *IEEE Transactions on Knowledge and Data Engineering*, 26(2) :376–390, 2014. xvii, xviii, 11, 42, 43, 48, 50, 53, 58, 68, 69, 80, 81, 85, 115, 117, 145, 146, 152, 154

[LYMA16] Fabien Labernia, Florian Yger, Brice Mayag, and Jamal Atif. Query-based learning of acyclic conditional preference networks from noisy data. In *EURO Mini Conference : "From Multiple Criteria Decision Aid to Preference Learning" (DA2PL'2016), Paderbon, Germany*, 2016. 58

[LYMA18] Fabien Labernia, Florian Yger, Brice Mayag, and Jamal Atif. Query-based learning of acyclic conditional preference networks from contradictory preferences. *EURO Journal on Decision Processes*, 6(1) :39–59, June 2018. 58

[LYX+13] Juntao Liu, Zhijun Yao, Yi Xiong, Wenyu Liu, and Caihua Wu. Learning conditional preference network from noisy samples using hypothesis testing. *Knowledge-Based Systems*, 40 :7–16, 2013. 41, 50, 85, 139

[LZLZ18] Zhaowei Liu, Zhaolin Zhong, Ke Li, and Chenghui Zhang. Structure learning of conditional preference networks based on dependent degree of attributes from preference database. *IEEE Access*, 2018. 42, 53

[LZM+17] Fabien Labernia, Bruno Zanuttini, Brice Mayag, Florian Yger, and Jamal Atif. Online learning of acyclic conditional preference networks from noisy data. In *2017 IEEE International Conference on Data Mining, ICDM 2017, New Orleans, LA, USA, November 18-21, 2017*, pages 247–256, 2017. 84, 85, 104

[McD89] Colin McDiarmid. On the method of bounded differences. *Surveys in combinatorics*, 141(1) :148–188, 1989. 104

[ML03] Cleve B. Moler and Charles Van Loan. Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later. *SIAM Review*, 45(1) :3–49, 2003. 43

[MPP14] Pierre Marquis, Odile Papini, and Henri Prade. *Panorama de l'intelligence artificielle, ses bases méthodologiques, ses développements*, volume 1. Cépaduès Éditions, 2014. 17

[Obi12] Sergei A. Obiedkov. Modeling preferences over attribute sets in formal concept analysis. In *Formal Concept Analysis - 10th International Conference, ICFCA 2012, Leuven, Belgium, May 7-10, 2012. Proceedings*, pages 227–243, 2012. 156, 166

[Obi13] Sergei A. Obiedkov. Modeling ceteris paribus preferences in formal concept analysis. In *Formal Concept Analysis, 11th International Conference, ICFCA 2013, Dresden, Germany, May 21-24, 2013. Proceedings*, pages 188–202, 2013. 156, 166

[OIKZ12] Sergei Obiedkov, DI Ignatov, SO Kuznetsov, and LE Zhukov. From preferences over objects to preferences over concepts. In *Proceedings of the 6th Multidisciplinary Workshop on Advances in Preference Handling, Montpellier, France, in conjunction with ECAI*, 2012. 156, 166

[Pan03] Liam Paninski. Estimation of entropy and mutual information. *Neural Computation*, 15(6) :1191–1253, 2003. 105

[Qui86] J. Ross Quinlan. Induction of decision trees. *Machine Learning*, 1(1) :81–106, 1986. 88

[Qui93] J. Ross Quinlan. *C4.5 : Programs for Machine Learning*. Morgan Kaufmann, 1993. 88

[RA05] Shyamsundar Rajaram and Shivani Agarwal. Generalization bounds for k-partite ranking. In *Proceedings of the NIPS Workshop on Learning to Rank, Vancouver, Canada*, pages 18–23, 2005. 23

[Raj79] David W. Rajala. Review of "decisions with multiple objectives : Preferences and value trade-offs" by ralph l. keeney and howard raiffa. *IEEE Transactions on Systems, Man, and Cybernetics : Systems*, 9(7) :403, 1979. 11, 18

[RC15] Rocco De Rosa and Nicolò Cesa-Bianchi. Splitting with confidence in decision trees with application to stream mining. In *2015 International Joint Conference on Neural Networks, IJCNN 2015, Killarney, Ireland, July 12-17, 2015*, pages 1–8, 2015. 106

[RC17] Rocco De Rosa and Nicolò Cesa-Bianchi. Confidence decision trees via online and active learning for streaming data. *Journal of Artificial Intelligence Research*, 60 :1031–1055, 2017. 89, 104, 105, 106

[Roy68] Bernard Roy. Classement et choix en présence de points de vue multiples. *Revue française d'automatique, d'informatique et de recherche opérationnelle. Recherche opérationnelle*, 2(1) :57–75, 1968. 25

[Roy13] Bernard Roy. *Multicriteria methodology for decision aiding*, volume 12. Springer Science & Business Media, 2013. 25

[RRS15] Francesco Ricci, Lior Rokach, and Bracha Shapira, editors. *Recommender Systems Handbook*. Springer, 2015. 3

[San07] Tuomas Sandholm. Expressive commerce and its application to sourcing : How we conducted $35 billion of generalized combinatorial auctions. *AI Magazine*, 28(3) :45–58, 2007. 11

[SL02] Amnon Shashua and Anat Levin. Ranking with large margin principle : Two approaches. In *Advances in Neural Information Processing Systems 15 [Neural Information Processing Systems, NIPS 2002, December 9-14, 2002, Vancouver, British Columbia, Canada]*, pages 937–944, 2002. 25

[SSBD14] Shai Shalev-Shwartz and Shai Ben-David. *Understanding machine learning : From theory to algorithms*. Cambridge university press, 2014. 52

[Tro05] Andrew Trotman. Learning to rank. *Information Retrieval*, 8(3) :359–381, 2005. 24

[Tso08] Alexis Tsoukiàs. From decision theory to decision aiding methodology. *European Journal of Operational Research*, 187(1) :138–161, 2008. 11

[VC15] Vladimir N Vapnik and A Ya Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. In *Measures of Complexity*, pages 11–30. Springer, 2015. 36

[VG10] Shankar Vembu and Thomas Gärtner. Label ranking algorithms : A survey. In *Preference Learning*, pages 45–64. Springer, 2010. 11, 18

[Wal07] Toby Walsh. Representing and reasoning with preferences. *AI Magazine*, 28(4) :59–70, 2007. 11

[WB10] Willem Waegeman and Bernard De Baets. A survey on roc-based ordinal regression. In *Preference Learning.*, pages 127–154. Springer, 2010. 18, 22, 23, 28

[Wil09] Rudolf Wille. Restructuring lattice theory : An approach based on hierarchies of concepts. In *Formal Concept Analysis, 7th International Conference, ICFCA 2009, Darmstadt, Germany, May 21-24, 2009, Proceedings*, pages 314–339, 2009. 156

[Wil11] Nic Wilson. Computational techniques for a simple theory of conditional preferences. *Artificial Intelligence*, 175(7-8) :1053–1091, 2011. 40

[WLZ10] Hongning Wang, Yue Lu, and Chengxiang Zhai. Latent aspect rating analysis on review text data : a rating regression approach. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, July 25-28, 2010*, pages 783–792, 2010. 69, 138

[WLZ11] Hongning Wang, Yue Lu, and ChengXiang Zhai. Latent aspect rating analysis without aspect keyword supervision. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Diego, CA, USA, August 21-24, 2011*, pages 618–626, 2011. 69, 138

[ZBHH10] Jianping Zhang, Jerzy W. Bala, Ali Hadjarian, and Brent Han. Ranking cases with classification rules. In *Preference Learning*, pages 155–177. Springer, 2010. 28

Résumé

La croissance exponentielle des données personnelles, et leur mise à disposition sur la toile, a motivé l'émergence d'algorithmes d'apprentissage de préférences à des fins de recommandation, ou d'aide à la décision. Les réseaux de préférences conditionnelles (CP-nets) fournissent une structure compacte et intuitive pour la représentation de telles préférences. Cependant, leur nature combinatoire rend leur apprentissage difficile : comment apprendre efficacement un CP-net au sein d'un milieu bruité, tout en supportant le passage à l'échelle ?

Notre réponse prend la forme de deux algorithmes d'apprentissage dont l'efficacité est soutenue par de multiples expériences effectuées sur des données réelles et synthétiques.

Le premier algorithme se base sur des requêtes posées à des utilisateurs, tout en prenant en compte leurs divergences d'opinions. Le deuxième algorithme, composé d'une version hors ligne et en ligne, effectue une analyse statistique des préférences reçues et potentiellement bruitées. La borne de McDiarmid est en outre utilisée afin de garantir un apprentissage en ligne efficace.

Mots Clés

Apprentissage de préférences, réseaux de préférences conditionnelles, apprentissage en ligne, préférences bruitées, borne de McDiarmid.

Abstract

The rapid growth of personal web data has motivated the emergence of learning algorithms well suited to capture users' preferences. Among preference representation formalisms, conditional preference networks (CP-nets) have proven to be effective due to their compact and explainable structure. However, their learning is difficult due to their combinatorial nature.

In this thesis, we tackle the problem of learning CP-nets from corrupted large datasets. Three new algorithms are introduced and studied on both synthetic and real datasets.

The first algorithm is based on query learning and considers the contradictions between multiple users' preferences by searching in a principled way the variables that affect the preferences. The second algorithm relies on information-theoretic measures defined over the induced preference rules, which allow us to deal with corrupted data. An online version of this algorithm is also provided, by exploiting the McDiarmid's bound to define an asymptotically optimal decision criterion for selecting the best conditioned variable and hence allowing to deal with possibly infinite data streams.

Keywords

Preference learning, conditional preference networks, online learning, noisy preferences, McDiarmid's bound.