Prise de décision multiattribut avec le modèle GAI

Jean-Philippe Dubus

Université Pierre et Marie Curie (PARIS VI)
UFR ingénierie

THESE

Pour l'obtention du titre de
DOCTEUR EN INFORMATIQUE

spécialité: aide à la décision

Prise de décision multiattribut avec le modèle GAI

Candidat: Jean-Philippe DUBUS

JURY

Directeur de thèse:	Christophe GONZALES Professeur à l'Université Pierre et Marie Curie
Rapporteurs:	Philippe LERAY Professeur à l'Université de Nantes
	Régis SABBADIN Chargé de recherche INRA (Toulouse)
Examinateurs:	Nicolas MAUDET Maître de conférences à l'Université Paris Dauphine
	Alix MUNIER-KORDON Professeur à l'Université Pierre et Marie Curie
	Patrice PERNY Professeur à l'Université Pierre et Marie Curie

Présentée et soutenue publiquement le 23 septembre 2010

L'université n'entend donner aucune approbation ni improbation aux opinions émises dans les thèses : ces opinions doivent être considérées comme propres à leurs auteurs.

Remerciements

En premier lieu, je tiens à exprimer une pensée à ceux qui ne sont pas là pour lire cette thèse : tout d'abord à ma mère qui nous a quittés à l'époque où j'étais scolairement très instable et à un âge où je n'étais pas le plus facile à vivre. J'aurais aimé qu'elle me voit évoluer et qu'elle assiste aux changements que l'entrée à l'université m'a apporté. Ainsi, ce travail lui est dédié. Ensuite, à mon premier directeur de thèse, Jean-Yves Jaffray, que je n'ai, hélas, pas assez connu dans le cadre de la thèse, mais envers qui j'ai un profond respect, de par la qualité de ses enseignements et de ses travaux, aussi bien que pour l'humilité et la gentillesse dont il faisait preuve lors de discussions.

Pour reprendre les remerciements sur un ton plus joyeux, je remercie mon directeur de thèse, Christophe Gonzales, pour m'avoir donné goût aux modèles graphiques, la rigueur dont il fait preuve, le temps qu'il m'a consacré, sa capacité à pouvoir discuter de sujets aussi bien très théoriques que très pratiques, et aussi pour avoir réussi à me supporter (dans tous les sens du terme) pendant trois ans (et plus, si on inclue le stage de M2 et le projet de M1).

Je remercie aussi particulièrement les membres du jury : Philippe Leray pour avoir accepté de plonger dans les profondeurs de ma thèse en acceptant le rôle de rapporteur. Régis Sabbadin pour avoir décortiqué le contenu de cette thèse en tant que rapporteur et m'avoir fait part de ses remarques avisées par mail. Alix Munier-Kordon avec qui j'ai eu de nombreuses interactions intéressantes dans le cadre d'enseignements, je suis très heureux qu'elle ait accepté de présider le jury. Nicolas Maudet, pour son rôle d'examinateur et pour représenter le LAMSADE au sein du jury. Et enfin, *last but not least*, Patrice Perny, avec qui j'ai beaucoup travaillé ces trois dernières années, et qui est un modèle de pédagogie et de droiture.

Je tiens aussi à remercier la *team aGrUM*[1] : Christophe Gonzales, Pierre-Henri Wuillemin (personnification de la bonne humeur et de la sympathie) et Lionel Torti, pour leur aide précieuse dans la compréhension des mystiques mécanismes internes au C++.

Cette thèse n'aurait jamais pu être telle qu'elle est actuellement sans la fine équipe d'amis qui s'est formée à la fois très naturellement et très rapidemment au sein du laboratoire : Charles Delort (puisse sa main de *Magic* ne jamais tarir), Thomas Génin (dont la voix précède son ombre), Gildas Jeantet (capable de trouver toutes les failles des règles d'un jeu en moins de temps qu'il n'est nécessaire pour lire ces règles), Jean-Mathieu Segura (pour nous avoir inculqué la joie de vivre marseillaise) et Lionel Torti (créature surnaturelle vivant sur deux plans d'existence : Paris et Bordeaux). Les excellentes soirées de rigolade, de jeux, les sympathiques pauses, les vacances à Preignac, aussi bien que le réconfort et les conseils que l'on a pu s'apporter mutuellement lors des passages à vide de la thèse ont permis d'aboutir à ce résultat.

Je remercie également toute l'équipe Décision, ainsi que l'équipe Recherche Opérationnelle, pour l'ambiance et la convivialité qu'elles ont réussi à instaurer durant ces années de doctorat : Abir, Ariele (futur chanteur de *Kiss* ?), Aurélien (nouvel adepte de Robin Hobb), Charles, Christophe, Claire, Emmanuel, Fanny, Gildas, Hacène, Isabelle,

1. Séquence publicitaire : `http://agrum.lip6.fr/`

Jean-Yves, Jie, Julien, Lionel, Lucie, Michel, Olivier, Oncu, Patrice, Patrick, Paul, Pierre, Pierre-Henri, Philippe, Safia, Sergio, Viet-Hung, Xiaoliang (Alex), Xuan Son et Yasmina (que la force de CPLEX soit avec toi).

J'aimerai conclure ces remerciements par des personnes n'appartenant pas au laboratoire ou au milieu de la recherche en informatique, mais qui m'ont beaucoup apporté, que ce soit dans ma construction en tant que personne, dans le soutien qu'ils ont pu me fournir tout au long de la thèse et au-delà, et tout simplement aussi parce-que ce sont des personnes qui comptent à mes yeux, et je tiens à les remercier d'être là. En premier lieu, ma famille la plus proche : François mon père, Jean-Christophe mon frère (et son talent pour la photo), Yvette ma grand-mère, mais aussi Aurore, Axelle, Céline, Emmanuelle, Françoise, Jean-Claude, Jean-Pierre, Jocelyne, Julien, Lisette, Marie-Agnès, Nénette, et Roland. Merci pour tout. Un grand merci aussi à ma chère et tendre Émilie, qui a accepté de sacrifier de nombreux moments où nous aurions pu nous voir pour que je puisse mener à bien la rédaction de cette thèse.

Mes plus sincères remerciements aussi à mes amis, que ce soient ceux de longue date aussi bien que ceux que j'ai eu l'occasion de découvrir ces dernières années, certains pourront s'étonner de figurer sur cette liste, car je ne les ai pas revus depuis un moment, mais je compte bien y remédier très rapidemment : Antoine (promis, on se fera un week-end de Civilisation bientôt), Aslèche (je suis très heureux que tu ais trouvé ta voie), Aurélie (barbare nomade ou princesse elfe ?), Caroline (portée par la musique), Charlotte D. (*leader* presque incontestée de la bande d'aventuriers), Charlotte P. (énergie à l'état pur), Étienne (on en aura vécu des choses en neuf ans), Fabien (le wookie en blouse blanche), Guillaume (grand maître de la blague matheuse), Lucie (on va bientôt pouvoir écrire un roman), Marie-Neige (promis, je vais essayer de dire *mardouk* et non plus *mare-duck*), Meriem (et sa joie de vivre), Michel (je débarque bientôt dans ton monde), Olivier (notre grand frère à tous), Sacha (Aristote des temps modernes), Samuel (notre futur espagnol), Xavier (Linux vaincra !) et, évidemment, toute la bande du laboratoire citée plus haut.

Table des matières

Introduction **1**

Notations utilisées **7**

I Etat de l'art **9**

1 Prise de décision multiattribut et multicritère **11**
1.1 Préférence et utilité . . . 12
1.1.1 Bases pour la prise de décision . . . 12
1.1.2 Relation de préférence . . . 13
1.1.3 Fonction d'utilité . . . 14
1.2 Utilité sur des attributs . . . 17
1.2.1 Utilité multiattribut . . . 17
1.2.2 Exemple de décomposition : les utilités additives . . . 18
1.3 Décision multicritère . . . 19
1.3.1 Problématiques . . . 20
1.3.2 Comparaison de solutions vectorielles . . . 22
1.3.3 Détermination de frontières . . . 23
1.4 Agrégateurs et solution agrégée optimale . . . 27
1.4.1 La somme ordonnée pondérée (OWA) . . . 28
1.4.2 La norme de Tchebycheff . . . 29
1.4.3 L'intégrale de Choquet . . . 31

2 Décision avec le modèle GAI **35**
2.1 Le modèle GAI . . . 36
2.1.1 GAI-décomposabilité . . . 36
2.1.2 Bases de théorie des graphes . . . 38
2.1.3 Un modèle graphiquement décomposable : les réseaux GAI . . . 42
2.2 Représentation des préférences par des réseaux GAI . . . 45
2.2.1 Elicitation de préférences GAI-décomposables . . . 45
2.2.2 Triangulation et construction automatique de réseaux GAI . . . 47
2.2.3 Considérations sur la triangulation . . . 53
2.2.4 Agrégation et intégration de contraintes dans les réseaux GAI . . . 55
2.3 Exploitation des réseaux GAI pour la résolution de problèmes décisionnels monocritères . . . 58
2.3.1 Détermination du choix optimal . . . 58

2.3.2 Rangement des meilleures alternatives 65
2.4 Utilisation des réseaux GAI en décision multicritère 69
2.4.1 Réseau GAI scalarisé 70
2.4.2 Détermination du choix agrégé optimal par rangement 72

II Algorithmes d'exploitation de réseaux GAI 75

3 Exploitation des réseaux GAI de forte treewidth 77
3.1 Élément critique et idée d'approche 78
3.1.1 L'élément critique : la treewidth 78
3.1.2 Un algorithme en trois moteurs 80
3.2 Vers les détails de l'algorithme 81
3.2.1 Moteur d'approximation : le couteau 82
3.2.2 Moteur d'analyse graphique : le boucher 89
3.3 Analyse expérimentale de l'algorithme 95
3.4 Extensions de l'algorithme 97
3.4.1 Rangement sous forte treewidth 97
3.4.2 Solution agrégée optimale sous forte treewidth 98
3.4.3 Extension des moteurs 99
3.4.4 Vers une nouvelle approche de la triangulation 100

4 Détermination exacte et approchée de la frontière de Pareto 103
4.1 La frontière de Pareto par collecte de messages 104
4.1.1 NP-complétude du problème 104
4.1.2 Réseau GAI vectoriel 105
4.1.3 Collecte vectorielle pour le calcul de la frontière de Pareto 107
4.2 Un algorithme approché avec garantie de performance 113
4.2.1 ϵ-dominance 114
4.2.2 Vers un algorithme approché avec garantie de performance 115
4.3 Analyse et conclusion 119
4.3.1 Analyse théorique 119
4.3.2 Analyse expérimentale 121
4.3.3 Conclusion 123

5 Approche filaire pour les réseaux GAI vectoriels 125
5.1 Fonctions heuristiques pour un réseau GAI vectoriel 127
5.1.1 Algorithme de détermination des heuristiques majorantes 127
5.1.2 Validité et complexité de l'algorithme 131
5.2 Détermination de la frontière de Pareto 134
5.2.1 Approche filaire sans heuristique 134
5.2.2 Intégration de l'heuristique 138
5.2.3 Validité et complexité de l'algorithme 138
5.3 Extension de l'algorithme à de nouveaux problèmes 142
5.3.1 Détermination de la frontière de Lorenz 142
5.3.2 Détermination de la solution agrégée optimale 144
5.4 Analyse expérimentale 146

Conclusion **153**

Introduction

Nous sommes tous amenés à prendre régulièrement des décisions, que ce soit pour choisir le contenu de son panier dans un magasin, ou bien encore pour déterminer la musique à passer en fond sonore d'une soirée. Nos prises de décision sont souvent guidées par nos préférences, parfois par des contraintes intervenant dans nos choix, ou bien encore par la recherche de compromis lors d'une décision de groupe. Fort heureusement, la plupart des décisions à prendre dans la vie de tous les jours sont assez simples pour que nous puissions instinctivement faire de bons choix. Néanmoins, il existe des situations complexes où le processus décisionnel n'est pas aussi évident comme, par exemple, les décisions à prendre à la tête d'une entreprise, ou encore la détermination de l'emplacement d'un nouvel aéroport (Keeney et Raiffa, 1993).

La *théorie de la décision*[2] étudie scientifiquement les différents éléments intervenant au sein d'une prise de décision, de manière à la comprendre, la reproduire et à l'étendre dans des domaines complexes. Cette théorie a mené à l'élaboration d'outils d'*aide à la décision*, souvent des logiciels jouant le rôle de « second cerveau » : ils visent à comprendre le comportement d'un utilisateur (comme, par exemple, ses préférences), pour ensuite exploiter toute la puissance computationelle et mémorielle d'un ordinateur pour proposer les meilleurs choix possibles selon le point de vue de l'utilisateur.

La démocratisation d'internet a favorisé l'émergence des sites commerciaux. A la fin du vingtième siècle, des nombreux sites se sont dôtés d'un *système de recommandation* : un moteur de calcul capable de proposer au client, pendant sa recherche de produits, d'autres articles pouvant potentiellement l'intéresser. De tels systèmes se fondent sur l'étude du comportement du client sur le site, de manière à coller au plus près des véritables préférences de celui-ci. Nous pouvons remarquer une forte analogie avec les systèmes d'aide à la décision, la principale différence étant que les interactions entre le le décideur et le système doivent parfois se faire de manière implicite. Par exemple, le système ne demandera pas à l'utilisateur ses préférences sur certains produits, mais devra réussir à les déduire en fonction de ses actions sur le site (par exemple ses précédents achats, les liens sur lesquels il a cliqué, etc...).

Le domaine du jeu vidéo a aussi beaucoup évolué ces vingt dernières années, que ce

2. Il est à noter que cette thèse ne s'inscrit pas directement dans le domaine de la théorie de la décision, mais au croisement entre ce domaine, l'optimisation combinatoire et l'intelligence artificielle. Durant les trois années de travaux ayant conduit à la rédaction de ce manuscrit a eu lieu à Venise la première conférence internationale (*Algorithmic Decision Theory*) ayant pour thème ce mélange de genres, et lui donnant un nom : Théorie de la décision algorithmique.

soit du point de vue des graphismes, du scénario, ou du moteur du jeu. Un but recherché par les éditeurs de jeu est l'immersion du joueur au sein d'un monde virtuel dans lequel il sera amené à interagir avec des entités artificielles, les personnages non-joueurs. Ces personnages sont gérés par une *intelligence artificielle*, et une telle immersion prendra toute son ampleur si ces entités ont l'air humaine dans leur comportement en cours de jeu. De nouveau, la théorie de la décision s'intègre parfaitement dans ce contexte en proposant des modèles calqués sur la décision humaine pour les choix auxquels sera confronté l'intelligence artificielle.

Pour résumer la façon de procéder évoquée dans les exemples d'application précédents, une procédure de prise de décision comporte deux facettes : *l'élicitation*, qui vise à modéliser le comportement du décideur (de façon interactive dans un logiciel d'aide à la décision, implicite dans un système de recommendation, ou encore définie par les concepteurs ou apprise dans une intelligence artificielle). La finalité de l'élicitation est une réprésentation de ce comportement exploitable par une machine. La seconde facette (que nous nommerons *le choix*) vise à exploiter la représentation des préférences élicitées pour proposer une solution du problème de prise de décision. Dans le cadre de cette thèse, nous supposerons que le comportement du décideur a été élicité, et donc que nous disposons d'une représentation de ce comportement.

Parmi les différents problèmes étudiés en théorie de la décision, nous pouvons définir plusieurs catégories :

- **La décision dans l'incertain** : Les conséquences des décisions possibles dépendent de facteurs que le décideur ne peut maîtriser. La prise de décision doit donc prendre en compte l'incertitude sur les conséquences des décisions prises et l'attitude du décideur vis-à-vis du risque.
- **La décision collective** : Cette catégorie définit les problèmes où plusieurs décideurs doivent se mettre d'accord sur un choix commun (par exemple, lors d'une élection), et est la digne héritière des débats de Condorcet et Borda sur la *théorie du choix social*.
- **La décision multicritère** : Les conséquences des décisions sont évaluées par le décideur selon plusieurs points de vue (les *critères*) et le décideur a des préférences bien établies selon chacun de ces points de vue. Ces préférences peuvent être contradictoires entre elles, menant à des problèmes de recherche de solutions de compromis entre les différents critères. Il est à noter qu'il existe un lien fort entre décision collective et décision multicritère. On peut en effet se représenter les différents critères d'un problème comme des décideurs à part entière devant prendre une décision en commun.
- **La décision multiattribut** : Les décisions possibles sont décrites sous forme de regroupements d'*attributs*, c'est-à-dire différentes facettes caractérisant chaque décision. C'est le cas, par exemple, du choix de la composition d'un menu dans un restaurant : les décisions possibles correspondent aux combinaisons d'entrée, de plat de résistance, de dessert et de vin.

Ces catégories peuvent se combiner entre elles. On peut donc parfaitement imaginer des problèmes multiattributs portant sur un choix collectif entre plusieurs décideurs, en présence de multiples critères et avec une incertitude planant sur les conséquences de la

décision à prendre.

Cette thèse porte sur la décision dans le certain. Nous supposerons donc que les conséquences d'une décision sont connues avec certitude au moment où la décision est prise. De plus, cette thèse est centrée sur le domaine de la décision multiattribut, c'est-à-dire que l'*espace des alternatives* (l'ensemble de toutes les décisions possibles) sera défini comme un produit cartésien d'ensembles (les attributs).

Une première difficulté liée à la présence de multiples attributs vient de la taille de l'espace des alternatives. Supposons qu'un problème porte sur un espace de n attributs comportant chacun deux valeurs possibles. Alors le nombre d'alternatives composant cet espace est exactement 2^n. Donc, représenter (et éliciter) l'intégralité des préférences d'un utilisateur devient impensable dès que le nombre d'attributs devient conséquent. Pour palier ce problème, différents modèles ont été inventés pour représenter les préférences d'un décideur, que ce soit d'une façon qualitative (comme les CP-nets (Boutilier et al., 2004a,b) ou les TCP-nets (Brafman et Domshlak, 2002)) ou quantitative (comme les UCP-nets (Boutilier et al., 2001), les CUI-nets (Engel et Wellman, 2006)).

Les modèles qualitatifs offrent des représentations compactes ne requérant qu'un nombre limité de questions pour éliciter les préférences d'un décideur, mais ne permettent pas de décrire finement ces préférences. Les modèles quantitatifs exploitent des fonctions d'utilité permettant une représentation plus approfondie de ces préférences, au détriment d'une phase d'élicitation plus longue. Nous avons choisi dans le cadre de cette thèse d'utiliser le modèle GAI (Generalized Additive Independence (Fishburn, 1970)), un modèle quantitatif exploitant dans sa représentation les indépendances entre attributs présentes dans les préférences du décideur. Ce modèle théorique a été étendu pour offrir des représentations compactes des préférences dans un formalisme proche des réseaux bayésiens : les réseaux GAI (Gonzales et Perny, 2004; Braziunas et Boutilier, 2005).

Objectifs de la thèse. Les réseaux GAI possèdent un fort pouvoir descriptif des préférences d'un décideur. L'algorithmique qui leur est associée permet de résoudre des problèmes comme la détermination de l'alternative préférée d'un décideur, la détermination des k meilleures alternatives (selon les préférences d'un décideur) ou encore, dans un contexte mixte entre décision multiattribut et décision multicrère, et sous certaines hypothèses, la détermination de l'alternative engendrant la meilleure solution de compromis entre les critères (solution agrégée) (Queiroz, 2008). Le but de cette thèse est d'étendre l'algorithmique associée aux réseaux GAI, en identifiant les problèmes computationnels engendrés par certains algorithmes et en proposant de nouvelles approches, ou encore en abordant de nouveaux problèmes de prise de décision encore non-résolus par une exploitation des réseaux GAI.

Structure de la thèse. Cette thèse est structurée en deux grandes parties : La première comporte deux chapitres et contient toutes les connaissances qu'il est nécessaire d'avoir pour lire la seconde partie. La second partie comporte trois chapitres et présente

les nouveaux algorithmes que nous avons élaborés pour contourner certains problèmes computationnels ou pour résoudre de nouveaux problèmes dans le domaine des réseaux GAI. Plus spécifiquement, le contenu des chapitres est le suivant :

- Le chapitre 1 est une présentation des domaines de la décision multiattribut et de la décision multicritère. Nous y abordons les problèmes de représentation des préférences d'un décideur, la nécessité d'élaborer des modèles en décision multiattribut, les problèmes de l'introduction de multiples critères et différentes approches pour résoudre ces problèmes (dominance de Pareto, de Lorenz, OWA, intégrale de Choquet, etc...).
- Le chapitre 2 présente le modèle GAI autour duquel gravite toute cette thèse, comment intégrer la notion de contraintes aux préférences représentées, et l'algorithmique associée aux réseaux GAI (et plus généralement aux forêts de jonction) dans la littérature.
- Le chapitre 3 reprend trois algorithmes vus dans le chapitre 2 pour traiter des structures de préférences pouvant rendre impossible l'utilisation de ces algorithmes. Une nouvelle approche y est d'abord proposée pour un de ces algorithmes, puis étendue aux deux autres.
- Le chapitre 4 traite d'un problème encore non résolu dans le domaine des réseaux GAI : La détermination de la frontière de Pareto en décision multiattribut et multicritère. Nous y présentons la NP-complétude du problème abordé. Nous proposons un algorithme exact pour le résoudre en exploitant la structure d'indépendance entre attributs des préférences du décideur, et une variante permettant d'approcher la solution du problème tout en ayant une garantie de performance paramétrable par le décideur.
- Le chapitre 5 présente un algorithme basé sur la notion de recherche heuristique dans les réseaux GAI. Nous y montrons comment effectuer un précalcul sur le contenu du réseau GAI pour générer automatiquement de bonnes heuristiques, puis nous présentons une approche générique capable de résoudre de nombreux problèmes de décision multiattribut et multicritère : détermination de la frontière de Pareto, de la frontière de Lorenz, et de la solution agrégée optimale (sous des hypothèses moins restrictives que l'algorithme de l'état de l'art).

Comment lire cette thèse ? Pour un lecteur ne connaissant pas les bases de la théorie de la décision, que ce soit dans le domaine multiattribut ou multicritère, la lecture du chapitre 1 est indispensable.

Pour un lecteur ne connaissant pas le modèle GAI et ses algorithmes, et plus généralement les algorithmes classiques pour les modèles graphiquement décomposables (triangulation, programmation dynamique non sérielle), il est vivement conseillé de lire le chapitre 2.

Concernant les travaux réalisés dans cette thèse, le chapitre 3 peut être lu de manière tout à fait indépendante des deux autres chapitres, mais une lecture du chapitre 4 est nécessaire avant d'aborder le chapitre 5. En s'inspirant de l'introduction de Dechter (2003), on obtient donc le graphe de dépendances entre chapitres de la figure 1, et les experts en réseaux bayésiens sauront déduire que, conditionnellement aux connaissances contenues dans les chapitre 1 et 2, le chapitre 3 est indépendant des chapitre 4 et 5 (voir l'arbre de jonction de la figure 2).

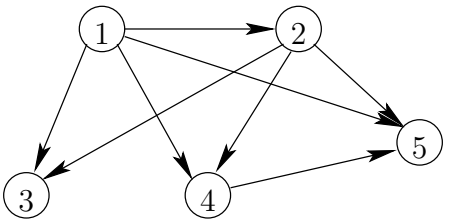


FIGURE 1 – Graphe de dépendances entre les chapitres

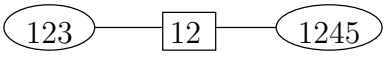


FIGURE 2 – Arbre de jonction des dépendances

Notations utilisées

$\mathcal{X}$	Ensemble des alternatives
n	Nombre d'attributs du problème
m	Nombre de critères du problème
$X_1, \ldots, X_n$	Attributs
x	Alternative ($x \in \mathcal{X}$)
x_i	Instanciation d'un attribut ($x_i \in X_i$)
x^*	Alternative préférée
$x^{(1)}, \ldots, x^{(k)}$	k meilleures alternatives
u	Fonction d'utilité (vectorielle en multicritère)
u_i	Sous-fonction d'utilité ($u = \sum u_i$)
C_i	Cliques (ensemble d'indices d'attributs)
C	Ensemble des cliques ($C = \{C_1, \ldots, C_q\}$)
S_{ij}	Séparateur entre la clique C_i et la clique C_j ($S_{ij} = C_i \cap C_j$)
x_{C_i}	Projection d'une alternative x sur les attributs indicés dans C_i
X_{C_i}	Produit cartésien des attributs indicés dans C_i ($X_{C_i} = \prod_{j \in C_i} X_j$)
$\hat{u}$	Utilité approximée
u^i	Fonction d'utilité du critère i ($u(x) = (u^1(x), \ldots, u^m(x))$)
v	Vecteur à m composantes
v^i	i-ème composante du vecteur v
$\mathcal{U}$	Ensemble des vecteurs de critères
$\mathrm{ND}_P(\mathcal{U})$	Ensemble de non-dominés (dominance de Pareto)
$\mathrm{ND}_L(\mathcal{U})$	Ensemble de non-dominés (dominance de Lorenz)
$\phi_{i,j}$	Message (hypermatrice) transitant sur le séparateur S_{ij} (algorithmes basés sur une collecte)
ψ_i	Hypermatrice contenue dans la clique C_i (algorithmes basés sur une collecte)

Première partie

Etat de l'art

Chapitre 1

Prise de décision multiattribut et multicritère

Résumé. *Dans ce chapitre, nous allons définir les notions de base pour représenter les préférences d'un décideur. Dans un premier temps, la notion de préférence sera formalisée, et nous présenterons les conditions permettant à un modèle qualitatif (la relation de préférence) d'être représenté par un modèle quantitatif (la fonction d'utilité). Puis nous resituerons ces notions dans le cadre d'attributs multiples et nous présenterons l'intérêt d'introduire un modèle qui soit simplificateur tout en permettant de représenter une grande diversité de comportements décisionnels. Dans un second temps, nous introduirons la notion de critère, et nous présenterons les difficultés qu'impliquent la prise en compte de critères multiples, avant de finalement introduire différentes problématiques de l'analyse multicritère (frontière de Pareto, de Lorenz, moyenne pondérée ordonnée, intégrale de Choquet, norme de Tchebycheff).*

1.1 Préférence et utilité

1.1.1 Bases pour la prise de décision

Qu'est-ce qu'une bonne décision ? Peut-on tout le temps expliquer les raisons de ses choix ? Comment différencier le comportement de personnes ? Comment les individus gèrent-ils l'incertitude sur les conséquences de leurs choix ? Comment des compromis peuvent-ils émerger d'une décision collective ? Comment gérer des critères de choix contradictoires ? Nous sommes confrontrés à ces questions dans la vie de tous les jours, mais nous avons souvent des difficultés pour y répondre. La prise de décision est souvent un processus très complexe, tant d'un point de vue cognitif que d'un point de vue computationnel. C'est pourquoi, dans la littérature comme dans cette thèse, nous nous focaliserons sur des modèles de décision simplificateurs relativement faciles à appréhender par le décideur tout en étant proches du « vrai » processus décisionnel du décideur et justifiables d'un point de vue rationnel.

Le but de cette thèse est d'élaborer des algorithmes capables de prendre des décisions automatiquement en se basant sur la perception d'un être humain ou artificiel, que nous nommerons *décideur*. Nous appellerons *alternative* une décision particulière. L'ensemble de toutes les décisions qu'un décideur peut potentiellement prendre sera *l'espace des alternatives*. Le décideur doit faire un *choix* parmi les différentes alternatives qui s'offrent à lui, et chaque décision implique des *conséquences*. Nous nous placerons dans le domaine du *certain*, c'est-à-dire que l'incertitude n'influera pas sur les conséquences d'une décision. Ainsi, conséquence et décision peuvent se confondre puisque le décideur connaît déjà exactement les conséquences de ses décisions (par exemple, pour le choix d'un meuble à acheter, nous connaissons déjà les tarifs, les dimensions du meuble, son apparence et sa qualité), et les *préférences* du décideur sur les conséquences de son choix seront exprimées sous forme de préférences sur les alternatives. De cette façon, le problème du choix optimal se résume à la détermination de l'alternative préférée du décideur.

Nous noterons $\mathcal{X}$ l'espace des alternatives, et nous supposerons que cet ensemble est fini (on est donc capable de borner le nombre de décisions possibles). Le long de cette thèse, nous nous intéresserons à une approche en deux phases, symbolisée par l'algorithme ci-dessous :

Algorithme 1 : Prise de décision en deux phases

```
1 M ← Eliciter(préférences du décideur);
2 retourner Exploiter(M);
```

Cet algorithme générique repose sur trois composantes :

- Le *modèle* $\mathcal{M}$: Il s'agit d'une structure de données permettant de représenter les préférences du décideur tout étant compacte en mémoire.
- Un algorithme d'*élicitation* : Il instancie le modèle, c'est-à-dire qu'il remplit la structure de données choisie pour la faire correspondre aux préférences du décideur. Ce remplissage peut se faire par l'intermédiaire de questions posées directement au décideur, ou de façon plus indirecte (comme, par exemple, en analysant ses actions sur un site internet).
- Un algorithme d'*exploitation* : Une fois le modèle $\mathcal{M}$ instancié, la phase d'exploi-

tation a pour but de proposer une décision à prendre qui soit optimale selon les préférences du décideur.

Cette thèse s'intéresse à la phase d'exploitation. Le but de ce chapitre est tout d'abord d'introduire les notions de base nécessaires à la représentation des préférences du décideur. Puis nous présenterons la notion de décision multiattribut (au sein de la section 1.2), c'est-à-dire le cas où une alternative peut être représentée comme un ensemble de caractéristiques ou attributs. Finalement, dans la section 1.3, nous nous intéresserons à l'introduction de critères, correspondant à l'introduction de différents « points de vue » sur un problème donné.

Bien que ce chapitre n'ait pas pour but d'entrer dans les détails d'un modèle donné, nous orienterons nos présentations des problématiques de manière à concorder avec le modèle qui sera présenté dans le chapitre 2.

1.1.2 Relation de préférence

La notion de préférences pour un décideur est très vaste : le décideur peut préférer une alternative à une autre, les juger incomparables, ou encore être indifférent entre deux alternatives. Cette diversité de comportements possibles peut être exprimée sous la forme d'une relation mathématique binaire sur l'espace des alternatives.

Définition 1 (Relation de préférence) *Soit $\mathcal{X}$ l'espace des alternatives, on définit $\succsim$ une relation de préférence comme étant un ensemble vérifiant $\succsim \subseteq \mathcal{X} \times \mathcal{X}$.*
Soit $x \in \mathcal{X}$ et $y \in \mathcal{X}$, on notera $x \succsim y$ l'assertion $(x, y) \in \succsim$.

Cette définition est classique en mathématiques. Toutefois nous allons lui attacher une sémantique liée à nos préférences. Ainsi, $x \succsim y$ sera lu « le décideur préfère x à y ou est indifférent entre les deux », et si, pour deux éléments, nous n'avons ni $x \succsim y$, ni $y \succsim x$, nous lirons « x et y sont incomparables ».

A partir de cette définition, nous pouvons extraire deux nouvelles relations pour formaliser au mieux les préférences du décideur : les préférences strictes et la relation d'indifférence.

Définition 2 (Relation de préférence stricte) *Soit $\mathcal{X}$ un espace d'alternatives et $\succsim$ une relation de préférence sur $\mathcal{X}$. On appelle relation de préférence stricte issue de $\succsim$ (et on notera $\succ$), la partie asymétrique de $\succsim$. On aura donc :*

$$\forall x \in \mathcal{X}\ \forall y \in \mathcal{X}\ x \succ y \iff x \succsim y \text{ et } Non(y \succsim x).$$

Définition 3 (Relation d'indifférence) *Soit $\mathcal{X}$ un espace d'alternatives et $\succsim$ une relation de préférence sur $\mathcal{X}$. On appelle relation d'indifférence issue de $\succsim$ (et on notera $\sim$), la partie symétrique de $\succsim$. On aura donc :*

$$\forall x \in \mathcal{X}\ \forall y \in \mathcal{X}\ x \sim y \Leftrightarrow x \succsim y \text{ et } y \succsim x.$$

Ces relations fournissent une base pour pouvoir raisonner qualitativement sur les préférences d'un décideur. Nous supposons que toute personne possède une relation de préférence, et le but d'une élicitation est d'être capable de déterminer une telle relation de préférence sous-jacente à un décideur en lui posant des questions.

1.1.3 Fonction d'utilité

Dans le cadre de nos travaux, nous avons supposé l'existence de certaines propriétés sur la structure de la relation de préférence du décideur. Ces hypothèses peuvent sembler restrictives. Toutefois, nous verrons qu'elles sont assez naturelles, et permettent d'utiliser une nouvelle représentation des préférences : la fonction d'utilité. De plus, de nombreux travaux en théorie de la décision dans l'incertain s'attachent à définir la rationalité d'un décideur et décrivent ce type de représentation (voir von Neumann et Morgenstern, 1947; Savage, 1954). Dans le certain, nous supposerons qu'un décideur rationnel vérifie les hypothèses que nous allons énoncer. Nous verrons toutefois qu'il peut exister des décideurs ne vérifiant pas ces hypothèses mais qui ne semblent pas irrationels pour autant. Nous allons, dans un premier temps, introduire le vocabulaire sur les structures particulières des relations de préférence que nous allons exploiter.

Définition 4 (Relation réflexive) *Soit R une relation sur un ensemble $\mathcal{X}$, on dit que R est réflexive si, et seulement si, $\forall x \in \mathcal{X}$ xRx.*

Du point de vue des préférences, la réflexivité correspond au fait que, lorque le décideur compare deux alternatives identiques, il doit être indifférent entre elles. Il semble assez naturel qu'un décideur considère toujours de la même façon une même alternative.

Définition 5 (Relation symétrique) *Soit R une relation sur un ensemble $\mathcal{X}$, on dit que R est symétrique si, et seulement si, $\forall (x, y) \in \mathcal{X}^2$ $xRy \Leftrightarrow yRx$.*

Considérons non plus la relation de préférence, mais la relation d'indifférence entre deux alternatives, cette relation doit être nécessairement symétrique et respectera cette propriété.

Définition 6 (Relation transitive) *Soit R une relation sur un ensemble $\mathcal{X}$, on dit que R est transitive si, et seulement si, $\forall (x, y, z) \in \mathcal{X}^3$ xRy et $yRz \implies xRz$.*

Des préférences transitives peuvent être vues comme une affirmation des choix du décideur : si un décideur préfère une alternative x à une autre y, ou est indifférent entre les deux, alors x sera forcément préféré ou indifférent à toute alternative qui était dépréciée par rapport à y.

Définition 7 (Relation complète) *Soit R une relation sur un ensemble $\mathcal{X}$, on dit que R est complète si, et seulement si, $\forall (x, y) \in \mathcal{X}^2$ xRy ou yRx.*

La complétude des préférences supprime la possibilité de ne pouvoir être capable de comparer deux alternatives. Donc, soit le décideur en préfère une des deux, soit il est indifférent entre les deux. Il semble de nouveau assez logique qu'un décideur puisse avoir des préférences complètes pour réussir à déterminer son alternative préférée.

Dans la suite de cette thèse, nous supposerons que la relation de préférence du décideur est réflexive, transitive et complète, et que sa relation d'indifférence est symétrique.

Remarque 1 *Il est à noter qu'il peut exister des décideurs dont la relation de préférence (et d'indifférence) ne vérifient pas les définitions énoncées précédemment. Pour reprendre un exemple classique dans Luce (1956), Luce et Raiffa (1957, page 346) et Krantz et al.*

(1989, page 301), considérons un décideur appréciant le café sucré et tentons de représenter ses préférences sur la concentration de sucre qu'il apprécie dans son café. Ce décideur ne sentira jamais la différence entre boire son café avec n grains de sucre ou $n+1$ grains de sucre. Donc, si on considère que l'espace des alternatives est l'ensemble des grains de sucre qu'il peut potentiellement mettre dans son café, on aura $n \sim n+1$. Or, si on suppose que $\sim$ est une relation transitive, on aura $0 \sim n$ quelle que soit la valeur de n, ce qui ne correspond pas aux préférences du décideur, puisqu'il préfère un café sucré à un café non sucré. Pourtant, les préférences du décideur semblent extrêmement naturelles.

Toutefois, nous pouvons relativiser ces exemples en citant les travaux de Wakker (1989) qui a recensé plusieurs domaines d'application où une modélisation des relations de préférence respecte les hypothèses énoncées précédemment (par exemple, la théorie du consommateur, la théorie du producteur, ...).

Les définitions suivantes correspondent à des regroupements de caractéristiques donnant à une relation des propriétés intéressantes à exploiter.

Définition 8 (Relation d'équivalence) *Une relation d'équivalence est une relation réflexive, symétrique et transitive.*

Sous les hypothèses que nous avons posées, la relation d'indifférence $\sim$ est une relation d'équivalence, au même titre que le symbole $=$ en algèbre. L'intérêt des relations d'équivalence est qu'elles permettent de définir des classes d'équivalence : il s'agit d'ensembles d'éléments liés entre eux par une telle relation.

Définition 9 (Classe d'équivalence) *Soit R une relation d'équivalence sur $\mathcal{X}$ et x un élement de $\mathcal{X}$. La classe d'équivalence de x par rapport à R est l'ensemble $\{y \in \mathcal{X}/xRy\}$.*

Ainsi, l'ensemble des classes d'équivalence de $\mathcal{X}$ selon la relation d'indifférence $\sim$ (aussi appelé ensemble quotient) correspond à des regroupements d'alternatives qui sont jugées totalement équivalentes entre elles. La notion de classe d'équivalence sera aussi utilisée par la suite dans la section 2.1.2 pour introduire la notion de composante connexe d'un graphe.

Définition 10 (Préordre complet) *Un préordre complet est une relation réflexive, transitive et complète.*

En utilisant les travaux de Debreu (1964), il existe une représentation alternative de la relation de préférence d'un décideur si celle-ci est un préordre complet, en supposant que $\mathcal{X}$ soit fini : la fonction d'utilité.

Propriété 1 (Fonction d'utilité) *Soit $\succsim$ une relation de préférence sur un ensemble d'alternatives $\mathcal{X}$ fini, telle que $\succsim$ soit un préordre complet. Alors, il existe une fonction $u : \mathcal{X} \to \mathbb{R}$ telle que :*

$$\forall (x, y) \in \mathcal{X}^2 \; x \succsim y \Leftrightarrow u(x) \geqslant u(y).$$

On dira que u est une fonction d'utilité de la relation de préférence $\succsim$.

Ainsi, la relation de préférence d'un décideur peut s'exprimer de façon quantitative par une fonction d'utilité, et il suffit donc d'associer à chaque alternative un nombre réel. Les préférences seront ainsi exprimées par l'intermédiaire des relations de comparaison classiques sur $\mathbb{R}$ (celle utilisée dans la définition des fonctions d'utilité, ou bien celles de la propriété 2).

Propriété 2 *Soit $\succsim$ une relation de préférence sur un ensemble d'alternatives $\mathcal{X}$ représentable par une fonction d'utilité u. Considérons $\succ$ et $\sim$, les relations de préférence stricte et d'indifférence issues de $\succsim$. Alors on aura :*

$$\forall (x,y) \in \mathcal{X}^2 \; x \sim y \Leftrightarrow u(x) = u(y),$$
$$\forall (x,y) \in \mathcal{X}^2 \; x \succ y \Leftrightarrow u(x) > u(y).$$

Il est à noter que, comme le montre l'exemple 1, la fonction d'utilité n'est pas unique. La propriété 3 nous montre que, pour une relation de préférence donnée, une infinité de fonctions d'utilité représentant cette relation existe. En pratique, il ne faut pas attacher de sémantique aux valeurs de la fonction d'utilité autre qu'une sémantique ordinale : les nombres ne servent qu'à se comparer entre eux et n'indiquent aucunement un niveau de préférence (par exemple, si $u(x) = 5$ et $u(y) = 10$, on peut en déduire que y est préféré à x, mais cela ne veut pas dire que le décideur préfère « deux fois plus » y à x).

Propriété 3 *Soit u une fonction d'utilité de la relation de préférence $\succsim$, et $f : \mathbb{R} \to \mathbb{R}$ une fonction strictement croissante, soit $v : \mathcal{X} \to \mathbb{R}$ la fonction définie par $v = f \circ u$, alors v est une fonction d'utilité de $\succsim$.*

Exemple 1 *Considérons un décideur ayant envie d'acheter de la musique dans un magasin ne contenant pas tous les styles de musique (pour simplifier). L'ensemble des alternatives sera $\mathcal{X} = \{Classique\ (C), Jazz\ (J), Métal\ (M), Rap\ (R)\}$. Ce décideur étant féru d'envolées de guitare, et de rythmes effrénés, il préfèrera acheter du métal par rapport aux autres styles musicaux. Toutefois, il reste amateur de structures musicales étudiées aussi bien que d'improvisations riches en émotions. Il est donc indifférent entre le jazz et la musique classique. Par contre, il n'a jamais pu supporter le rap, et préfèrera de toutes façons les autres styles de musique à celui-ci. On peut décrire sa relation de préférence par : $M \succsim M, J \succsim J, C \succsim C, R \succsim R, M \succsim C, M \succsim J, M \succsim R, J \succsim C, C \succsim J, J \succsim R$ et $C \succsim R$. On en déduit les relations de préférence stricte : $M \succ J, M \succ C, M \succ R, J \succ R, C \succ R$ et d'indifférence : $M \sim M, J \sim J, C \sim C, R \sim R, J \sim C, C \sim J$. La relation $\succsim$ étant un préordre complet, on peut donc créer des fonctions d'utilité pour la représenter. Par exemple, en posant :*

$$\begin{array}{llll} u(M) = 666 & u(J) = 100 & u(C) = 100 & u(R) = 93 \\ v(M) = 1 & v(J) = 0.999 & v(C) = 0.999 & v(R) = 0.998 \\ w(M) = 10^9 & w(J) = 10^{-9} & w(C) = 10^{-9} & w(R) = -10^{200} \end{array}$$

On peut remarquer que u, v et w sont trois fonctions d'utilité possibles pour représenter $\succsim$. Seule la comparaison entre les nombres importe, les valeurs ne sont pas importantes en tant que telles.

1.2 Utilité sur des attributs

Dans la section précédente, nous avons vu que la relation de préférence d'un décideur peut se représenter par une fonction d'utilité. Ainsi, la phase d'élicitation de certains logiciels d'aide à la décision revient à déterminer une fonction d'utilité représentant les préférences du décideur, puis la détermination des décisions optimales correspondra à un algorithme exploitant la fonction élicitée. Nous allons maintenant introduire la notion d'attributs et étudier les difficultés de représentation des utilités multiattributs.

1.2.1 Utilité multiattribut

Dans l'exemple 1, le décideur devait choisir le style de musique qu'il désirait acheter, l'ensemble des styles de musique correspondait à l'ensemble des alternatives $\mathcal{X}$. On peut objecter à cet exemple son aspect extrêmement simpliste. En pratique, il est rare d'entrer dans un magasin pour acheter un album d'un style sans se soucier des particularités d'un groupe de musique, mais ces particularités peuvent être multiples, et chaque groupe de musique peut être vu comme une « collection » de ces particularités. Adopter un tel point de vue sur l'ensemble des alternatives revient à raisonner en présence de multiples attributs. Pour formaliser et généraliser une telle approche, il est nécessaire de redéfinir l'espace des alternatives pour intégrer la notion d'attributs.

Définition 11 (Attributs) *Soit $\mathcal{X}$ un espace d'alternatives, et $X_1, \ldots, X_n$ des ensembles. On dit que $X_1, \ldots, X_n$ sont les attributs constituant $\mathcal{X}$ si, et seulement si, $\mathcal{X} = \prod_{i=1}^{n} X_i$. Les éléments d'un attribut X_i sont appelés modalités de X_i.*
Fixer la valeur d'un attribut à une de ses modalités se nomme aussi instancier X_i et la modalité choisie est dite instanciation de X_i.

Exemple 2 *Notre décideur mélomane a décidé de revoir sa façon d'appréhender ses goûts musicaux. Pour lui, chaque album possède son style de musique (X_1), une note de la critique spécialisée (X_2), un prix d'achat (X_3), et un niveau de rareté (X_4). Son ensemble d'alternatives $\mathcal{X}$ devient donc le produit cartésien (ou, dans certains cas, un sous-ensemble du produit cartésien) de ces attributs ($\mathcal{X} = X_1 \times X_2 \times X_3 \times X_4$), et chaque alternative sera ainsi représentée par un n-uplet. Par exemple $(M, 5, 10, Courant)$ correspondra à un album du style M (voir l'exemple 1), noté 5 par la critique, coûtant 10 euros, et considéré comme facile à trouver en magasin.*

L'espace des alternatives devient donc un produit cartésien d'ensembles appelés attributs. Ainsi, une alternative particulière sera un n-uplet composé d'instanciations de ces attributs. Dans la suite de cette thèse, nous noterons soit x pour se référer à l'intégralité du n-uplet, soit $(x_1, \ldots, x_n)$ avec $\forall i \in \{1, \ldots, n\}\ x_i \in X_i$. Nous allons supposer que tous les attributs sont des ensembles finis (donc leur produit cartésien sera un ensemble fini). Sous les hypothèses énoncées en 1.1.3, la relation de préférence du décideur est de nouveau exprimable sous forme d'une fonction d'utilité u, et nous noterons indifféremment $u(x)$ ou $u(x_1, \ldots, x_n)$.

Nous supposerons que $\forall i \in \{1, \ldots, n\}\ |X_i| \geqslant 2$. Il est en effet difficile d'envisager un attribut ne contenant pas d'éléments, et un attribut contenant un unique élément ne présente que peu d'intérêt : il suffit d'instancier à chaque fois cet attribut avec sa seule valeur possible. La propriété 4 montre que l'espace $\mathcal{X}$ contient un nombre d'alternatives

qui est exponentiel par rapport au nombre d'attributs du problème. Cette explosion combinatoire rend déraisonable une représentation exhaustive d'une fonction d'utilité portant sur l'espace des alternatives dès lors que le nombre d'attributs est élevé. De plus, il devient inenvisageable d'éliciter les préférences du décideur, car le nombre de questions à poser est bien trop élevé.

Propriété 4 (Ordre de grandeur de l'ensemble des alternatives) *Si* $\mathcal{X} = \prod_{i=1}^{n} X_i$ *avec* $\forall i \in \{1, \ldots, n\}\ |X_i| \geqslant 2$, *alors* $|\mathcal{X}| \in \Omega(2^n)$.

Preuve. Supposons que $\mathcal{X} = \prod_{i=1}^{n} X_i$ avec $\forall i \in \{1, \ldots, n\}\ |X_i| \geqslant 2$, alors on aura $|\mathcal{X}| = \prod_{i=1}^{n} |X_i|$, d'où $|\mathcal{X}| \geqslant \prod_{i=1}^{n} 2 = 2^n$. On a donc bien $\forall n \geqslant 0\ \exists n_0 \in \mathbb{N}\ n > n_0 \implies |\mathcal{X}| \geqslant 2^n$. En passant en notation asymptotique, on a donc : $|\mathcal{X}| \in \Omega(2^n)$. ■

Une conséquence de cette observation est la nécessité de définir un nouveau mode de représentation des préférences pour réussir à contourner ce problème de stockage et d'élicitation. Il existe dans la littérature de nombreuses propositions de modèles, que ce soit d'un point de vue qualitatif (comme les CP-nets (Boutilier et al., 2004a,b) ou les TCP-nets (Brafman et Domshlak, 2002)), ou quantitatif, en travaillant sur la fonction d'utilité (comme les UCP-nets (Boutilier et al., 2001), les CUI-nets (Engel et Wellman, 2006), les utilités additives (Fishburn, 1970) ou encore les utilités multilinéaires (Keeney et Raiffa, 1993; Bacchus et Grove, 1995)). Dans le cadre de cette thèse, nous avons choisi de travailler dans le formalisme des réseaux GAI (Gonzales et Perny, 2004). Ce modèle réalise un bon compromis entre la compacité de la représentation et son expressivité, tout en restant simple d'utilisation. Les réseaux GAI feront l'objet de l'intégralité du chapitre 2.

1.2.2 Exemple de décomposition : les utilités additives

Une première approche pour réduire la taille de la représentation de la fonction d'utilité est de considérer que chaque attribut possède sa propre fonction d'utilité, et que la fonction d'utilité de la relation de préférence du décideur est obtenue par sommation de l'utilité de chaque attribut (chaque attribut sera donc considéré indépendamment des autres, voir Fishburn (1970); Krantz et al. (1971); Bacchus et Grove (1995)).

Définition 12 (Décomposition additive) *Soit u une fonction d'utilité définie sur l'espace* $\mathcal{X} = \prod_{i=1}^{n} X_i$, *on dit que u se décompose additivement si, et seulement si,* $\forall i \in \{1, \ldots, n\}\ \exists u_i : X_i \to \mathbb{R}$ *vérifiant* $\forall (x_1, \ldots, x_n) \in \mathcal{X}\ u(x_1, \ldots, x_n) = \sum_{i=1}^{n} u_i(x_i)$.

Propriété 5 (Ordre de grandeur de la décomposition additive) *Soit u une fonction d'utilité définie sur l'espace* $\mathcal{X} = \prod_{i=1}^{n} X_i$ *décomposable additivement en* $u(x_1, \ldots, x_n) = \sum_{i=1}^{n} u_i(x_i)$, *alors l'espace occupé en mémoire par cette représentation est en* $O(kn)$ *avec* $k = \max_{i=1}^{n} |X_i|$.

Preuve. Pour représenter u en mémoire, il suffit de représenter chaque u_i et, à chaque alternative $x \in \mathcal{X}$, on pourra associer $u(x)$ par sommation des u_i. Soit $k = \max_{i=1}^{n} |X_i|$. On a donc $\forall i \in \{1, \ldots, n\}\ |X_i| \leqslant k$. On en déduit une borne supérieure du nombre de valeurs réelles à stocker pour représenter tous les u_i : $\sum_{i=1}^{n} |X_i| \leqslant \sum_{i=1}^{n} k = kn$. Chaque valeur réelle ayant une représentation de taille constante en mémoire, on en déduit que

l'espace occupé en mémoire par cette représentation est en $O(kn)$. ■

Une telle approche est à la fois simple et compacte (voir la propriété 5), mais suppose une indépendance complète en terme de préférences entre les attributs du problème, indépendance qu'il est rare de rencontrer en pratique, comme le montre l'exemple 3, très inspiré de Gonzales et Perny (2004).

Exemple 3 *Notre mélomane préféré, après s'être délecté de douces symphonies, sent soudainement la faim lui tirailler le ventre. Il se rend donc dans un restaurant et doit choisir les aliments faisant partie de son menu. Le menu comprend un plat de résistance à choisir parmi* $X_1 = \{viande\ (V), poisson\ (P)\}$, *un dessert* $X_2 = \{cake\ (C), sorbet\ (S)\}$ *et du vin* $X_3 = \{rouge\ (R), blanc\ (B)\}$. *L'ensemble des menus composables correspondra donc à* $\mathcal{X} = X_1 \times X_2 \times X_3$.

Ce décideur appréciant que le vin s'accorde avec son plat, il préfèrera boire du vin rouge avec de la viande, et du vin blanc avec du poisson. En sachant qu'il préfère en second lieu la viande au poisson, et que, dans une moindre mesure, il préfèrera manger du sorbet s'il a mangé de la viande (afin d'éviter que le repas ne soit trop lourd), et du cake s'il a mangé du poisson.

Ces préférences sont incompatibles avec une représentation des préférences par des utilités additives : En sachant que $(V,R,C) \succ (P,R,C)$, *on aura* $u_1(V) + u_2(R) + u_3(C) > u_1(P) + u_2(R) + u_3(C)$ *et donc* $u_1(V) > u_1(P)$, *mais en prenant en compte que* $(P,B,C) \succ (V,B,C)$, *on aura en suivant le même raisonnement* $u_1(P) > u_1(V)$. *Cette incompatibilité est dûe aux interactions entre les attributs : le plat et le vin ne peuvent se déterminer séparément. Il en est de même pour le plat et le dessert. Toutefois, le problème reste tout de même décomposable : Posons deux fonctions* $u_{1,2} : X_1 \times X_2 \to \mathbb{R}$ *et* $u_{1,3} : X_1 \times X_3 \to \mathbb{R}$, *alors il est possible de déterminer* $u_{1,2}$ *et* $u_{1,3}$ *de telle sorte que* $\forall (x_1,x_2,x_3) \in \mathcal{X}\ u(x_1,x_2,x_3) = u_{1,2}(x_1,x_2) + u_{1,3}(x_1,x_3)$, *en posant par exemple les valeurs suivantes :*

$$\begin{array}{llll} u_{1,3}(V,R) = 6 & u_{1,3}(P,B) = 4 & u_{1,3}(V,B) = 2 & u_{1,3}(P,R) = 0 \\ u_{1,2}(V,S) = 1 & u_{1,2}(P,C) = 1 & u_{1,2}(V,C) = 0 & u_{1,2}(P,S) = 0 \end{array}$$

Une telle décomposition est dite additive généralisée ou GAI-décomposition. Nous l'étudierons en détail dans le chapitre 2. Supposons maintenant que le menu soit plus varié et que le nombre de plats, desserts ou vins différents ne soit plus de 2 mais de k, *alors stocker exhaustivement la fonction d'utilité* u *nécessitera la représentation de* k^3 *réels, tandis que stocker* $u_{1,2}$ *et* $u_{1,3}$ *nécessitera de représenter* $2k^2$ *réels. On observera donc un gain de stockage à partir du moment où* $k > 2$.

1.3 Décision multicritère

Dans les sections précédentes, nous nous sommes intéressés à un décideur ne possédant qu'un seul point de vue sur l'espace des alternatives, ce point de vue étant représenté par une relation de préférence. La décision multicritère traite des problèmes liés à la présence de plusieurs points de vue de la part du décideur, chaque point de vue ayant sa propre relation de préférence. Nous appellerons *critères* les différentes relations de préférence correspondant à ces points de vue. À un décideur, nous n'associerons donc plus une relation $\succsim$ mais m relations $\succsim^1, \ldots, \succsim^m$, où $\succsim^i$ correspondra à la relation de préférence

du décideur associée au critère i. En procédant ainsi, la relation de préférence globale du décideur n'est plus forcément représentable par une fonction d'utilité (contrairement aux relations liées aux critères). De nouveaux problèmes apparaissent donc naturellement : Comment peut-on comparer deux alternatives entre elles ? Qu'est-ce qu'une alternative préférée en présence de multiples critères ? Comment chercher une solution de compromis entre plusieurs critères contradictoires ?

Cette section présentera dans un premier temps les problématiques liées à la prise de décision en présence de multiples critères (en 1.3.1) pour aboutir à une première formulation de problèmes sous la forme d'une recherche d'un ensemble d'alternatives intéressantes. Dans la section 1.4, nous étudierons une autre approche des problèmes multicritères et nous reformulerons un nouvel ensemble de problèmes visant à rechercher une alternative préférée.

1.3.1 Problématiques

Le premier problème que nous aborderons est lié à la comparaison de deux alternatives. Comment, par exemple, comparer une alternative x et une alternative y, en sachant que $x \succsim^1 y$ et $y \succsim^2 x$? Une approche classique, *l'agrégation des préférences*, consiste à exploiter les relations de préférence de chaque critère pour construire une relation globale de préférence $\succsim$ ou d'indifférence $\sim$. Toutefois, établir une telle relation semble difficile car elle peut nécessiter plus d'informations de la part du décideur que ses simples préférences sur chaque critère, comme par exemple l'importance qu'il attache aux différents critères, ou encore la possibilité d'une synergie sur un sous-ensemble de critères. Mais même sans posséder toutes les informations nécessaires, on peut tout de même remarquer qu'il semble logique qu'un décideur préfère x à y si $\forall i \in \{1, \ldots, m\}\ x \succsim^i y$. Effectuer des *comparaisons* entre alternatives ne nécessite donc pas forcément de relation de préférence agrégée. Parmi les différentes approches développées dans le domaine de la décision multicritère, nous pouvons retenir deux catégories :

- **Agréger puis comparer** : Cette approche consiste à bâtir une relation agrégée, puis à effectuer des comparaisons en utilisant cette relation.
- **Comparer puis agréger** : Cette approche exploite d'abord les relations $\succsim^1, \ldots, \succsim^m$ pour comparer les alternatives entre elles (ou par rapport à un profil de référence), avant de bâtir une relation de préférence globale $\succsim$.

Dans le cadre de cette thèse, nous nous situerons dans la catégorie « agréger puis comparer ». De plus, de la même façon que nous avions précédemment supposé que la relation de préférence du décideur était représentable par une fonction d'utilité, nous allons supposer chaque relation $\succsim^i$ est représentable par une utilité u^i (Vincke, 1989). Toutefois, pour découvrir d'autres façons de procéder en décision multicritère, nous vous renvoyons à la lecture de Roy (1985); Jacquet-Lagrèze (1975); Roubens et Vincke (1985); Greco et al. (2000)).

Nous disposons donc des fonctions d'utilité $u^1, \ldots, u^m$. Comment comparer deux alternatives entre elles ? Considérons $x \in \mathcal{X}$ et $y \in \mathcal{X}$, ainsi que deux fonctions d'utilité u^1 et u^2 associées à deux critères. Dans le cas où $u^1(x) = 1$, $u^2(x) = 0$, $u^1(y) = 0$ et $u^2(y) = 1$ (on aura donc $x \succ^1 y$ et $y \succ^2 x$), il semble difficile de comparer x et y sans faire d'hypothèse supplémentaire sur l'importance que le décideur attache aux différents critères. De plus, lorsque nous avons introduit la notion de fonction d'utilité, nous avons précisé que la seule sémantique à leur attacher est ordinale, les valeurs d'utilité de chaque critère

ne sont donc pas forcément commensurables entre elles. Cette difficulté supplémentaire pour comparer deux alternatives se répercute sur certaines problématiques classiques de la prise de décision (comme, par exemple, la détermination de l'alternative préférée d'un décideur ou le rangement de ces alternatives (Roy, 1985; Vincke, 1992)). Il est à noter qu'en pratique, le décideur humain réussit à bâtir une relation de dominance générale pour comparer les alternatives (Montgomery, 1987; Berthoz, 2003).

Une solution couramment répandue est d'utiliser une somme (éventuellement pondérée) entre les valeurs d'utilité de chaque critère, et de comparer les alternatives en fonction de cette somme. Cette solution peut paraître simple et logique. Nous avons longtemps été conditionné à ce raisonnement (par exemple, les fameuses moyennes pour classer des étudiants à un examen), mais l'exemple 4 (inspiré de Bouyssou et al. (2000)) nous montre que cette première intuition n'est pas forcément la plus intéressante.

Exemple 4 *Considérons un laboratoire ayant obtenu des financements pour recruter un doctorant en décision algorithmique. Une annonce est effectuée au sein des étudiants de master désirant continuer en thèse. Ces candidats sont classés selon deux aspects : leur résultat dans une unité d'enseignement théorique, et leur résultat dans une unité d'enseignement liée à la programmation. Les financements ne permettant de recruter qu'un seul thésard, il devra exceller dans ces deux domaines de façon à concevoir des algorithmes reposant sur des bases théoriques intéressantes, tout en étant capable de les programmer efficacement pour rédiger les sections expérimentales de ses futurs articles. Huit candidats se présentent. Leurs notes (sur 20) en programmation (P) et en décision (D) sont représentées par des cercles figure 1.1. Nous avons donc :* $(5,5)$, $(6,10)$, $(9,18)$, $(10,15)$, $(12,4)$, $(13,13)$, $(15,11)$ *et* $(18,9)$.

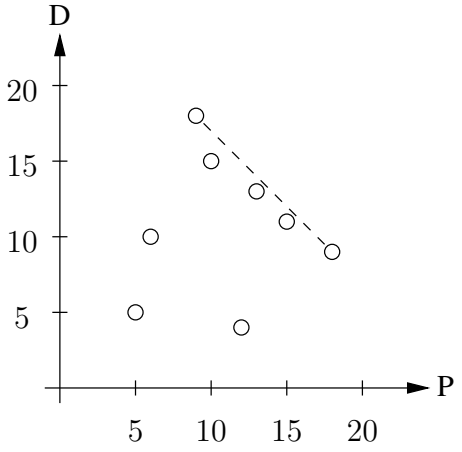


FIGURE 1.1 – Exemple des notes

Bien que cet exemple contienne un aspect très subjectif, nous pouvons classer les candidats en trois catégories : tout d'abord ceux correspondant aux notes $(5,5)$, $(6,10)$ *et* $(12,4)$ *semblent assez inintéressants par rapport aux autres candidats, les deux candidats* $(9,18)$ *et* $(18,9)$ *sont les meilleurs sur une des matières, mais semblent assez faibles sur l'autre enseignement. La dernière catégorie de candidats (contenant* $(10,15)$, $(13,13)$ *et* $(15,11)$*) semble réaliser un compromis entre les deux matières, et le candidat possédant*

13 *à ces deux enseignements a l'air d'être le plus équilibré. Classons les candidats par rapport à la moyenne qu'ils ont obtenu, nous avons donc :*

Notes :	$(5,5)$	$(6,10)$	$(9,18)$	$(10,15)$	$(12,4)$	$(13,13)$	$(15,11)$	$(18,9)$
Moyenne :	5	8	13.5	12.5	8	13	13	13.5

Les candidats ayant les meilleures moyennes sont donc les spécialistes d'un seul domaine, ce qui ne semble pas correspondre au profil recherché. On peut tenter d'intégrer des pondérations au sein de la moyenne, mais elles s'avèrent inefficaces pour obtenir n'importe quel autre candidat. En effet, une moyenne pondérée ne permet d'atteindre que des candidats appartenant à l'enveloppe convexe de la figure (en pointillé nous avons indiqué la zone de l'enveloppe convexe intervenant dans le cadre d'une moyenne sans pondération). Ce phénomène s'explique assez simplement : Supposons que nous cherchions une pondération de manière à ce que le candidat $(13,13)$ *soit avantagé par rapport au candidat* $(18,9)$*, il s'agit donc de déterminer deux coefficients* λ_1 *et* λ_2 *tels que* $13\lambda_1 + 13\lambda_2 > 18\lambda_1 + 9\lambda_2$*, on aura donc* $4\lambda_2 > 5\lambda_1$ *d'où :* $\lambda_2 > \frac{5}{4}\lambda_1$*. En suivant un raisonnement similaire entre* $(13,13)$ *et* $(9,18)$*, on en déduit que pour que* $(13,13)$ *puisse battre les deux spécialistes, les coefficients doivent respecter :*

$$\begin{cases} \lambda_1 > \frac{5}{4}\lambda_2, \\ \\ \lambda_2 > \frac{5}{4}\lambda_1. \end{cases}$$

Il n'existe donc pas une telle pondération, et donc on ne pourra jamais choisir $(13,13)$*. Pourtant il semble être le meilleur compromis entre les deux critères.*

Les problèmes multicritères ont été dans un premier temps étudiés en économétrie pour devenir un domaine ayant pris une grande importance ces quarante dernières années. De ces travaux ont émergé différents outils pour traiter ce type de problèmes, que nous pouvons ranger en deux catégories de problèmes : la détermination d'un ensemble d'alternatives intéressantes (que nous présenterons dans le reste de cette section pour finalement formaliser les problèmes en 1.3.3), et la recherche d'une unique alternative (qui sera le thème de l'intégralité de la section 1.4).

1.3.2 Comparaison de solutions vectorielles

Dans la suite de ce chapitre, nous nous situerons dans un problème à m critères, et nous associerons à chaque critère une fonction d'utilité $u^j : \mathcal{X} \to \mathbb{R}$ (pour $j \in \{1,\dots,m\}$). Dans l'exemple 4, nous avons manipulé les candidats à une thèse sous forme de vecteurs de notes. En suivant un raisonnement similaire, nous pouvons définir la notion de fonction d'utilité multicritère.

Définition 13 (Fonction d'utilité multicritère) *Soit* $\mathcal{X}$ *un espace d'alternatives et* $(u^1,\dots,u^m)$ *une famille de fonctions d'utilité sur* $\mathcal{X}$*. On appelle fonction d'utilité multicritère issue de* $(u^1,\dots,u^m)$ *la fonction* $u : \mathcal{X} \to \mathbb{R}^m$ *telle que* $\forall x \in \mathcal{X}\ u(x) = (u^1(x),\dots,u^m(x))$*.*

A chaque alternative, il est donc possible d'associer un vecteur correspondant aux valeurs d'utilité dans les différents critères. Toujours par analogie avec l'exemple 4, nous

avions représenté les candidats au sein d'une figure, cette figure était une représentation dans l'espace des critères.

Définition 14 (Espace des critères) *Soit $\mathcal{X}$ un espace d'alternatives, et $u : \mathcal{X} \to \mathbb{R}^m$ une fonction d'utilité multicritère, on définit l'espace des critères comme étant l'ensemble $\mathcal{U} \subset \mathbb{R}^m$ défini comme étant l'image de $\mathcal{X}$ par u, c'est-à-dire :*

$$\forall l \in \mathcal{U}\ \exists x \in \mathcal{X}\ u(x) = l.$$

Les travaux en décision multicritère manipulant des valeurs numériques se réfèrent à une comparaison de base pour évaluer des vecteurs entre eux : la dominance de Pareto. Issue de l'économie, cette relation exprime une idée assez intuitive. Supposons que nous ayons deux vecteurs : $v \in \mathcal{U}$ et $v' \in \mathcal{U}$, on sera sûr de préférer v à v' si v est au moins aussi bon que v' sur tous les critères (on parle alors de dominance faible), tout en étant strictement meilleur sur au moins un critère. Cette notion, une fois formalisée, donne les définitions suivantes :

Définition 15 (Dominance faible de Pareto) *Soit $v \in \mathbb{R}^m$ et $v' \in \mathbb{R}^m$ deux vecteurs. On dit que v domine v' au sens faible de Pareto, et on note $v \succsim_P v'$, si, et seulement si, $\forall j \in \{1, \ldots, m\}\ v^j \geqslant v'^j$.*

Définition 16 (Dominance de Pareto) *Soit $v \in \mathbb{R}^m$ et $v' \in \mathbb{R}^m$ deux vecteurs. On dit que v domine v' au sens de Pareto, et on note $v \succ_P v'$, si, et seulement si, $v \succsim_P v'$ et $\exists j \in \{1, \ldots, m\}\ v^j > v'^j$.*

Ces définitions peuvent s'interpréter schématiquement : étant donné un vecteur v dans l'espace des critères, on peut extraire une zone de l'espace dans laquelle tous les vecteurs sont dominés au sens de Pareto par v : on parle alors de *cône de dominance*, en référence à la forme de cette zone (voir l'exemple 5).

Exemple 5 *Reprenons l'exemple de nos prétendants au doctorat. Nous avions déterminé intuitivement trois types de candidats : ceux réalisant des prouesses dans une matière au détriment de l'autre, ceux semblant être de bons compromis entre les deux matières, et ceux ne semblant pas intéressants. En appliquant la dominance de Pareto sur les vecteurs de notes des candidats (la représentation des cônes de dominance pour chaque vecteur est donnée sur la figure 1.2), nous pouvons nous apercevoir que les trois candidats ne semblant pas intéressants (en noir sur la figure) sont tous dominés par au moins un vecteur, tandis que les deux autres catégories de candidats sont incomparables : aucun de ces vecteurs n'en domine un autre.*

1.3.3 Détermination de frontières

L'exemple 5 fait apparaître que la dominance de Pareto permet d'éliminer des vecteurs qui ne sont pas intéressants. En effet, si un vecteur est dominé au sens de Pareto par un autre vecteur, alors l'autre vecteur ne peut qu'éveiller notre intérêt à son égard, puisqu'il est au moins aussi bon que le premier sur tous les critères. En se basant sur ce raisonnement, on aboutit à la définition d'un premier type de problème : la détermination de la frontière de Pareto.

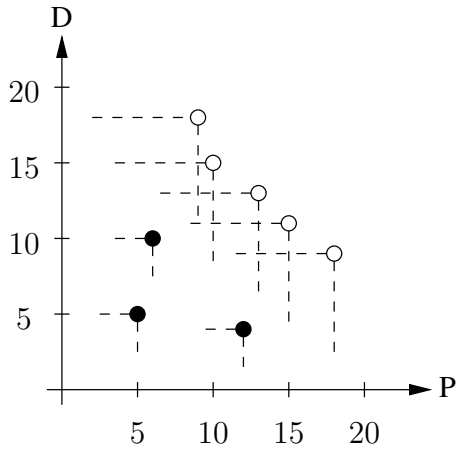


FIGURE 1.2 – Représentation des cônes de dominance

Définition 17 (Frontière de Pareto) *Soit $\mathcal{X}$ un espace d'alternatives, u une fonction d'utilité multicritère, et $\mathcal{U}$ l'espace des critères engendré par u. On appelle frontière de Pareto (ou ensemble de non-dominés au sens de Pareto), et on note $ND_P(\mathcal{U})$ l'ensemble de vecteurs défini par $ND_P(\mathcal{U}) = \{v \in \mathcal{U} / \nexists v' \in \mathcal{U}\ v' \succ_P v\}$.*
On introduit de la même façon l'ensemble des alternatives engendrant la frontière de Pareto par : $ArgND_P(\mathcal{X}, u) = \{x \in \mathcal{X} / \nexists x' \in \mathcal{X}\ u(x') \succ_P u(x)\}$.

Exemple 6 *Dans la figure 1.2, les vecteurs composant la frontière de Pareto sont : $\{(9,18),(10,15),(13,13),(15,11),(18,9)\}$. En effet, il n'existe aucun autre vecteur pour les dominer, tandis que pour les trois autres vecteurs, on a $(9,18) \succ_P (5,5)$, $(9,18) \succ_P (6,10)$ et $(13,13) \succ_P (12,4)$.*

Déterminer les alternatives engendrant la frontière de Pareto permet donc d'obtenir un ensemble de solutions pouvant potentiellement intéresser le décideur. Toutefois, il est possible de construire des instances de problèmes tels que $\mathrm{ND}_P(\mathcal{U}) = \mathcal{U}$, comme le montre l'exemple 7 (inspiré de Hansen (1980)). Donc, si le nombre de vecteurs contenus dans $\mathcal{U}$ est grand (comme par exemple dans le cas d'un produit cartésien ou d'un espace combinatoire), alors $\mathrm{ND}_P(\mathcal{U})$ peut potentiellement être aussi grand.

Exemple 7 *Considérons un espace d'alternatives construit comme un produit cartésien d'attributs : $\mathcal{X} = \prod_{i=1}^{n} X_i$ avec $\forall i \in \{1,\ldots,n\}$ $X_i = \{0,1\}$, et deux fonctions d'utilité u^1 et u^2 telles que : $u^1(x_1,\ldots,x_n) = \sum_{i=1}^{n} 2^i x_i$ et $u^2(x_1,\ldots,x_n) = \sum_{i=1}^{n} 2^i(1-x_i)$. On a donc $\forall x \in \mathcal{X}$ $u^1(x) + u^2(x) = \sum_{i=1}^{n} 2^i[x_i + (1-x_i)] = \sum_{i=1}^{n} 2^i = 2^{n+1} - 2$. Soit u l'utilité multicritère engendrée par u^1 et u^2, et $\mathcal{U} \subset \mathbb{R}^2$ l'espace des critères. L'ensemble des vecteurs $(v^1, v^2) \in \mathcal{U}$ seront donc alignés selon l'équation de droite $v^1 + v^2 = 2^{n+1} - 2$. Donc ils seront tous incomparables au sens de Pareto (voir la figure 1.3 avec $n = 3$). On en déduit donc que $ND_P(\mathcal{U}) = \mathcal{U}$. De plus, tous les vecteurs étant distincts, on aura donc $|ND_P(\mathcal{U})| = |\mathcal{X}| = 2^n$, la frontière de Pareto contiendra donc 2^n éléments.*

La frontière de Pareto contient donc toutes les solutions potentiellement intéressantes pour un décideur, mais avant même de chercher un algorithme pour résoudre ce problème,

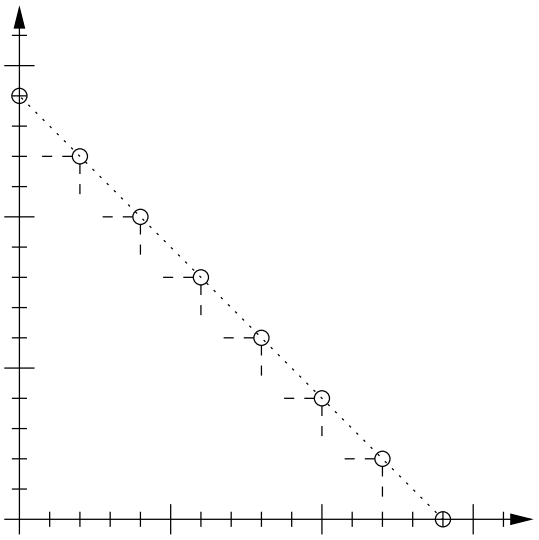


FIGURE 1.3 – Exemple de $\mathrm{ND}_P(\mathcal{U}) = \mathcal{U}$

nous avons pu remarquer l'éventualité d'une saturation de la mémoire dans la simple représentation de la solution du problème. Cette approche n'est pas à rejeter pour autant : il est rare que $|\mathrm{ArgND}_P(\mathcal{X}, u)| = |\mathcal{X}|$ en pratique, mais cette remarque fait apparaître un manque d'expressivité de cette frontière et la nécessité de définir d'autre types de problèmes tenant compte de nouvelles propriétés.

Nous allons, dans un premier temps, songer à « renforcer » la dominance de Pareto, c'est-à-dire déterminer une nouvelle relation de dominance telle que tout vecteur dominé par un autre au sens de Pareto le soit aussi au sens de la nouvelle relation (voir la propriété 6). On peut donc songer à spécialiser une relation de dominance de façon à privilégier certains types de vecteurs en particulier. Nous avons évoqué dans ce chapitre de façon intuitive la notion de compromis, sans jamais l'avoir véritablement défini. Une manière de voir les vecteurs réalisant des compromis entre plusieurs critères correspond à maintenir un équilibre entre les différents critères. La théorie du choix social a formalisé cette recherche d'équilibre sous la forme d'un axiome connu sous le nom de *transfert de Pigou-Dalton* (voir la propriété 7, et plus généralement le livre de Sen et Foster (1997)).

Propriété 6 (Compatibilité avec la dominance de Pareto (P-monotonie)) *Soit $v \in \mathbb{R}^m$, $v' \in \mathbb{R}^m$, et $\succsim_?$ une relation sur $\mathbb{R}^m$. On dit que $\succsim_?$ est compatible avec la dominance de Pareto, ou encore que $\succsim_?$ est P-monotone, si et seulement si, on a :*

$$v \succsim_P v' \implies v \succsim_? v'.$$

Propriété 7 (Principe de transfert) *Soit $v \in \mathbb{R}^m$, $\succsim_?$ une relation de dominance sur $\mathbb{R}^m$, et $(i,j) \in \{1,\ldots,m\}^2$ tels que $v^i > v^j$. On note e^z avec $z \in \{1,\ldots,m\}$ le vecteur composé de 0 sur toutes ses composantes, sauf la z-ième qui prendra une valeur de 1. On dit que $\succsim_?$ respecte le principe de transfert si, et seulement si, pour tout ϵ vérifiant $0 \leqslant \epsilon \leqslant \frac{v^i - v^j}{2}$, on a $v - \epsilon e^i + \epsilon e^j \succsim_? v$.*

Le principe de transfert pour caractériser les relations privilégiant les vecteurs de compromis est assez intuitif à saisir : si on considère deux critères i et j tels que l'un

est meilleur que l'autre ($v^i > v^j$), alors tout vecteur équilibrant ces deux critères sans modifier les valeurs sur les autres critères ($v - \epsilon e^i + \epsilon e^j$, le fait que $0 \leqslant \epsilon \leqslant \frac{v^i - v^j}{2}$ garantit l'équilibrage de ces deux critères) sera au moins aussi bon que le vecteur initial.

Dans la littérature, Kostreva et Ogryczak (1999) ont utilisé une relation pour un problème multicritère particulièrement intéressante : la dominance de Lorenz généralisée. Cette relation est compatible avec la dominance de Pareto et respecte le principe de transfert (voir les travaux de Hardy et al. (1934)). En pratique, elle correspond à la dominance de Pareto appliquée à une transformation des vecteurs.

Définition 18 (Transformée de Lorenz) *Soit $v \in \mathbb{R}^m$ un vecteur. On appelle transformée de Lorenz du vecteur v, et on note $L(v)$, le vecteur des sommes cumulées du tri en ordre croissant de v. Ainsi, si $v^{(j)}$ correspond à la j-ième plus petite composante de v, on aura donc :*

$$L(v) = \left(v^{(1)}, v^{(1)} + v^{(2)}, \ldots, \sum_{j=1}^{m} v^{(j)}\right)$$

Définition 19 (Dominance (généralisée) de Lorenz) *Soit $v \in \mathbb{R}^m$ et $v' \in \mathbb{R}^m$ deux vecteurs. On dit que v domine v' au sens Lorenz, et on note $v \succsim_L v'$, si, et seulement si, $L(v) \succsim_P L(v')$.*

Propriété 8 *La relation de dominance généralisée de Lorenz est compatible avec la dominance de Pareto et respecte le principe de transfert.*

La dominance de Lorenz n'est pas, au même titre que la dominance de Pareto, une relation complète sur $\mathbb{R}^m$. Il existe donc des vecteurs qui sont incomparables entre eux. La formulation d'un problème de recherche de solutions optimales au sens de Lorenz est donc proche de la frontière de Pareto.

Définition 20 (Frontière de Lorenz) *Soit $\mathcal{X}$ un espace d'alternatives, u une fonction d'utilité multicritère, et $\mathcal{U}$ l'espace des critères engendré par u. On appelle frontière de Lorenz (ou ensemble de non dominés au sens de Lorenz), et on note $ND_L(\mathcal{U})$ l'ensemble de vecteurs défini par $ND_L(\mathcal{U}) = \{v \in \mathcal{U} / \nexists v' \in \mathcal{U} \ v' \succ_L v\}$.*
On introduit de la même façon l'ensemble des alternatives engendrant la frontière de Lorenz par : $ArgND_L(\mathcal{X}, u) = \{x \in \mathcal{X} / \nexists x' \in \mathcal{X} \ u(x') \succ_L u(x)\}$.

Exemple 8 *Reprenons l'exemple de nos candidats à une thèse. Sur le tableau ci-dessous, nous indiquons pour chaque couple de notes des candidats, leur tranformée de Lorenz.*

Notes :	$(5,5)$	$(6,10)$	$(9,18)$	$(10,15)$	$(12,4)$	$(13,13)$	$(15,11)$	$(18,9)$
Lorenz :	$(5,10)$	$(6,16)$	$(9,27)$	$(10,25)$	$(4,16)$	$(13,26)$	$(11,26)$	$(9,27)$

Nous sommes dans le cas particulier d'un espace à deux critères. Appliquer la dominance de Pareto aux transformées de Lorenz revient donc à comparer les vecteurs initiaux selon leur moyenne et leur note minimale. La frontière de Pareto des transformées de Lorenz sera donc : $\{(9,27),(13,26)\}$, et on aura donc $ND_L(\mathcal{U}) = \{(9,18),(13,13),(18,9)\}$, c'est-à-dire que les deux solutions spécialistes auront été conservées de par leur haute moyenne, mais on ne conservera qu'une solution de compromis, qui est la plus équilibrée au niveau des résultats obtenus.

La frontière de Lorenz est donc l'ensemble des solutions non-dominées au sens de cette nouvelle relation. Une conséquence directe de la compatibilité de la dominance de Lorenz avec la dominance de Pareto est donnée dans la propriété 9. Toutefois, il est possible de créer de nouveau une instance pathologique où $\mathrm{ND}_L(\mathcal{U}) = \mathrm{ND}_P(\mathcal{U}) = \mathcal{U}$ (voir l'exemple 9, inspiré de Perny et al. (2006)).

Propriété 9 *$ND_L(\mathcal{U}) \subseteq ND_P(\mathcal{U}) \subseteq \mathcal{U}$*

Exemple 9 *Considérons un décideur devant prendre une décision dans un espace d'alternatives à n attributs : $\mathcal{X} = \prod_{i=1}^{n} X_i$ avec $\forall i \in \{1, \ldots, n\}$ $X_i = \{0, 1\}$. Deux critères interviennent dans sa décision, leurs fonctions d'utilité associées, u^1 et u^2, étant définies par :*

$$\begin{cases} u^1(x_1, \ldots, x_n) = & \sum_{i=1}^{n} 2^{i-1} x_i \\ u^2(x_1, \ldots, x_n) = & 2^{n+1} + 1 + \sum_{i=1}^{n} 2^i (1 - x_i) \end{cases}$$

Soit u la fonction d'utilité multicritère associée à u^1 et u^2. Etant donné que $\forall x \in \mathcal{X}$ $u^1(x) \leqslant u^2(x)$, on aura donc :

$$\begin{aligned} \forall x \in \mathcal{X}\ L(u(x_1, \ldots, x_n)) &= \left(\sum_{i=1}^{n} 2^{i-1} x_i, 2^{n+1} + 1 + \sum_{i=1}^{n} 2^i (1 - x_i) + \sum_{i=1}^{n} 2^{i-1} x_i \right) \\ &= \left(\sum_{i=1}^{n} 2^{i-1} x_i, 2^{n+1} + 1 + 2 \sum_{i=1}^{n} 2^{i-1} (1 - x_i) + \sum_{i=1}^{n} 2^{i-1} x_i \right) \\ &= \left(\sum_{i=1}^{n} 2^{i-1} x_i, 2^{n+1} + 1 + 2 \sum_{i=1}^{n} 2^{i-1} - \sum_{i=1}^{n} 2^{i-1} x_i \right) \\ &= \left(\sum_{i=1}^{n} 2^{i-1} x_i, 2^{n+1} + 1 + 2^{n+1} - 2 - \sum_{i=1}^{n} 2^{i-1} x_i \right) \\ &= \left(\sum_{i=1}^{n} 2^{i-1} x_i, 2^{n+2} - 1 - \sum_{i=1}^{n} 2^{i-1} x_i \right) \\ &= \left(y, 2^{n+2} - 1 - y \right) \text{ avec } y = \sum_{i=1}^{n} 2^{i-1} x_i \end{aligned}$$

Sous cette forme, on remarque que $\forall (x, x') \in \mathcal{X}^2\ L(u(x))^1 > L(u(x'))^1 \implies L(u(x))^2 < L(u(x'))^2$, nous pouvons donc en déduire que l'intégralité des transformées de Lorenz du problème sont incomparables au sens de Lorenz, et donc $ND_L(\mathcal{U}) = \mathcal{U}$ et $ArgND_L(\mathcal{X}, u) = \mathcal{X}$.

1.4 Agrégateurs et solution agrégée optimale

Dans la section précédente, l'incomplétude des relations de dominance imposait de formuler un problème de décision sous la forme de la détermination d'un ensemble de vecteurs non dominés. Nous allons maintenant nous intéresser à des relations de dominance complètes : deux vecteurs ne pourront pas être incomparables, et, pour les mêmes raisons que celles permettant de passer d'une relation de préférence à une fonction d'utilité, nous représenterons ces relations par des fonctions qui, à un vecteur de $\mathbb{R}^m$, associent une valeur dans $\mathbb{R}$.

Définition 21 (Agrégateur) *Soit $\mathcal{X}$ un espace d'alternatives, $u : \mathcal{X} \to \mathbb{R}^m$ une fonction d'utilité multicritère et $\mathcal{U}$ l'espace des critères engendré par u. On appelle agrégateur de $\mathcal{U}$ toute fonction $A : \mathcal{U} \to \mathbb{R}$.*
On appelle solution agrégée de l'alternative $x \in \mathcal{X}$ la valeur $A(u(x))$.

Il en résulte naturellement qu'étant donné un agrégateur, on souhaite déterminer l'alternative engendrant la solution agrégée optimale, c'est-à-dire $\arg\max_{x \in \mathcal{X}} A(u(x))$. Dans cette thèse, nous nous intéresserons principalement à trois agrégateurs : OWA, la

norme de Tchebycheff et l'intégrale de Choquet, mais la plupart des algorithmes que nous présenterons peuvent se généraliser à de nombreux autres agrégateurs. Pour plus de détails sur les autres agrégateurs et leurs propriétés, nous vous renvoyons à la lecture de Grabisch et Perny (2003) et Fodor et Roubens (1994).

1.4.1 La somme ordonnée pondérée (OWA)

Nous avons vu précédemment que la moyenne était un agrégateur pouvant favoriser l'apparition de solutions spécialistes. Cet agrégateur peut toutefois être modifié pour avoir un comportement beaucoup plus expressif (Yager, 1988). L'idée fondamentale de l'agrégateur OWA est de ne pas assigner une pondération à un critère particulier, mais au positionnement des critères triés par ordre croissant. En raisonnant ainsi, on obtient l'agrégateur OWA (Ordered Weighted Average) :

Définition 22 (OWA) *Etant donné un espace des critères $\mathcal{U} \subseteq \mathbb{R}^m$, et un vecteur de pondérations $\mu = (\mu_1, \ldots, \mu_m)$, vérifiant $\forall j \in \{1, \ldots, m\}$ $\mu_j \in [0, 1]$ et $\sum_{j=1}^{m} \mu_j = 1$, la fonction $OWA_\mu : \mathcal{U} \to \mathbb{R}$ est un agrégateur OWA selon μ si, et seulement si :*

$$OWA_\mu(v^1, \ldots, v^m) = \sum_{j=1}^{m} \mu_j v^{(j)}$$

$v^{(j)}$ étant la j-ième plus petite valeur du vecteur $(v^1, \ldots, v^m)$.

Avec cette définition, OWA généralise certains agrégateurs connus, comme par exemple le maximin (qui consiste en la maximisation de la valeur minimale des critères, l'agrégateur est donc la fonction qui à un vecteur d'utilité cherchera sa composante minimale), qui revient à affecter la valeur 1 à μ_1 et 0 aux autres coefficients. La moyenne est aussi un OWA particulier : il suffit de poser $\forall j \in \{1, \ldots, m\}$ $\mu_j = \frac{1}{m}$.

Exemple 10 *Revenons à nos candidats à la thèse, et comparons les selon un agrégateur OWA. Nous allons définir deux pondérations particulières : $\mu^1 = (\frac{3}{4}, \frac{1}{4})$ et $\mu^2 = (\frac{1}{4}, \frac{3}{4})$. Le calcul des différentes valeurs agrégées est représenté ci-dessous :*

Notes :	$(5,5)$	$(6,10)$	$(9,18)$	$(10,15)$	$(12,4)$	$(13,13)$	$(15,11)$	$(18,9)$
OWA_{μ^1} :	5	7	11,25	11,25	6	13	12	11,25
OWA_{μ^2} :	5	9	15,75	13,75	10	13	14	15,75

Comme nous pouvions nous y attendre, la pondération μ^1 privilégie les résultats dont la plus basse note n'est pas critique, tandis que μ^2 s'intéresse surtout aux notes maximales. Il est toutefois intéressant de constater autre chose : la pondération μ^1 a pour solution optimale agrégée $(13,13)$, qui est parfaitement équilibrée, et d'une façon générale μ^1 a tendance à préférer les solutions équilibrées.

Dans le cadre de la dominance de Lorenz, nous avions présenté la représentation de la notion d'équité par l'intermédiaire des transferts de Pigou-Dalton. Ogryczak (2000) a montré que l'agrégateur OWA respectait ce principe de tranfert à condition d'utiliser un vecteur de pondération décroissant (voir la propriété 10).

Propriété 10 *Soit $\mu = (\mu_1, \ldots, \mu_m)$ un vecteur de pondération vérifiant $\forall j \in \{1, \ldots, m\}\ \mu_j \in [0, 1]$, $\sum_{j=1}^{m} \mu_j = 1$ et $\mu_1 \geqslant \ldots \geqslant \mu_m$. L'agrégateur OWA_μ respecte le principe de transfert de Pigou-Dalton.*

Dans le cadre de cette thèse, nous nous intéresserons essentiellement au cas où la pondération est décroissante. Il est à noter que les travaux récents de Golden et Perny (2010) ont permis d'établir un lien entre l'agrégateur OWA et une application successive de transformées de Lorenz. Il résulte de ces travaux un jeu de poids particulièrement adapté à la recherche de solutions équilibrées en posant $\mu_j = sin(\frac{(m+1-j)\pi}{2m+1})$.

1.4.2 La norme de Tchebycheff

Nous avons pour l'instant raisonné sur l'espace des critères en supposant qu'il y ait un sens à effectuer des sommes entre différentes valeurs d'utilité des critères. En pratique, nous avons vu dans la section 1.1.3 qu'une fonction d'utilité exploite ses valeurs pour comparer des alternatives, mais n'attache en aucun cas une importance sur le nombre en soi (si $u(x) = 1$ et $u(y) = 10$, y n'est pas 10 fois préférable à x). Ainsi, nous pouvons parfaitement avoir une fonction d'utilité pour le premier critère dont toutes les valeurs seront comprises dans l'intervalle $[0, 1]$, et une autre fonction d'utilité pour un second critère dont les valeurs seront comprises dans l'intervalle $[-10^9, 10^{51}]$. La norme de Tchebycheff, utilisée dans le cadre de la programmation multi-objectifs dans Steuer et Choo (1983), est une tentative d'agrégateur pour résoudre ce problème. Son principe consiste en la détermination de deux vecteurs (le point idéal et le point Nadir), qui vont servir à redimensionner l'espace de manière à ce qu'un critère ne soit pas prépondérant par rapport aux autres uniquement parce que les valeurs de son utilité sont significativement plus grandes que les valeurs des utilités des autres critères.

Définition 23 (Point idéal) *Soit $\mathcal{X}$ un espace d'alternatives, $u : \mathcal{X} \to \mathbb{R}^m$ une fonction d'utilité multicritère formée à partir des fonctions d'utilité $(u^1, \ldots, u^m)$. Le point idéal selon u est le vecteur $I = (I^1, \ldots, I^m) \in \mathbb{R}^m$ vérifiant $\forall j \in \{1, \ldots, m\}\ I^j = \max_{x \in \mathcal{X}} u^j(x)$.*

Le point idéal est donc un vecteur n'existant pas nécessairement dans $\mathcal{U}$, composé des meilleures valeurs qu'il est possible d'obtenir à partir des différentes fonctions d'utilité. Il domine donc au sens de Pareto tous les vecteurs de $\mathcal{U}$ et peut être vu comme fournissant une « borne supérieure » des solutions de $\mathcal{U}$.

Le point Nadir va fournir la borne opposée, mais contrairement à ce que l'on pourrait intuitivement attendre, ce vecteur n'est pas composé des pires valeurs qu'il est possible d'obtenir dans chaque critère, mais des pires valeurs des solutions spécialistes :

Définition 24 (Point Nadir) *Soit $\mathcal{X}$ un espace d'alternatives, $u : \mathcal{X} \to \mathbb{R}^m$ une fonction d'utilité multicritère formée à partir des fonctions d'utilité $(u^1, \ldots, u^m)$. On note $\forall j \in \{1, \ldots, m\}\ x_j^* = \arg\max_{x \in \mathcal{X}} u^j(x)$ et $(v_j^1, \ldots, v_j^m) = u(x_j^*)$. Le point Nadir selon u est le vecteur $N = (N^1, \ldots, N^m) \in \mathbb{R}^m$ vérifiant $\forall j \in \{1, \ldots, m\}\ N^j = \min_{i \in \{1, \ldots, m\}} v_i^j$.*

Remarque 2 *Nous allons maintenant introduire la norme de Tchebycheff. Sa formulation peut sembler étonnante étant donné que le nom de l'agrégateur contient le terme « norme » alors que les valeurs renvoyées sont des nombres négatifs. Cette impression est uniquement dûe à une formulation cohérente de la définition de cet agrégateur à un*

problème de maximisation de la valeur agrégée. En réalité, la définition de la norme de Tchebycheff est plus générale que celle présentée dans cette section, mais nous avons choisi de la présenter de cette façon pour deux raisons :

- *En la définissant par rapport à un point idéal et un point Nadir, cette norme remplit toutes les conditions nécessaires pour être compatible avec la dominance de Pareto, comme nous le verrons dans la propriété 11.*
- *Cette définition est celle utilisée en pratique par Gonzales et al. (2008). L'utilisation d'un point idéal permet d'interpréter la formule de la norme comme traitant directement sur les regrets de chaque critère.*

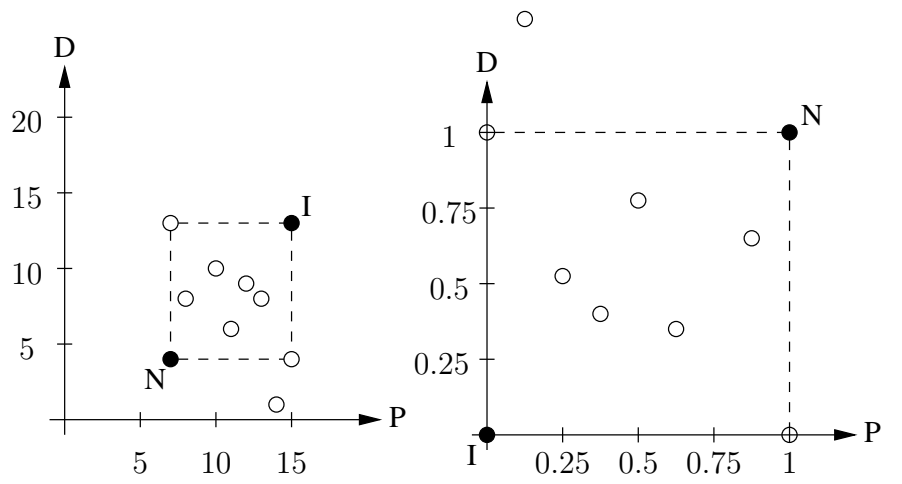
Définition 25 (Norme de Tchebycheff) *Soit $\mathcal{X}$ un espace d'alternatives, $u : \mathcal{X} \to \mathbb{R}^m$ une fonction d'utilité multicritère engendrant l'espace des critère $\mathcal{U}$. On considère I le point idéal selon u et N le point Nadir selon u. Une norme de Tchebycheff est un agrégateur défini par :*

$$Tcheby(v^1, \ldots, v^m) = - \max_{j \in \{1,\ldots,m\}} \frac{I^j - v^j}{I^j - N^j}.$$

Cette formule peut s'interpréter de la façon suivante : le point idéal définit comme son nom l'indique un idéal que l'on aimerait atteindre. Le point Nadir va permettre d'identifier une zone de l'espace contenant les solutions intéressantes (voir exemple 11). L'espace des critères subit une transformation : à chaque composante v^j d'un vecteur on associe $\frac{I^j - v^j}{I^j - N^j}$, $I^j - v^j$ représente une distance à l'idéal, et la division par $I^j - N^j$ est une mise à l'échelle de cette distance de façon à harmoniser les différents critères. Pour chaque vecteur, on s'intéresse au critère maximisant cette valeur, donc à la valeur $D(v)$ de la plus grande distance harmonisée. La solution agrégée optimale au sens de la norme de Tchebycheff sera celle qui maximisera $-D(v)$, donc celle qui minimisera $D(v)$, ce qui revient donc à chercher la solution qui sera la moins éloignée de l'idéal (au sens de la norme infinie et après harmonisation des différents critères).

Exemple 11 *Une nouvelle année s'annonce, et de nouveaux candidats à la thèse sont arrivés. Les unités d'enseignement n'ont pas changé mais les correcteurs ne sont pas les mêmes. Le nouveau correcteur en programmation note maintenant sur 15 les étudiants, et a tendance donner facilement des bonnes notes. De son côté, le nouveau correcteur en décision note sur 20 et a la main lourde sur ses notations. Les notes des nouveaux candidats sont* $(7, 13)$, $(10, 10)$, $(8, 7)$, $(11, 6)$, $(13, 8)$, $(15, 4)$, $(14, 1)$ *et* $(12, 9)$. *Le point idéal sera donc issu des solutions spécialistes* $(15, 4)$ *et* $(7, 13)$, *et on aura donc* $I = (15, 13)$. *Le point Nadir sera les pires valeurs des spécialistes et donc* $N = (7, 4)$. *La figure 1.4(a) représente les candidats et ces deux points.*

Le calcul d'une norme de Tchebycheff peut se décomposer en deux étapes : tout d'abord l'espace des critères subit une transformation en fonction de I et N. Dans notre exemple, à chaque vecteur (v^1, v^2), on associera le vecteur $\left(\frac{15-v^1}{8}, \frac{13-v^2}{9}\right)$ (cette transformation est représentée sur la figure 1.4(b)). Puis, les candidats seront classés dans l'espace transformé en fonction de leur distance maximale sur chaque critère au point idéal. Cette



(a) Les nouveaux candidats, avec le point idéal et le point Nadir

(b) Changement d'échelle dans la norme de Tchebycheff

FIGURE 1.4 – Exemple d'application de la norme de Tchebycheff

succession de calculs est représentée dans le tableau suivant :

Notes	*Transformation*	*Distance max*
$(7, 13)$	$(1, 0)$	1
$(10, 10)$	$(5/8, 1/3)$	5/8
$(8, 7)$	$(7/8, 2/3)$	7/8
$(11, 6)$	$(1/2, 7/9)$	7/9
$(13, 8)$	$(1/4, 5/9)$	5/9
$(15, 4)$	$(0, 1)$	1
$(14, 1)$	$(1/8, 4/3)$	4/3
$(12, 9)$	$(3/8, 4/9)$	4/9

Il ne reste plus qu'à maximiser l'opposé des distances maximales, c'est-à-dire à déterminer le minimum des distances maximales. La solution agrégée optimale au sens de la norme de Tchebycheff est donc celle dont la distance maximale est $\frac{4}{9}$*. Il s'agit du candidat ayant obtenu pour résultats scolaires* $(12, 9)$*.*

Wierzbicki (1986) a étudié les propriétés de la norme de Tchebycheff dans sa formulation générale. En utilisant la définition avec un point idéal et un point Nadir, cette norme vérifie la compatibilité avec la dominance faible de Pareto.

Propriété 11 *La norme de Tchebycheff est compatible avec la dominance faible de Pareto :*

$$\forall v \in \mathcal{U}\ \forall v' \in \mathcal{U}\ v \succsim_P v' \implies Tcheby(v) \geqslant Tcheby(v').$$

1.4.3 L'intégrale de Choquet

L'intégrale de Choquet est un agrégateur appartenant à la catégorie des intégrales floues (Sugeno, 1974) et permettant de quantifier l'importance des critères d'un décideur.

Les quantifications sont souvent vues comme des pondérations, mais les pondérations ne permettent pas d'exprimer des importances conjointes entre les critères (par exemple, un décideur peut avoir peu d'intérêt pour les critères 1 et 2 pris individuellement, mais avoir un fort intérêt lorsque ces deux critères ont coinjointement de grandes valeurs). L'intégrale de Choquet remplace les pondérations par une notion plus générale : les capacités (Choquet, 1953).

Définition 26 (Capacité) *Soit E un ensemble et 2^E l'ensemble des parties de E. Une capacité sur E est une fonction $\mu : 2^E \to [0,1]$ vérifiant :*
- $\mu(\varnothing) = 0$
- $\mu(E) = 1$
- $\forall F \in 2^E \ \forall F' \in 2^E \ F \subseteq F' \implies \mu(F) \leqslant \mu(F')$.

Dans le cadre qui nous intéresse, on aura $E = \{1, \ldots, m\}$. Donc une capacité prendra en argument un sous-ensemble de l'espace des critères, et renverra un nombre correspondant à l'intérêt que l'on peut éprouver du point de vue de l'ensemble conjoint des critères passés en argument, ce qui permet de quantifier la notion de synergie entre critères (Grabisch, 1996). Une intégrale de Choquet se calcule de la façon suivante (Murofushi et Sugeno, 1991) :

Définition 27 (Intégrale de Choquet) *Soit $\mathcal{X}$ un espace d'alternatives, $u : \mathcal{X} \to \mathbb{R}^m$ une fonction d'utilité multicritère engendrant l'espace des critères $\mathcal{U}$, et μ une capacité de $\{1, \ldots, m\}$. L'intégrale de Choquet selon μ est une fonction $Choquet_\mu : \mathcal{U} \to \mathbb{R}$ définie par :*

$$Choquet_\mu(v^1, \ldots, v^m) = \sum_{j=1}^{m} (T(j) - T(j+1)) v^{(j)} = v^{(1)} + \sum_{j=2}^{m} (v^{(j)} - v^{(j-1)}) T(j)$$

Avec $(.)$ un opérateur de classement (on aura donc $v^{(1)} \leqslant \ldots \leqslant v^{(m)}$) et $T(j)$ l'ensemble des $m - j + 1$ indices des meilleurs critères pour v : $T(j) = \mu(\{(j), \ldots, (m)\})$. On aura donc $T(m+1) = \mu(\varnothing) = 0$ et $T(1) = \mu(\{1, \ldots, m\}) = 1$.

L'intégrale de Choquet possède donc deux formulations équivalentes. La deuxième écriture de la définition est la plus simple à interpréter : supposons que l'on ait $v = (5, 1, 10)$, l'intégrale de Choquet s'écrira donc $1\mu(\{1,2,3\}) + (5-1)\mu(\{1,3\}) + (10-5)\mu(\{3\})$. On est sûr d'avoir une valeur d'au moins 1 sur tous les critères, puis nous sommes sûrs, si l'on excepte le critère 2 que cette valeur minimale soit augmentée de $5 - 1$. On pondère donc cette augmentation par l'importance des critères 1 et 3 pris conjointement, et finalement, nous sommes sûrs de gagner au moins $10 - 5$ sur le critère 3, et on pondère de nouveau ce gain pour l'intérêt porté au troisième critère. D'une façon générale, cette formule peut donc être vue comme une somme de gains en utilité (représentée par la différence d'utilités classées) pondérée par l'importance conjointe des critères restant (représentée par la capacité).

La forme de la capacité possède des propriétés intéressantes en représentation des préférences (voir la définition 28). Ainsi, une capacité convexe donnera une plus forte valeur aux combinaisons de critères par rapport à une somme de sous-combinaisons et permet de représenter des décideurs préférant des solutions de compromis entre les critères, tandis qu'une capacité concave représentera plus d'attachement aux critères pris

individuellement qu'à leur regroupement, et correspondra plutôt à des décideurs préférant des solutions spécialistes.

Définition 28 (Capacités particulières) *Une capacité* $\mu : 2^E \rightarrow [0,1]$ *est dite convexe (ou supermodulaire) si, et seulement si,* $\forall A \in 2^E \ \forall B \in 2^E \ \mu(A \cup B) + \mu(A \cap B) \geqslant \mu(A) + \mu(B)$.

Une capacité $\mu : 2^E \rightarrow [0,1]$ *est dite concave (ou sousmodulaire) si, et seulement si,* $\forall A \in 2^E \ \forall B \in 2^E \ \mu(A \cup B) + \mu(A \cap B) \leqslant \mu(A) + \mu(B)$.

Une capacité $\mu : 2^E \rightarrow [0,1]$ *est dite additive si, et seulement si, elle est à la fois convexe et concave, soit :* $\forall A \in 2^E \ \forall B \in 2^E \ \mu(A \cup B) + \mu(A \cap B) = \mu(A) + \mu(B)$.

Une capacité $\mu : 2^E \rightarrow [0,1]$ *est dite symétrique si, et seulement si,* $\forall A \in 2^E \ \forall B \in 2^E \ |A| = |B| \Leftrightarrow \mu(A) = \mu(B)$.

Soit $\mu : 2^E \rightarrow [0,1]$ *une capacité, la capacité duale de* μ *est une capacité* $\bar{\mu} : 2^E \rightarrow [0,1]$ *vérifiant* $\forall A \in 2^E \ \bar{\mu}(A) = 1 - \mu(E \backslash A)$.

Propriété 12 *Une capacité* $\mu : 2^E \rightarrow [0,1]$ *est additive si, et seulement si,* $\forall A \in 2^E \ \mu(A) = \sum_{e \in A} \mu(\{e\})$.

Une capacité additive permet une représentation moins coûteuse en mémoire des capacités, car la propriété 12 permet de définir la valeur de la capacité d'un ensemble en fonction des singletons contenus dans l'ensemble. Toutefois, calculer une intégrale de Choquet en utilisant une capacité additive revient en réalité à calculer une somme pondérée, étant donné que $\sum_{j=1}^{m}(T(j)-T(j+1))v^{(j)} = \sum_{j=1}^{m}(\mu(\{(j),\ldots,(m)\}) - \mu(\{(j+1),\ldots,(m)\}))v^{(j)} = \sum_{j=1}^{m}\mu(\{(j)\})v^{(j)} = \sum_{j=1}^{m}\mu(\{j\})v^{j}$.

Remarque 3 *Dans le domaine de la représentation de l'incertain, une capacité additive correspond à une probabilité. E est l'univers,* 2^E *les évènements et les singletons sont les évènements élémentaires.*

Une propriété intéressante des intégrales de Choquet est qu'elles généralisent les agrégateurs OWA : Prenons une capacité symétrique, alors la valeur de la capacité d'un ensemble A ne dépendra pas du contenu de A mais du nombre de critères contenus dans A. Posons $\mu_j = \mu(A) - \mu(B)$ avec $|A| = j$ et $|B| = j-1$. Par définition, on a $T(j) = \mu(\{(j),\ldots,(m)\})$, donc $T(j)$ est une valeur de capacité d'un ensemble à $m-j+1$ éléments, et donc $T(j) - T(j+1) = \mu_{m-j+1}$. Il est donc possible de réécrire l'expression de l'intégrale de Choquet : $\sum_{j=1}^{m}(T(j) - T(j+1))v^{(j)} = \sum_{j=1}^{m}\mu_{m-j+1}v^{(j)}$, ce qui est exactement l'expression d'un OWA.

Chapitre 2

Décision avec le modèle GAI

Résumé. *Ce chapitre introduit le modèle autour duquel gravite cette thèse : les réseaux GAI. Nous étudierons dans un premier temps (section 2.1) l'intérêt d'une telle représentation, aussi bien du point de vue de l'expressivité que de la compacité mémorielle. Puis, nous présenterons l'algorithmique associée aux réseaux GAI dans la littérature : La section 2.2 sera centrée sur l'algorithmique liée à la représentation des préférences par des réseaux GAI (triangulation, intégration de contraintes et élicitation des préférences). La section 2.3 présentera deux problèmes monocritères et leur algorithme de résolution associé : la détermination du choix optimal (alternative préférée d'un décideur) et du rangement optimal (k meilleures alternatives selon les préférences du décideur). La section 2.4 abordera l'utilisation des réseaux GAI en décision multicritère et présentera une façon de résoudre la détermination de la solution agrégée optimale sous l'hypothèse de l'existence d'une borne supérieure additivement décomposable pour l'agrégateur choisi.*

Dans le chapitre précédent, nous avons présenté des problématiques liées à la présence de multiples attributs ou de multiples critères. Nous avons pu nous rendre compte de la nécessité de choisir un modèle qui puisse à la fois réduire le nombre de questions à poser au décideur durant la phase d'élicitation, qui soit compact en mémoire, qui soit également capable de représenter une grande diversité de comportements et pouvant être exploité efficacement dans un but d'aide à la décision. Ce chapitre présente le modèle qui a été choisi pour les travaux que nous avons menés et l'algorithmique qui lui est associé.

La section 2.1 présentera la GAI-décomposabilité puis effectuera des rappels de théorie des graphes qui seront par la suite utilisés pour définir un modèle exploitant cette décomposition : les réseaux GAI. La section 2.2 abordera l'utilisation des réseaux GAI pour représenter des préférences. Bien que cette thèse ne porte pas sur l'élicitation des préférences, on ne peut songer à exploiter le modèle sans une élicitation préalable. Nous présenterons donc l'idée générale des algorithmes d'élicitation en 2.2.1, ainsi que des références vers les articles détaillant ces algorithmes, pour avoir une vision complète du processus décisionel. Des travaux ayant déjà été menés sur l'exploitation des préférences dans le modèle GAI seront exposés en 2.3 afin de pouvoir étudier leurs limites, et proposer par la suite une nouvelle approche pour résoudre les problèmes algorithmiques qui se posent. Finalement, la section 2.4 abordera l'utilisation des réseaux GAI pour déterminer (sous certaines conditions) l'alternative engendrant la solution agrégée optimale pour des problèmes multicritères.

2.1 Le modèle GAI

2.1.1 GAI-décomposabilité

Nous avons pu voir dans l'exemple 3 qu'il était possible d'exploiter une indépendance dans les préférences du décideur, mais que cette indépendance n'était pas compatible avec une décomposition en utilités additives, ce qui nous avait amené à introduire la notion de décomposition additive généralisée, ou GAI-décomposition (Fishburn, 1970). GAI signifie « Generalized Additive Independance » et repose sur un concept d'indépendance moins restrictif qu'une indépendance additive sur la totalité des attributs du problème. Avant d'introduire une définition formelle de cette façon de décomposer l'utilité globale, il est nécessaire de définir ce qu'est la projection d'un n-uplet.

Définition 29 (Projection d'ensemble et de n-uplet) *Soit $\mathcal{X} = \prod_{i=1}^{n} X_i$ et $C \subseteq \{1, \ldots, n\}$. On définit le projeté de $\mathcal{X}$ sur les attributs indicés par C (on écrira X_C par abus de notation) comme étant $X_C = \prod_{j \in C} X_j$.*
Soit $x = (x_1, \ldots, x_n)$ un n-uplet de $\mathcal{X}$. On définit le projeté de x sur les attributs indicés par C (et on notera x_C) par un élément de l'ensemble X_C comportant la même instanciation que x sur chaque attribut indicé par C.

Par abus de notation, lorsqu'il n'y a pas d'ambiguïté sur l'ensemble C, on pourra omettre les accolades dans l'écriture de l'ensemble.

Exemple 12 *Soit $\mathcal{X} = \prod_{i=1}^{9} X_i$ et $x \in \mathcal{X}$ tel que $x = (x_1, \ldots, x_9)$. Alors $x_{\{2,3,5,7\}} = (x_2, x_3, x_5, x_7) \in X_{\{2,3,5,7\}}$. Comme aucune ambiguïté n'est possible entre les attributs, on pourra écrire $x_{\{2,3,5,7\}} = x_{2357}$ et $X_{\{2,3,5,7\}} = X_{2357}$.*

Définition 30 (GAI-décomposition) *Soit $\mathcal{X} = \prod_{i=1}^{n} X_i$ un espace d'alternatives, $\succsim$ une relation de préférence sur $\mathcal{X}$ représentable par une fonction d'utilité u, et $C_1, \ldots, C_q$ des ensembles vérifiant :*

- $\forall j \in \{1, \ldots, q\}\ C_j \subseteq \{1, \ldots, n\}$
- $\bigcup_{j=1}^{q} C_j = \{1, \ldots, n\}$

On dit que u est GAI-décomposable selon $C_1, \ldots, C_q$ si, et seulement si, il existe des fonctions $u_j : \prod_{i \in C_j} X_i \to \mathbb{R}$ vérifiant $\forall x \in \mathcal{X}\ u(x) = \sum_{j=1}^{q} u_j(x_{C_j})$.

Remarque 4 *Il est à noter que les ensembles C_j ne forment pas nécessairement une partition de $\{1, \ldots, n\}$, mais un recouvrement, c'est-à-dire que leurs intersections peuvent être non vides.*

Exemple 13 *En reprenant l'exemple 3 dans le formalisme présenté, nous aurons donc $C_1 = \{1, 2\}$ et $C_2 = \{1, 3\}$ et les sous-fonctions d'utilité $u_1 : X_1 \times X_2 \to \mathbb{R}$ et $u_2 : X_1 \times X_3 \to \mathbb{R}$ vérifiant $\forall x \in \mathcal{X}\ u(x) = u(x_1, x_2, x_3) = u_1(x_{C_1}) + u_2(x_{C_2}) = u_1(x_1, x_2) + u_2(x_1, x_3)$.*

Une telle définition réalise une bon compromis entre compacité et expressivité : l'idée étant d'exploiter les indépendances entre attributs dans les préférences du décideur par des sous-fonctions d'utilité afin de réduire l'espace de stockage nécessaire pour représenter l'utilité globale sans introduire plus d'indépendance de manière à pouvoir garantir que les préférences du décideur sont bien représentées après la décomposition.

Propriété 13 (Ordre de grandeur d'une GAI-décomposition) *Soit $\mathcal{X} = \prod_{i=1}^{n} X_i$ un espace d'alternatives, $\succsim$ une relation de préférence sur $\mathcal{X}$ représentable par une fonction d'utilité u, GAI-décomposable en $u(x) = \sum_{j=1}^{q} u_j(x_{C_j})$. L'espace mémoire nécessaire pour représenter la fonction u est en $\Theta(\sum_{j=1}^{q} \prod_{i \in C_j} |X_i|)$.*

Preuve. De la même façon que pour l'ordre de grandeur du stockage des utilités additives, l'utilité gloable peut se représenter en mémorisant les sous-fonctions d'utilité. $\forall j \in \{1, \ldots, q\}$, le stockage de u_j nécessitera de représenter $\prod_{i \in C_j} |X_i|$ valeurs réelles. Donc, représenter toutes les sous-fonctions d'utilités correspondra à représenter $\sum_{j=1}^{q} \prod_{i \in C_j} |X_i|$ valeurs. En supposant une représentation à taille fixe de ces valeurs, l'ordre de grandeur de l'espace mémoriel pour représenter u sera donc en $\Theta(\sum_{j=1}^{q} \prod_{i \in C_j} |X_i|)$. ■

Remarque 5 *Nous avons choisi d'illustrer l'ordre de grandeur avec une borne asymptotique précise. Il est possible de déterminer d'autres bornes asymptotiques moins précises (en $O(.)$) mais plus concises dans les notations (notamment avec la notion de treewidth que nous étudierons plus tard).*

Remarque 6 *La GAI-décomposabilité permet de représenter les utilités additives (si $q = n$ et $\forall j \in \{1, \ldots, n\}\ C_j = \{j\}$) ainsi que l'absence complète d'indépendance (si $q = 1$ et $C_1 = \{1, \ldots, n\}$).*

Nous avons présenté la GAI-décomposabilité en tant que modèle permettant d'exploiter au maximum les indépendances du décideur pour représenter de façon compacte ses

préférences. Toutefois, cette modélisation ne permet pas de déduire facilement une procédure d'élicitation efficace ou une manière d'exploiter les préférences du décideur pour guider ce dernier dans ses choix (détermination de ou des alternative(s) préférée(s)). Ces problèmes peuvent être traités efficacement par l'utilisation d'une structure de données exploitant conjointement GAI-décomposabilité et théorie des graphes : les réseaux GAI. Nous allons brièvement présenter des bases de théorie des graphes dans la section 2.1.2 afin de pouvoir définir les réseaux GAI (dans la section 2.1.3) et de s'intéresser à l'utilisation d'une telle structure en 2.2.

2.1.2 Bases de théorie des graphes

Un graphe est constitué de deux ensembles, l'ensemble des *sommets* (ou *nœuds*), et un ensemble modélisant une relation entre ces sommets (les *arêtes* si la relation est symétrique, les *arcs* dans le cas contraire). Les graphes disposent d'une représentation schématique facilitant la compréhension de certains problèmes pour un humain. Cette thèse ne se situe pas dans le domaine de la théorie des graphes, mais l'utilise comme un outil pour résoudre certains problèmes, donc nous n'introduirons que les définitions et propriétés qui sont exploitées dans cette thèse.

Définition 31 (Graphe non orienté) *On dit que $G = (V, E)$ est un graphe non orienté si, et seulement si, $E \subseteq \mathcal{P}_2(V)$ ($\mathcal{P}_2(V)$ étant l'ensemble des parties à deux éléments de V). Les éléments de V sont appelés sommets ou nœuds, et les élements de E sont les arêtes.*

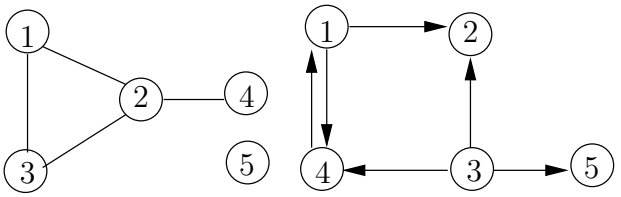
Définition 32 (Graphe orienté) *On dit que $G = (V, A)$ est un graphe orienté si, et seulement si, $A \subseteq V \times V$ et $\forall (x, y) \in A \; x \neq y$. Les éléments de V sont appelés sommets ou nœuds, et les élements de A sont les arcs.*

Remarque 7 *Pour des raisons de simplicité, les définitions introduites sont en réalité les définitions des graphes simples orientés et non orientés (pas plus d'une arête entre deux sommets, pas de boucles, ...).*

La représentation schématique des graphes correspond à dessiner la totalité des éléments de V, puis à tracer un lien entre deux élements x et y si $\{x, y\} \in E$ dans le cas non orienté, ou une flèche de x vers y si $(x, y) \in A$ dans le cas orienté.

Exemple 14 *On considère le graphe non orienté $G_1 = (\{1, 2, 3, 4, 5\}, \{\{1, 2\}, \{2, 3\}, \{3, 1\}, \{2, 4\}\})$ et le graphe orienté $G_2 = (\{1, 2, 3, 4, 5\}, \{(1, 2), (3, 2), (3, 4), (4, 1), (1, 4), (3, 5)\})$. Leur représentation schématique respective correspond aux figures 2.1(a) et 2.1(b).*

Aux graphes sont associés des définitions intuitives quand on considère une représentation schématique. On peut s'intéresser à la possibilité de parcourir le graphe de manière à aller d'un sommet à un autre en ne passant que par des arêtes (ou arcs) du graphe. Il est à noter que les définitions introduites dans le cadre des graphes non orientés sont valides pour les graphes orientés en considérant que tous les arcs sont des arêtes (on perd ainsi la notion d'orientation).



(a) Le graphe non orienté G_1 (b) Le graphe orienté G_2

FIGURE 2.1 – Exemple de graphes

Définition 33 (Chaîne) *Soit $G = (V, E)$ un graphe non orienté, et $(s_1, \ldots, s_p)$ une famille de sommets de V. On dit que $(s_1, \ldots, s_p)$ est une chaîne de longueur p si, et seulement si, $\forall i \in \{1, \ldots, p-1\}$ $\{s_i, s_{i+1}\} \in E$.*
L'ensemble des arêtes $\{s_i, s_{i+1}\}$ sera nommé ensemble des arêtes parcourues par $(s_1, \ldots, s_p)$.

Définition 34 (Chemin) *Soit $G = (V, A)$ un graphe orienté, et $(s_1, \ldots, s_p)$ une famille de sommets de V. On dit que $(s_1, \ldots, s_p)$ est un chemin de longueur p si, et seulement si, $\forall i \in \{1, \ldots, p-1\}$ $(s_i, s_{i+1}) \in A$.*
L'ensemble des arcs (s_i, s_{i+1}) sera nommé ensemble des arcs parcourus par $(s_1, \ldots, s_p)$.

Définition 35 (Cycle) *Soit $G = (V, E)$ un graphe non orienté, et $(s_1, \ldots, s_p)$ une chaîne de G, on dit que $(s_1, \ldots, s_p)$ est un cycle si, et seulement si, $s_1 = s_p$.*

Définition 36 (Circuit) *Soit $G = (V, A)$ un graphe orienté, et $(s_1, \ldots, s_p)$ un chemin de G, on dit que $(s_1, \ldots, s_p)$ est un circuit si, et seulement si, $s_1 = s_p$.*

Exemple 15 *Sur les graphes G_1 et G_2 de la figure 2.1, $(1, 2, 3)$ est une chaîne de G_1 mais n'est pas un chemin de G_2 car il n'existe pas d'arc $(2, 3)$. Par transposition de la définition de chaîne dans le cadre des graphes orientés, $(1, 2, 3)$ est une chaîne de G_2. $(1, 2, 3, 1)$ est un cycle de G_1 et $(1, 4, 1)$ est un circuit de G_2.*

Définition 37 (Corde) *Soit $G = (V, E)$ un graphe non orienté, et C un cycle de G, on dit que l'arête $\{s_i, s_j\} \in E$ est une corde de C si, et seulement si, s_i et s_j sont des sommets du cycle C et $\{s_i, s_j\}$ n'appartient pas à l'ensemble des arêtes parcourues par C.*

En considérant que l'ensemble des arêtes ou des arcs correspond à une mise en relation de paires de sommets, les définitions de chaîne et de chemin permettent d'exprimer des successions de mises en relation, et les cycles et circuits correspondront à des successions revenant à l'élément de départ. La notion de corde peut être vue comme l'existence d'un « raccourci » coupant un cycle. En reprenant la notion de chaîne, on peut mettre en évidence des familles de sommets : les composantes connexes. Ainsi, les sommets appartenant à une même composante connexe sont en relation directe ou indirecte entre eux (il existe un chemin ou une chaîne les reliant).

Définition 38 (Connexité) *Soit $G = (V, E)$ un graphe non orienté. On définit la relation de connexité comme étant une relation binaire R sur V telle que pour tout couple de sommets (x, y), on a xRy si, et seulement si, il existe une chaîne reliant x et y.*
Un graphe tel qu'il existe au moins une chaîne reliant chaque couple de sommets est dit connexe.

Propriété 14 *La relation de connexité dans un graphe est une relation d'équivalence.*

Définition 39 (Composantes connexes) *Soit $G = (V, E)$ un graphe non orienté, et R la relation de connexité. On définit les composantes connexes d'un graphe comme étant les classes d'équivalence de la relation de connexité.*

Les définitions données précédemment caractérisent des familles de sommets ou d'arêtes. Nous allons maintenant nous intéresser à des formes particulières de graphe. Ces topologies peuvent être intéressantes que ce soit dans un sous-graphe ou dans la globalité du graphe. Il convient de définir tout d'abord ce qu'est un sous-graphe.

Définition 40 (Sous-graphe partiel) *Soit $G = (V, E)$ un graphe non orienté, et $E' \subseteq E$. Le graphe G' est dit sous-graphe partiel de G selon E' si, et seulement si, $G' = (V, E')$.*

Définition 41 (Sous-graphe induit) *Soit $G = (V, E)$ un graphe non orienté, et $V' \subseteq V$. Le graphe G' est dit sous-graphe de G induit par V' si, et seulement si, $G' = (V', E')$ et $E' = \{\{s_i, s_j\} \in E / s_i \in V', s_j \in V'\}$.*

Définition 42 (Sous-graphe) *Soit $G = (V, E)$ et $G' = (V', E')$ deux graphes non orientés. On dit que G' est un sous-graphe de G si, et seulement si, G' est le sous-graphe partiel selon E' du sous-graphe de G induit par V'.*

Une première topologie intéressante est celle d'une clique, il s'agit d'un sous-graphe dont tous les sommets sont interconnectés. Identifier les cliques d'un graphe permet d'avoir une idée des sommets qui sont fortement en relation entre eux et donne une idée de la structure interne d'une relation au sein de l'ensemble des sommets.

Définition 43 (Graphe complet) *Soit $G = (V, E)$ un graphe non orienté. On dit que G est un graphe complet si, et seulement si, $\forall (s_i, s_j) \in V \times V \ \{s_i, s_j\} \in E$.*

Définition 44 (Clique) *Soit $G = (V, E)$ un graphe non orienté, et $G' = (V', E')$ un sous-graphe de G. On dit que G' est une clique si, et seulement si, G' est un sous-graphe complet de G et il n'existe pas de graphe $G'' = (V'', E'')$ tel que G'' soit un sous-graphe complet de G et $V' \subset V''$.*

Une autre topologie intéressante correspond aux forêts. Ce sont des graphes tels que, pour tout couple de sommets, soit il existe une unique chaîne les reliant, soit il n'en existe pas. Les sommets reliés par une chaîne forment une composante connexe, et chaque composante sera nommée arbre.

Définition 45 (Acyclicité) *Soit $G = (V, E)$ un graphe non orienté. On dit que G est acyclique si, et seulement si, il n'existe pas de cycle au sein de G.*

Définition 46 (Forêt) *Soit G un graphe non orienté. On dit que G est une forêt si, et seulement si, G est acyclique.*

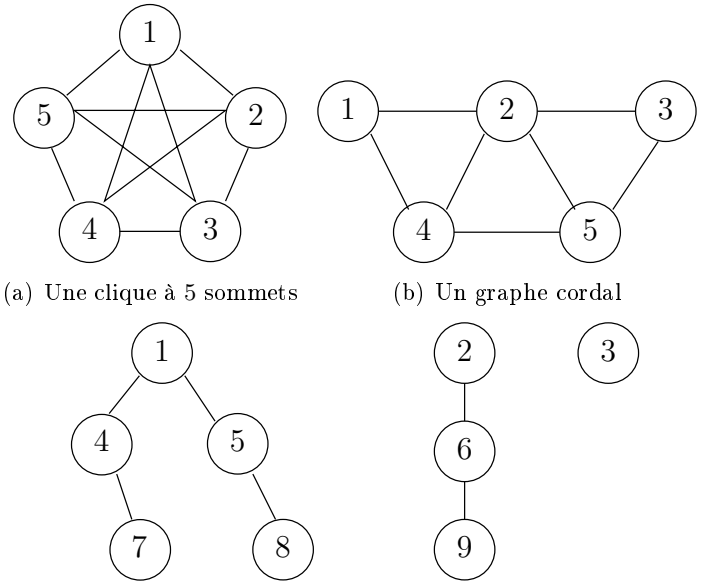
Définition 47 (Arbre) *Un arbre est une forêt connexe.*

Propriété 15 *Les composantes connexes d'une forêt sont des arbres.*

Propriété 16 *Soit $G = (V, E)$ un arbre, alors pour tout couple $(s, s') \in V \times V$ de sommets, il existe une unique chaîne reliant ces deux sommets.*

Dans le cadre de cette thèse, nous nous intéresserons à une dernière topologie importante : les graphes cordaux qui établissent un pont entre les graphes non orientés et les hypergraphes que nous allons introduire ensuite.

Définition 48 (Graphe cordal) *Un graphe non orienté G est dit cordal si, et seulement si, tout cycle contenant au moins 4 arêtes possède au moins une corde.*



(a) Une clique à 5 sommets (b) Un graphe cordal

(c) Une forêt contenant 3 arbres (maximaux)

FIGURE 2.2 – Topologies particulières de graphe

Exemple 16 *Les figures 2.2 donnent des exemples de topologies de graphe : une clique à 5 sommets sur la figure 2.2(a), et n'importe lequel de ses sous-graphes induits correspondra lui-même à un sous-graphe complet. La figure 2.2(b) présente un graphe cordal : conformément aux propriétés des graphes cordaux, tout cycle de longueur strictement supérieure à trois possède une corde. Par exemple, le cycle $(1, 2, 3, 5, 4, 1)$ a pour corde $\{5, 2\}$, et le cycle $(1, 2, 5, 4, 1)$ a pour corde $\{2, 4\}$. La figure 2.2(c) correspond à une forêt : il n'existe donc pas de cycle au sein de ce graphe. Il contient trois composantes connexes : $\{1, 4, 5, 7, 8\}$, $\{2, 6, 9\}$ et $\{3\}$ qui sont des arbres (voir la propriété 15). Chaque couple de sommets de chaque arbre est bien relié par une unique chaîne. Par exemple, les sommets 1 et 8 sont reliés par la chaîne $(1, 5, 8)$ (voir la propriété 16).*

Un graphe est un outil permettant de modéliser et de raisonner sur une relation binaire. Les graphes (simples) ont été par la suite généralisés aux relations n-aires, ce qui a donné naissance aux hypergraphes. Nous allons présenter succinctement ce qu'est un hypergraphe, puis nous montrerons quelques modélisations des hypergraphes utilisant des graphes.

Définition 49 (Hypergraphe) *Soit V un ensemble de sommets. On dit que $G = (V, S)$ est un hypergraphe si, et seulement si, $\forall S_i \in S\ S_i \subseteq V$ et $S_i \neq \varnothing$.*
Les ensembles contenus dans S sont nommés hyperarêtes.

La seule différence entre un hypergraphe et un graphe non orienté vient donc des hyperarêtes qui ne lient plus deux sommets entre eux, mais une quantité variable de sommets. Une représentation schématique des hyperarêtes est plus difficile à obtenir, et est moins lisible : les hyperarêtes deviennent des zones du schéma (voir la figure 2.3(a)). En pratique, les graphes étant des structures faciles à manipuler, il est courant d'utiliser des graphes non orientés pour modéliser un hypergraphe. Les deux modélisations les plus répandues sont le graphe primal et le graphe dual d'un hypergraphe.

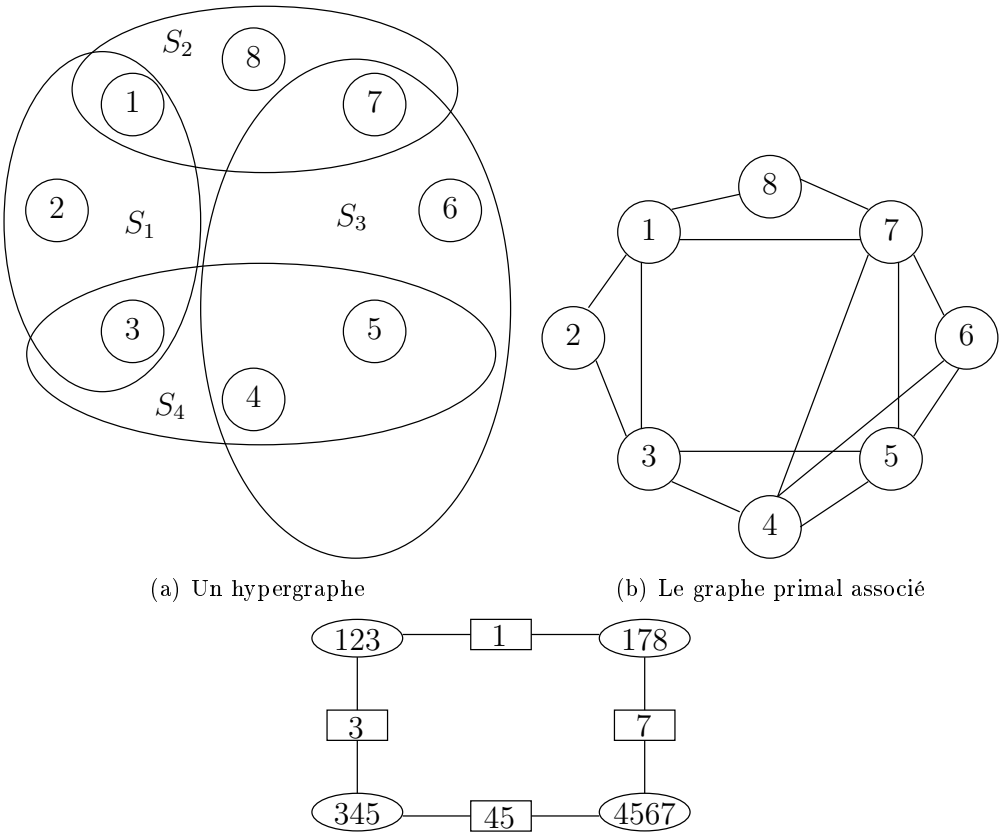
Définition 50 (Graphe primal d'un hypergraphe) *Soit $G_H = (V, S)$ un hypergraphe et $G = (V, E)$ un graphe non orienté. On dit que G est le graphe primal de l'hypergraphe G_H si, et seulement si, G est le plus petit graphe (selon le nombre d'arêtes) vérifiant $\forall \{s_i, s_j\} \in E\ \exists S_k \in S\ \{s_i, s_j\} \subseteq S_k$.*

Définition 51 (Graphe dual d'un hypergraphe) *Soit $G_H = (V, S)$ un hypergraphe et $G = (S, E)$ un graphe non orienté. On dit que G est le graphe dual de l'hypergraphe G_H si, et seulement si, $E = \{\{S_i, S_j\} / S_i \in S, S_j \in S, S_i \cap S_j \neq \varnothing\}$.*

Exemple 17 *Considérons l'hypergraphe $G_H = (\{1, \ldots, 8\}, \{S_1, S_2, S_3, S_4\})$ avec $S_1 = \{1, 2, 3\}$, $S_2 = \{1, 7, 8\}$, $S_3 = \{4, 5, 6, 7\}$ et $S_4 = \{3, 4, 5\}$. La représentation schématique de G_H correspond à la figure 2.3(a). Le graphe primal de G_H se déduit facilement en gardant l'ensemble des sommets, et en générant des arêtes de façon à ce que chaque hyperarête corresponde à un sous-graphe complet dans le graphe dual. On obtiendra donc le graphe de la figure 2.3(b). Le graphe dual contiendra un sommet par hyperarête (on représentera les sommets par des ellipses contenant l'hyperarête), et en connectant deux à deux les sommets ainsi obtenus par une arête si l'intersection des hyperarêtes qu'ils représentent est non vide. Dans la figure 2.3(c), l'intersection des hyperarêtes est représentée par un rectangle sur l'arête contenant cette intersection.*

2.1.3 Un modèle graphiquement décomposable : les réseaux GAI

Nous allons maintenant faire le lien entre GAI-décomposabilité et théorie des graphes. La représentation qui se prête le mieux pour modéliser une GAI-décomposition est un hypergraphe : l'ensemble des sommets correspondra aux attributs du problème, et il suffira de créer une hyperarête pour chaque sous-fonction d'utilité contenant l'ensemble des attributs sur lesquels porte le domaine de définition de la sous-utilité. En pratique, cette représentation ne facilite pas les calculs à effectuer sur les sous-utilités (comme par exemple la détermination de l'alternative préférée du décideur). Les réseaux GAI sont inspirés des travaux menés dans le domaine des réseaux bayésiens, et en particulier des algorithmes basés sur les arbres de jonction. La représentation est proche du graphe dual d'un hypergraphe tout en imposant une contrainte supplémentaire : le graphe dual doit être une forêt.



(a) Un hypergraphe (b) Le graphe primal associé

(c) Le graphe dual associé

FIGURE 2.3 – Représentation des hypergraphes

Définition 52 (Forêt de jonction) *Soit $G = (V, E)$ un graphe non orienté, et $C_1, \ldots, C_q$ des ensembles vérifiant $\forall j \in \{1, \ldots, q\}$ $C_j \subseteq \{1, \ldots, n\}$. On dit que G est un arbre de jonction de $X_1, \ldots, X_n$ selon $C_1, \ldots, C_q$ si, et seulement si :*

- *$V = \{X_{C_1}, \ldots, X_{C_q}\}$.*
- *G est une forêt.*
- *$\forall (X_{C_i}, X_{C_j}) \in E \times E$ $C_i \cap C_j \neq \varnothing$.*
- **(Propriété d'intersection courante)** *Pour tout couple de sommets (X_{C_i}, X_{C_j}) vérifiant $S_{ij} = C_i \cap C_j \neq \varnothing$, il existe une chaîne les reliant dans G et vérifiant pour chaque nœud X_{C_z} de la chaîne, $S_{ij} \subseteq C_z$.*

Les composantes connexes de la forêt de jonction seront appelées arbres de jonction, les sommets seront nommés les cliques et les arêtes les séparateurs.

Définition 53 (Réseau GAI) *Soit $\mathcal{X} = \prod_{i=1}^{n} X_i$ un espace d'alternatives, $\succsim$ une relation de préférence représentable selon la fonction d'utilité u, GAI-décomposable selon $C_1, \ldots, C_q$ en $u = \sum_{j=1}^{q} u_j$. On dit que $(G, \{u_j\})$ est un réseau GAI représentant u si, et seulement si, G est une forêt de jonction de $\{X_1, \ldots, X_n\}$ selon $C_1, \ldots, C_q$ et $\forall j \in \{1, \ldots, q\}$ $u_j : X_{C_j} \to \mathbb{R}$.*
On dira que la sous-fonction d'utilité u_j est associée à la clique C_j.

Remarque 8 *On peut objecter à cette définition qu'il peut ne pas exister de réseau GAI représentant certaines GAI-décompositions. Par exemple,* $u(x_1, x_2, x_3, x_4, x_5) = u_1(x_1, x_2) + u_2(x_2, x_3) + u_3(x_3, x_4) + u_4(x_4, x_5) + u_5(x_1, x_5)$ *ne peut se représenter directement sous forme de forêt de jonction : soit la propriété d'intersection courante ne sera pas vérifiée, soit il existera un cycle dans le graphe. Mais cette GAI-décomposition peut s'écrire* $u(x_1, x_2, x_3, x_4, x_5) = v_1(x_1, x_2, x_4) + v_2(x_1, x_4, x_5) + v_3(x_2, x_3, x_4)$*, en posant :*

$$\begin{cases} v_1(x_1, x_2, x_4) = u_1(x_1, x_2) \\ v_2(x_1, x_4, x_5) = u_4(x_4, x_5) + u_5(x_1, x_5) \\ v_3(x_2, x_3, x_4) = u_2(x_2, x_3) + u_3(x_3, x_4) \end{cases}$$

Nous supposerons pour l'instant que toute GAI-décomposition est représentable en réseau GAI, et nous présenterons en 2.2.2 une méthode pour générer automatiquement le réseau GAI d'une GAI-décomposition par triangulation de son graphe primal.

Les forêts de jonction et réseaux GAI se représenteront schématiquement de façon identique aux graphes duaux des hypergraphes : les cliques seront des ellipses contenant les attributs d'une sous-fonction d'utilité, et les séparateurs contiendront un rectangle dans lequel nous noterons le contenu de $X_{S_{ij}}$, l'intersection entre deux cliques X_{C_i} et X_{C_j} adjacentes.

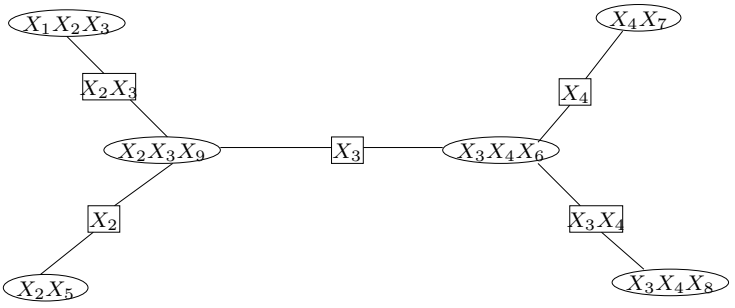


FIGURE 2.4 – Un réseau GAI

Exemple 18 *Considérons la fonction d'utilité* u *d'un décideur portant sur l'espace* $\mathcal{X} = \prod_{i=1}^{9} X_i$*, GAI-décomposable en* $u(x_1, \ldots, x_9) = u_1(x_1, x_2, x_3) + u_2(x_2, x_5) + u_3(x_2, x_3, x_9) + u_4(x_3, x_4, x_6) + u_5(x_4, x_7) + u_6(x_3, x_4, x_8)$*. Un réseau GAI correspondant à cette décomposition est représenté sur la figure 2.4. Nous pouvons constater que la propriété d'intersection courante est respectée : quelle que soit la chaîne reliant deux cliques, on trouvera dans toutes les cliques et tous les séparateurs constituant cette chaîne l'intersection de ces deux cliques, par exemple la chaîne reliant* $X_1X_2X_3$ *à* $X_3X_4X_8$ *contient uniquement des cliques et des séparateurs contenant* $\{X_1, X_2, X_3\} \cap \{X_3, X_4, X_8\} = \{X_3\}$*.*

Les réseaux GAI permettent de représenter les valeurs des sous-utilités (il suffit d'associer à chaque clique X_{C_j} la fonction u_j sous forme d'hypermatrice), et les indépendances entre groupes d'attributs : si deux cliques appartiennent à des arbres de jonction différents, alors les attributs de la première clique seront indépendants des attributs de la seconde clique. De plus, fixer la valeur d'un ou plusieurs attributs revient à éliminer ces attributs du graphe, et peut donc séparer une composante connexe en plusieurs

composantes connexes, comme le montre l'exemple 19, ce qui est une transposition des indépendances généralisées au sein de la GAI-décomposition.

Définition 54 (Indépendance généralisée) *Soit $\succsim$ une relation de préférence sur $\mathcal{X} = \prod_{i=1}^{n} X_i$, représentable par une fonction d'utilité u. Soit C_i, C_j et C_z une partition de $\{1, \ldots, n\}$, on dit que X_{C_i} est indépendant de X_{C_j} conditionnellement à X_{C_z} dans u si, et seulement si, il existe des sous-fonctions d'utilité u_1 et u_2 telles que $\forall x \in \mathcal{X}\ u(x) = u_1(x_{C_i}, x_{C_z}) + u_2(x_{C_j}, x_{C_z})$.*

Exemple 19 *Reprenons le réseau GAI de la figure 2.4. Chaque séparateur exploite une indépendance généralisée présente dans les préférences du décideur. Ainsi le séparateur X_2X_3 transcrit l'indépendance $u(x) = u_1(x_1, x_{23}) + v_1(x_{23}, x_{456789})$ avec $v_1 = u_2 + u_3 + u_4 + u_5$, le séparateur X_2 exprime l'indépendance $u(x) = u_2(x_5, x_2) + v_2(x_{1346789}, x_2)$ avec $v_2 = u_1 + u_3 + u_4 + u_5$, et ainsi de suite. Il est à noter que l'indépendance additive est englobée par cette généralisation en posant $C_z = \varnothing$ et sera représentée dans le réseau GAI par différents arbres de jonction. Par exemple, si $u(x_1, x_2) = u_1(x_1) + u_2(x_2)$, on aura X_1 indépendant de X_2 conditionnellement à $\varnothing$, et le réseau GAI correspondant contiendra les cliques X_1 et X_2 sans aucun séparateur.*

2.2 Représentation des préférences par des réseaux GAI

Nous avons défini précédemment toutes les bases pour représenter des préférences sur des domaines multiattributs à l'aide de réseaux GAI, nous allons maintenant nous intéresser à l'obtention d'utilités GAI-décomposables dans la section 2.2.1, puis nous étudierons de façon plus fine les interactions entre attributs dans les modèles graphiques pour générer automatiquement des réseaux GAI (en 2.2.2), et la manière d'utiliser ces méthodes pour intégrer des contraintes dans les réseaux GAI afin d'avoir une approche mixte entre préférences et contraintes au sein de la section 2.2.4.

2.2.1 Elicitation de préférences GAI-décomposables

Comme indiqué précédemment, cette thèse ne porte pas sur l'élicitation des préférences, mais un algorithme d'exploitation des préférences d'un décideur ne peut s'exécuter sans avoir préalablement instancié le modèle au moyen d'un algorithme d'élicitation. Il nous a donc semblé important de présenter brièvement un algorithme d'élicitation, et de faire un rapide résumé sur les techniques utilisées. Un algorithme d'élicitation des préférences doit donc être capable de produire un réseau GAI complet pour représenter les préférences d'un décideur. Comme évoqué dans l'introduction, les domaines d'application de ces travaux sont divers, et ne présupposent pas tous les mêmes méthodes d'élicitation des préférences. Certains poseront explicitement des questions au décideur, d'autres tenteront de déduire ses préférences en fonction des interactions que celui-ci peut avoir avec une application. Certains coderont directement les préférences d'un agent artificiel ou encore on peut imaginer des algorithmes d'apprentissage des préférences en fonction d'expérimentations (voir à ce sujet les travaux de Degris (2007) et notamment l'application d'un algorithme d'apprentissage pour générer des personnages non joueurs dans un jeu de combat en temps réel).

Parmi les travaux réalisés pour la génération d'un réseau GAI élicité, Gonzales et Perny (2004) présente un algorithme d'élicitation dans l'incertain et (Gonzales et Perny, 2005) son pendant dans le certain. Dans (Braziunas et Boutilier, 2005), une autre approche de l'élicitation est présentée, et la dernière section de l'article fournit les bases d'une procédure d'élicitation myope, c'est-à-dire n'interrogeant pas explicitement le décideur, mais mettant à jour sa fonction d'utilité GAI-décomposable au fur et à mesure de ses interactions avec une application.

Remarque 9 *Dans cette thèse, les réseaux GAI présentés sont identiques à ceux des articles de Christophe Gonzales et Patrice Perny. Les travaux sur l'élicitation de Darius Braziunas et Craig Boutilier portent sur des réseaux GAI qui correspondent dans cette thèse au graphe dual de l'hypergraphe des interactions entre attributs. Toutefois ce n'est pas restrictif et des parallèles peuvent se faire entre les deux représentations grâce aux algorithmes de triangulation que nous présenterons en 2.2.2.*

Les travaux évoqués ont pour hypothèse que la structure graphique du réseau GAI est déjà connue. Pour reprendre et synthétiser les travaux cités, tous se servent de la GAI-décomposition pour élaborer une procédure comportant deux opérations importantes :

- L'élicitation d'une sous-fonction d'utilité : En remarquant que si on instancie les séparateurs du réseau GAI, les sous-fonctions d'utilité sont additives (non généralisées), elles deviennent facilement élicitables. L'élicitation d'une sous-fonction d'utilité consiste à assigner les différentes valeurs de l'hypermatrice représentant cette sous-fonction d'utilité. Ces valeurs sont issues des réponses du décideur à différentes questions faisant intervenir des loteries (élicitation dans l'incertain) ou des séquences standards (élicitation dans le certain). Ces valeurs sont attribuées en fonction d'une alternative de référence.
- La mise à l'échelle : La sous-fonction d'utilité élicitée est ensuite mise à l'échelle par rapport aux sous-fonctions d'utilité déjà élicitées en utilisant l'alternative de référence et les attributs constituant les séparateurs du réseau.

Cette façon de procéder est classique dans l'utilisation des réseaux GAI (et plus généralement des forêts de jonction), et nous aurons l'occasion de présenter plusieurs autres algorithmes basés sur cette exploitation de la structure. Il est à noter qu'il existe un courant ne visant pas nécessairement à représenter explicitement une fonction d'utilité mais à modéliser une distribution de probabilité sur l'ensemble des fonctions d'utilité potentielles. L'élicitation consiste alors à essayer de transformer cette distribution en une loi de Dirac (Chajewska et Koller, 2000; Boutilier, 2002). Un autre courant modélise des ensembles d'utilités possibles et a donné lieu à la théorie des utilités multiattributs imprécises (« Imprecisely specified multiattribute utility theory ») et exploitant de l'incertain non probabilisé, voir White et al. (1983); Anandalingam et White (1993); Weber (1987); Blythe (2002); Braziunas et Boutilier (2007). Pour plus d'informations sur l'élicitation des fonctions d'utilité factorisées et les différentes approches utilisées, voir Braziunas et Boutilier (2008).

Ce qu'il faut retenir de cette sous-section est que s'intéresser aux algorithmes d'exploitation des réseaux GAI a un sens car les algorithmes d'élicitation existent, et qu'il est donc possible de définir des processus de prise de décision dans leur globalité.

2.2.2 Triangulation et construction automatique de réseaux GAI

Dans la remarque 8, nous avions évoqué le fait qu'obtenir une forêt de jonction à partir d'un ensemble de sous-fonctions d'utilité n'était pas tout le temps évident : nous avions une fonction d'utilité $u(x_1, x_2, x_3, x_4, x_5) = u_1(x_1, x_2) + u_2(x_2, x_3) + u_3(x_3, x_4) + u_4(x_4, x_5) + u_5(x_1, x_5)$. Si on crée un réseau ayant pour cliques X_1X_2, X_2X_3, X_3X_4, X_4X_5 et X_1X_5, il n'est pas possible de placer des séparateurs respectant à la fois la propriété d'intersection courante et l'acyclicité du réseau (pour que le réseau soit une forêt de jonction). Pourtant, nous avons vu qu'il existe une manière de réécrire la GAI-décomposition pour créer un réseau GAI représentant cette fonction d'utilité. Cette façon de faire peut être automatisée au moyen d'une structure de graphe intermédiaire (le graphe primal de l'hypergraphe, que nous appellerons graphe markovien, par analogie avec les réseaux bayésiens) et de deux algorithmes (la triangulation et la réduction).

Définition 55 (Graphe markovien) *Soit $\gtrsim$ une relation de préférence sur un ensemble $\mathcal{X} = \prod_{i=1}^{n} X_i$, représentable par une fonction d'utilité u, GAI-décomposable en $\forall x \in \mathcal{X}\ u(x) = \sum_{j=1}^{q} u_j(x_{C_j})$. Le graphe markovien G_M de la GAI-décomposition est le graphe primal de l'hypergraphe $G_H = (V, S)$, avec $V = \{X_i\}_{i=1}^{n}$, et $S = \{X_{C_j}\}_{j=1}^{q}$.*

Le graphe markovien se construit donc très simplement : Il suffit de créer un sommet par attribut composant $\mathcal{X}$, et de générer un sous-graphe complet pour chaque ensemble d'attributs composant une sous-fonction d'utilité. L'étude d'un graphe markovien permet de construire la forêt de jonction nécessaire pour définir le réseau GAI de la GAI-décomposition d'une fonction d'utilité. Cette construction se nomme triangulation, et le théorème 1 (adapté de Gavril (1974)) permet de caractériser les forêts de jonction dans lesquelles chaque clique correspond exactement aux attributs d'une fonction d'utilité.

Théorème 1 (Caractérisation d'une triangulation parfaite) *Soit G_M le graphe markovien d'une GAI-décomposition. Il existe une forêt de jonction F réprésentant la GAI-décomposition telle que toute clique de F corresponde à une clique maximale de G_M si, et seulement si, G_M est cordal.*

La reconnaissance d'un graphe markovien cordal est un problème polynômial qui peut être résolu avec des algorithmes simples et efficaces (Rose et al., 1976; Ohtsuki, 1976; Tarjan et Yannakakis, 1984). Ce type d'algorithme permet aussi de contruire ce que l'on nomme une forêt d'élimination. Une forêt d'élimination est une forêt de jonction dont les cliques ne correspondent pas nécessairement à des cliques maximales du graphe markovien. Dans le cas des graphes non cordaux, un algorithme de triangulation doit ajouter des arêtes au graphe markovien pour le rendre cordal. Ces arêtes sont nommées « fill-ins », et le graphe résultat sera le graphe triangulé, ou la triangulation du graphe markovien.

Définition 56 (Graphe triangulé, fill-ins) *Soit $G_M = (V_M, E_M)$ un graphe non orienté. On dit que le graphe non orienté $G_T = (V_M, E_T)$ est une triangulation du graphe G_M (ou encore que G_T est un graphe triangulé de G_M) , si et seulement si, $E_T = E_M \cup E_F$ et G_T est cordal. On nommera les arêtes de E_F les fill-ins.*

Algorithme 2 : Triangulation

Entrée : $G_M = (V_M, E_M)$ un graphe markovien
Sorties :
G_T : graphe triangulé,
G_A : forêt d'élimination,
σ : ordre d'élimination des sommets

créer le graphe $G_A = (V_A, E_A)$ avec $V_A = \varnothing$ et $E_A = \varnothing$;
créer le graphe $G_T = (V_T, E_T)$ avec $V_T = V_M$ et $E_T = \varnothing$;
créer le graphe Temp $= (V, E)$ avec $V = V_M$ et $E = E_M$;
pour *i variant de* 1 *à* $|V_M|$ **faire**
  choisir un sommet $X_{\sigma(i)}$ de V (σ : permutation de $\{1, \ldots, n\}$);
  créer Voisins $\leftarrow$ Ensemble des sommets voisins de $X_{\sigma(i)}$ dans Temp ;
  créer $C_i \leftarrow$ Voisins $\cup \{X_{\sigma(i)}\}$;
  créer Clique $\leftarrow \{\{X_{\sigma(i)}, X_k\}/X_k \in$ Voisins $\} \cup \{\{X_k, X_l\}/X_k, X_l \in$ Voisins $\}$;
  $E_T \leftarrow E_T \cup$ Clique ;
  $V_A \leftarrow V_A \cup \{C_i\}$;
  $E \leftarrow E \cup$ Clique ;
  $E \leftarrow E - \{\{X_{\sigma(i)}, X_k\} \in E\}$;
  $V \leftarrow V - \{X_{\sigma(i)}\}$;
fin
pour *i variant de* 1 *à* $|V_M|$ **faire**
  soit $k = \min_{X_{\sigma(j)} \in C_i}\{j/j \neq i\}$;
  si *k existe* **alors**
    $E_A \leftarrow E_A \cup \{\{C_i, C_k\}\}$;
  fin
fin
retourner (G_T, G_A, σ);

Rose (1970) a montré que trianguler un graphe revient à effectuer l'algorithme générique 2 (dans cet algorithme, nous construirons conjointement le graphe triangulé et la forêt d'élimination)

Dans cet algorithme, le graphe G_A va correspondre à la forêt d'élimination créée par la triangulation, et G_T au graphe triangulé. Temp est un graphe temporaire qui est exploité uniquement pendant la première boucle. Les lignes 01 à 03 initialisent ces trois graphes : G_A sera un graphe vide, G_T un graphe sans arêtes et contenant tous les attributs en tant que sommets, et Temp une copie de G_M qui sera réduite au fur et à mesure (sans impact pour G_M). La première boucle Pour détermine (à chaque itération i) à la ligne 05 un sommet de Temp $X_{\sigma(i)}$, crée deux variables locales : Voisins à la ligne 06 qui contient l'ensemble des sommets voisins de $X_{\sigma(i)}$ dans Temp, et Clique qui contiendra l'ensemble des arêtes possibles entre $X_{\sigma(i)}$ et ses voisins. La variable C_i correspond juste aux attributs de Voisins et $X_{\sigma(i)}$, puis les trois graphes sont modifiés :

- La forêt d'élimination G_A : on ajoute un sommet correspondant à C_i. Celui-ci devient une clique de la forêt de jonction (ligne 10).
- Le graphe triangulé G_T : on forme une clique entre $X_{\sigma(i)}$ et ses voisins existant encore dans Temp (ligne 09).
- Le graphe temporaire Temp : Après modification sur les deux autres graphes de la boucle, on forme une clique entre $X_{\sigma(i)}$ et ses voisins existant encore (ligne 11), puis on supprime $X_{\sigma(i)}$ et toutes ses arêtes adjacentes (lignes 12 et 13).

A la fin de la première boucle Pour (ligne 14), le graphe G_T correspond à une triangulation du graphe markovien G_M, et la forêt d'élimination G_A contient un ensemble de cliques, mais aucune arête. La seconde boucle Pour (ligne 15) va créer les arêtes de G_A en choisissant (si c'est possible) de lier la clique C_i à la première clique ayant été créée après C_i par l'élimination d'un attribut de C_i (lignes 16 et 17). A la fin de cette boucle (ligne 18), on peut garantir que G_A est une forêt de jonction. L'ensemble des fill-ins du graphe triangulé est donc l'ensemble des arêtes qu'il est nécessaire d'ajouter pour former une clique entre $X_{\sigma(i)}$ et ses voisins à la ligne 08 (donc la variable locale Clique contient des arêtes existant déjà dans E et des fill-ins).

Il est à noter que la forêt d'élimination générée et le graphe triangulé ne sont pas forcément uniques. Il est possible d'effectuer un autre choix de sommets (représenté par la fonction σ) pour obtenir une triangulation différente. Cette succession de choix de sommets sera nommée « séquence d'élimination ».

Définition 57 (Séquence d'élimination) *Soit $G_M = (V_M, E_M)$ un graphe markovien tel que $V_M = \{X_1, \ldots, X_n\}$. La séquence d'élimination de l'application de l'algorithme générique de triangulation sur G_M est une bijection $\sigma : \{1, \ldots, n\} \to \{1, \ldots, n\}$ telle que $\forall i \in \{1, \ldots, n\}$ $X_{\sigma(i)}$ est le sommet de V_M choisi à la i-ième itération de l'algorithme (ligne 05).*

Remarque 10 *Lorsque nous avions présenté les forêts de jonction, on pouvait s'interroger sur le choix d'appeler chaque sommet « une clique », alors que ce nom existait déjà en théorie des graphe pour symboliser un sous-graphe complet maximal. Cette dénomination est liée à la triangulation, car chaque clique de la forêt d'élimination G_A représente un sous-graphe complet du graphe triangulé G_T, et donc G_A n'est qu'une représentation factorisée de G_T respectant les propriétés de forêt et d'intersection courante.*

Il est aussi à noter que nous avons choisi de présenter les algorithmes de triangulation sous la forme d'une élimination de sommets. Cette approche n'est pas unique (Para et Scheffler, 1997), mais sera celle qui est utilisée en pratique dans nos expérimentations.

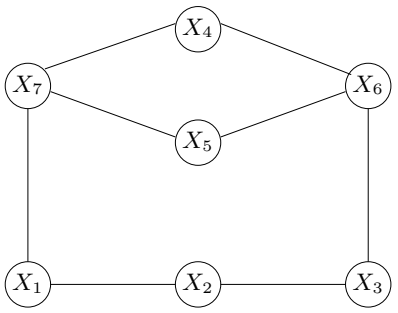
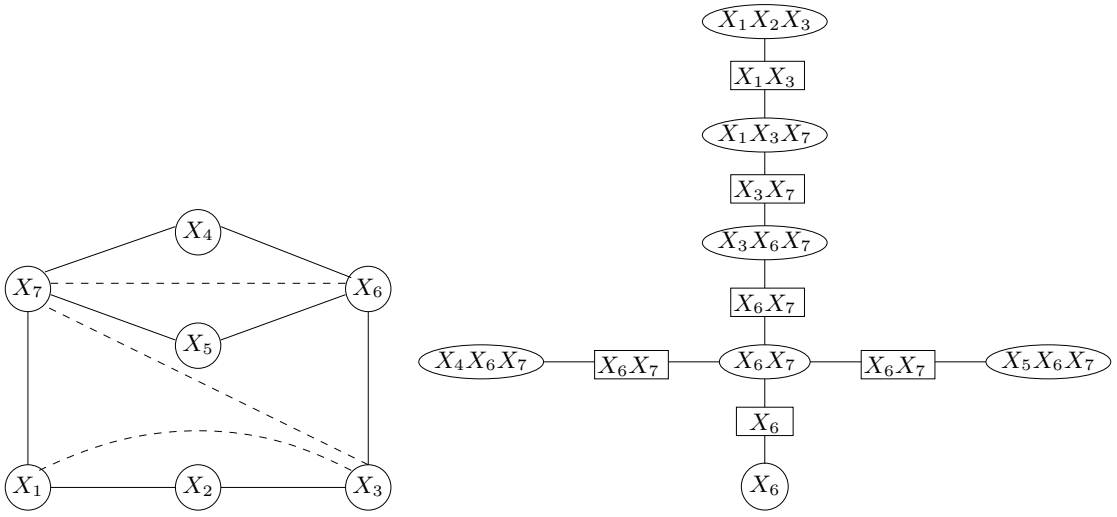


FIGURE 2.5 – La graphe markovien à trianguler



(a) Le graphe cordal obtenu (les fill-ins sont les arêtes en pointillés)

(b) La forêt d'élimination obtenue

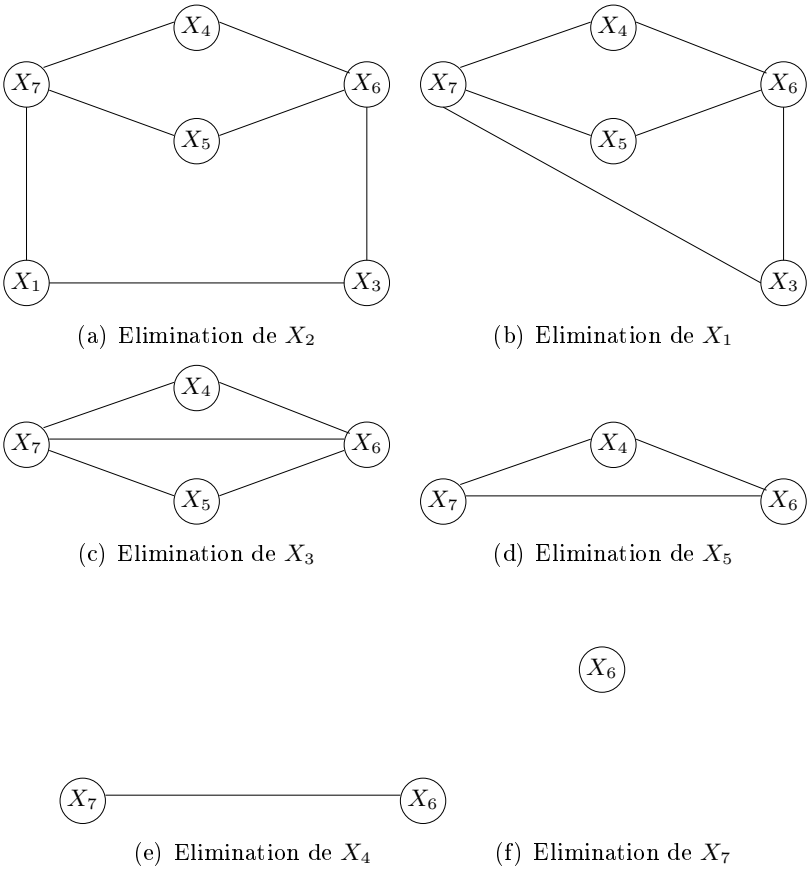
FIGURE 2.6 – Graphes produits par l'algorithme de triangulation

Exemple 20 *Considérons le graphe markovien représenté dans la figure 2.5, éxécutons la procédure de triangulation avec la séquence d'élimination* $(X_2, X_1, X_3, X_5, X_4, X_7, X_6)$ *et observons le déroulement en fonction des différents graphes intervenant dans l'algorithme :*

- *Temp (voir figures 2.7) est initialisé comme une copie du graphe markovien donné en argument. A la première itération,* X_2 *ayant pour voisin* X_1 *et* X_3*, le fill-in* $\{X_1, X_3\}$ *sera créé, puis le sommet* X_2 *sera supprimé (avec ses arêtes incidentes). A la fin de la première itération, on obtiendra donc le graphe de la figure 2.7(a). De la même façon sur le graphe résultant, l'élimination de* X_1 *engendrera le fill-in* $\{X_3, X_7\}$ *(figure 2.7(b)), l'élimination de* X_3 *ajoutera le fill-in* $\{X_6, X_7\}$ *(fi-*

gure 2.7(c)). Le graphe étant devenu une clique, toutes les autres éliminations n'ajouterons pas de fill-ins (figures 2.7(d), 2.7(e) et 2.7(f)).

- *G_T est une copie des sommets du graphe markovien sans arêtes. A chaque itération, on ajoute au graphe des arêtes appartenant au graphe markovien initial et des fill-ins. Il en résulte qu'à la fin de la triangulation, G_T correspondra au graphe markovien et aux fill-ins ajoutés (représenté dans la figure 2.6(a)).*
- *G_A est l'arbre d'élimination, ses cliques sont $X_1X_2X_3$ (élimination de X_2), $X_1X_3X_7$ (élimination de X_1), $X_3X_6X_7$ (élimination de X_3), $X_5X_6X_7$ (élimination de X_5), $X_4X_6X_7$ (élimination de X_4), X_6X_7 (élimination de X_6)et X_7 (élimination de X_7). La seconde boucle crée les séparateurs de cet arbre d'élimination, le premier sommet éliminé de $X_1X_2X_3$ différent de X_2 est X_1, $X_1X_2X_3$ sera donc lié à $X_1X_3X_7$. En suivant un raisonnement identique pour toutes les autres cliques, on obtient finalement le graphe de la figure 2.6(b).*



(a) Elimination de X_2 (b) Elimination de X_1

(c) Elimination de X_3 (d) Elimination de X_5

(e) Elimination de X_4 (f) Elimination de X_7

FIGURE 2.7 – Exemple de triangulation

Nous avons vu comment obtenir un graphe triangulé et une forêt d'élimination à partir d'un graphe markovien. Toutefois (comme le montre l'exemple 20) cette forêt n'est pas obligatoirement compacte (certaines cliques peuvent être entièrement contenues dans d'autres cliques). Obtenir une forêt de jonction compacte est le but d'un algorithme de réduction comme présenté dans l'algorithme 3.

Algorithme 3 : Réduction

Entrée : $G_A = (V_A, E_A)$ une forêt d'élimination
Sortie : G_A une forêt de jonction

```
1 tant que ∃{C_i, C_j} ∈ E_A avec C_i ⊂ C_j faire
2 |   créer Voisins ← {C_k/{C_i, C_k} ∈ E_A};
3 |   Voisins ← Voisins \{C_j};
4 |   E_A ← E_A\{{C_i, C_k}/{C_i, C_k} ∈ E_A};
5 |   V_A ← V_A\{C_i};
6 |   pour tous les C_k ∈ Voisins faire
7 |   |   E_A ← E_A ∪ {{C_j, C_k}};
8 |   fin
9 fin
```

Le principe de fonctionnement de cet algorithme est d'identifier deux cliques voisines dont l'une (C_i) est complètement incluse dans l'autre (C_j) à la ligne 01 (on présente cette recherche sous forme d'arêtes grâce à la propriété d'intersection courante). Si de telles cliques existent, on stocke l'ensemble des cliques voisines de C_i dans la variable locale Voisins (ligne 02), à l'exception de C_j (ligne 03), puis on supprime toutes les arêtes adjacentes à C_i (ligne 04) pour finalement supprimer C_i du graphe (ligne 05) et réassigner toutes les anciennes cliques voisines de C_i comme de nouvelles voisines de C_j (ligne 06 à 09). Ainsi les propriétés de forêt et d'intersection courante sont conservées.

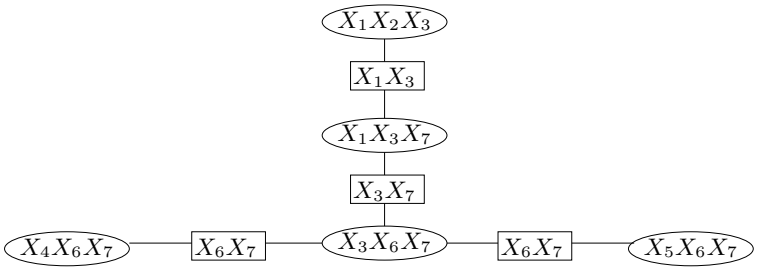


FIGURE 2.8 – La forêt de jonction après réduction

Exemple 21 *Considérons la procédure de réduction appliquée à la forêt d'élimination de l'exemple 20. La clique X_6 est complètement contenue dans la clique voisine X_6X_7. Elle n'a pas d'autres voisins. On peut donc supprimer X_6 et l'arête reliant X_6 à X_6X_7. X_6X_7 est complètement contenue dans la clique voisine $X_3X_6X_7$ et possède d'autres voisins ($X_4X_6X_7$ et $X_5X_6X_7$). Il est donc nécessaire de créer deux arêtes reliant $X_4X_6X_7$ et $X_5X_6X_7$ à $X_3X_6X_7$ avant de supprimer X_6X_7. Après cette opération, aucune autre itération n'est possible. On a donc obtenu la forêt de jonction de la figure 2.8.*

Nous disposons donc de deux procédures : la triangulation permet la génération d'une forêt d'élimination à partir du graphe markovien d'une GAI-décomposition via une triangulation, et la réduction permet de compacter la forêt d'élimination en une forêt de jonction qui correspondra à la structure du réseau GAI. Pour construire automatiquement un réseau GAI à partir d'une GAI-décomposition, il suffit donc d'assigner à chaque

clique de la forêt de jonction une hypermatrice correspondant à une sous-fonction d'utilité. L'ensemble des attributs du domaine de définition de chaque sous-fonction d'utilité est intégralement contenu dans une clique de la forêt de jonction. Si deux sous-fonctions d'utilité doivent être assignées à une même clique, il suffit de les additionner.

2.2.3 Considérations sur la triangulation

Dans la sous-section précédente, nous avons vu qu'il est possible de compiler toute expression GAI-décomposable sous la forme de réseau GAI, mais qu'il est parfois nécessaire de reformuler la GAI-décomposition. Cette reformulation a un impact sur le coût de stockage du réseau GAI. L'ordre de grandeur donnée dans la proposition 13 doit s'appliquer à la reformulation de la GAI-décomposition, dont les ensembles de définition des sous-fonctions d'utilité contiennent plus d'attributs. Il est possible de reformuler cet ordre de grandeur de manière graphique sur la forêt de jonction :

Propriété 17 (Ordre de grandeur d'un réseau GAI) *Soit $G = (C, S)$ un réseau GAI. A chaque clique $C_i \in C$ est associée la sous-fonction d'utilité u_i. Alors le coût de stockage nécessaire pour représenter un tel réseau est en $\Theta(\sum_{C_j \in C} \prod_{X_i \in C_j} |X_i|)$.*

Cet ordre de grandeur est analogue à celui exprimé dans la proposition 13, la différence étant l'exploitation de la structure graphique. Nous avons vu que le réseau GAI engendré par triangulation et réduction dépend de la séquence d'élimination des attributs utilisée dans l'algorithme. Il en vient naturellement qu'en pratique, on ne cherche pas n'importe quelle triangulation du graphe markovien, mais une triangulation visant à minimiser l'espace de stockage nécessaire au réseau GAI qui sera engendré. Nous allons reformuler l'ordre de grandeur de façon moins précise mais faisant bien apparaître ce qui est critique dans le réseau GAI engendré :

Définition 58 (Treewidth induite) *Soit $F = (C, S)$ une forêt de jonction, on définit $\forall C_j \in C$ la largeur d'une clique comme étant le nombre d'attributs contenus dans cette clique moins 1 : $|C_j| - 1$. La treewidth induite t d'une forêt de jonction est la largeur maximale des cliques de cette forêt : $t = \max_{C_i \in C} |C_i| - 1$.*

Propriété 18 (Ordre de grandeur d'un réseau GAI (reformulation)) *Soit $G = (C, S)$ un réseau GAI. A chaque clique $C_j \in C$ est associée la sous-fonction d'utilité u_j. Alors le coût de stockage nécessaire pour représenter un tel réseau est en $O(|C|k^{t+1})$, avec t la treewidth de la forêt de jonction, et k la modalité maximale des attributs.*

Preuve. Inspirée de Dechter (2003). Soit $k = \max_{i=1}^{n} |X_i|$ la modalité maximale des attributs. A chaque clique C_j on associe une sous-fonction d'utilité u_j définie sur le produit cartésien des attributs contenus dans C_j. Il est donc nécessaire pour chaque sous-fonction d'utilité de stocker $\prod_{X_i \in C_j} |X_i|$ valeurs. Or, par définition de k, on a $\forall i \in \{1, \ldots, n\}\ |X_i| \leqslant k$. Donc, $\prod_{X_i \in C_j} |X_i| \leqslant k^{|C_j|}$ et par définition de la treewith, on a $\forall j \in \{1, \ldots, q\}\ |C_j| \leqslant t + 1$, donc $k^{|C_j|} \leqslant k^{t+1}$. On en déduit donc que l'espace de stockage nécessaire à la représentation du réseau GAI est : $\sum_{C_j \in C} \prod_{X_i \in C_j} |X_i| \leqslant \sum_{C_j \in C} k^{t+1} \leqslant |C|k^{t+1}$. Donc, la complexité en espace asymptotique de cette représentation est en $O(|C|k^{t+1})$. ■

De ces deux reformulations, il se dégage les observations suivantes :
- Le nombre maximal d'attributs (représenté par la treewidth dans la propriété 18) contenus dans une clique est critique pour la complexité en espace : une augmentation de ce nombre se traduit par une augmentation exponentielle de l'espace mémoire occupé par le réseau.
- L'augmentation du nombre de cliques d'un réseau GAI fait augmenter linéairement la complexité mémorielle, mais la taille des cliques est souvent bien en dessous de k^{t+1}. Il est donc plus important de regarder en premier lieu le contenu des cliques plutôt que le nombre de cliques.

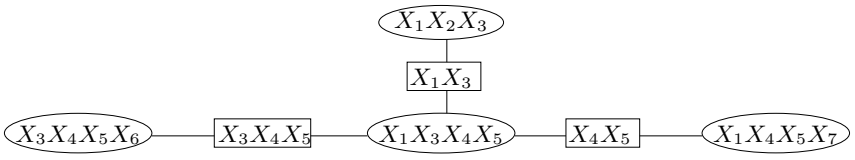


FIGURE 2.9 – Une autre triangulation (et réduction) du graphe de la figure 2.5

Exemple 22 *Reprenons l'exemple de triangulation de la section 2.2.2. Après triangulation et réduction nous avions obtenu la forêt de jonction de la figure 2.8. Toutefois nous avons vu qu'une triangulation dépend de la séquence d'élimination utilisée. Si nous essayons de trianguler le même graphe avec la séquence d'élimination* $X_6X_7X_2X_5X_1X_2X_3$*, nous obtenons (après réduction) la forêt de jonction de la figure 2.9. Prenons en compte maintenant que ces opérations ont été faites dans le but d'obtenir des réseaux GAI, et qu'il faut donc associer à chaque clique une hypermatrice de nombres (pour représenter les sous-fonctions d'utilité). Si on suppose que chaque attribut peut prendre* 10 *valeurs différentes, alors la forêt de jonction de la figure 2.8 (de treewidth* 2*) nécessitera la représentation de* $10^3 + 10^3 + 10^3 + 10^3 = 4000$ *valeurs. La forêt d'élimination de la figure 2.9 (de treewidth* 3*) implique le stockage de* $10^3 + 10^4 + 10^4 + 10^4 = 31000$ *valeurs. En prenant en compte que ce sont deux triangulations d'un petit graphe à* 7 *attributs, les problèmes concrets peuvent nécessiter des espaces de stockage prohibitifs si on néglige la séquence d'élimination.*

Nous avons donc d'une part la treewidth qui est un élément critique dans la complexité du réseau GAI, et d'autre part les triangulations du graphe makovien qui ne sont pas uniques. Il en vient naturellement que la séquence d'élimination à utiliser dans une triangulation devrait être calculée pour faire en sorte de minimiser la complexité du réseau GAI. Dans la littérature, différents critères ont été énoncés pour caractériser cette baisse de complexité au sein des triangulations, allant de la minimisation du nombre de fill-ins (dans le cadre qui nous intéresse, cela correspondra à minimiser les dépendances ajoutées par la triangulation entre les attributs) à la minimisation de la treewidth du réseau GAI engendré. Ces problèmes sont NP-difficiles (Arnborg et al., 1987; Yannakakis, 1981), mais il existe de bonnes heuristiques pour calculer des séquences intéressantes (Kjærulff, 1990; Leimer, 1993). De plus :
- Le recherche d'une séquence d'élimination optimale peut se paralléliser de manière à optimiser les temps de calcul (Dahlhaus et Karpinski, 1989).
- Nous avons présenté une façon de calculer la forêt de jonction à partir d'un graphe triangulé (en construisant la forêt d'élimination et en la réduisant), toutefois cette

approche n'est pas unique : Golumbic (1980) a défini une approche basée sur un ordre lexicographique entre nombre de variables contenues dans un séparateur et le produit des modalités des attributs d'une clique.
- Certaines règles basées sur la topologie du graphe permettent d'affirmer que l'élimination d'un sommet n'augmentera pas plus la treewidth que le choix d'un autre sommet (van den Eijkhof et Bodlaender, 2002).
- Il existe des algorithmes visant à produire des triangulations telles que la suppression d'un fill-in entraine la suppression de la cordalité du graphe (Kjærulff, 1990; Peyton, 2001; Berry et al., 2006).
- Un algorithme de triangulation incrémentale basé sur la structure des Maximal Prime Subgraph permet d'analyser plus finement l'impact de l'ajout d'une arête ou de l'élimination d'un sommet (Flores et al., 2003; Madsen et Olesen, 2002).
- Parmi les algorithmes ne cherchant pas à déterminer la triangulation optimale, il existe des algorithmes avec garantie de performance (Bodlaender et al., 1995; Shoikhet et Geiger, 1997; Eyal, 2001).

2.2.4 Agrégation et intégration de contraintes dans les réseaux GAI

Les réseaux GAI représentent des préférences. Toutefois pour certains problèmes, il est nécessaire de mêler ces préférences avec des contraintes (budgétaires par exemple). Les réseaux GAI permettent de mélanger ces deux concepts, de façon similaire aux réseaux de contraintes flexibles (Weighted-CSP (Dechter, 2003; Larrosa et Schiex, 2003)), au moyen d'une opération que nous nommerons agrégation.

Définition 59 (Agrégation de graphes markoviens) *Soit $G_M^1 = (V_M^1, E_M^1)$ et $G_M^2 = (V_M^2, E_M^2)$ deux graphes markoviens. On dit que $G_M = (V_M, E_M)$ est le graphe markovien agrégé de G_M^1 et G_M^2 si, et seulement si, $V_M = V_M^1 \cup V_M^2$ et $E_M = E_M^1 \cup E_M^2$.*
On notera $G_M = G_M^1 + G_M^2$.
Note : l'intersection de V_M^1 et V_M^2 n'est pas nécessairement vide.

L'agrégation de deux graphes markoviens consiste à former un nouveau graphe markovien contenant tous les attributs des deux graphes, et l'union de toutes les arêtes de ces graphes. En procédant de cette façon, nous pouvons être sûrs que toutes les dépendances sont respectées : toute clique représentant une fonction de sous-utilité se retrouvera dans le graphe markovien agrégé. En pratique, nous ne manipulons jamais directement les graphes markoviens, mais une représentation utilisant une forêt de jonction. Nous avons vu précédemment que la transposition d'un graphe markovien en une forêt de jonction est simple, et correspond à une triangulation (suivie d'une réduction). La définition de l'agrégation de graphes markoviens trouve donc naturellement sa transposée pour les forêts de jonction :

Définition 60 (Agrégation de forêts de jonction) *Soit F^1 et F^2 deux forêts de jonction, de graphes markoviens respectifs G_M^1 et G_M^2. Une forêt de jonction agrégée de F^1 et F^2 est une forêt de jonction obtenue par triangulation (et réduction) de $G_M^1 + G_M^2$.*

L'agrégation de deux forêts de jonction est donc une opération respectant les dépendances de ces forêts étant donné que ce respect des dépendances existe pour l'agrégation des graphes markoviens et pour la procédure de triangulation. Supposons maintenant

que dans le cadre d'un problème donné, nous soyons amenés à manipuler deux réseaux GAI représentant deux fonctions d'utilité u^1 et u^2, et que le problème nous amène à considérer la fonction d'utilité $u = u^1 + u^2$. Cette fonction reste GAI-décomposable, et ses sous-fonctions d'utilité sont les sous-fonctions d'utilité de u^1 et u^2, le réseau GAI représentant u doit donc être un réseau respectant toutes les dépendances de u^1 et u^2. Il en découle donc la définition suivante :

Définition 61 (Agrégation par sommation de réseaux GAI) *Soit $\mathcal{X} = \prod_{i=1}^{n} X_i$ un espace d'alternatives, u^1 et u^2 deux fonctions d'utilité GAI-décomposables représentables par deux réseaux GAI dont les forêts de jonction sont F^1 et F^2. Le réseau GAI agrégé de u^1 et u^2 est un réseau GAI représentant u vérifiant :*

- *Sa forêt de jonction est une agrégation de F^1 et F^2.*
- *$\forall x \in \mathcal{X}\ u(x) = u^1(x) + u^2(x)$.*

Puisque la forêt de jonction du réseau GAI représentant u respecte toutes les dépendances de u^1 et u^2, pour toute sous-fonction d'utilité u^i_j, il existera une clique formant un sur-ensemble des attributs du domaine de définition de u^i_j. Il suffit donc d'assigner à chaque sous-fonction d'utilité sa clique correspondante par sommation.

Remarque 11 *Nous avons choisi de préciser « par sommation » car nous développerons une nouvelle façon d'agréger des réseaux GAI. Toutefois, quelle que soit la façon d'agréger ces réseaux, les définitions de base (sur les graphes markoviens et les forêts de jonction) restent inchangées. Il est donc important de prendre en compte deux aspects de l'agrégation : la prise en compte des dépendances (aspect qui s'exprime graphiquement) et la manière de placer les sous-fonctions d'utilité dans les cliques (ici, par sommation).*

Une application directe de ces définitions est l'intégration de contraintes dans les réseaux GAI. Dans la littérature, les contraintes sont souvent représentées par des ensembles représentant des restrictions, mais pour faciliter l'intégration des contraintes dans le contexte qui nous intéresse, nous allons les définir directement de manière fonctionnelle. Cette définition n'est pas restrictive et est particulièrement adaptée aux raisonnements par forêt de jonction (voir le livre de Dechter (2003)).

Définition 62 (Contrainte) *Soit $\mathcal{X} = \prod_{i=1}^{n} X_i$ un espace d'alternatives. On définit une contrainte sur les attributs indicés par $N \subseteq \{1, \ldots, n\}$ comme étant une fonction $c : \prod_{j \in N} X_j \to \{0, -\infty\}$.*

Cette définition étant proche des sous-fonctions d'utilité, il semble naturel de l'utiliser au sein d'un réseau GAI pour représenter des associations d'attributs interdites ($-\infty$) ou autorisées (0).

Définition 63 (Fonction d'utilité étendue) *Soit $\mathcal{X}$ un espace d'alternatives, $\succsim$ une relation de préférence sur $\mathcal{X}$ représentable par une fonction d'utilité $u : \mathcal{X} \to \mathbb{R}$ et $\{c_1, \ldots, c_{q'}\}$ des fonctions représentant des contraintes. La fonction d'utilité u étendue aux contraintes $\{c_1, \ldots, c_{q'}\}$ (de domaines de définition respectifs $X_{C_1}, \ldots, X_{C_{q'}}$) est une fonction $\bar{u} : \mathcal{X} \to \mathbb{R} \cup \{-\infty\}$ vérifiant $\forall x \in \mathcal{X}\ \bar{u}(x) = u(x) + \sum_{i=1}^{q'} c_i(x_{C_i})$.*

Propriété 19 (Respect des contraintes) *Soit $\bar{u} : \mathcal{X} \to \mathbb{R} \cup \{-\infty\}$ une fonction d'utilité u étendue à un ensemble de contraintes. $\forall x \in \mathcal{X}$ on a :*

- *Si $\bar{u}(x) = -\infty$ alors x ne respecte pas au moins une des contraintes de l'extension.*
- *Si $\exists a \in \mathbb{R}\ \bar{u}(x) = a$ alors x respecte toutes les contraintes de l'extension, et $u(x) = a$.*

Ainsi, si un réseau GAI ne représente non plus la fonction d'utilité u mais son extension à un ensemble de contraintes $\bar{u}$, il est aisé de savoir si une contrainte est violée, et, dans le cas contraire, la valeur de l'utilité initiale. Dans le cadre de fonctions GAI-décomposables, on aura donc une fonction $\bar{u}$ de la forme $\bar{u} = \sum_i u_i + \sum_j c_j$. L'agrégation par sommation nous permet de générer, à partir d'un réseau GAI représentant u, un réseau GAI représentant $\bar{u}$ si on considère qu'à chaque contrainte, il est possible d'associer un graphe markovien de la façon suivante :

Définition 64 (Graphe markovien d'une contrainte) *Soit $c_i : X_{C_i} \to \{0, -\infty\}$ une contrainte. Le graphe markovien associé à c_i est un graphe non orienté $G_M = (V_M, E_M)$ tel que $V_M = \{X_j / j \in C_i\}$ et $E_M = \{\{X_j, X_k\} / j \in C_i$ et $k \in C_i\}$.*

Le graphe markovien d'une contrainte est donc une clique sur l'ensemble des attributs intervenant dans la contrainte. En exploitant cette définition, on en déduit l'algorithme 4 d'intégration de contraintes.

Algorithme 4 : IntégrerContraintes

Entrées :
F : forêt de jonction,
$\{u_1, \ldots, u_q\}$: sous-fonctions d'utilité associées aux cliques de F,
$\{c_1, \ldots, c_{q'}\}$: ensemble de contraintes à respecter
Sortie : F' : réseau GAI intégrant les contraintes

```
01 G_M ← générer graphe markovien de F;
02 pour j variant de 1 à q' faire
03 |   G'_M ← générer le graphe markovien de c_j;
04 |   G_M ← G_M + G'_M;
05 fin
06 F' ← trianguler (et réduire) G_M;
07 pour i variant de 1 à q faire
08 |   Placer par sommation u_i dans F';
09 fin
10 pour j variant de 1 à q' faire
11 |   Placer par sommation c_j dans F';
12 fin
```

Ce pseudo-code contient deux parties : l'agrégation des graphes markoviens (lignes 01 à 06) et le placement des sous-fonctions d'utilité (lignes 07 à 12). L'agrégation respecte les définitions énoncées plus haut : la forêt de jonction F' obtenue à la ligne 06 est une triangulation d'un graphe markovien agrégé (G_M) contenant toutes les dépendances de la forêt de jonction initiale (ligne 01) et toutes les dépendances des contraintes (ligne 03).

Nous disposons donc d'une extension des réseaux GAI permettant de prendre en compte conjointement préférences et contraintes. Dans la suite de cette thèse, nous raisonnerons sur des réseaux GAI « classiques », mais l'intégralité des algorithmes exploitant des réseaux GAI s'appliqueront aussi aux réseaux prenant en compte les contraintes.

2.3 Exploitation des réseaux GAI pour la résolution de problèmes décisionnels monocritères

Nous avons défini dans les sections précédentes ce qu'était le modèle GAI, et sa structure de données associée : les réseaux GAI. Nous avons évoqué l'existence d'algorithmes d'élicitation des préférences pour ce modèle, ainsi que les méthodes de triangulation pour intégrer des contraintes dans les préférences élicitées. Nous allons maintenant nous intéresser aux travaux ayant déjà été menés pour l'exploitation de ces réseaux GAI afin de résoudre deux problèmes :

- Le problème du choix optimal : il consiste en la détermination de l'alternative préférée d'un décideur
- Le problème du rangement : problème proche du choix optimal, il ne s'agit plus de déterminer l'alternative préférée, mais les k meilleures alternatives (au sens de la relation de préférence du décideur), pour un k quelconque.

Remarque 12 *Pour tous les algorithmes d'exploitation des réseaux GAI, que ce soit dans l'état de l'art ou pour les approches que nous proposerons, nous supposerons que le réseau GAI est un arbre de jonction avec des sous-fonctions d'utilité stockées dans les cliques. Une telle approche est plus restrictive que le cas général, mais nous aborderons toujours à la fin d'une présentation d'algorithme le cas où le réseau GAI a une structure de forêt de jonction pour appliquer l'algorithme dans le cas général (qui consiste généralement à appliquer l'algorithme pour chaque arbre de la forêt, puis à fusionner les résultats obtenus pour considérer la forêt dans sa globalité).*

2.3.1 Détermination du choix optimal

Le problème du choix optimal consiste à déterminer l'alternative préférée d'un décideur (voir définition 65). Comme nous nous plaçons dans le cadre des modèles GAI, nous pouvons garantir que cette alternative préférée existe (propriété 20). Dans ce formalisme, cette recherche consiste à déterminer l'alternative maximisant la fonction d'utilité (propriété 21).

Définition 65 (Alternative préférée) *Soit $\mathcal{X}$ un espace d'alternatives, $\succsim$ une relation de préférence sur $\mathcal{X}$, et $\succ$ la relation de préférence stricte issue de $\succsim$. On dit que l'alternative $x^* \in \mathcal{X}$ est une alternative préférée de $\succsim$ si, et seulement si, $\nexists x \in \mathcal{X}$ vérifiant $x \succ x^*$.*

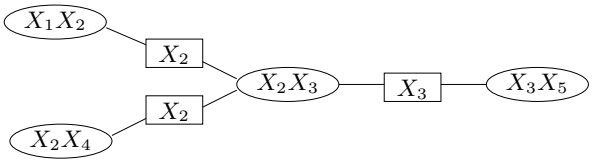
Propriété 20 (Condition suffisante d'existence) *Soit $\mathcal{X}$ un espace d'alternatives fini et $\succsim$ une relation de préférence sur $\mathcal{X}$. Si $\succsim$ est représentable par une fonction d'utilité, alors il existe une alternative $x^* \in \mathcal{X}$, préférée au sens de $\succsim$.*

Propriété 21 *Soit $\mathcal{X}$ un espace d'alternatives fini, $\succsim$ une relation de préférence sur $\mathcal{X}$, représentable par une fonction d'utilité $u : \mathcal{X} \to \mathbb{R}$, et $x^* \in \mathcal{X}$. On a :*

$$x^* \text{ alternative préférée de } \succsim \Leftrightarrow x^* = \arg\max_{x \in \mathcal{X}} u(x)$$

Remarque 13 *Cette alternative préférée existe mais n'est pas forcément unique : plusieurs alternatives peuvent partager la même valeur d'utilité maximale. Le problème du choix optimal ne traite dans ce cas que la détermination d'une des alternatives préférées, mais on peut obtenir l'ensemble des alternatives préférées en utilisant l'algorithme de rangement qui sera présenté ensuite.*

Une première approche simple consiste à énumérer toutes les alternatives de $\mathcal{X}$ et à conserver celle d'utilité maximale, mais nous avons vu que la taille de $\mathcal{X}$ augmentait de façon exponentielle selon le nombre d'attributs. Il n'est donc pas raisonnable d'envisager cette approche dès que le nombre d'attributs devient grand. Dans (Gonzales et Perny, 2004), un algorithme a été proposé pour réduire les calculs nécessaires à la résolution de ce problème en exploitant la structure du réseau GAI (et donc les indépendances entre attributs du décideur). Cette approche est une application directe de la programmation dynamique non sérielle (Bertele et Brioschi, 1972), axiomatisée dans Shafer et Shenoy (1990); Shenoy et Shafer (1990). Nous allons dans un premier temps utiliser un exemple pour présenter cet algorithme, puis nous le formaliserons avant d'étudier ses limites.



(a) Forêt de jonction

X_2	X_1 x_{11}	x_{12}
x_{21}	5	7
x_{22}	3	10
x_{23}	15	4

(b) $u_1(x_1, x_2)$

X_4	X_2 x_{21}	x_{22}	x_{23}
x_{41}	8	1	2
x_{42}	10	12	3

(c) $u_2(x_2, x_4)$

X_3	X_2 x_{21}	x_{22}	x_{23}
x_{31}	6	5	10
x_{32}	8	8	14

(d) $u_3(x_2, x_3)$

X_5	X_3 x_{31}	x_{32}
x_{51}	1	8
x_{52}	10	11
x_{53}	5	2

(e) $u_4(x_3, x_5)$

FIGURE 2.10 – Réseau GAI de l'exemple 23

Exemple 23 *Considérons l'espace d'alternatives $\mathcal{X} = X_1 \times X_2 \times X_3 \times X_4 \times X_5$ et la fonction GAI-décomposable $u(x_1, x_2, x_3, x_4, x_5) = u_1(x_1, x_2) + u_2(x_2, x_4) + u_3(x_2, x_3) + u_4(x_3, x_5)$ représentée par le réseau GAI de la figure 2.10. La détermination de l'alternative préférée revient à déterminer $x^* \in \mathcal{X}$ tel que $u(x^*) = \max_{x \in \mathcal{X}} u(x)$, ce qui correspond donc à déterminer $\max_{x_1 \in X_1} \max_{x_2 \in X_2} \max_{x_3 \in X_3} \max_{x_4 \in X_4} \max_{x_5 \in X_5} u_1(x_1, x_2) + u_2(x_2, x_4) + u_3(x_2, x_3) + u_4(x_3, x_5)$. Or, en utilisant les propriétés de commutativité et d'associativité de l'addition, et le fait que les sous-fonctions d'utilité ne sont pas définies*

sur l'intégralité des attributs du problème, il est possible de réécrire la formule de la façon suivante :

$$u(x^*) = \max_{x_5 \in X_5} \max_{x_3 \in X_3} \left(u_4(x_3, x_5) + \max_{x_2 \in X_2} \left[u_3(x_2, x_3) + \max_{x_1 \in X_1} u_1(x_1, x_2) + \max_{x_4 \in X_4} u_2(x_2, x_4) \right] \right)$$

En introduisant des fonctions $\phi_{i,j}$ et ψ_i pour montrer la décomposition des calculs, cette expression s'écrit de façon plus lisible :

$$\begin{cases} \psi_2(x_2, x_4) & = u_2(x_2, x_4) \\ \phi_{2,3}(x_2) & = \max_{x_4 \in X_4} \ \psi_2(x_2, x_4) \\ \psi_1(x_1, x_2) & = u_1(x_1, x_2) \\ \phi_{1,3}(x_2) & = \max_{x_1 \in X_1} \ \psi_1(x_1, x_2) \\ \psi_3(x_2, x_3) & = u_3(x_2, x_3) + \phi_{1,3}(x_2) + \phi_{2,3}(x_2) \\ \phi_{3,4}(x_3) & = \max_{x_2 \in X_2} \ \psi_3(x_2, x_3) \\ \psi_4(x_3, x_5) & = u_4(x_3, x_5) + \phi_{3,4}(x_3) \\ u(x^*) & = \max_{x_5 \in X_5} \max_{x_3 \in X_3} \ \psi_4(x_3, x_5) \end{cases}$$

En utilisant ces notations, les ψ_i sont des fonctions utilisées pour représenter une sous-fonction d'utilité et pour réaliser certaines sommes de l'expression, tandis que les $\phi_{i,j}$ sont des réductions des ψ_i sur un sous-ensemble d'attributs en utilisant la maximisation. On peut faire trois constatations :

- *Les fonctions ψ_i sont exactement définies sur les attributs de la clique contenant l'utilité u_i.*
- *Les fonctions $\phi_{i,j}$ sont exactement définies sur les attributs du séparateur connectant la clique C_i à la clique C_j.*
- *Ce séquençage de calculs suit un certain ordre le long du graphe.*

On peut représenter ce séquençage comme une série d'envois de messages le long des séparateurs de la forêt de jonction jusqu'à atteindre la clique X_3X_5 (voir la figure 2.11). On nommera cette façon de procéder la « phase de collecte ».

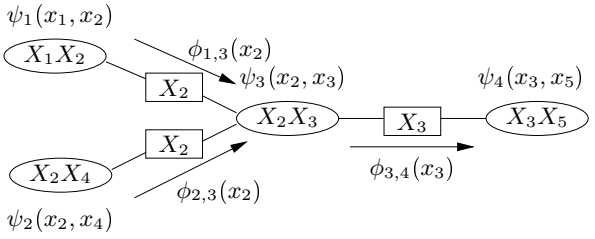


FIGURE 2.11 – Phase de collecte

Posons maintenant $x^* = (x_1^*, x_2^*, x_3^*, x_4^*, x_5^*)$. Le dernier calcul de la phase de collecte pour déterminer $u(x^*)$ consiste à maximiser complètement $\psi_4(x_3, x_5)$. Lors de cette maximisation, on peut déterminer (en effectuant un $\arg\max$*) les instanciations x_3^* et x_5^* composant l'alternative x^*, ce qui ne résoud pas complètement notre problème. Si on reprend la manière dont nous avons défini les différentes fonctions, on peut se rendre compte que $\psi_4(x_3^*, x_5^*) = u_4(x_3^*, x_5^*) + \phi_{3,4}(x_3^*) = u_4(x_3^*, x_5^*) + \max_{x_2 \in X_2} \ \psi_3(x_2, x_3^*)$, donc $x_2^* = \arg\max_{x_2 \in X_2} \ \psi_3(x_2, x_3^*)$. En poursuivant ce raisonnement, on obtient donc le sys-*

tème suivant :

$$\begin{cases} (x_3^*, x_5^*) & = \arg\max_{(x_3,x_5)\in X_3\times X_5} \psi_4(x_3, x_5) \\ x_2^* & = \arg\max_{x_2\in X_2} \psi_3(x_2, x_3^*) \\ x_1^* & = \arg\max_{x_1\in X_1} \psi_1(x_1, x_2^*) \\ x_4^* & = \arg\max_{x_4\in X_4} \psi_2(x_2^*, x_4) \end{cases}$$

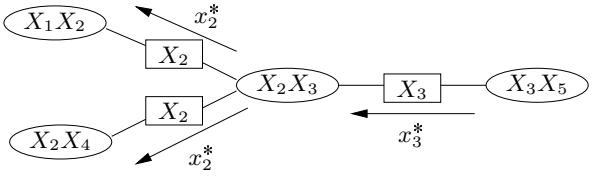


FIGURE 2.12 – Phase de diffusion

De la même façon que pour la phase de collecte, une telle opération peut s'interpréter de manière graphique (voir la figure 2.12) : la dernière clique de la phase de collecte permet d'obtenir une instanciation partielle de x^*. Il suffit ensuite de remonter les messages de la phase de collecte en sens inverse, en transmettant à chaque fois la projection de l'instanciation partielle sur le séparateur parcouru. Avec le message transitant et la fonction ψ_i calculée pendant la phase de collecte, il est possible de reconstituer au fur et à mesure l'intégralité de l'alternative optimale x^*. Cette seconde phase se nomme la « phase de diffusion ».

En effectuant les calculs des fonctions intermédiaires (voir figure 2.13), on aura $(x_3^*, x_5^*) = (x_{32}, x_{52})$ (puisque 42 maximise toutes les autres valeurs de ψ_4), or $\phi_{3,4}(x_3^*) = 31 = \psi_3(x_2^*, x_3^*)$, donc $x_2^* = x_{23}$. De la même façon $\phi_{1,3}(x_2^*) = 15 = \psi_1(x_1^*, x_2^*)$ et $\phi_{2,3}(x_2^*) = 3 = \psi_1(x_2^*, x_4^*)$, donc $x_1^* = x_{11}$ et $x_4^* = x_{42}$. On en déduit donc que l'alternative optimale est $x^* = (x_{11}, x_{23}, x_{32}, x_{42}, x_{52})$.

X_2	
x_{21}	7
x_{22}	10
x_{23}	15

(a) $\phi_{1,3}(x_2)$

X_2	
x_{21}	10
x_{22}	12
x_{23}	3

(b) $\phi_{2,3}(x_2)$

	X_2		
X_3	x_{21}	x_{22}	x_{23}
x_{31}	23	27	27
x_{32}	25	30	31

(c) $\psi_3(x_2, x_3)$

X_3	
x_{31}	27
x_{32}	31

(d) $\phi_{3,4}(x_3)$

	X_3	
X_5	x_{31}	x_{32}
x_{51}	28	39
x_{52}	37	42
x_{53}	32	33

(e) $\psi_4(x_3, x_5)$

FIGURE 2.13 – Calcul des fonctions intermédiaires

Cet exemple nous a permis de montrer qu'un raisonnement purement algébrique pouvait s'interpréter sur la structure du réseau GAI. Une telle approche est démontrable en appliquant l'axiomatique de la programmation dynamique non sérielle (voir (Shenoy et Shafer, 1990)) sur ces sous-fonctions d'utilité avec les opérateurs de maximisation et d'addition appropriés.

Pour formaliser les techniques utilisées dans l'exemple, il est nécessaire de définir une clique au sein de l'arbre de jonction, que l'on nommera la clique racine. Le rôle de cette clique est de faire le pont entre la phase de collecte et la phase de diffusion, tous les messages de la phase de collecte seront orientés vers cette clique, et elle sera à l'origine du premier message de la phase de diffusion (le réseau ayant une structure d'arbre, il

existe une unique chaîne reliant une clique quelconque à cette clique. Dans l'exemple 23, il s'agissait de la clique X_3X_5). L'indice de cette clique est transmise dans les deux arguments de l'algorithme de collecte (voir algorithme 5).

La seule différence entre l'éxécution sur l'exemple et cet algorithme est la création à la ligne 10 de la fonction $ArgMaxTab_{act,succ}$. Son unique but est d'optimiser les calculs durant la phase de diffusion, ainsi il n'y a plus besoin de parcourir deux fois les hypermatrices (une fois pour calculer un max, l'autre pour un arg max) : toutes les informations auront déjà été calculées pendant la phase de collecte, et on aura donc d'une part une fonction ϕ qui, à une instanciation de séparateur, renverra la maximisation de la sous-utilité, et d'autre part une fonction $ArgMaxTab$ qui, à une instanciation de séparateur, renverra l'instanciation n'appartenant pas au séparateur ayant servi à obtenir cette valeur.

Remarque 14 *Pour faciliter la lecture des algorithmes présentés dans cette thèse, nous utiliserons des opérateurs directement sur les hypermatrices.*

Ainsi, étant donnée une fonction $\psi : X_D \to \mathbb{R}$ et une fonction $\phi : X_E \to \mathbb{R}$, la valeur de retour de $\psi + \phi$ est une fonction $f : X_{D\cup E} \to \mathbb{R}$ vérifiant $\forall x \in X_{D\cup E}\ f(x) = \psi(x_D) + \phi(x_E)$.

De la même façon, pour des opérateurs unaires, étant donnée une fonction $\psi : X_E \to \mathbb{R}$, $\max_{X_E} \psi$ correspond à la valeur maximale contenue dans la table représentant ψ (on maximise sur tout le domaine de définition). Tandis que, pour $S \subset E$, l'expression $\max_{X_S} \psi$ s'évalue en une fonction f, définie sur $X_{E\setminus S}$ (les attributs n'ayant pas été pris en compte par la maximisation), et vérifiant $\forall x \in X_{E\setminus S}\ f(x) = \max_{y\in X_S} \psi(x, y)$. Il en est de même pour l'expression $\arg\max_{X_S} \psi$, elle s'évalue en une fonction g définie sur $X_{E\setminus S}$ et vérifiant $\forall x \in X_{E\setminus S}\ g(x) = \arg\max_{y\in X_S} \psi(x, y)$.

Par exemple, étant donnée une fonction $u : X_1 \times X_2 \to \mathbb{R}$ avec $X_1 = \{x_{11}, x_{12}\}$ et $X_2 = \{x_{21}, x_{22}\}$ et vérifiant $u(x_{11}, x_{21}) = 27$, $u(x_{12}, x_{21}) = 1664$, $u(x_{11}, x_{22}) = 51$ et $u(x_{12}, x_{22}) = 33$. Si $v = \max_{X_1} u$, alors on aura $v(x_{21}) = \max\{27, 1664\} = 1664$ et $v(x_{22}) = \max\{51, 33\} = 51$. Si $v = \max_{X_1\times X_2} u$, comme u est définie sur $X_1 \times X_2$, v est un scalaire et on aura $v = \max\{27, 1664, 51, 33\} = 1664$.

Considérons une fonction ψ définie sur $X_1 \times X_2$ (avec $X_1 = \{x_{11}, x_{12}\}$ et $X_2 = \{x_{21}, x_{22}\}$) et telle que $\psi(x_{11}, x_{21}) = 3$, $\psi(x_{11}, x_{22}) = 2$, $\psi(x_{12}, x_{21}) = 1$ et $\psi(x_{12}, x_{22}) = 4$. Notons $ArgMaxTab = \arg\max_{X_1} \psi$, alors $ArgMaxTab$ sera défini sur X_2 et on aura $ArgMaxTab(x_{21}) = x_{11}$ (étant donné que $\psi(x_{11}, x_{21}) > \psi(x_{12}, x_{21})$) et $ArgMaxTab(x_{22}) = x_{12}$ (étant donné que $\psi(x_{12}, x_{22}) > \psi(x_{11}, x_{22})$).

Exemple 24 *En appliquant le pseudo-code à l'exemple précédent (les cliques C_i sont les cliques contenant la sous-fonction d'utilité u_i), et en supposant que la boucle Pour parcourt les voisins dans l'ordre croissant des indices des cliques, on obtiendra la séquence de calculs suivante :*

Appel à `Collecte(4, 4)`
  Créer $\psi_4 \leftarrow u_4$
  Appel à `Collecte(3, 4)`
    Créer $\psi_3 \leftarrow u_3$
    Appel à `Collecte(1, 3)`
      Créer $\psi_1 \leftarrow u_1$

Algorithme 5 : Collecte

Entrées :
act : indice d'une clique,
succ : indice d'une clique

Créer $\psi_{act} \leftarrow u_{act}$;
pour chaque *clique* C_j *voisine de* C_{act} **faire**
  si $j \neq succ$ **alors**
    `Collecte(`j, act`)`;
    $\psi_{act} \leftarrow \psi_{act} + \phi_{j,act}$;
  fin
fin
si $act \neq succ$ **alors**
  Créer $\phi_{act,succ} \leftarrow \max_{C_{act} \setminus S_{act,succ}} \psi_{act}$;
  Créer $ArgMaxTab_{act,succ} \leftarrow \arg\max_{C_{act} \setminus S_{act,succ}} \psi_{act}$;
fin

Créer $\phi_{1,3} \leftarrow \max_{X_1} \psi_1$
Créer $ArgMaxTab_{1,3} \leftarrow \arg\max_{X_1} \psi_1$
Retour dans `Collecte(3, 4)`
$\psi_3 \leftarrow \psi_3 + \phi_{1,3}$
Appel à `Collecte(2, 3)`
Créer $\psi_2 \leftarrow u_2$
Créer $\phi_{2,3} \leftarrow \max_{X_4} \psi_2$
Créer $ArgMaxTab_{2,3} \leftarrow \arg\max_{X_4} \psi_2$
Retour dans `Collecte(3, 4)`
$\psi_3 \leftarrow \psi_3 + \phi_{2,3}$
Créer $\phi_{3,4} \leftarrow \max_{X_2} \psi_3$
Créer $ArgMaxTab_{3,4} \leftarrow \arg\max_{X_2} \psi_3$
Retour dans `Collecte(4, 4)`
$\psi_4 \leftarrow \psi_4 + \phi_{3,4}$

Il est à noter que l'algorithme de collecte définie une hiérarchie sur les cliques d'un arbre de jonction : l'ordre de collecte. De cet ordre, on peut déduire deux notions importantes sur les cliques : les prédecesseurs et le successeur (définition 66). Ces notions seront exploitées par la suite dans d'autres algorithmes que nous présenterons.

Définition 66 (Successeur, prédecesseurs) *Étant donné un algorithme de collecte sur un arbre de jonction, on définit le successeur d'une clique C_i comme étant la clique C_j reçevant un message $\phi_{i,j}$ (on note $Succ(i) = j$, et on pose, si C_{root} est la clique racine, $Succ(root) = root$). De la même façon, les prédecesseurs d'une clique C_j sont les cliques C_i ayant engendré un message $\phi_{i,j}$ vers C_j (on note $Pred(j)$ la fonction renvoyant l'ensemble des indices des cliques ayant produit ces messages).*

La phase de diffusion utilise les $ArgMaxTab$ calculés dans la phase de collecte. Sa structure est la même que l'algorithme de collecte : l'indice de la clique racine est passé

en double argument de la fonction, et les appels récursifs correspondent à un parcours en profondeur de graphe (voir algorithme 6).

Algorithme 6 : Diffusion

Entrées :
act : indice d'une clique,
succ : indice d'une clique,
x^* : instanciation des attributs du réseau

```
01 si act = succ alors
02 |  Instancier partiellement x* avec arg max_{X_{C_act}} ψ_act;
03 sinon
04 |  Instancier partiellement x* avec ArgMaxTab_{act,succ}(x*_{S_{act,succ}});
05 fin
06 pour chaque clique C_j voisine de C_act faire
07 |  si j ≠ succ alors
08 |  |  Diffusion(j, act, x*);
09 |  fin
10 fin
```

Le problème du choix optimal est donc résolu par un algorithme en deux phases. Il est intéressant de constater que la phase de collecte effectue les calculs coûteux en temps, tandis que la phase de diffusion réexploite ces calculs mais est bien plus rapide (voir propriété 22).

Propriété 22 (Complexité temporelle de l'algorithme) *La phase de collecte s'exécute en un temps de l'ordre de* $\Omega(\sum_{C_j \in C} \prod_{X_i \in C_j} |X_i|)$ *et* $O(|C| \max_{C_i \in C} \prod_{X_i \in C_j} |X_i|)$. *La phase de diffusion s'éxécute en un temps de l'ordre de* $O(|C| \max_{C_i \in C} |C_i|)$ *(Dechter, 2003).*

Preuve. Pour la phase de collecte, l'ordre de grandeur en Ω se démontre trivialement en observant que chaque valeur contenue dans une sous-utilité doit être parcourue au moins une fois dans l'algorithme. Pour la complexité en O, posons $K = \max_{C_i \in C} \prod_{X_i \in C_j} |X_i|$, alors les opérations des lignes 05, 09 et 10 sont en $O(K)$. À chaque appel récursif, elles doivent être éxécutées $O(d(C_{act}))$ fois (avec $d(C_{act})$ le nombre de voisins de la clique C_{act}). Donc une récursion a un coût en $O(d(C_{act})K)$. Comme on effectue une récursion pour chaque clique, et que le réseau a une structure d'arbre, on aura donc $O(\sum_{C_i \in C} d(C_i)K) = O(|C|K)$.
Pour la phase de diffusion, chaque appel récursif instanciera $O(C_{act}) \subseteq O(\max_{C_i \in C} |C_i|)$ attributs. On effectue un appel récursif par cliques, on aura donc $O(|C| \max_{C_i \in C} |C_i|)$. ■

Supposons maintenant que le réseau GAI ait une structure de forêt de jonction. Ses composantes connexes sont donc des arbres de jonction, et deux attributs contenus dans deux arbres différents sont donc totalement indépendants. L'algorithme de choix optimal se généralise donc facilement : il suffit d'éxécuter l'algorithme indépendamment sur chaque arbre du réseau, puis de recomposer chaque sous-alternative obtenue en une alternative globale.

2.3.2 Rangement des meilleures alternatives

Le problème du rangement des k meilleures alternatives selon la fonction d'utilité u consiste à déterminer une suite d'alternatives x^i vérifiant :

$$\forall i \in \{1, \ldots, k\}\ x^i = \arg\max_{x \in \mathcal{X}^i} u(x) \quad \text{avec} \quad \mathcal{X}^i = \begin{cases} \mathcal{X} & \text{si } i = 1 \\ \mathcal{X}^{i-1} \backslash \{x^{i-1}\} & \text{sinon.} \end{cases}$$

x^1 sera donc l'alternative préférée du décideur (x^* dans la sous-section précédente), x^2 l'alternative préférée si enlève x^1 de l'espace des alternatives, et ainsi de suite. On aura donc $u(x^1) \geqslant u(x^2) \geqslant \ldots \geqslant u(x^k)$. Gonzales et al. (2008) ont montré que ce problème peut être résolu par un algorithme proche de celui utilisé pour résoudre le problème du choix optimal en se basant sur les décompositions d'espace de Nilsson (1998).

Exemple 25 *Reprenons le réseau GAI de l'exemple 23. Après une phase de collecte et une phase de diffusion, on aura calculé l'alternative préférée* $x^1 = x^* = (x_{11}, x_{23}, x_{32}, x_{42}, x_{52})$. *La détermination du second élément préféré doit donc s'effectuer dans l'espace* $\mathcal{X}\backslash\{x^1\}$, *ou écrit d'une autre façon :*

$$X_1 \times X_2 \times X_3 \times X_4 \times X_5 \backslash \{x_{11}, x_{23}, x_{32}, x_{42}, x_{52}\}$$

Posons $x^2 = (x_1^2, x_2^2, x_3^2, x_4^2, x_5^2)$, *cette alternative devra donc vérifier une des conditions suivantes :*

$$\begin{cases} & (x_3^2, x_5^2) \neq (x_3^*, x_5^*) & \\ ou & \left[(x_3^2, x_5^2) = (x_3^*, x_5^*) \right. & \left. et\ (x_2^2) \neq (x_2^*)\right] \\ ou & \left[(x_2^2, x_3^2, x_5^2) = (x_2^*, x_3^*, x_5^*) \right. & \left. et\ (x_4^2) \neq (x_4^*)\right] \\ ou & \left[(x_2^2, x_3^2, x_4^2, x_5^2) = (x_2^*, x_3^*, x_4^*, x_5^*) \right. & \left. et\ (x_1^2) \neq (x_1^*)\right] \end{cases}$$

Cette réécriture exploite la structure d'arbre de jonction de l'exemple, et l'ordre de passage dans les cliques utilisé durant la phase de collecte. Nous avions utilisé l'ordre X_1X_2, X_2X_4, X_2X_3 *et* X_3X_5. *Cette décomposition de l'espace est liée à cet ordre en sens inverse : nous fixons d'abord que* x^2 *doit différer de* x^1 *selon les attributs* X_3 *et/ou* X_5 *(clique* X_3X_5*). Puis les instanciations de* X_3 *et* X_5 *sont fixées à celles contenues dans* x^1, *mais la différence se situe en* X_2 *(clique* X_2X_3, *le message envoyé à partir de* X_2 *dans la phase de collecte a transité sur le séparateur* X_3*). Et ainsi de suite... Avec une telle écriture, il est possible de déterminer facilement, pour chacun des quatre ensembles d'alternatives, l'alternative préférée. Prenons le premier ensemble (*$(x_3^2, x_5^2) \neq (x_3^*, x_5^*)$*), l'alternative préférée de cette restriction sera la seconde meilleure alternative de la fonction* ψ_4. *Pour le second ensemble (*$(x_2^2, x_3^2, x_5^2) = (x_2^*, x_3^*, x_5^*)$ *et* $(x_4^2) \neq (x_4^*)$*), il faudra déterminer la seconde meilleure alternative de* ϕ_3 *à valeur de* X_3 *fixée. En procédant ainsi sur les quatre ensembles, on obtiendra quatre alternatives, et il suffira de prendre celle de plus forte utilité.*

Supposons maintenant que nous ayons déterminé la seconde meilleure alternative. La recherche d'une troisième alternative préférée x^3 *se fera de la même façon : on devra chercher dans l'espace* $\mathcal{X}\backslash\{x^1, x^2\}$. *Toutefois, supposons que la seconde meilleure solution soit issue de la décomposition de l'espace* $(x_3^2, x_5^2) = (x_3^*, x_5^*)$ *et* $(x_2^2) \neq (x_2^*)$, *alors générer une décomposition de la même façon n'a pas de sens sur la clique* X_3X_5 *étant donné que nous aurons* $x_3^1 = x_3^2$ *et* $x_5^1 = x_5^2$. *Il suffira donc de suivre l'ordre de collecte en sens*

inverse à partir de la clique ayant généré la décomposition dont est issue x^2 *(*X_2X_3 *en l'occurence), on obtiendra donc :*

$$\left\{\begin{array}{lll} & (x_3^3, x_5^3) \neq (x_3^1, x_5^1) & \\ ou & \left[(x_3^3, x_5^3) = (x_3^1, x_5^1) \right. & et\ (x_2^3) \notin \{(x_2^1), (x_2^2)\}\left.\right] \\ ou & \left[(x_2^3, x_3^3, x_5^3) = (x_2^1, x_3^1, x_5^1) \right. & et\ (x_4^3) \neq (x_4^1)\left.\right] \\ ou & \left[(x_2^3, x_3^3, x_5^3) = (x_2^2, x_3^2, x_5^2) \right. & et\ (x_4^3) \neq (x_4^2)\left.\right] \\ ou & \left[(x_2^3, x_3^3, x_4^3, x_5^3) = (x_2^1, x_3^1, x_4^1, x_5^1) \right. & et\ (x_1^3) \neq (x_1^1)\left.\right] \\ ou & \left[(x_2^3, x_3^3, x_4^3, x_5^3) = (x_2^2, x_3^2, x_4^2, x_5^2) \right. & et\ (x_1^3) \neq (x_1^2)\left.\right] \end{array}\right.$$

En analysant cet exemple, nous pouvons nous apercevoir que trois aspects sont améliorables :

- Au niveau de l'écriture des ensembles restreignant l'espace des alternatives, la recherche d'une alternative partielle revient en réalité à chercher la i-ème meilleure alternative d'une fonction ψ à valeur de séparateur fixé.
- De plus, il est nécessaire d'obtenir en un temps de calcul rapide une alternative complète à partir d'une alternative partielle. Dans le problème du choix optimal, ce problème correspondait à la phase de diffusion et était résolu efficacement par la création du tableau $ArgMaxTab$ durant la phase de collecte, qui était ensuite réexploité pour diffuser au sein du réseau des instanciations partielles et recomposer de proche en proche l'instanciation globale.
- Une fois que des solutions sont assignées à chaque ensemble, il est nécessaire de les comparer entre elles. Une alternative sera celle qui sera élue pour être la k-ième meilleure alternative, les autres sont des candidats potentiels pour la $(k+1)$-ième meilleure alternative. Plutôt que de recalculer à chaque fois ces solutions potentielles, il est possible de représenter cet ensemble de candidats sous la forme d'un ensemble dynamique qui vérifiera les propriétés de tas.

En se basant sur ces constatations, il est possible d'élaborer un algorithme basé sur trois aspects : une collecte permettant de stocker efficacement la i-ième alternative partielle préférée d'une clique (à valeur de séparateur fixé), une diffusion pouvant recomposer une alternative à partir de n'importe quelle clique, et un moteur de rangement permettant de générer les candidats portentiels à la k-ième meilleure alternative (voir algorithme 7).

Remarque 15 *Au sein de l'algorithme 7, nous noterons* $Table(X_D, \mathbb{R})$ *l'ensemble des tables indexées par* X_D *et à valeur dans* $\mathbb{R}$*. Ainsi, définir* $Toto : X_E \rightarrow Table(X_D, \mathbb{R})$ *revient à écrire que la variable* $Toto$ *est une table indexée par* X_E *dont le contenu* $(Toto(x_E))$ *est lui-même une table, indexée par* X_D *et contenant des nombres réels.*

Pour l'algorithme de collecte, nous nous servirons d'une fonction Tri de la façon suivante : Tri(ψ_{act}, x) (ligne 11) avec x un argument partiel de ψ_{act} renvoie un tableau T de couples tel que $\forall i\ T[i] = (x', a)$, où $\psi_{act}(x, x') = a$ et a est la i-ème meilleure valeur de ψ_{act} pour x fixé. Si aucune valeur de x n'est précisée (ligne 16 : Tri(ψ_{act})), le tri s'effectuera sur l'intégralité de l'hypermatrice. Cet algorithme est donc similaire à l'algorithme de collecte, seul le stockage des alternatives préférées est modifié.

L'algorithme de diffusion (algorithme 8) est donc modifié en conséquence de façon à exploiter le changement de structure, et être capable de diffuser à partir de n'importe quelle clique. Si la clique à partir de laquelle la diffusion commence est la clique racine,

Algorithme 7 : RankingCollecte

Entrées :
act : indice d'une clique,
$succ$: indice d'une clique

créer $\psi_{act} \leftarrow u_{act}$;
pour chaque *clique* C_j *voisine de* C_{act} **faire**
  si $j \neq succ$ **alors**
    `RankingCollecte(`j, act`)`;
    $\psi_{act} \leftarrow \psi_{act} + \phi_{j,act}$;
  fin
fin
si $act \neq succ$ **alors**
  créer $ArgMaxTab_{act,succ} : X_{S_{act,succ}} \rightarrow \text{Table}(X_{C_{act}}, \mathbb{R})$;
  pour chaque $x \in X_{S_{act,succ}}$ **faire**
    $ArgMaxTab_{act,succ}(x) \leftarrow \text{Tri}(\psi_{act}, x)$;
  fin
  créer $\phi_{act,succ} \leftarrow \max_{C_{act} \setminus S_{act,succ}} \psi_{act}$;
sinon
  créer $ArgMaxRacine : \text{Table}(X_{C_{act}}, \mathbb{R})$;
  $ArgMaxRacine \leftarrow \text{Tri}(\psi_{act})$;
fin

Algorithme 8 : RankingDiffusion

Entrées :
act : indice d'une clique,
$succ$: indice d'une clique,
x : instanciation des attributs du réseau (éventuellement partiellement remplie),
i : entier

si $act = succ$ **alors**
  instancier partiellement x avec $ArgMaxRacine[i]$;
sinon
  instancier partiellement x avec $ArgMaxTab_{act,succ}(x_{S_{actsucc}})[i]$;
fin
pour chaque *clique* C_j *voisine de* C_{act} **faire**
  si $j \neq succ$ **alors**
    `RankingDiffusion(`j, act, x, 1`)`;
  fin
fin

on aura $act = succ$ lors du premier appel de la fonction, sinon on utilisera dans succ le successeur de act dans la phase de collecte.

Avec ces deux briques de base, il est donc possible d'écrire l'algorithme de rangement des k meilleures alternatives (algorithme 9).

Remarque 16 *Il est important de noter dans l'algorithme 9 que la fonction Succ définie à la page 63 est utilisée, mais que la notion de cliques C_z antérieures à C_C ne correspond pas à la notion de prédecesseurs introduite dans cette même page, mais des cliques C_z ayant produit leur message $\phi_{z,Succ(z)}$ avant C_c, c'est-à-dire les cliques C_z dont l'appel résursif à* `RankingCollecte(`z, $Succ(z)$`)` *s'est achevé avant l'appel récursif à* `RankingCollecte(`c, $Succ(c)$`)`.

Algorithme 9 : Rangement

Entrée : k : entier
1 **créer** $KMeilleur$: Tableau de taille k;
2 **créer** x : instanciation des attributs;
3 **créer** $Candidats$: Tas;
4 **créer** $j \leftarrow 1$;
5 **choisir** C_{root} : une clique du réseau;
6 `RankingCollecte(`$root$, $root$`)`;
7 `RankingDiffusion(`$root$, $root$, x, 1`)`;
8 **insérer** dans $Candidats$: $(x, root, 1)$ indexé par $u(x)$;
9 **tant que** $j \leqslant k$ **faire**
10 $(x, c, a) \leftarrow$ Extraire élément de clé maximale de $Candidats$;
11 $KMeilleur[j] \leftarrow x$;
12 **créer** $y \leftarrow x$;
13 `RankingDiffusion(`c, $Succ(c)$, y, $a+1$`)` (si $a+1$ possible);
14 **insérer** dans $Candidats$: $(y, c, a+1)$ indexé par $u(y)$;
15 **pour chaque** *clique C_z, antérieure à C_c dans la collecte* **faire**
16 **créer** $y \leftarrow x$;
17 `RankingDiffusion(`z, $Succ(z)$, y, 1`)`;
18 **insérer** dans $Candidats$: $(y, z, 1)$ indexé par $u(y)$;
19 **fin**
20 $j \leftarrow j + 1$;
21 **fin**
22 **retourner** $KMeilleur$;

Dans cet algorithme, chaque candidat est représenté par une association (z, x, i) contenant la clique C_z d'où est issu le candidat (qui représente, dans les restrictions d'ensembles, les attributs sur lesquels porte la restriction), une instanciation x des attributs du problème (la solution de cette restriction), et un entier i correspondant à la recherche de la i-ème meilleure valeur (à valeur de séparateur fixée). Ces associations sont stockées dans le tas $Candidats$, indexé par l'utilité de x. A chaque itération, on choisit le candidat d'utilité maximale, et on génère de nouveaux candidats en fonction des nouvelles restrictions du problème.

Remarque 17 *Le problème du rangement est écrit sous la forme de la recherche des k meilleures alternatives. En pratique, on peut avoir besoin pour certains problèmes, d'itérer pour obtenir les k meilleures alternatives sans connaître la valeur de k, mais en ayant un critère d'arrêt des itérations (nous en verrons une illustration en décision multicritère). Il suffit de modifier l'algorithme de rangement à la ligne 09 pour insérer son critère de fin des itérations. En procédant ainsi, on peut voir l'algorithme sous deux aspects : l'initialisation, et la recherche d'une nouvelle alternative, et l'algorithme peut se représenter symboliquement par l'algorithme 10 où l'initialisation correspond aux lignes 01 à 08 de l'algorithme de rangement, et l'extraction d'une nouvelle solution correspond aux lignes 10 à 20.*

Algorithme 10 : RangementSymbolique

```
1 Initialiser(MoteurRangement);
2 tant que Critère non atteint faire
3 |  ExtraireNouvelleSolution(MoteurRangement);
4 fin
```

Du point de vue du temps de calcul, les tris effectués dans la phase de collecte rendent plus coûteuse cette phase par rapport à l'algorithme de détermination du choix optimal, mais la complexité asymptotique de la phase de diffusion reste inchangée en notation O. L'algorithme de rangement effectue donc une phase de collecte plus coûteuse, et un grand nombre des phases de diffusion. D'une façon générale, la complexité temporelle de la recherche des k meilleures alternatives est (Gonzales et al., 2007) :

Propriété 23 (Complexité temporelle du rangement) *Le temps de calcul de l'algorithme de rangement des k meilleures alternatives d'un réseau GAI est en* $O(q\bar{c} + qk + qlog(qk) + q\bar{c}log(\bar{c}))$ *avec q le nombre de cliques et* $\bar{c}$ *la moyenne du nombre de valeurs stockées dans une hypermatrice.*

Dans le cas où le réseau GAI a une structure de forêt, l'algorithme est inchangé : il suffit de définir un ordre global de collecte par concaténation des ordres de collectes obtenus sur chaque arbre. En procédant ainsi, l'algorithme fonctionnera comme si la forêt de jonction était un arbre, en reliant implicitement les différents arbres contenus dans la forêt par un séparateur vide.

2.4 Utilisation des réseaux GAI en décision multicritère

Dans le chapitre 1, nous avons eu un aperçu des problématiques posées par la présence de plusieurs critères, et nous avons formulé des problèmes à résoudre sans toutefois proposer d'algorithme pour le faire. Dans la littérature associée aux réseaux GAI, Gonzales et al. (2008) ont proposé une approche pour déterminer la solution agrégée optimale en décision multiattribut et multicritère. Nous allons présenter dans cette section l'approche développée, en définissant dans un premier temps ce qu'est un réseau GAI scalarisé (section 2.4.1), puis en montrant que la résolution du problème peut être vue comme un rangement du réseau GAI scalarisé (section 2.4.2).

Il est à noter que nous ne présenterons pour l'instant aucune solution pour la détermination des frontières de Pareto ou de Lorenz, mais nous aborderons ces sujets dans de nouveaux algorithmes au sein des chapitres 4 et 5.

2.4.1 Réseau GAI scalarisé

Nous allons dans un premier temps reprendre les notions introduites pour définir l'énoncé du problème abordé. En utilisant le même formalisme que dans le chapitre 1, nous considérons un espace des alternatives composé de n attributs : $\mathcal{X} = \prod_{i=1}^{n} X_i$. La prise de décision fait intervenir m critères, et on associe à chaque critère j une fonction d'utilité $u^j : \mathcal{X} \to \mathbb{R}$. Notons $u : \mathcal{X} \to \mathbb{R}^m$ la fonction d'utilité multicritère associée au problème. On a donc $\forall x \in \mathcal{X}\ u(x) = (u^1(x), \ldots, u^m(x))$ et $\mathcal{U}$ l'espace des critères associé. On suppose que chaque fonction d'utilité u^j est GAI-décomposable selon le réseau GAI $(G^j, \{u_z^j\})$. Nous allons considérer un agrégateur $A : \mathcal{U} \to \mathbb{R}$, et nous cherchons à déterminer l'alternative $x^* \in \mathcal{X}$ engendrant la solution agrégée selon A optimale :

$$x^* = \arg\max_{x \in \mathcal{X}} A(u(x)).$$

La principale difficulté du problème vient du fait que les trois agrégateurs que nous avons présentés dans le chapitre 1 ne sont pas linéaires. L'approche proposée par Gonzales et al. (2008) consiste à construire dans un premier temps le réseau GAI scalarisé du problème : il s'agit d'un réseau GAI englobant toutes les dépendances entre attributs de chaque critère, et dont la valeur d'utilité associée à chaque alternative est une borne supérieure de l'agrégation de tous les critères.

Définition 67 (Réseau GAI scalarisé) *Soit $\mathcal{X} = \prod_{i=1}^{n} X_i$ un espace d'alternatives, $u^1, \ldots, u^m$ des fonctions d'utilité telles que, à chaque fonction u^j, on puisse associer une représentation en réseau GAI $R^j = (F^j, \{u_z^j\})$, et A un agrégateur défini sur l'espace des critères $\mathcal{U}$ engendré par la fonction d'utilité multicritère issue de $u^1, \ldots, u^m$.*
On dit que le réseau GAI $\hat{R} = (\hat{F}, \{\hat{u}_z\})$ est un réseau GAI scalarisé de $R^1, \ldots, R^m$ selon A si, et seulement si :
- *$\hat{F}$ est une agrégation des forêts de jonction $F^1, \ldots, F^m$,*
- *$\forall x \in \mathcal{X}\ \hat{u}(x) \geqslant A(u(x))$.*

D'un point de vue structurel, exploiter les indépendances d'un réseau GAI scalarisé revient à exploiter des indépendances présentes dans tous les critères. Nous avons vu dans la section 2.2.4 un algorithme permettant de générer automatiquement la forêt de jonction d'un réseau GAI scalarisé. Le principal problème dans la génération d'un réseau GAI scalarisé est donc la construction des sous-fonctions d'utilité, de façon à ce que l'utilité globale borne supérieurement l'utilité agrégée pour toutes les alternatives. Ces sous-utilités dépendent fortement de l'agrégateur choisi. Il est donc nécessaire d'étudier au cas par cas la génération de ces sous-fonctions.

Pour l'agrégateur OWA, Galand (2008) a démontré que, dans le cadre d'un jeu de poids décroissant, il était possible de borner la valeur agrégée par une moyenne (voir la propriété 24, qui est un cas particulier de l'inégalité de la somme de Tchebycheff).

Propriété 24 (Borne supérieure d'un OWA) *Soit $\mathcal{U} \subseteq \mathbb{R}^m$ un espace de critères et $\mu_1, \ldots, \mu_m$ une famille de réels de l'intervale $[0,1]$ vérifiant $\mu_1 \geqslant \ldots \geqslant \mu_m$ et $\sum_{j=1}^m \mu_j = 1$, alors $\forall v = (v^1, \ldots, v^m) \in \mathcal{U}$ on aura :*

$$OWA_\mu(v) \leqslant \sum_{j=1}^{m} \mu_j v^j.$$

Cette borne est particulièrement intéressante, étant donné que les vecteurs v sont issus de la somme des sous-fonctions d'utilité. On aura donc une décomposition de la forme $\sum_{j=1}^m \mu_j \sum_z u_z^j(x_{C_z}) = \sum_{j=1}^m \sum_z \mu_j u_z^j(x_{C_z})$. Il suffit donc de pondérer chaque sous-fonction d'utilité de chaque réseau GAI par un coefficient μ_j, et d'intégrer ces sous-fonctions par sommation dans la clique leur correspondant.

Pour les autres agrégateurs, le raisonnement sera similaire : se ramener à une borne facilement décomposable par sommation. Ainsi, l'intégrale de Choquet peut être bornée en utilisant une capacité additive issue du noyau de la capacité convexe (propriété 25, démontrée par Schmeidler (1986)).

Définition 68 (Noyau d'une capacité convexe) *Soit $\mu : 2^E \to [0,1]$ une capacité convexe, et $\bar{\mu}$ sa capacité duale (concave), le noyau (core) de μ est défini par :*

$$core(\mu) = \{\lambda \in \mathcal{L} / \forall A \in 2^E \ \mu(A) \leqslant \lambda(A) \leqslant \bar{\mu}(A)\}$$

Avec $\mathcal{L}$ l'ensemble des capacités additives définies sur 2^E.

Propriété 25 (Borne supérieure d'une intégrale de Choquet) *Soit $\mu : 2^E \to [0,1]$ une capacité convexe, et λ une capacité additive telle que $\lambda \in core(\mu)$, alors on a :*

$$\forall v \in \mathcal{U} \ Choquet_\mu(v) \leqslant \sum_{j=1}^{m} \lambda(\{j\}) v^j$$

Il est à noter que Shapley (1971) a montré que si μ est une capacité convexe, le noyau de μ est non vide, et donc une telle capacité additive λ existe. Pour déterminer une telle capacité, on peut se baser sur les valeurs de Shapley, ou bien encore sur les travaux de Jaffray (1995) permettant d'obtenir une capacité du noyau d'entropie maximale (voir l'algorithme 11, la capacité λ renvoyée est entièrement définie par ses singletons, puisqu'elle est additive).

La borne supérieure de la norme de Tchebycheff s'obtient en remarquant que $\max_j f^j(v) \geqslant \frac{1}{m} \sum_j f^j(v)$. En remplaçant les fonctions f^j par la transformation faite dans le calcul de la norme de Tchebycheff sur chaque critère, on obtient la propriété 26.

Propriété 26 (Borne supérieure d'une norme de Tchebycheff) *Soit $\mathcal{U} \subseteq \mathbb{R}^m$ un espace des critères, de point idéal I et de point Nadir N, alors on aura :*

$$\forall v \in \mathcal{U} \ Tcheby(v) \leqslant -\frac{1}{m} \sum_{j=1}^{m} \frac{I^j - v^j}{I^j - N^j}.$$

Pour résumer, il est donc aisé d'obtenir un réseau scalaire à partir d'un ensemble de réseaux GAI en utilisant l'agrégation structurelle classique, puis en intégrant par sommation les sous-fonctions d'utilité modifiées par la propriété adéquate.

Algorithme 11 : Construction d'une capacité additive d'entropie maximale

Entrée : μ : capacité convexe définie sur 2^E

soit $\bar{\mu}$ la capacité duale de μ;
$A \leftarrow \varnothing$;
$B \leftarrow \varnothing$;
tant que $B \neq E$ **faire**
  $A \leftarrow \arg\min\{\frac{\bar{\mu}(B \cup F) - \bar{\mu}(B)}{|F|} / F \subseteq N \backslash B \text{ et } F \neq \varnothing\}$;
  pour tous les $j \in A$ **faire**
    $\lambda(\{j\}) \leftarrow \frac{\bar{\mu}(B \cup A) - \bar{\mu}(B)}{|A|}$;
  fin
  $B \leftarrow B \cup A$;
fin
retourner λ;

2.4.2 Détermination du choix agrégé optimal par rangement

Nous avons vu dans la sous-section précédente comment obtenir un réseau GAI scalaire pour représenter un ensemble de réseaux GAI formant un problème multicritère. Cette sous-section s'intéresse à l'exploitation d'un tel réseau pour déterminer de manière exacte la solution agrégée optimale.

Il est possible de ramener le problème de détermination de la solution agrégée optimale en un problème de rangement du réseau scalaire en se basant sur deux constatations :

- Pour une alternative donnée $x \in \mathcal{X}$, il est aisé (et rapide) de reconstituer le vecteur d'utilité multicritère $v \in \mathcal{U}$ qui lui est associé : il suffit d'instancier chaque sous-fonction d'utilité de chaque réseau GAI représentant un critère : $v = (v^1, \ldots, v^m) = (u^1(x), \ldots, u^m(x)) = (\sum_z u^1_z(x), \ldots, \sum_z u^m_z(x))$.
- Soit $A : \mathcal{U} \rightarrow \mathbb{R}$ une fonction d'agrégation, et $\hat{u} : \mathcal{X} \rightarrow \mathbb{R}$ la fonction d'utilité du réseau GAI scalarisé. Par définition, on aura $A(u^1(x), \ldots, u^m(x)) \leqslant \hat{u}(x)$. Notons $x^1, \ldots, x^k$ les k-meilleures alternatives selon $\hat{u}$, et notons $a_i = \max_{i \in \{1,\ldots,k\}} A(u^1(x^i), \ldots, u^m(x^i))$. Supposons qu'à l'indice k nous ayons $a_k > \hat{u}(x^k)$. On aura donc $\forall z > k,\ a_k > \hat{u}(x^k) \geqslant \hat{u}(x^z) \geqslant A(u^1(x^z), \ldots, u^m(x^z))$. On aura donc $a_k = \max_{x \in \mathcal{X}} A(u^1(x), \ldots, u^m(x))$. a_k est la solution optimale agrégée.

L'algorithme consiste donc à construire le réseau GAI scalarisé, puis à itérer l'algorithme de rangement jusqu'à ce que la valeur maximale obtenue par agrégation sur les alternatives rangées soit plus grande que l'utilité scalarisée de l'alternative courante. On aura donc l'algorithme 12.

Nous avons pu voir un aperçu des problématiques qui se posent en décision multicritères, et nous avons présenté un algorithme mêlant décision multicritère et décision multiattribut pour résoudre la recherche de la solution agrégée optimale. Dans les conclusions du chapitre 3, nous présenterons une manière d'améliorer cet algorithme dans un contexte de réseaux GAI de forte treewidth, puis dans le chapitre 4, nous introduirons un nouvel algorithme capable de résoudre le problème de la détermination de la frontière de Pareto. Finalement, dans le chapitre 5, nous présenterons un algorithme heuristique général capable de résoudre tous les problèmes d'agrégation et de détermination de frontière

Algorithme 12 : AgregationOptimale

Entrées :
A : agrégateur,
$\{R_j\}$: m réseaux GAI représentant l'utilité u^j

```
1  R̂ ← réseau GAI scalarisé des {R_j} (d'utilité û);
2  Ranking ← MoteurRangement(R̂);
3  x^1 ← Ranking.solutionSuivante();
4  AgregationMax ← A(u^1(x^1), ..., u^m(x^1));
5  BorneSup ← û(x^1);
6  MeilleureAlternative ← x^1;
7  tant que AgregationMax ⩽ BorneSup et Ranking.pasFini() faire
8  |  x^i ← Ranking.solutionSuivante();
9  |  BorneSup ← û(x^i);
10 |  AgregationCourante ← A(u^1(x^i), ..., u^m(x^i));
11 |  si AgregationCourante > AgregationMax alors
12 |  |  AgregationMax ← AgregationCourante ;
13 |  |  MeilleureAlternative ← x^i;
14 |  fin
15 fin
16 retourner MeilleureAlternative;
```

introduits dans ce chapitre.

Deuxième partie

Algorithmes d'exploitation de réseaux GAI

Chapitre 3

Exploitation des réseaux GAI de forte treewidth [1]

Résumé. *Nous avons pour l'instant étudié trois problèmes de la littérature des réseaux GAI : le choix optimal, le rangement et la solution agrégée optimale, et nous avons présenté des algorithmes résolvant ces trois problèmes. Au sein de cette section, nous nous situerons dans des situations où les réseaux GAI sont de forte treewidth. La complexité de ces réseaux a un impact fort sur les temps de calcul présentés jusqu'à présent, et nous proposerons une nouvelle approche pour résoudre ces trois problèmes dans ces nouvelles situations.*
L'algorithme en trois moteurs que nous proposerons sera dans un premier temps présenté pour résoudre uniquement le problème du choix optimal. Toutefois, nous verrons qu'il peut être généralisé et résoudre aussi bien le rangement que la détermination du choix agrégée optimal.
Finalement, nous étudierons les nouvelles perspectives ouvertes par cette approche, et nous présenterons de pistes exploitant la modularité de l'algorithme pour l'améliorer.

1. Travaux publiés dans Dubus et al. (2009a) et Dubus et al. (2009c)

Dans les chapitres précédents, nous avons présenté les notions de base pour la prise de décision multiattribut et multicritère dans le contexte des réseaux GAI, ainsi que les algorithmes de la littérature qui leur sont associés. Dans le cadre de ce chapitre, nous supposerons que la prise de décision ne fera intervenir qu'un unique critère, et nous étudierons une limitation forte des algorithmes présentés précédemment. Puis nous proposerons un nouvel algorithme contournant ce problème. Nous nous intéresserons dans un premier temps à un problème de détermination de l'alternative préférée du décideur, mais après présentation et étude expérimentale de cet algorithme, nous verrons en guise de conclusion comment il peut être étendu pour des problèmes de rangement et de détermination de la solution agrégée optimale dans les réseaux GAI scalarisés.

3.1 Élément critique et idée d'approche

3.1.1 L'élément critique : la treewidth

Nous avons, pour l'instant, étudié trois algorithmes pour résoudre trois problèmes différents. Le problème du choix optimal d'un décideur a été résolu en 2.3.1 par un algorithme en deux phases : la collecte qui exploitait la structure du réseau GAI pour effectuer un séquençage de calculs, et la diffusion qui permettait de recomposer entièrement l'alternative préférée d'un décideur. Lors de l'analyse de cet algorithme, nous avons vu que le temps de calcul nécessaire à l'exécution de la phase de collecte s'exprimait comme une fonction en $O(|C| \max_{C_i \in C} \prod_{X_j \in C_i} |X_j|)$, tandis que la phase de diffusion était beaucoup plus rapide ($O(|C| \max_{C_i \in C} |C_i|)$) car elle exploitait les valeurs calculées dans la phase précédente.

L'algorithme de rangement des k meilleures alternatives a été présenté en 2.3.2 comme une adaptation de l'algorithme de choix optimal, comportant une phase de collecte modifiée et une série de phases de diffusion modifiées. En supposant que le nombre k d'alternatives à ranger reste raisonnable, la phase de collecte de cet algorithme reste critique et est plus coûteuse que la phase de collecte de l'algorithme de choix optimal, de par le tri nécessaire de chaque sous-utilité au lieu de la détermination d'une seule valeur maximale. L'algorithme de détermination de la solution agrégée optimale (présenté en 2.4) possède le même coût temporel que l'algorithme de rangement à partir du moment où le réseau GAI scalarisé a été construit, le principal problème étant qu'il est difficile de connaître à l'avance une borne précise du nombre de solutions à itérer avant de pouvoir cesser le rangement.

De ces trois algorithmes, il ressort qu'un élément critique pour le temps de calcul est lié au parcours (ou au rangement) des hypermatrices contenant les sous-fonctions d'utilité. Dans la section 2.2.3, nous avions introduit la notion de treewidth de manière à analyser de façon simple la qualité d'une triangulation. La treewidth est liée de manière exponentielle à la taille de chaque hypermatrice contenue dans le réseau GAI (en effet, si t est la treewidth d'un réseau GAI, et k la modalité maximale des attributs, alors $\forall C_j \in C \ \prod_{X_i \in C_j} |X_i| \leqslant k^{t+1}$), et plus la treewidth d'un réseau augmente, plus les temps de calculs des algorithmes exploitant ce réseau augmentent de manière exponentielle.

On peut objecter à cette remarque qu'il est difficile de se représenter un décideur dont le réseau GAI aurait une forte treewidth, car cela signifierait que les attributs du problème ont une très forte interaction entre eux, ce qu'il est rare de rencontrer en pratique étant

donné les limitations cognitives des êtres humains. L'exemple 26 montre que, d'une façon générale, même si les préférences d'un décideur se modélisent sous forme d'un réseau GAI ayant une faible treewidth, l'utilisation d'une agrégation (pour intégrer des contraintes ou pour former un réseau GAI scalarisé) peut avoir un impact désastreux sur la treewidth, et donc sur la complexité temporelle et spatiale des algorithmes.

Exemple 26 *Nous allons considérer un décideur devant prendre une décision sur l'ensemble d'alternatives* $\mathcal{X} = X_1 \times X_2 \times X_3 \times X_4 \times X_5 \times X_6 \times X_7$ *avec* $\forall i \in \{1, \ldots, 7\}\ |X_i| = 10$. *Ses préférences sont représentables par une fonction d'utilité GAI-décomposable en* $u(x_1, \ldots, x_7) = u_1(x_1, x_2) + u_2(x_2, x_3) + u_3(x_2, x_4) + u_4(x_3, x_5) + u_5(x_5, x_6) + u_6(x_3, x_7)$. *Une telle GAI-décomposition possède une treewidth de* 1, *puisque toutes les cliques du réseau contiennent* 2 *attributs et que ce modèle GAI ne nécessite pas de triangulation. De manière plus précise, une telle décomposition nécessitera de stocker* 600 *valeurs au sein des hypermatrices. Toutefois, les choix de notre décideur doivent respecter certaines contraintes dures imposées pour obtenir des alternatives réalisables. Supposons en effet que toute alternative* $(x_1, \ldots, x_7)$ *réalisable doit respecter le système :*

$$\begin{cases} x_1 \neq x_6 \\ x_1 \neq x_7 \\ x_6 \neq x_7 \\ x_4 \neq x_6 \end{cases}$$

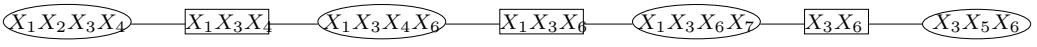


FIGURE 3.1 – Réseau GAI après intégration des contraintes

En utilisant la méthode d'agrégation par sommation et la séquence d'élimination $X_2X_4X_5X_3X_1X_6X_7$ *lors de la triangulation, on obtient le réseau GAI représenté sur la figure 3.1. Ce réseau a une treewidth de* 3 *et nécessite de stocker* 31000 *éléments dans ses hypermatrices, soit une augmentation d'un facteur supérieur à* 51 *pour l'intégration de* 4 *contraintes simples seulement !*

Cette possibilité d'augmentation de la treewidth se retrouve de la même façon dans la construction du réseau GAI scalarisé, étant donné que le problème est lié à la structure, et non à la construction des hypermatrices. Dans notre exemple, à séquence d'élimination égale durant la triangulation, il suffit de remplacer les quatre contraintes par une fonction d'utilité représentant un second critère, GAI-décomposable en $u^2(x_1, \ldots, x_7) = u_1^2(x_1, x_7) + u_2^2(x_1, x_6) + u_3^2(x_4, x_6) + u_4^2(x_6, x_7) + u_5^2(x_2) + u_6^2(x_3) + u_7^2(x_5)$, *et la structure du réseau GAI scalarisé sera la même que celle du réseau GAI de la figure 3.1.*

Ce chapitre va porter sur l'exploitation de réseaux GAI de forte treewidth, et présenter les résultats obtenus dans Dubus et al. (2009a) et Dubus et al. (2009c). Dans un premier temps, nous nous intéresserons uniquement à la résolution du problème de choix optimal dans un contexte de réseau GAI de forte treewidth, mais nous verrons par la suite comment étendre l'approche que nous allons proposer aux deux problèmes laissés de côté (rangement et solution agrégée optimale).

3.1.2 Un algorithme en trois moteurs

L'idée principale de l'algorithme que nous allons proposer est de générer un réseau GAI qui soit plus simple à traiter par un algorithme d'inférence, et donc qui aura une treewidth plus faible que le réseau GAI originel. Toutefois, en supposant que le problème de treewidth ne soit pas dû à un « simple » problème de mauvaise séquence d'élimination dans la triangulation, la création d'un tel réseau ne pourra représenter exactement la relation de préférence du décideur. Ce réseau GAI doit donc être une approximation du réseau GAI originel, et l'algorithme pour le traiter doit donc être capable de traiter une approximation pour obtenir un **résultat qui sera optimal pour le réseau GAI originel**.

En se basant sur le principe développé dans la section 2.4, un tel problème peut se ramener à un problème de rangement. La proposition 1 dresse une hypothèse (de borne supérieure) permettant d'obtenir un critère d'arrêt du rangement suffisant pour connaître le choix optimal du réseau originel.

Proposition 1 (Condition suffisante d'arrêt) *Soit $u : \mathcal{X} \to \mathbb{R}$ une fonction d'utilité, et $\hat{u} : \mathcal{X} \to \mathbb{R}$ une fonction vérifiant $\forall x \in \mathcal{X}\ \hat{u}(x) \geqslant u(x)$. Notons $x^1, \ldots, x^k$ les k meilleures alternatives selon $\hat{u}$, alors on aura :*

$$\hat{u}(x^k) < \max_{j\in\{1,\ldots,k\}} u(x^j) \implies \arg\max_{j\in\{1,\ldots,k\}} u(x^j) = \arg\max_{x\in\mathcal{X}} u(x)$$

Preuve. Pour montrer que $\arg\max_{j\in\{1,\ldots,k\}} u(x^j) = \arg\max_{x\in\mathcal{X}} u(x)$, par décomposition de $\mathcal{X}$, il suffit de montrer que $\max_{j\in\{1,\ldots,k\}} u(x^j) \geqslant \max_{k<j\leqslant|\mathcal{X}|} u(x^j)$.
Supposons que $\hat{u}(x^k) < \max_{j\in\{1,\ldots,k\}} u(x^j)$. On sait que, par rangement, $\forall j > k\ \hat{u}(x^k) \geqslant \hat{u}(x^j)$. De plus, par définition de $\hat{u}$, on a $\forall x \in \mathcal{X}\ \hat{u}(x) \geqslant u(x)$. On a donc $\forall j > k\ \hat{u}(x^k) \geqslant u(x^j)$. Or, par hypothèse, $\hat{u}(x^k) < \max_{j\in\{1,\ldots,k\}} u(x^j)$, il en résulte que $\max_{j\in\{1,\ldots,k\}} u(x^j) \geqslant \max_{k<j\leqslant|\mathcal{X}|} u(x^j)$.
On a donc bien $\hat{u}(x^k) < \max_{j\in\{1,\ldots,k\}} u(x^j) \implies \arg\max_{j\in\{1,\ldots,k\}} u(x^j) = \arg\max_{x\in\mathcal{X}} u(x)$
■

Il est donc possible de ramener le problème de la détermination du choix optimal sur des préférences représentées par une fonction d'utilité u à un problème de rangement d'une fonction d'utilité $\hat{u}$, à condition que $\forall x \in \mathcal{X}\ \hat{u}(x) \geqslant u(x)$. L'algorithme résolvant ce problème dans le cadre de réseaux GAI de forte treewidth doit donc être capable de générer un réseau GAI représentant $\hat{u}$ (que nous noterons par la suite $\hat{R}$) de manière à ce que $\hat{u}$ soit une borne supérieure de u, et $\hat{R}$ doit avoir une treewidth acceptable. Si une telle génération est possible, alors une exploitation directe de la propriété 1 se traduira par l'algorithme 13.

Nous appellerons cet algorithme, l'algorithme aux trois moteurs, car trois briques élémentaires sont nécessaires à sa mise au point :

- **Le moteur de rangement :** Quelle que soit la méthode de génération du réseau $\hat{R}$ utilisée, il sera nécessaire de résoudre un problème de rangement à l'aide de ce réseau.
- **Le moteur d'analyse :** Afin de générer un réseau $\hat{R}$ qui soit une bonne approximation de R, il est nécessaire de comprendre les dépendances entre attributs causant une représentation de forte treewidth. Avec une bonne analyse des dépendances,

Algorithme 13 : ChoixOptTreewidth

```
   Entrée : R : réseau GAI représentant l'utilité u
 1 R̂ ← réseau GAI (d'utilité û) bornant supérieurement u;
 2 Ranking ← MoteurRangement(R̂);
 3 x¹ ← Ranking.solutionSuivante();
 4 UtilitéMax ← u(x¹);
 5 BorneSup ← û(x¹);
 6 MeilleureAlternative ← x¹;
 7 tant que (UtilitéMax ⩽ BorneSup) et Ranking.pasFini() faire
 8 |   xⁱ ← Ranking.solutionSuivante();
 9 |   BorneSup ← û(xⁱ);
10 |   UtilitéCourante ← u(xⁱ);
11 |   si UtilitéCourante > UtilitéMax alors
12 |   |   UtilitéMax ← UtilitéCourante ;
13 |   |   MeilleureAlternative ← xⁱ;
14 |   fin
15 fin
16 retourner MeilleureAlternative ;
```

il peut être possible de conserver des dépendances importantes au sein de $\hat{R}$ et d'élaguer d'autres dépendances moins importantes, ou entrainant directement une augmentation drastique de la treewidth.

– **Le moteur d'approximation :** La propriété de borne supérieure de $\hat{u}$ est directement liée aux sous-fonctions d'utilité composant u. Le moteur d'approximation doit être capable de collaborer avec le moteur d'analyse de manière à générer des sous-fonctions d'utilité pour $\hat{u}$ qui respectent la structure générée par le moteur d'analyse, tout en réalisant de bonnes approximations des sous-fonctions d'utilité de u.

3.2 Vers les détails de l'algorithme

Nous avons vu dans la section précédente l'idée générale d'un algorithme pour résoudre le problème du choix optimal dans un contexte de réseau GAI de forte treewidth. Cette idée s'articule autour de trois moteurs : les moteurs d'approximation et d'analyse servent à la génération d'un réseau GAI de treewidth raisonnable ayant la propriété de borne supérieure des utilités initiales, tandis que le moteur de rangement permet de déterminer l'optimum à partir du réseau approximant. Cette section va décrire en détail les deux moteurs nécessaires à la génération du réseau approximant. Avant d'entrer dans les détails de cette génération, il est important de garder en tête deux points essentiels et contradictoires entre eux :

– Le réseau initial est trop complexe pour être exploité. Toutefois cette complexité est liée aux attributs contenus dans ses cliques, pas nécessairement aux sous-fonctions d'utilité qui le composent. Ainsi, une clique contenant les attributs $X_1X_2X_3X_4X_5X_6$ devrait contenir une hypermatrice définie sur tous ces attributs, mais cette hyper-

matrice peut être elle-même composée d'une somme de plus petites fonctions (nous verrons que, dans ce cas, les approximations peuvent être optimisées). Par exemple, après une triangulation, la sous-fonction d'utilité associée à une clique est l'expression d'une combinaison de plus petites fonctions. La réduction de la treewidth de ce réseau entraînera des réductions dans la taille des cliques, et donc dans les sous-fonctions d'utilité. Cette réduction s'exprimera sous forme d'approximations, donc plus le réseau sera réduit, plus des approximations seront nécessaires à la génération du réseau approximant.

– Les approximations doivent générer des hypermatrices de manière à obtenir une borne supérieure de l'utilité initiale. Intuitivement, cette borne supérieure doit être la plus proche possible de l'utilité initiale, une mauvaise approximation pouvant augmenter le nombre d'itérations de rangement lors des calculs effectués après la génération. De la même façon, quelle que soit la qualité des approximations, démultiplier ces approximations augmentera le nombre d'alternatives rangées, et donc la complexité globale de l'algorithme.

Dans toute cette section, nous noterons $\hat{R} = (\hat{F}, \{\hat{u}_j\})$ le réseau approximant, et nous noterons Z l'ensemble des cliques Z_j du réseau $\hat{R}$, tandis que le réseau originel utilisera les notations classiques précédemment introduites (cliques C_i, etc...).

3.2.1 Moteur d'approximation : le couteau

Au sein de cette sous-section, nous nous intéresserons uniquement aux approximations, donc à des aspects numériques, sans aucune considération pour les aspects graphiques du réseau GAI. Ce moteur peut donc être vu comme un couteau, le moteur d'analyse étant le boucher, et les sous-fonctions d'utilité de grandes pièces de viande. Le boucher déterminera quelles pièces découper et quelle forme il désire obtenir, et le couteau exécutera ce découpage sans remettre en question les choix du boucher.

Avant d'entrer dans les détails de ce qu'est une bonne approximation, nous savons que, quelle que soit l'approximation à réaliser, il est nécessaire qu'elle soit une borne supérieure de l'utilité initiale. On aura donc $\forall x \in \mathcal{X}\ \hat{u}(x) \geqslant u(x)$, ou encore du point de vue de la GAI-décomposition, $\forall x \in \mathcal{X}\ \sum_{Z_j \in Z} \hat{u}_j(x_{Z_j}) \geqslant \sum_{C_j \in C} u_j(x_{C_j})$.

Une bonne approximation doit être la plus proche possible de l'utilité initiale pour toute alternative. Cette propriété peut être vue comme une distance à minimiser pour chaque alternative. Cette notion peut se traduire, pour une alternative x donnée, par la minimisation de $|\hat{u}(x) - u(x)|$. Étant donné que $\hat{u}$ est une borne supérieure de u, on aura donc : $|\hat{u}(x) - u(x)| = \hat{u}(x) - u(x)$. Cette quantité étant positive, on peut modéliser la généralisation à toutes les alternatives par sommation en : $\min_{\{\hat{u}\}} \sum_{x \in \mathcal{X}} \big(\hat{u}(x) - u(x)\big)$. Il est important de noter que nous cherchons le minimum pour toutes les fonctions $\hat{u}$ admissibles. Celles-ci pouvent être décrites par des hypermatrices sur $\mathcal{X}$, ceci revient à dire que nous cherchons un minimum sur un ensemble de $|\mathcal{X}|$ paramètres (les valeurs des cases de l'hypermatrice $\hat{u}$).

En mélangeant les contraintes de borne supérieure et la notion de distance d'approximation à minimiser, on en déduit une première modélisation de l'approximation sous forme de programme linéaire :

$$\begin{cases} \min_{\{\hat{u}_j\}} & \sum_{x \in \mathcal{X}} \left(\sum_{Z_j \in Z} \hat{u}_j(x_{Z_j}) - \sum_{C_j \in C} u_j(x_{C_j}) \right) \\ \text{s.c.} & \sum_{Z_j \in Z} \hat{u}_j(x_{Z_j}) \geqslant \sum_{C_j \in C} u_j(x_{C_j})\ \forall x \in \mathcal{X} \end{cases}$$

Étant donné que $\hat{u}$ se décompose comme $\sum_{Z_j \in Z} \hat{u}_j$, il suffit de minimiser sur l'ensemble des paramètres de tous les $\hat{u}_j$. Donc, dans ce programme linéaire, les variables du problème correspondent au contenu des hypermatrices à générer, c'est-à-dire les $\hat{u}_j(x_{Z_j})$, tandis que les valeurs de $u_j(x_{C_j})$ sont des constantes issues du réseau GAI représentant les préférences du décideur. L'exemple 27 présente un cas concret d'approximation en utilisant cette méthode.

Exemple 27 *Supposons que nous désirions réduire la fonction d'utilité GAI-décomposable* $u(x_1, x_2, x_3) = u_1(x_1, x_2) + u_2(x_2, x_3)$ *en une fonction approximante* $\hat{u}(x_1, x_2, x_3) = \hat{u}_1(x_1) + \hat{u}_2(x_2) + \hat{u}_3(x_3)$*, avec les fonctions* u_i *représentées au sein de la figure 3.2.*

	X_2	
X_1	x_{21}	x_{22}
x_{11}	1	10
x_{12}	6	4

(a) $u_1(x_1, x_2)$

	X_2	
X_3	x_{21}	x_{22}
x_{31}	8	3
x_{32}	12	5

(b) $u_2(x_2, x_3)$

	X_1			
	x_{11}		x_{12}	
	X_2		X_2	
X_3	x_{21}	x_{22}	x_{21}	x_{22}
x_{31}	9	13	14	7
x_{32}	13	15	18	9

(c) $u_1(x_1, x_2) + u_2(x_2, x_3)$

FIGURE 3.2 – Fonctions d'utilité à réduire

Au sein de chaque sous-fonction approximante, on introduit les variables du programme linéaire que nous avons représentées dans la figure 3.3. Les attributs étant binaires, et chaque sous-fonction approximante ne comportant qu'un seul attribut, nous aurons donc $2 + 2 + 2 = 8$ *variables dans le programme linéaire.*

X_1	
x_{11}	$\hat{u}_{11}$
x_{12}	$\hat{u}_{12}$

(a) $\hat{u}_1(x_1)$

X_2	
x_{21}	$\hat{u}_{21}$
x_{22}	$\hat{u}_{22}$

(b) $\hat{u}_2(x_2)$

X_3	
x_{31}	$\hat{u}_{31}$
x_{32}	$\hat{u}_{32}$

(c) $\hat{u}_3(x_3)$

FIGURE 3.3 – Variables dans les fonctions à générer

Il suffit ensuite de générer la fonction objectif et les contraintes comme nous l'avons présenté précédemment : Pour ce faire, nous parcourons toutes les valeurs d'utilité $u(x_1, x_2, x_3)$ *possibles et, pour chaque valeur, nous générons une contrainte et nous enrichissons la fonction objectif.*

$$\begin{cases} \min & \hat{u}_{11} + \hat{u}_{21} + \hat{u}_{31} - 9 \\ & +\hat{u}_{11} + \hat{u}_{21} + \hat{u}_{32} - 13 \\ & +\hat{u}_{11} + \hat{u}_{22} + \hat{u}_{31} - 13 \\ & +\hat{u}_{11} + \hat{u}_{22} + \hat{u}_{32} - 15 \\ & +\hat{u}_{12} + \hat{u}_{21} + \hat{u}_{31} - 14 \\ & +\hat{u}_{12} + \hat{u}_{21} + \hat{u}_{32} - 18 \\ & +\hat{u}_{12} + \hat{u}_{22} + \hat{u}_{31} - 7 \\ & +\hat{u}_{12} + \hat{u}_{22} + \hat{u}_{32} - 9 \\ s.c. & \hat{u}_{11} + \hat{u}_{21} + \hat{u}_{31} \geqslant 9 \\ & \hat{u}_{11} + \hat{u}_{21} + \hat{u}_{32} \geqslant 13 \\ & \hat{u}_{11} + \hat{u}_{22} + \hat{u}_{31} \geqslant 13 \\ & \hat{u}_{11} + \hat{u}_{22} + \hat{u}_{32} \geqslant 15 \\ & \hat{u}_{12} + \hat{u}_{21} + \hat{u}_{31} \geqslant 14 \\ & \hat{u}_{12} + \hat{u}_{21} + \hat{u}_{32} \geqslant 18 \\ & \hat{u}_{12} + \hat{u}_{22} + \hat{u}_{31} \geqslant 7 \\ & \hat{u}_{12} + \hat{u}_{22} + \hat{u}_{32} \geqslant 9 \end{cases}$$

Il est à noter que la fonction objectif peut s'écrire de manière plus factorisée en déterminant les coefficients devant les variables et en supprimant les valeurs de l'utilité initiale, qui sont des constantes n'intervenant pas dans la détermination d'un minimum. On aura donc finalement le programme linéaire suivant :

$$\begin{cases} \min & 4\hat{u}_{11} + 4\hat{u}_{12} + 4\hat{u}_{21} + 4\hat{u}_{22} + 4\hat{u}_{31} + 4\hat{u}_{32} \\ s.c. & \hat{u}_{11} + \hat{u}_{21} + \hat{u}_{31} \geqslant 9 \\ & \hat{u}_{11} + \hat{u}_{21} + \hat{u}_{32} \geqslant 13 \\ & \hat{u}_{11} + \hat{u}_{22} + \hat{u}_{31} \geqslant 13 \\ & \hat{u}_{11} + \hat{u}_{22} + \hat{u}_{32} \geqslant 15 \\ & \hat{u}_{12} + \hat{u}_{21} + \hat{u}_{31} \geqslant 14 \\ & \hat{u}_{12} + \hat{u}_{21} + \hat{u}_{32} \geqslant 18 \\ & \hat{u}_{12} + \hat{u}_{22} + \hat{u}_{31} \geqslant 7 \\ & \hat{u}_{12} + \hat{u}_{22} + \hat{u}_{32} \geqslant 9 \end{cases}$$

Nous avons choisi de laisser les coefficients 4 *dans l'expression de la fonction objectif au sein de cet exemple, les coefficients étant généralement différents d'une variable à une autre. Cette écriture factorisée sera étudiée plus en détails en fin de section, lorsque l'optimisation des programmes linéaires à générer sera abordée.*

En analysant cette modélisation des approximations, nous pouvons nous rendre compte que le programme linéaire possède autant de contraintes que d'alternatives dans $\mathcal{X}$, et autant de variables que de valeurs à générer au sein des sous-utilités de $\hat{R}$, soit $O(\sum_{Z_j \in Z} \prod_{X_i \in Z_j} |X_i|)$ variables et $O(|\mathcal{X}|)$ contraintes. Une telle approche n'est donc pas viable en pratique, et il est nécessaire de trouver une nouvelle modélisation.

En gardant le même principe de modélisation, nous allons nous intéresser au découpage d'une sous-fonction d'utilité $u_j : X_{C_j} \to \mathbb{R}$. Cette fonction sera découpée en un certain nombre de sous-fonctions définies sur un sous-ensemble de X_{C_j} et telles que l'union de ces domaines de définition corresponde à X_{C_j}. Ainsi, il sera possible d'associer à chaque sous-fonction d'utilité du réseau approximant une unique sous-fonction d'utilité

du réseau initial. La formalisation de cette notion nous permet de définir l'ensemble de découpages.

Définition 69 (Ensemble de découpages) *Soit C_i un ensemble. On appelle découpage de C_i un ensemble $\mathcal{Z}_i = \{Z_j\}$ vérifiant :*
- $\forall Z_j \in \mathcal{Z}_i \; Z_j \subseteq C_i$
- $\bigcup_{Z_j \in \mathcal{Z}_i} Z_j = C_i$

Soit $C = \{C_1, \ldots, C_q\}$. On appelle ensemble de découpages de C l'ensemble $\mathcal{Z} = \{\mathcal{Z}_1, \ldots, \mathcal{Z}_q\}$ vérifiant $\forall i \in \{1, \ldots, q\}$ $\mathcal{Z}_i$ est un découpage de C_i.

Dans une écriture plus formelle, à chaque clique C_i est associée une sous-fonction d'utilité u_i et un découpage $\mathcal{Z}_i$. Ce découpage est une réprésentation des fonctions approximantes à générer : on aura une fonction $\hat{u}_{i,j}$ pour chaque $Z_j \in \mathcal{Z}_i$ vérifiant $\hat{u}_{i,j} : X_{Z_j} \to \mathbb{R}$. Une telle écriture permet de découper le système de contraintes à respecter (pour obtenir une approximation majorante) en un ensemble de systèmes de taille réduite, comme le montre la propriété 27.

Propriété 27 (Contraintes découpées) *Etant donné $\mathcal{Z} = \{\mathcal{Z}_1, \ldots, \mathcal{Z}_q\}$ l'ensemble des découpages de $C = \{C_1, \ldots, C_q\}$, on a :*

$$\forall C_i \in C \forall x_{C_i} \in X_{C_i} \sum_{Z_j \in \mathcal{Z}_i} \hat{u}_{i,j}(x_{Z_j}) \geqslant u_i(x_{C_i}) \implies \forall x \in \mathcal{X} \sum_{\mathcal{Z}_i \in \mathcal{Z}} \sum_{Z_j \in \mathcal{Z}_i} \hat{u}_{i,j}(x_{Z_j}) \geqslant u(x).$$

Preuve. Cette propriété s'obtient facilement par sommation : considérons une alternative $x \in \mathcal{X}$ et supposons que, $\forall C_i \in C$, on ait $\sum_{Z_j \in \mathcal{Z}_i} \hat{u}_{i,j}(x_{Z_j}) \geqslant u_i(x_{C_i})$, alors, en sommant sur toutes les inégalités liées aux $C_i \in C$, on aura : $\sum_{\mathcal{Z}_i \in \mathcal{Z}} \sum_{Z_j \in \mathcal{Z}_i} \hat{u}_{i,j}(x_{Z_j}) \geqslant \sum_{C_i \in C} u_i(x_{C_i}) = u(x)$. ■

La propriété 27 va nous permettre de découper le système de contraintes en q systèmes, chacun d'entre eux étant associé à une clique C_i contenant $\prod_{X_j \in C_i} |X_j|$ inégalités. Afin de pouvoir généraliser ce découpage au programme linéaire que nous avions initialement, il est nécessaire de modifier la fonction objectif. Pour ce faire, nous allons approcher cette fonction : à chaque clique C_i, on peut associer l'écart entre la sous-fonction d'utilité qui lui est associée et les fonctions approximantes : $\sum_{x_{C_i} \in X_{C_i}} (\sum_{Z_j \in \mathcal{Z}_i} u_{i,j}(x_{Z_j}) - u_i(x_{C_i}))$. Il en résulte que nous pouvons bâtir une suite PL_i de programmes linéaires pour déterminer de bonnes approximations des fonctions découpées :

$$(PL_i) \begin{cases} \min_{\{\hat{u}_{i,j}\}} & \sum_{x_{C_i} \in X_{C_i}} \sum_{Z_j \in \mathcal{Z}_i} \hat{u}_{i,j}(x_{Z_j}) - u_i(x_{C_i}) \\ \text{s.c.} & \sum_{Z_j \in \mathcal{Z}_i} \hat{u}_{i,j}(x_{Z_j}) \geqslant u_i(x_{C_i}) \; \forall x_{C_i} \in X_{C_i} \end{cases}$$

De nouveau, les variables du problème du programme linéaire (PL_i) correspondent au contenu des hypermatrices à générer (les $\hat{u}_{i,j}(x_{Z_j})$), et les $u_i(x_{C_i})$ sont des constantes issues du réseau GAI initial. L'exemple 28 présente un cas concret d'approximation utilisant des programmes linéaires de la forme (PL_i).

Exemple 28 *Le moteur prend maintenant en argument une unique sous-fonction d'utilité, à découper en plusieurs sous-fonctions portant sur un nombre réduit d'attributs. Supposons que notre moteur d'approximation doive répondre à une requête pour découper*

	X_1			
	x_{11}		x_{12}	
	X_2		X_2	
X_3	x_{21}	x_{22}	x_{21}	x_{22}
x_{31}	1	2	6	7
x_{32}	8	5	4	3

(a) $u_1(x_1, x_2, x_3)$

	X_2	
X_1	x_{21}	x_{22}
x_{11}	$\hat{u}_{111}$	$\hat{u}_{112}$
x_{12}	$\hat{u}_{121}$	$\hat{u}_{122}$

(b) $\hat{u}_1(x_1, x_2)$

	X_2	
X_3	x_{21}	x_{22}
x_{31}	$\hat{u}_{211}$	$\hat{u}_{221}$
x_{32}	$\hat{u}_{212}$	$\hat{u}_{222}$

(c) $\hat{u}_2(x_2, x_3)$

FIGURE 3.4 – Fonction d'utilité à réduire et représentation des variables

$u_1(x_1, x_2, x_3)$ *en* $\hat{u}_1(x_1, x_2)$ *et* $\hat{u}_2(x_2, x_3)$. *Le contenu de la fonction* u_1 *et les variables associées aux fonctions* $\hat{u}_1$ *et* $\hat{u}_2$ *sont représentées sur la figure 3.4.*

En appliquant le nouveau principe d'approximation, on en déduit un programme linéaire effectuant une approximation locale à la sous-fonction u_1 *de manière indépendante d'éventuelles autres fonctions d'utilité. On aura donc :*

$$\begin{cases} \min & \hat{u}_{111} + \hat{u}_{211} - 1 + \hat{u}_{111} + \hat{u}_{212} - 8 \\ & +\hat{u}_{112} + \hat{u}_{221} - 2 + \hat{u}_{112} + \hat{u}_{222} - 5 \\ & +\hat{u}_{121} + \hat{u}_{211} - 6 + \hat{u}_{121} + \hat{u}_{212} - 4 \\ & +\hat{u}_{122} + \hat{u}_{221} - 7 + \hat{u}_{122} + \hat{u}_{222} - 3 \\ s.c. & \hat{u}_{111} + \hat{u}_{211} \geqslant 1 \\ & \hat{u}_{111} + \hat{u}_{212} \geqslant 8 \\ & \hat{u}_{112} + \hat{u}_{221} \geqslant 2 \\ & \hat{u}_{112} + \hat{u}_{222} \geqslant 5 \\ & \hat{u}_{121} + \hat{u}_{211} \geqslant 6 \\ & \hat{u}_{121} + \hat{u}_{212} \geqslant 4 \\ & \hat{u}_{122} + \hat{u}_{221} \geqslant 7 \\ & \hat{u}_{122} + \hat{u}_{222} \geqslant 3 \end{cases}$$

Ce qui peut être réécrit sous la forme :

$$\begin{cases} \min & 2\hat{u}_{111} + 2\hat{u}_{112} + 2\hat{u}_{121} + 2\hat{u}_{122} \\ & 2\hat{u}_{211} + 2\hat{u}_{212} + 2\hat{u}_{221} + 2\hat{u}_{222} \\ s.c. & \hat{u}_{111} + \hat{u}_{211} \geqslant 1 \\ & \hat{u}_{111} + \hat{u}_{212} \geqslant 8 \\ & \hat{u}_{112} + \hat{u}_{221} \geqslant 2 \\ & \hat{u}_{112} + \hat{u}_{222} \geqslant 5 \\ & \hat{u}_{121} + \hat{u}_{211} \geqslant 6 \\ & \hat{u}_{121} + \hat{u}_{212} \geqslant 4 \\ & \hat{u}_{122} + \hat{u}_{221} \geqslant 7 \\ & \hat{u}_{122} + \hat{u}_{222} \geqslant 3 \end{cases}$$

On peut toutefois remarquer que l'attribut X_2 *apparaît dans les deux sous-fonctions approximantes. L'ensemble des contraintes du programme linéaire peut être vu comme l'union de deux ensembles de contraintes correspondant aux deux valeurs possibles de* X_2 *:*

- *à la valeur* x_{21} *sont liées les contraintes* $\hat{u}_{111}+\hat{u}_{211} \geqslant 1$, $\hat{u}_{111}+\hat{u}_{212} \geqslant 8$, $\hat{u}_{121}+\hat{u}_{211} \geqslant 6$ *et* $\hat{u}_{121} + \hat{u}_{212} \geqslant 4$

– *à la valeur* x_{22} *sont liées les contraintes* $\hat{u}_{112}+\hat{u}_{221} \geqslant 2$, $\hat{u}_{112}+\hat{u}_{222} \geqslant 5$, $\hat{u}_{122}+\hat{u}_{221} \geqslant 7$ *et* $\hat{u}_{122}+\hat{u}_{222} \geqslant 3$.

Pour une valeur de X_2 *donnée, on remarque que chaque variable apparaissant dans une contrainte liée à cette valeur n'apparaîtra pas dans une contrainte liée à une autre valeur. Nous sommes donc en présence de deux ensembles indépendants de variables au sein du programme linéaire :* $\{\hat{u}_{111}, \hat{u}_{121}, \hat{u}_{211}, \hat{u}_{212}\}$ *et* $\{\hat{u}_{112}, \hat{u}_{122}, \hat{u}_{221}, \hat{u}_{222}\}$. *La fonction objectif étant linéaire, il est donc possible d'exprimer notre précédent programme linéaire, en deux programmes linéaires indépendants à résoudre :*

$$\begin{cases} \min & 2\hat{u}_{111}+2\hat{u}_{121}+2\hat{u}_{211}+2\hat{u}_{212} \\ s.c. & \hat{u}_{111}+\hat{u}_{211} \geqslant 1 \\ & \hat{u}_{111}+\hat{u}_{212} \geqslant 8 \\ & \hat{u}_{121}+\hat{u}_{211} \geqslant 6 \\ & \hat{u}_{121}+\hat{u}_{212} \geqslant 4 \end{cases} \quad \begin{cases} \min & 2\hat{u}_{112}+2\hat{u}_{122}+2\hat{u}_{221}+2\hat{u}_{222} \\ s.c. & \hat{u}_{112}+\hat{u}_{221} \geqslant 2 \\ & \hat{u}_{112}+\hat{u}_{222} \geqslant 5 \\ & \hat{u}_{122}+\hat{u}_{221} \geqslant 7 \\ & \hat{u}_{122}+\hat{u}_{222} \geqslant 3 \end{cases}$$

Les programmes ayant moins de variables et de contraintes, ils pourront donc être résolus plus rapidement. Une autre façon d'appréhender cette optimisation est de raisonner directement sur la sous-fonction à réduire et les sous-fonctions à générer : Découper $u_1(x_1,x_2,x_3)$ *en* $\hat{u}_1(x_1,x_2)$ *et* $\hat{u}_2(x_2,x_3)$ *revient à découper* $u_1^{X_2=x_{21}}(x_1,x_3)$ *en* $\hat{u}_1^{X_2=x_{21}}(x_1)$ *et* $\hat{u}_2^{X_2=x_{21}}(x_3)$ *et à découper* $u_1^{X_2=x_{22}}(x_1,x_3)$ *en* $\hat{u}_1^{X_2=x_{22}}(x_1)$ *et* $\hat{u}_2^{X_2=x_{22}}(x_3)$, *en posant :* $u_1^{X_2=x_{2i}}(x_1,x_3) = u_1(x_1,x_{2i},x_3)$, $\hat{u}_1^{X_2=x_{2i}}(x_1) = \hat{u}_1(x_1,x_{2i})$ *et* $\hat{u}_2^{X_2=x_{2i}}(x_3) = \hat{u}_2(x_{2i},x_3)$.

Le principe d'optimisation de l'écriture des programmes linéaires introduit dans l'exemple 28 peut être généralisé, il ne reste donc plus qu'à le formaliser.

Supposons que la requête adressée au moteur d'approximation soit de réduire la sous-fonction d'utilité $u_i(x_{C_i})$ selon un découpage $\mathcal{Z}_i$ (on aura donc une suite $\hat{u}_j(x_{Z_j})\ \forall Z_j \in \mathcal{Z}_i$ de sous-fonctions approximantes à générer). L'ensemble des attributs en commun dans le découpage sera représenté par $Y_i = \bigcap_{Z_j \in \mathcal{Z}_i} Z_j$. Si $Y_i = \varnothing$, alors l'optimisation du programme linéaire en sous-programmes linéaires de tailles réduites ne sera pas possible. Nous supposerons donc que $Y_i \neq \varnothing$. Il est nécessaire de définir dans un premier temps l'application partielle d'une fonction (terme inspiré des langages de programmation fonctionnelle), c'est-à-dire le mécanisme permettant de considérer un sous-ensemble de valeurs d'une hypermatrice représentant une fonction.

Définition 70 (Application partielle d'une fonction) *Soit* $u : X_C \to \mathbb{R}$, *et soit* $Y \neq \varnothing$ *tel que* $Y \subset C$. *On définit l'application partielle de* u *sur* $x_Y \in X_Y$ *comme étant une fonction notée* $u^{X_Y=x_Y} : X_{C\setminus Y} \to \mathbb{R}$ *vérifiant* $\forall x_C \in X_C\ u^{X_Y=x_Y}(x_{C\setminus Y}) = u(x_Y, x_{C\setminus Y})$.

En utilisant cette notion, il est possible d'exprimer une suite de programmes linéaires dépendant des valeurs de x_Y et correspondant à une approximation locale de l'hypermatrice :

$$(PL_i^{X_Y=x_Y}) \begin{cases} \min_{\{\hat{u}_{i,j}^{X_Y=x_Y}\}} & \sum_{x_{C_i\setminus Y} \in X_{C_i\setminus Y}} \sum_{Z_j \in \mathcal{Z}_i} \hat{u}_{i,j}^{X_Y=x_Y}(x_{Z_j\setminus Y}) - u_i^{X_Y=x_Y}(x_{C_i\setminus Y}) \\ \text{s.c.} & \sum_{Z_j \in \mathcal{Z}_i} \hat{u}_{i,j}^{X_Y=x_Y}(x_{Z_j\setminus Y}) \geqslant u_i^{X_Y=x_Y}(x_{C_i\setminus Y})\ \forall x_{C_i\setminus Y} \in X_{C_i\setminus Y} \end{cases}$$

Il reste donc à prouver qu'il est équivalent de résoudre PL_i et la suite $PL_i^{X_Y=x_Y}$. Ce résultat nous est fourni par la proposition 2. L'exploitation de cette équivalence nous indique qu'il est possible de calculer une approximation selon le découpage $\mathcal{Z}_i$ en exécutant $|X_Y|$ programmes linéaires contenant $\Theta(\sum_{Z_j \in \mathcal{Z}_i} |X_{Z_j \setminus Y}|)$ variables et $\Theta(|X_{C_i \setminus Y}|)$ contraintes.

Proposition 2 *Soit $u_i : X_{C_i} \to \mathbb{R}$ une fonction, $\mathcal{Z}_i$ un découpage de C_i et notons $\mathcal{O}(PL_i)$ la valeur de l'objectif d'un programme linéaire PL_i à l'optimum, alors $\forall Y \neq \varnothing$ vérifiant $\forall Z_j \in \mathcal{Z}_i$ $Y \subset Z_j$, on aura :*

$$\sum_{x_Y \in X_Y} \mathcal{O}(PL_i^{X_Y=x_Y}) = \mathcal{O}(PL_i)$$

Preuve. Reprenons la définition de la fonction objectif du programme linéaire PL_i, on a $\sum_{x_{C_i} \in X_{C_i}} \sum_{Z_j \in \mathcal{Z}_i} \hat{u}_{i,j}(x_{Z_j}) - u_i(x_{C_i})$. En faisant apparaître les composantes en X_Y de chaque fonction, on aura donc : $\sum_{x_{C_i} \in X_{C_i}} \sum_{Z_j \in \mathcal{Z}_i} \hat{u}_{i,j}(x_Y, x_{Z_j \setminus Y}) - u_i(x_Y, x_{C_i \setminus Y})$, ce qui (par définition des applications partielles) est équivalent à

$$\sum_{x_Y \in X_Y} \sum_{x_{C_i \setminus Y} \in X_{C_i \setminus Y}} \sum_{Z_j \in \mathcal{Z}_i} \hat{u}_{i,j}^{X_Y=x_Y}(x_{Z_j \setminus Y}) - u_i^{X_Y=x_Y}(x_{C_i \setminus Y})$$

Ce qui est l'expression de $\sum_{x_Y \in X_Y} \mathcal{O}(PL_i^{X_Y=x_Y})$. Il existe donc une bijection entre l'ensemble des variables de PL_i et l'ensemble de toutes les variables des $PL_i^{X_Y=x_Y}$. De plus, il n'existe aucune variable commune entre les différents $PL_i^{X_Y=x_Y}$, il est donc possible d'optimiser séparément ces programmes linéaires pour résoudre PL_i.

Considérons maintenant l'ensemble des contraintes de PL_i, on a $\sum_{Z_j \in \mathcal{Z}_i} \hat{u}_{i,j}(x_{Z_j}) \geqslant u_i(x_{C_i})$ $\forall x_{C_i} \in X_{C_i}$, ce qui peut se réécrire $\sum_{Z_j \in \mathcal{Z}_i} \hat{u}_{i,j}(x_Y, x_{Z_j \setminus Y}) \geqslant u_i(x_Y, x_{C_i \setminus Y})$ $\forall x_{C_i} \in X_{C_i}$, soit :

$$\sum_{Z_j \in \mathcal{Z}_i} \hat{u}_{i,j}^{X_Y=x_Y}(x_{Z_j \setminus Y}) \geqslant u_i^{X_Y=x_Y}(x_{C_i \setminus Y}) \ \forall x_{C_i} \in X_{C_i}$$

On peut donc en conclure que chaque programme linéaire $PL_i^{X_Y=x_Y}$ porte sur un sous-ensemble de variables de PL_i, et contient l'intégralité des contraintes associées à ces variables. La concaténation de l'ensemble des solutions optimales des $PL_i^{X_Y=x_Y}$ est une solution optimale pour PL_i, et réciproquement. On aura donc :

$$\sum_{x_Y \in X_Y} \mathcal{O}(PL_i^{X_Y=x_Y}) = \mathcal{O}(PL_i)$$

■

On peut donc déduire directement l'algorithme 14 générant automatiquement les approximations. Ses arguments seront une sous-fonction d'utilité à découper (u_i) et un découpage de cette sous-fonction $(\mathcal{Z}_i)$. Cet algorithme génère dans un premier temps l'intersection des attributs contenus dans les sous-fonctions approximantes à générer (Y). Si cet ensemble est vide, le problème d'approximation sera résolu par le programme linéaire PL_i, sinon l'ensemble des $|X_Y|$ programmes linéaires $PL_i^{X_Y=x_Y}$ sera exécuté.

Algorithme 14 : Approximer

Entrées :
$u_i : X_{C_i} \to \mathbb{R}$: sous-fonction d'utilité
$\mathcal{Z}_i$: découpage de la fonction

```
créer Y ← ⋂_{Z_j ∈ Z_i} Z_j ;
si Y = ∅ alors
    résoudre PL_i ;
sinon
    pour tous les x_Y ∈ X_Y faire
        résoudre PL_i^{X_Y = x_Y} ;
    fin
fin
```

Remarque 18 *Il est à noter que nous avons choisi de présenter les programmes linéaires sous forme d'une somme de différences entre fonctions approximantes et fonction approximée pour plus de clarté, mais il est possible de déterminer automatiquement les coefficients associés aux variables dans la fonction objectif. Étant donné que l'expression de l'objectif de* $PL_i^{X_Y=x_Y}$ *est* $\sum_{x_{C_i\setminus Y} \in X_{C_i\setminus Y}} \sum_{Z_j \in \mathcal{Z}_i} \hat{u}_{i,j}^{X_Y=x_Y}(x_{Z_j\setminus Y}) - u_i^{X_Y=x_Y}(x_{C_i\setminus Y})$*, le coefficient d'une variable* $\hat{u}_{i,j}^{X_Y=x_Y}(x_{Z_j\setminus Y})$ *sera son nombre d'occurences dans les sommes de différences, soit* $|X_{C_i\setminus Z_j}|$*. De plus, les constantes de l'objectif (*$u_i^{X_Y=x_Y}(x_{C_i\setminus Y})$*) n'intervenant pas dans la détermination d'un minimum, l'objectif peut se réécrire*

$$\min_{\{\hat{u}_{i,j}^{X_Y=x_Y}\}} \sum_{Z_j \in \mathcal{Z}_i} \sum_{x_{Z_j\setminus Y} \in X_{Z_j\setminus Y}} |X_{C_i\setminus Z_j}| \hat{u}_{i,j}^{X_Y=x_Y}(x_{Z_j\setminus Y})$$

3.2.2 Moteur d'analyse graphique : le boucher

Nous avons défini notre « couteau » dans la sous-section précédente, toutefois nous n'avons pas précisé la manière dont les découpages (les ensembles $\mathcal{Z}_i$) seront générés. Le rôle du moteur d'analyse graphique est de manipuler ce couteau pour déterminer de tels découpages. D'une façon plus formelle, deux aspects entrent en jeu dans la conception d'un tel moteur :

- **La structure du réseau** : Le problème initial étant que le réseau GAI à traiter a une trop forte treewidth, le moteur doit procéder a une analyse fine du réseau GAI pour déterminer les relations entre attributs posant des problèmes computationels. Cette analyse permettra de déterminer un élagage des arêtes du graphe markovien, et donc de générer un ensemble de découpages à fournir au moteur d'approximation.
- **Le contenu des sous-fonctions d'utilité** : Le moteur doit élaguer dans la structure de dépendance des attributs. Toutefois toutes les dépendances ne se valent pas entre elles. Supposons que le moteur, après une analyse de la structure graphique, ait le choix entre supprimer la dépendance entre X_1 et X_2 ou la dépendance entre X_2 et X_3, peut-être que l'élagage de la première dépendance conduirait à une approximation catastrophique, tandis que l'approximation de la seconde dépendance serait quasiment parfaite dans les valeurs obtenues. Il est donc difficile de s'intéresser à un critère uniquement graphique, le moteur devra donc traiter la structure

du réseau de façon conjointe avec des indicateurs sur la qualité de l'approximation que nous pourrions obtenir.

Pour la conception de ce moteur, nous avons choisi de nous intéresser au graphe markovien du réseau GAI plutôt qu'à la structure de forêt de jonction, d'une part pour partir d'une représentation plus fine des relations entre attributs, et d'autre part pour permettre aux réseaux GAI obtenus par triangulation de ce graphe d'être analysés en ayant connaissance des arêtes fill-ins ajoutées durant le processus de triangulation.

La première question à se poser avant d'entrer dans le vif du sujet est : Sur quel critère se baser pour préférer utiliser l'approche que nous proposons dans ce chapitre plutôt qu'un algorithme d'inférence classique sans approximation ? Un premier indicateur que nous avons présenté précédemment est la treewidth du réseau. Mais, pour raisonner de façon plus fine dans le cadre de ce moteur, nous allons nous intéresser à la taille des hypermatrices à stocker dans les cliques après triangulation du graphe markovien, c'est-à-dire le nombre de valeurs à stocker au sein d'une hypermatrice. Cet indicateur est lié à la treewidth lorsque tous les attributs contiennent le même nombre de modalités (on aura k^{t+1} comme taille d'hypermatrice, avec k le nombre de modalités d'un attribut et t la treewidth). Nous allons supposer ici l'existence d'une valeur T indiquant une taille d'hypermatrice à ne pas dépasser pour obtenir une inférence raisonnable en temps de calcul. Ainsi, une sous-fonction d'utilité $u_i : X_{C_i} \rightarrow \mathbb{R}$ sera jugée trop grosse si $\prod_{j \in C_i} |X_j| > T$, ce qui peut s'écrire également $\sum_{j \in C_i} \log(|X_j|) > \log(T)$, et il sera donc nécessaire de procéder à un découpage de u_i, ce qui se traduira dans le graphe markovien par une suppression d'arêtes dans la clique des attributs de u_i.

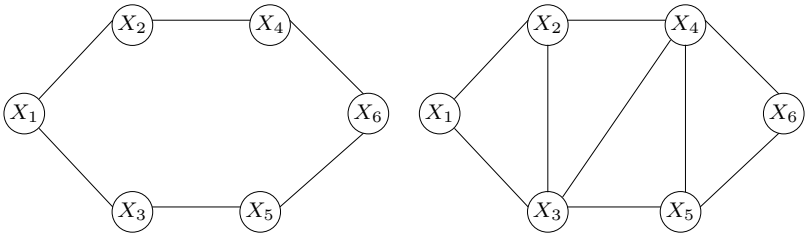
La difficulté d'une telle approche est qu'il n'est pas possible de supprimer n'importe quelle arête : le graphe markovien après triangulation est censé être un graphe cordal, et donc la suppression d'une arête peut potentiellement effacer une corde d'un cycle de longueur supérieure à 3. Il en résulte que le moteur d'analyse graphique est difficilement disjoint d'un processus de triangulation. Nous avons donc choisi de traiter conjointement ces deux aspects. Pour rappel, l'algorithme de triangulation a été présenté en 2.2.2 symboliquement de la façon suivante :

- A chaque itération i, un sommet $X_{\sigma(i)}$ du graphe Temp est choisi.
- Si ce sommet ne forme pas une clique avec son ensemble de voisins dans Temp, des arêtes fill-ins sont ajoutées, une nouvelle clique se forme dans la forêt de jonction, et les fill-ins sont ajoutés dans le graphe markovien triangulé.
- Le sommet et ses arêtes adjacentes sont supprimés de Temp et une nouvelle itération commence.

Le choix d'un sommet $X_{\sigma(i)}$ à chaque itération de la triangulation engendre une clique. Il est aisé de déterminer si la clique ainsi formée possède une taille supérieure à T. Supposons qu'une clique formée ne soit pas de taille acceptable et que nous décidions de supprimer une arête adjacente à $X_{\sigma(i)}$. Cette arête $\{X_{\sigma(i)}, X_{\sigma(j)}\}$ était peut-être nécessaire pour assurer la propriété de graphe cordal comme le montre l'exemple 29 : Supposons qu'il existe une sommet $X_{\sigma(z)}$ avec $z < i$ et $z < j$ ($X_{\sigma(z)}$ a donc été éliminé avant $X_{\sigma(i)}$ et $X_{\sigma(j)}$ dans le processus de triangulation), ainsi que les arêtes $\{X_{\sigma(z)}, X_{\sigma(i)}\}$ et $\{X_{\sigma(z)}, X_{\sigma(j)}\}$ dans le graphe triangulé final (G_T). Cette arête, malgré sa suppression, aurait de toute façon été ajoutée en tant que fill-in lors de l'élimination de $X_{\sigma(z)}$ et était donc nécessaire pour assurer la cordalité du graphe. Ainsi, si on souhaite supprimer $\{X_{\sigma(i)}, X_{\sigma(j)}\}$, il est nécessaire de supprimer également $\{X_{\sigma(z)}, X_{\sigma(i)}\}$ ou $\{X_{\sigma(z)}, X_{\sigma(j)}\}$

afin de préserver la cordalité du graphe.

Exemple 29 *Considérons le graphe G_M de la figure 3.5(a) et l'ordre d'élimination X_1, X_2, X_3, X_4, X_5, X_6 (donc, pour alléger les notations, nous aurons pour cet exemple $X_{\sigma(i)} = X_i$). Le graphe triangulé G_T correspondant sera donc celui représenté dans la figure 3.5(b).*



(a) G_M (b) G_T

FIGURE 3.5 – Les graphes de l'exemple 29

Supposons que nous décidions de supprimer $\{X_4, X_5\}$. Cette arête est un fill-in dû à l'élimination de X_3. Il est donc nécessaire que X_3 ne soit pas voisin à la fois de X_4 et X_5 lors de la triangulation. Il est donc nécessaire de supprimer également soit $\{X_3, X_5\}$, soit $\{X_3, X_4\}$. Toutefois, cela ne suffit pas. Supposons que nous ayons décidé de supprimer $\{X_3, X_4\}$ comme arête supplémentaire. En suivant le même raisonnement, il sera nécessaire de supprimer en plus une arête parmi $\{X_2, X_3\}$ et $\{X_2, X_4\}$, la suppression de $\{X_2, X_3\}$ engendrant elle-même un choix de suppression entre $\{X_1, X_2\}$ et $\{X_1, X_3\}$.

Si nous associons à chaque arête $\{X_i, X_j\}$ une variable $e_{ij} \in \{0, 1\}$ telle que $e_{ij} = 1$ si nous décidons de supprimer l'arête, et 0 sinon, le problème de la suppression de $\{X_4, X_5\}$ sera résolu par le système :

$$\begin{cases} e_{23} \leqslant e_{12} + e_{13} \\ e_{34} \leqslant e_{23} + e_{24} \\ e_{45} \leqslant e_{34} + e_{35} \\ e_{45} = 1 \end{cases}$$

L'idée de ce système linéaire réside dans le fait que chaque contrainte $e_{ij} \leqslant e_{ik} + e_{jk}$ oblige e_{ik} ou e_{jk} à être égal à 1 dès lors que $e_{ij} = 1$. Autrement dit, la suppression de l'arête $\{X_i, X_j\}$ implique aussi celle de $\{X_i, X_k\}$ et/ou $\{X_j, X_k\}$. Pour formaliser cette façon de procéder, il est nécessaire d'associer à chaque arête un ensemble d'arêtes qui seront impliquées lors d'une suppression, ce qui nous amène à définir les prédecesseurs d'une arête.

Définition 71 (Prédecesseurs d'une arête) *Soit $G_T = (V_T, E_T)$ un graphe triangulé selon l'ordre d'élimination σ. Soit trois nœuds $X_{\sigma(k)}, X_{\sigma(r)}, X_{\sigma(z)} \in V_T$ tels que E_T contient les arêtes $(X_{\sigma(k)}, X_{\sigma(r)})$, $(X_{\sigma(z)}, X_{\sigma(k)})$ et $(X_{\sigma(z)}, X_{\sigma(r)})$. $X_{\sigma(z)}$ est un nœud prédécesseur de $(X_{\sigma(k)}, X_{\sigma(r)})$ si et seulement si $z < k$ et $z < r$. On note $\mathrm{Pred}(X_{\sigma(k)}, X_{\sigma(r)})$ l'ensemble des nœuds prédécesseurs de $(X_{\sigma(k)}, X_{\sigma(r)})$.*

On définit l'ensemble des arêtes prédécesseurs de $(X_{\sigma(k)}, X_{\sigma(r)})$ par $\mathcal{P}(X_{\sigma(k)}, X_{\sigma(r)}) = \{(X_{\sigma(z)}, X) \in E_T / X_{\sigma(z)} \in \mathrm{Pred}(X_{\sigma(k)}, X_{\sigma(r)})$ et $X \in \{X_{\sigma(k)}, X_{\sigma(r)}\}\}$. Autrement dit, $\mathcal{P}(X_{\sigma(k)}, X_{\sigma(r)})$ est l'ensemble des arêtes adjacentes à $(X_{\sigma(k)}, X_{\sigma(r)})$ et à un prédécesseur de $(X_{\sigma(k)}, X_{\sigma(r)})$.

Soit $\mathcal{S}(X_{\sigma(k)}, X_{\sigma(r)}) = \{S \subseteq \mathcal{P}(X_{\sigma(k)}, X_{\sigma(r)})/\forall X_{\sigma(z)} \in \text{Pred}(X_{\sigma(k)}, X_{\sigma(r)}),\ \exists X \in \{X_{\sigma(k)}, X_{\sigma(r)}\}/(X_{\sigma(z)}, X) \in S\}$. $\mathcal{S}(X_{\sigma(k)}, X_{\sigma(r)})$ *est donc l'ensemble des ensembles d'arêtes tels que le sous-graphe induit par* $X_{\sigma(k)}$, $X_{\sigma(r)}$ *et* $\text{Pred}(X_{\sigma(k)}, X_{\sigma(r)})$ *est connexe.*

Par abus de notation, $\forall S \subseteq E_T$, *nous noterons* $\mathcal{S}(S) = \bigcup_{(X_{\sigma(k)}, X_{\sigma(r)}) \in S} \mathcal{S}(X_{\sigma(k)}, X_{\sigma(r)})$.

Avec une telle définition, il est aisé de voir que la suppression d'une arête $\{X_{\sigma(k)}, X_{\sigma(r)}\}$ doit s'accompagner de la suppression d'un sous-ensemble S^1 d'arêtes tel que $S^1 \in \mathcal{S}(X_{\sigma(k)}, X_{\sigma(r)})$. De la même façon, la suppression des arêtes de S^1 devra s'accompagner de la suppression d'un ensemble d'arêtes $S^2 \in \mathcal{S}(S^1)$ et ainsi de suite... de manière à ce que le graphe résultant soit cordal (ce qui est traduit par la notion de suppression saine dans la définition 72). La proposition 3 établit le lien entre les définitions précédentes et la notion de suppression saine. Conformément au raisonnement que nous avons présenté dans l'exemple 29, cette proposition introduit un système de contraintes à variables booléennes exploitant la notion de prédécesseur, de manière à montrer que le respect des contraintes implique une suppression saine.

Définition 72 (Suppression saine) *Soit* $G_T = (V_T, E_T)$ *un graphe cordal, et* $S \subseteq E_T$. *On dit que* S *est une suppression saine si, et seulement si, le graphe* $G'_T = (V_T, E_T \backslash S)$ *est cordal.*

Proposition 3 *Soit* $G_T = (V_T, E_T)$ *un graphe cordal obtenu par triangulation selon l'ordre d'élimination* σ, *et* C *un ensemble de contraintes sur des variables booléennes* $e_{kr} \in \{0, 1\}$ $\forall \{X_{\sigma(k)}, X_{\sigma(r)}\} \in E_T$, *tel que* C *vérifie :*

$$e_{kr} \leqslant e_{kz} + e_{zr}\ \forall \{X_{\sigma(k)}, X_{\sigma(r)}\} \in E_T\ \forall X_{\sigma(z)} \in \text{Pred}(X_{\sigma(k)}, X_{\sigma(r)}).$$

Posons $S = \{\{X_{\sigma(k)}, X_{\sigma(r)}\} \in E_T / e_{kr} = 1\}$, *alors on a :*

$$\{e_{kr}\} \textit{ vérifie les contraintes de } C \implies S \textit{est une suppression saine.}$$

Preuve. Cette démonstration s'obtient en se basant sur la procédure de triangulation : si une triangulation de $(V_T, E_T \backslash S)$ n'ajoute aucun fill-in, alors le graphe est cordal. Triangulons $(V_T, E_T \backslash S)$ selon l'ordre d'élimination σ, et notons i le plus petit entier tel qu'à la i-ième itération (élimination de $X_{\sigma(i)}$), la procédure doive ajouter un fill-in entre $X_{\sigma(k)}$ et $X_{\sigma(r)}$. On aura donc $\{X_{\sigma(k)}, X_{\sigma(r)}\} \in S$ (sinon le fill-in aurait aussi été ajouté dans le graphe triangulé G_T). De plus, $X_{\sigma(k)}$ et $X_{\sigma(r)}$ sont des voisins de $X_{\sigma(i)}$ vérifiant $k > i$ et $r > i$ (sinon aucun fill-in n'aurait été ajouté). Or, il existe une contrainte dans C vérifiant $e_{kr} \leqslant e_{ki} + e_{ir}$, l'appartenance à S de l'arête entraîne donc que $e_{kr} = 1$, donc on aura $e_{ki} = 1$ ou $e_{ir} = 1$ ce qui contredit le voisinage de $X_{\sigma(k)}$ et $X_{\sigma(r)}$, et donc l'ajout de fill-in.

Donc, par l'absurde, il n'existe pas d'itération ajoutant de fill-in dans une triangulation de $(V_T, E_T \backslash S)$. Donc, $(V_T, E_T \backslash S)$ est cordal et S est une suppression saine. ■

La proposition 3 introduit donc un système de contraintes comportant $|E_T|$ variables booléennes, chaque variable e_{ij} étant associée à l'arête $\{X_{\sigma(i)}, X_{\sigma(j)}\}$. Si $e_{ij} = 1$, alors l'arête doit être supprimée du graphe G_T, sinon elle doit être conservée. On peut remarquer qu'en posant $e_{ij} = 0$ $\forall \{X_{\sigma(i)}, X_{\sigma(j)}\} \in E_T$, le système de contraintes sera vérifié, et on aura $S = \varnothing$ qui est une suppression saine (ce qui est logique, puisque G_T est cordal). L'intérêt d'un tel système est de « fixer » la suppression d'une arête, c'est-à-dire

qu'à partir du moment où on décide de supprimer une arête $\{X_{\sigma(i)}, X_{\sigma(j)}\}$, il suffit de poser $e_{ij} = 1$ et de déterminer une solution de ce système de contraintes pour obtenir un ensemble S d'arêtes à supprimer pour garantir la cordalité du graphe.

Toutefois, une telle modélisation n'est pas suffisante : il suffirait par exemple de poser $e_{kr} = 1\ \forall k \forall r$ pour vérifier le système de contraintes. Or, nous n'avons aucune envie de supprimer toutes les relations de dépendance entre attributs du réseau GAI, mais uniquement de supprimer un minimum d'arêtes pour obtenir une approximation du réseau qui soit la plus proche possible du réseau initial, tout en contournant les problèmes computationels dûs à une trop forte treewidth. Il est important de bien voir que chaque approximation générée augmentera le nombre de solutions à énumérer lors du rangement du réseau approximé. Doit-on supprimer un minimum d'arêtes ? Nous avons choisi de ne pas nous baser sur ce critère, car toutes les suppressions d'arêtes ne sont pas équivalentes entre elles. En effet, nous pouvons aisément imaginer que la suppression d'une arête augmente drastiquement le nombre de solutions à ranger, alors que la suppression de deux autres arêtes ne l'augmente que très légèrement, car elles génèrent des approximations presque parfaites. Nous avons donc choisi de prendre plutôt en compte la qualité d'approximation que l'on pourrait obtenir pour choisir quelles arêtes supprimer.

Idéalement, une distance d'approximation parfaite serait une fonction $d : 2^{E_T} \rightarrow \mathbb{R}$ permettant de discriminer les ensembles d'arêtes à supprimer, en posant $d(A) < d(B)$ si la suppression de l'ensemble d'arêtes composant A engendre une approximation de meilleure qualité que la suppression des arêtes de B. Il suffit alors de chercher l'ensemble $S \subseteq E_T$ minimisant $d(S)$ tout en respectant le système de contraintes précédemment énoncé. En pratique, une telle fonction nécessite non seulement de déterminer énormément de valeurs (un coût en mémoire en $O(e^{n^2})$ pour une représentation exhaustive), mais demande aussi de déterminer la qualité d'approximation que l'on pourrait obtenir avec n'importe quel ensemble d'arêtes. La complexité d'un tel système ne conduit pas à une méthode opérationnelle en pratique.

Pour contourner ce problème, nous allons supposer que la fonction d est additivement décomposable sur les arêtes composant son argument. Ainsi, nous aurons $d(S) = \sum_{e \in S} d(e)$ et le problème de la détermination d'un ensemble d'arêtes d'approximation minimale à supprimer pourra s'exprimer sous la forme du programme linéaire suivant, quelle que soit la fonction d utilisée (pour des raisons de clarté, nous noterons $N(i) = \{j > i / \{X_{\sigma(i)}, X_{\sigma(j)}\} \in E_T\}$) :

$$(A) \begin{cases} \min_{e_{kr}} & \sum_{\{X_{\sigma(k)}, X_{\sigma(r)}\} \in E_T} d(\{X_{\sigma(k)}, X_{\sigma(r)}\}) e_{kr} \\ \text{s.c.} & e_{kr} \leqslant e_{kz} + e_{zr} \forall \{X_{\sigma(k)}, X_{\sigma(r)}\} \in E_T\ \forall X_{\sigma(z)} \in \text{Pred}(X_{\sigma(k)}, X_{\sigma(r)}) \\ & \log(|X_{\sigma(i)}|) + \sum_{j \in N(i)} \log(|X_{\sigma(j)}|)(1 - e_{ij}) \leqslant log(T)\ \forall i \in \{1, \ldots, n\} \\ & e_{kr} \in \{0, 1\}\ \forall \{X_{\sigma(k)}, X_{\sigma(r)}\} \in E_T \end{cases}$$

Le programme (A) permet d'intégrer toutes les notions vues précédemment :

- À chaque arête $\{X_{\sigma(k)}, X_{\sigma(r)}\} \in E_T$ est associée une variable booléenne e_{kr}.
- La fonction objectif associe à chaque arête supprimée (symbolisée par la variable e_{kr}) sa distance d'approximation $d(\{X_{\sigma(k)}, X_{\sigma(r)}\})$. Ainsi, l'objectif est de minimiser la distance d'approximation de l'ensemble $S = \{\{X_{\sigma(k)}, X_{\sigma(r)}\} \in E_T / e_{kr} = 1\}$.
- L'ensemble de contraintes $e_{kr} \leqslant e_{kz} + e_{zr}$ est une application de la proposition 3

et permet d'assurer que l'ensemble S des arêtes à supprimer génère bien une suppression saine. Donc, le graphe $G_T = (V_T, E_T \backslash S)$ est cordal.

- Les contraintes $\log(|X_{\sigma(i)}|) + \sum_{j \in N(i)} \log(|X_{\sigma(j)}|)(1 - e_{ij}) \leqslant log(T)$ permettent d'assurer l'acceptabilité des tailles des cliques produites dans la forêt de jonction après suppression des arêtes dans le graphe triangulé : une hypermatrice associée à une clique ne pourra contenir plus de T valeurs. Si tel n'est pas le cas dans la forêt de jonction associée à G_T, alors ces contraintes initient la suppression d'arêtes.

Toutefois, un problème subsiste : Ce programme à variables booléennes contient $O(|E_T|)$ variables et $O(n|E_T|)$ contraintes. Résoudre un programme linéaire à variables booléennes peut entraîner un temps de calcul prohibitif. Pour pallier ce problème, il est possible d'approximer le résultat de (A) par programmation dynamique en définissant une fonction de coût cumulée c à minimiser de la façon suivante :

- On associe à une arête n'ayant aucun prédecesseur dans la triangulation sa distance d'approximation.
- S'il existe au moins un prédecesseur, on associe à cette arête sa distance d'approximation à laquelle on ajoute, pour chaque prédecesseur $X_{\sigma(z)}$, le plus petit coût cumulé de $\{X_{\sigma(k)}, X_{\sigma(z)}\}$ et $\{X_{\sigma(z)}, X_{\sigma(r)}\}$. Autrement dit, si nous devons supprimer $\{X_{\sigma(k)}, X_{\sigma(z)}\}$ ou $\{X_{\sigma(z)}, X_{\sigma(r)}\}$, on préfèrera supprimer parmi ces deux arêtes celle dont le coût d'approximation est le plus petit.

Supposons que nous décidions de la suppression d'une arête $\{X_{\sigma(k)}, X_{\sigma(r)}\}$. Si elle n'a pas de prédecesseur, elle peut donc se supprimer sans engendrer de suppression supplémentaire. Sinon, en évaluant la fonction de coût cumulé, et en supprimant les arêtes engendrant les plus petits coûts cumulés utilisés dans la somme, le système de contraintes de la proposition 3 sera respecté, et la fonction de coût cumulé sera une estimation de la distance d'approximation de toutes les suppressions d'arêtes. D'un point de vue mathématique, cette approche se formalise de la façon suivante :

$$c(X_{\sigma(k)}, X_{\sigma(r)}) = \begin{cases} d(\{X_{\sigma(k)}, X_{\sigma(r)}\}) \\ \text{si } \mathrm{Pred}(X_{\sigma(k)}, X_{\sigma(r)}) = \varnothing \\ \\ d(\{X_{\sigma(k)}, X_{\sigma(r)}\}) + \\ \sum_{X_{\sigma(z)} \in \mathrm{Pred}(X_{\sigma(k)}, X_{\sigma(r)})} \min\{c(\{X_{\sigma(k)}, X_{\sigma(z)}\}), c(\{X_{\sigma(z)}, X_{\sigma(r)}\})\} \\ \text{sinon} \end{cases}$$

Ainsi, l'exécution d'une analyse graphique pourra s'exécuter de la façon suivante :

- On triangule le graphe markovien par élimination d'attributs.
- Si à une itération donnée de l'algorithme de triangulation, une clique trop grosse est formée, on sélectionne des arêtes à supprimer de coût cumulé minimal (en exécutant la programmation dynamique sur le graphe triangulé courant).
- A la fin de la triangulation, on génère le graphe triangulé, on supprime l'ensemble des arêtes sélectionnées par la programmation dynamique et on déduit les découpages de cliques nécessaires au moteur d'approximation.

Concernant la distance d à utiliser, elle peut être multiple, à partir du moment où sa valeur est plus faible sur une arête que l'on préfère supprimer par rapport à une autre arête. Dans le cadre de nos expérimentations, nous avons utilisé une façon simple d'obtenir une distance : Lors de la conception du moteur d'approximation, nous avons défini par

programmation linéaire une manière de générer les hypermatrices approximantes à partir d'une hypermatrice représentant une sous-fonction d'utilité et d'un ensemble de découpage. Un tel programme linéaire minimise une fonction objectif servant d'indicateur sur la qualité d'approximation. Pour connaître la distance d'approximation associée à l'arête $\{X_i, X_j\}$, il suffit donc de déterminer l'ensemble des hypermatrices contenant à la fois l'attribut X_i et l'attribut X_j, d'exécuter les programmes linéaires permettant de séparer chaque table $u_j(X_i, X_j X_A)$ en $\hat{u}_j^1(X_i, X_A)$ et $\hat{u}_j^2(X_j, X_A)$, et d'additionner les valeurs de l'objectif à l'optimum. En procédant de cette façon, nous disposons d'une fonction de distance d'approximation exploitant directement les méthodes d'approximation que nous utilisons.

Remarque 19 *Nous n'avons pas expliqué comment déterminer le seuil T de l'algorithme. Il s'agit d'un paramètre pouvant être déterminé par l'utilisateur de l'algorithme en fonction des caractéristiques techniques de son ordinateur (en supposant que l'algorithme s'exécute sur une machine dédiée). De manière plus simple, on peut se demander dans le cas où tous les attributs sont binaires, quelle treewidth t on est prêt à accepter, effectuer le calcul de ce seuil (2^{t+1}), et utiliser ce paramètre dans le cas général.*

3.3 Analyse expérimentale de l'algorithme

Afin d'évaluer la viabilité de cette approche, nous avons comparé les temps de calcul de l'algorithme classique de choix optimal avec l'algorithme en trois moteurs proposé dans ce chapitre. Pour ce faire, nous avons dans un premier temps généré des graphes makoviens contenant 20 attributs, chaque attribut pouvant contenir de 3 à 5 valeurs. Un nombre croissant de sous-fonctions d'utilité sont insérées en formant une clique entre les attributs sur lesquels elles sont définies, leur contenu étant généré aléatoirement avec des valeurs comprises entre 0 et 1000. La structure de forêt de jonction est finalement obtenue par triangulation du graphe markovien, et les sous-fonctions associées à une même clique sont additionnées entre elles. Enfin, les deux algorithmes sont exécutés sur les mêmes instances (avec un seuil pour notre algorithme correspondant à $T = 500000$).

La figure 3.6 représente les résultats de comparaison obtenus. Sur l'axe des abscisses, nous avons représenté la somme des tailles des cliques du réseau GAI généré (donc $\sum_{C_j \in C} \prod_{i \in C_j} |X_i|$) afin de servir d'indicateur de la complexité du réseau. Sur l'axe des ordonnées, nous avons représenté les temps d'exécution des deux algorithmes en secondes. Afin de rendre la courbe plus lisible, l'axe des ordonnées est représenté sur une échelle logarithmique. Il apparaît clairement qu'il existe un seuil (vers une somme de taille de 10^6) à partir duquel l'approche en trois moteurs devient plus avantageuse à utiliser que l'algorithme de choix optimal. En deçà de ce seuil, les précalculs effectués par notre algorithme pour construire un réseau approximé (correspondant au réseau initial lorsque la plus grande clique possède une taille inférieure à T) et l'utilisation du rangement le rendent désavantageux par rapport à l'algorithme classique. Au delà de ce seuil, nous pouvons voir qu'il prend tout son intérêt.

Afin d'évaluer plus finement l'algorithme, nous avons cherché dans la littérature un algorithme tentant de traiter des instances complexes tout en étant capable de renvoyer la solution optimale du problème original. Kask et Dechter (1999) ont introduit l'approche par Mini-Buckets avec Branch & Bound pour traiter le problème MPE (Most

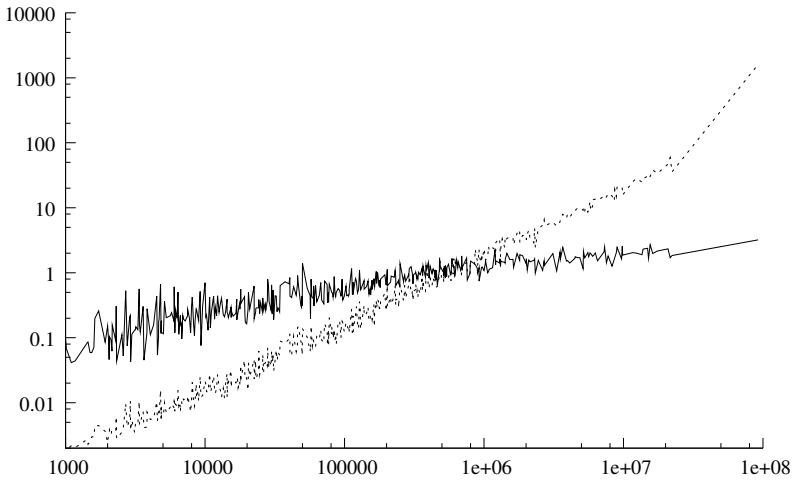


FIGURE 3.6 – Comparaison des temps de calcul entre l'algorithme de choix optimal (en pointillé) et l'approche en trois moteurs (en ligne pleine)

Probable Explanation) dans le domaine des réseaux bayésiens. Cette approche peut être adaptée aux réseaux GAI de façon à résoudre le problème du choix optimal. Nous avons implémenté cette adaptation de l'algorithme et nous avons effectué une comparaison avec l'approche en trois moteurs de la même façon que précédemment, à la seule différence que les réseaux GAI générés contiennent maintenant 40 attributs.

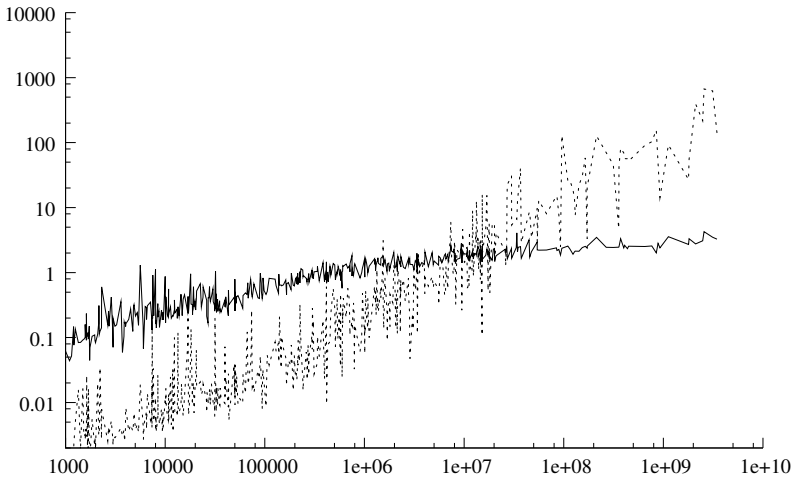


FIGURE 3.7 – Comparaison des temps de calcul entre Mini-Buckets avec Branch & Bound (en pointillé) et l'approche en trois moteurs (en ligne pleine)

La figure 3.7 montre qu'il existe de nouveau un seuil (aux alentours de 10^7 cette fois-ci) à partir duquel l'algorithme en trois moteurs est plus efficace que l'adaptation des Mini-Buckets avec Branch & Bound. Ce seuil existe de nouveau car la méthode de génération du réseau approximant utilisée dans notre algorithme procède à une analyse plus fine du réseau. En deçà du seuil, cette analyse est bien plus coûteuse en temps de calcul que les maximisations et le Branch&Bound des Mini-Buckets, mais sur des plus

grandes tailles de problème, nous pouvons nous apercevoir qu'une telle analyse prend tout son sens.

3.4 Extensions de l'algorithme

Au sein de cette section, nous allons généraliser cette approche pour deux autres problèmes présentés dans les chapitres précédents : le rangement d'un réseau GAI et la détermination de la solution agrégée optimale. Puis, nous discuterons de la modularité de cet algorithme et de la possibilité de remplacer un « moteur » par un autre sans impact sur le bon déroulement de l'algorithme. Finalement, nous montrerons que l'approche présentée dans ce chapitre ouvre la voie vers un nouveau concept de triangulation.

3.4.1 Rangement sous forte treewidth

Nous avons présenté dans les sections précédentes une nouvelle approche pour résoudre le problème de choix dans un contexte de réseau GAI de forte treewidth. Cette approche peut être étendue au problème de rangement dans ce même contexte : Supposons que le réseau GAI R soit approximé par un réseau GAI $\hat{R}$ en utilisant les mêmes techniques que celles vues précédemment, et que nous désirons déterminer les k meilleures alternatives $x^1, \ldots, x^k$ en fonction de l'utilité u représentable par le réseau GAI R. Appliquer la même approche que pour le choix optimal revient à déterminer un nouveau critère d'arrêt du rangement de $\hat{R}$ et à exploiter les alternatives itérées dans le moteur de rangement.

Du point de vue du stockage, il suffit d'implémenter un tas permettant de stocker les informations au fur et à mesure des itérations, l'idée étant de disposer de la primitive `Inserer`(x) fonctionnant de la façon suivante :

- Si la structure ne contient pas k alternatives, x est inséré dans le tas (et indexé par $u(x)$).
- Si la structure contient k alternatives et si $u(x)$ possède une valeur supérieure à la plus petite utilité du tas, alors x est indexé dans le tas, et la tête du tas (d'utilité minimale) est supprimée.
- Si la structure contient k alternatives et si $u(x)$ ne possède pas une valeur supérieure à la plus petite utilité du tas, la structure n'est pas modifiée.

L'insertion sera donc réalisée en $O(\log(k))$, et il sera donc aisé en fin d'algorithme de convertir cette structure en un tableau d'alternatives triées selon leur valeur d'utilité (même principe que le tri par tas, soit un coût de conversion en $O(k \log(k))$).

La détermination du point d'arrêt sera fera d'une manière similaire au problème de choix optimal, l'unique différence étant qu'au lieu de comparer l'utilité approchée de l'itération courante avec la meilleure utilité stockée jusqu'à présent, nous la comparerons avec la plus petite utilité stockée dans notre structure de solutions (donc la tête de tas, accessible en $O(1)$).

Proposition 4 (Condition suffisante d'arrêt (rangement de u)) *Soit $u : \mathcal{X} \to \mathbb{R}$ une fonction d'utilité, et $\hat{u} : \mathcal{X} \to \mathbb{R}$ une fonction vérifiant $\forall x \in \mathcal{X}\ \hat{u}(x) \geqslant u(x)$. Notons $x^1, \ldots, x^k$ les k meilleures alternatives selon u, $y^1, \ldots, y^z$ les z meilleures alternatives (selon $\hat{u}$), et $\bar{y}$ la k-ième meilleure alternative des y^i selon u (pour $i \in \{1, \ldots, z\}$), alors*

on a :

$$\hat{u}(y^z) < u(\bar{y}) \implies \bar{y} = x^k$$

Preuve. La démonstration est similaire à la condition suffisante d'arrêt utilisée pour le problème du choix optimal. Supposons que $\hat{u}(y^z) < u(\bar{y})$. Par définition du rangement, on aura donc que $\forall l > z\ \hat{u}(y^z) \geqslant \hat{u}(y^l)$, et par définition de $\hat{u}$, on aura que $\hat{u}(y^l) \geqslant u(y^l)$. Donc, on aura $\forall l > z\ u(\bar{y}) > u(y^l)$. Or, $\bar{y}$ est définie comme la k-ième meilleure alternative des y^i selon u (pour $i \in \{1, \ldots, z\}$), donc $\bar{y}$ est la k-ième meilleure alternative sur tout $\mathcal{X}$, donc $\bar{y} = x^k$. ■

Il résulte de la condition d'arrêt l'algorithme 15 permettant de résoudre le problème de rangement dans un contexte de forte treewidth.

Algorithme 15 : RangementTreewidth

Entrée : R : réseau GAI représentant l'utilité u

```
1  R̂ ← réseau GAI (d'utilité û) bornant supérieurement u;
2  Ranking ← MoteurRangement(R̂);
3  Structure ← InitStructure();
4  x ← Ranking.solutionSuivante();
5  Structure.Insérer(x, u(x));
6  BorneSup ← û(x);
7  tant que Structure.PlusPetiteUtilité() ⩽ BorneSup et Ranking.pasFini()
   faire
8  |  x ← Ranking.solutionSuivante();
9  |  BorneSup ← û(x);
10 |  Structure.Insérer(x, u(x));
11 fin
12 retourner Structure ;
```

3.4.2 Solution agrégée optimale sous forte treewidth

Dans la section 2.4.2, nous avons présenté une approche par rangement pour résoudre des problèmes de décision multicritère et multiattribut (la détermination de la solution agrégée optimale). En reprenant cette approche et en la mêlant avec l'algorithme de rangement sous forte treewidth, il est possible de déduire directement un algorithme de détermination de solution agrégée optimale sous forte treewidth. Cet algorithme est d'autant plus intéressant que l'algorithme originel se base sur une agrégation des réseaux GAI représentant les critères (le réseau GAI scalarisé). Or, si les structures de dépendances sont dissemblables entre les différents critères, la treewidth du réseau GAI scalarisé aura tendance à être élevée.

De plus, la construction du réseau GAI approché se base sur une analyse de la structure du réseau agrégé pour déterminer un découpage des utilités. Dans ce cas de figure, le lien entre utilité scalarisée et utilité des réseaux GAI d'origine peut être exploité. Ainsi, le moteur d'analyse graphique étudiera la structure du réseau scalarisé, mais le moteur d'approximation sera exploité directement sur les sous-fonctions d'utilité propres à chaque critère.

3.4.3 Extension des moteurs

L'approche en trois moteurs peut être représentée symboliquement par la figure 3.8 : La construction du réseau GAI approché nécessite un moteur d'analyse graphique qui génèrera des découpages pour le moteur d'approximation. Ce dernier génèrera les sous-fonctions approchées nécessaires à la construction du réseau. Finalement, le moteur de rangement appliqué au réseau approché permettra de déduire la solution du problème. Nous avons incorporé dans le moteur d'analyse une notion de distance renforçant les interactions entre analyse et approximation dans la formulation que nous lui avons donné.

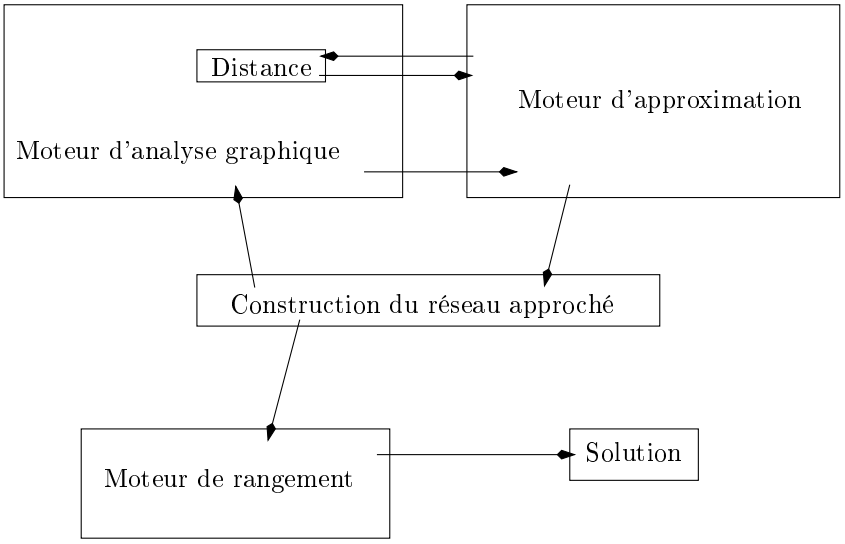


FIGURE 3.8 – Vision symbolique de l'approche par moteurs

L'algorithme présenté au sein de ce chapitre est tout à fait adaptable à de nouveaux moteurs, et il est sûrement possible de l'améliorer en se basant sur les constatations suivantes :

- La notion de distance : Nous avons choisi une formulation simple de la notion de distance d'approximation en utilisant un moteur d'approximation fondé sur la programmation linéaire. Toutefois, la distance peut devenir un moteur à part entière et on peut explorer de nouvelles voies telles que l'adaptation de la divergence de Kullback-Leibler (Kullback et Leibler, 1951) aux fonctions d'utilité, ou encore en réussissant à déterminer une manière de calculer la divergence entre deux relations de préférence.
- L'analyse graphique : Nous avons formulé l'analyse graphique comme un problème général mais trop complexe à résoudre. Cette constatation nous a amené à faire certaines hypothèses (distance générale décomposable en somme des distances liées aux arêtes) pour reformuler le problème général en un problème approché solvable en un temps convenable par programmation dynamique. Il est possible d'imaginer de nouvelles techniques d'analyse graphique afin d'améliorer le choix des arêtes à supprimer du réseau approché.
- L'approximation : Lors de l'élaboration du moteur, nous avons défini un système de contraintes à respecter. Ce système est indispensable au bon fonctionnement de l'algorithme (sinon le critère d'arrêt du moteur de rangement ne pourra être vérifié).

Toutefois, nous avons choisi une fonction objectif qui nous semblait intéressante pour générer de bonnes approximations, mais il se peut que d'autres techniques améliorent la qualité des approximations obtenues.

3.4.4 Vers une nouvelle approche de la triangulation

De façon plus générale, sans entrer dans les détails des différents moteurs ou dans la génération des sous-fonctions, les travaux que nous avons réalisés pour développer cette nouvelle approche peuvent être vus comme la possibilité de supprimer des arêtes dans le graphe markovien tout en ayant un algorithme capable de résoudre le problème initial de triangulation. La notion de triangulation utilisée pour générer nos réseaux GAI agrégés ou approchés consiste en général à partir d'un graphe markovien $G_M = (V_M, E_M)$ et à déterminer un ensemble d'arêtes fill-ins T à ajouter au graphe pour le rendre cordal. On obtient ainsi le graphe triangulé $G_T = (V_M, E_M \cup T)$. La possibilité de supprimer des arêtes peut donc nous conduire à une nouvelle approche de la triangulation consistant en la détermination d'un ensemble de fill-ins T et un ensemble d'arêtes à supprimer D tels que $G_T = (V_M, (E_M \backslash D) \cup T)$ soit cordal.

Chapitre 4

Détermination exacte et approchée de la frontière de Pareto [1]

Résumé. *Dans ce chapitre, nous présenterons un algorithme permettant le calcul de la frontière de Pareto lorsque les m critères sont représentés par des réseaux GAI. Dans un premier temps (section 4.1) nous présenterons la notion de réseau GAI vectoriel. Nous démontrerons la NP-complétude du problème et nous proposerons un algorithme exact pour le résoudre. Puis, dans la section 4.2, face aux problèmes computationels résultant de la NP-complétude du problème, nous adapterons l'algorithme exact pour proposer un algorithme approché avec garantie de performance. Finalement, les deux algorithmes seront analysés théoriquement et expérimentalement dans la section 4.3.*

1. Travaux publiés dans Dubus et al. (2009d)

Le reste de cette thèse se situera dans un contexte mêlant décision multiattribut et décision multicritère. Dans le chapitre 2, nous avions présenté différentes problématiques liées à l'aspect multicritère, ainsi qu'un algorithme pour résoudre les problèmes de détermination de la solution agrégée optimale. Le but de ce chapitre est de traiter un des problèmes que nous avions laissé de côté : la détermination de la frontière de Pareto. Dans un premier temps (au sein de la section 4.1), nous introduirons une modélisation du problème adaptée aux réseaux GAI, et nous présenterons un algorithme conceptuellement proche de l'algorithme de choix optimal pour le résoudre. Étant donné que la frontière de Pareto peut potentiellement contenir l'ensemble de toutes les instanciations des attributs du problème, la section 4.2 aura pour but d'exploiter un nouveau concept de dominance permettant de contourner les problèmes de temps de calcul et de représentation mémorielle. Cette section s'achèvera sur l'introduction d'un algorithme d'approximation avec garantie de performance. Finalement, dans la section 4.3, nous analyserons l'algorithme théoriquement et expérimentalement, avant de conclure sur la viabilité de cette approche.

Au sein de ce chapitre, nous supposerons que les fonctions d'utilité associées aux différents critères sont de la forme : $u^j : \mathcal{X} \rightarrow \mathbb{Z}_+$. Cette hypothèse n'est nullement restrictive étant donné qu'il est aisé d'ajouter à chaque sous-utilité une constante qui la rend positive.

4.1 La frontière de Pareto par collecte de messages

Lors du chapitre 2, nous avons introduit les notions nécessaires à l'intégration de l'aspect multicritère dans le modèle GAI. Toutefois, l'algorithme de l'état de l'art se basait sur le réseau GAI scalarisé qui était construit à partir des m réseaux GAI représentant les critères, et une fonction de borne supérieure de l'agrégateur souhaité. Après avoir démontré la NP-complétude du problème de la détermination de la frontière de Pareto (section 4.1.1), nous introduirons un nouveau type de réseau GAI agrégé dans la section 4.1.2 nous permettant de manipuler directement des vecteurs d'utilité. Finalement, nous utiliserons les propriétés de la dominance de Pareto pour déduire un algorithme exploitant les indépendances du réseau pour calculer efficacement la frontière de Pareto (section 4.1.3).

4.1.1 NP-complétude du problème

Nous avons vu précédemment (au sein de l'exemple 7) une première difficulté posée par la frontière de Pareto : l'ensemble des vecteurs dans la frontière peut être potentiellement aussi grand que $\mathcal{X}$. Nous reviendrons sur cette difficulté dans la section 4.2, mais une seconde difficulté apparaît en étudiant le problème du point de vue de la théorie de la complexité.

Définition 73 (P_v : problème de décision (frontière de Pareto)) *On notera P_v le problème de décision (au sens de la théorie de la complexité) associé à la détermination de la frontière de Pareto dont la formulation est :*

- **Données :** *Un ensemble de n attributs X_i avec $\forall i \in \{1,\ldots,n\}|X_i| \geqslant 2$, m ensembles C^j vérifiant $\forall C_i^j \in C^j\ C_i^j \subseteq \{1,\ldots,n\}$, auxquels on associe des fonctions $u_i^j : \prod_{z \in C_i^j} X_z \rightarrow \mathbb{Z}_+$, et un vecteur $v \in \mathbb{Z}_+^m$. On notera $u^j = \sum_{C_i^j \in C^j} u_i^j$.*

– **Question :** *Existe-t-il une instanciation x de $\mathcal{X}$ dont le vecteur d'utilité associé $(u^1(x), \ldots, u^m(x))$ domine au sens de Pareto le vecteur v ?*

La proposition 5 montre que le problème P_v est NP-complet. Il est donc impossible de le traiter en un temps polynômial (à moins que P = NP).

Proposition 5 *P_v est un problème NP-complet.*

Preuve. Nous allons établir la preuve pour un problème bicritère ($m = 2$). Pour ce faire, nous allons considérer la version décisionnelle du problème de sac à dos (que nous noterons KP), dont la formulation est la suivante :

– **Données :** un vecteur d'utilité $(v_1, \ldots, v_m) \in \mathbb{Z}_+^m$, un vecteur de pondération $(w_1, \ldots, w_m) \in \mathbb{Z}_+^m$ et deux entiers U et W.
– **Question :** Existe-t-il $x \in \{0,1\}^m$ tel que $\sum_{j=1}^m v_j x_j \geqslant U$ et $\sum_{j=1}^m w_j x_j \leqslant W$?

Étant donné une instance de KP, construisons (en un temps polynômial) une instance de notre problème : Soit m attributs booléens $X_j = \{0,1\} \forall j \in \{1, \ldots, m\}$, $C_i^1 = C_i^2 = i \; \forall i \in \{1, \ldots, m\}$, et m sous-fonctions d'utilité pour chaque critère vérifiant $u_j^1(x_j) = v_j x_j$ et $u_j^2 = w_j(1 - x_j)$. Ainsi, le vecteur d'utilité associé à une instanciation des attributs correspondra à $u(x) = (u^1(x), u^2(x)) = (\sum_{j=1}^m v_j x_j, \sum_{j=1}^m w_j(1 - x_j))$. Donc, déterminer l'existence d'un vecteur x tel que $\sum_{j=1}^m v_j x_j \geqslant U$ et $\sum_{j=1}^m w_j x_j \leqslant W$ revient à déterminer s'il existe un vecteur x dont l'utilité domine le vecteur $(U, \sum_{j=1}^m w_j - W)$.

KP étant NP-complet (Garey et Johnson, 1979), notre problème est donc NP-difficile. De plus, il est aisé de tester polynômialement si, pour une instanciation x donnée, on a $u(x) \succ_P v$. Donc, P_v est aussi de la classe NP. Le problème P_v est donc NP-complet. ■

4.1.2 Réseau GAI vectoriel

Le but de la nouvelle formulation des réseaux GAI que nous exploiterons au sein de ce chapitre (et du chapitre suivant) est d'utiliser une expression des utilités propice à l'utilisation de la dominance de Pareto, donc une formulation vectorielle. De la même façon que pour la construction du réseau GAI scalarisé, la structure sera bâtie par agrégation afin d'englober toutes les dépendances. Les sous-fonctions d'utilité contenues dans les diverses cliques d'un réseau représentant le j-ème critère seront placées dans la clique correspondante du réseau agrégé, au sein de la j-ème composante d'un vecteur. Le réseau GAI vectoriel sera donc conforme à la définition 74.

Définition 74 (Réseau GAI vectoriel) *Soit $\mathcal{X} = \prod_{i=1}^n X_i$ un espace d'alternatives, $u^1, \ldots, u^m$ des fonctions d'utilité telles que, à chaque fonction u^j, on puisse associer une représentation en réseau GAI $R^j = (F^j, \{u_z^j\})$.*
On dit que le réseau GAI $R = (F, \{u_z\})$ est un réseau GAI vectoriel de $R^1, \ldots, R^m$ si, et seulement si :

– *F est une agrégation des forêts de jonction $F^1, \ldots, F^m$*
– *$\forall x \in \mathcal{X} \; u(x) = (u^1(x), \ldots, u^m(x))$.*

Remarque 20 *Tout le long de ce chapitre, ainsi que dans le chapitre suivant, nous exploiterons des réseaux GAI vectoriels et nous manipulerons des vecteurs à m dimensions. Nous utiliserons les opérations et relations suivantes pour manipuler les vecteurs :*

– *L'addition :* $\forall v \in \mathbb{Z}_+^m \forall w \in \mathbb{Z}_+^m \; v + w = (v^1 + w^1, \ldots, v^m + w^m)$.
– *La soustraction :* $\forall v \in \mathbb{Z}_+^m \forall w \in \mathbb{Z}_+^m \; v - w = (v^1 - w^1, \ldots, v^m - w^m)$.
– *La comparaison :* $\forall v \in \mathbb{Z}_+^m \forall w \in \mathbb{Z}_+^m \; v \leqslant w \Leftrightarrow \forall i \in \{1, \ldots, m\} \; v^i \leqslant w^i$.

De plus, le nom de réseau GAI vectoriel a un sens : on peut en effet étendre la GAI-décomposabilité sur des sous-fonctions d'utilité vectorielles en utilisant l'addition vectorielle. Soit $C = \{C_1, \ldots, C_q\}$ *les cliques du réseau GAI vectoriel, alors on aura* $\forall x \in \mathcal{X} \; u(x) = \sum_{C_i \in C} u_i(x_{C_i})$ *avec* $u_i : X_{C_i} \to \mathbb{Z}_+^m$ *des sous-fonctions à valeurs vectorielles associées aux cliques. Il est donc important de noter qu'à partir de maintenant, toute fonction notée u ou* u_i *sera vectorielle, et que nous ne nous référerons à des fonctions à valeur scalaire uniquement par* u^i *ou* u_j^i.

Le problème de la détermination de la frontière de Pareto au sein d'un réseau GAI vectoriel consistera donc à calculer $\text{ArgND}_P(\mathcal{X}, u) = \{x \in \mathcal{X} / \nexists x' \in \mathcal{X} \; u(x') \succ_P u(x)\}$. Un tel algorithme devra être capable d'exploiter la structure d'indépendance des utilités de manière à optimiser les calculs.

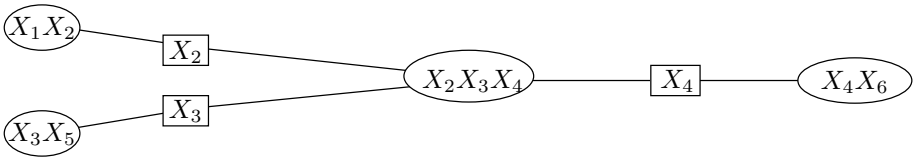


FIGURE 4.1 – Réseau GAI vectoriel de l'exemple 30

Exemple 30 *Considérons le réseau GAI vectoriel de la figure 4.1 et tentons de déterminer comment reconstituer l'ensemble* $\mathcal{U}$ *de vecteurs d'utilité représenté dans ce réseau vectoriel. L'ensemble* $\mathcal{U}$ *peut s'écrire sous la forme*

$$\mathcal{U} = \bigcup_{x \in \mathcal{X}} \{ \sum_{C_j \in C} u_j(x_{C_j}) \}$$

Posons $\mathcal{U}_j$ *la fonction qui, à une instanciation* $x_{C_j} \in X_{C_j}$, *renvoie le singleton* $\{u_j(x_{C_j})\}$, *et* $\boxplus$ *l'opérateur qui à deux ensembles de vecteurs, renvoie l'ensemble des combinaisons possibles par sommation de ces vecteurs, c'est-à-dire :*

$$E \boxplus F = \{v + w / v \in E \text{ et } w \in F\}$$

En pratique, nous n'allons pas manipuler directement des ensembles, mais des sous-fonctions d'utilité renvoyant des ensembles. Nous allons étendre l'opérateur $\boxplus$ *au traitement de ce type de fonctions. Considérons deux fonctions E et F, définies respectivement sur* $X_Y \times X_S$ *et* $X_Z \times X_S$ *avec* $Y \cap Z = \varnothing$, $Y \cap S = \varnothing$ *et* $Z \cap S = \varnothing$. *Nous allons définir* $G = E \boxplus F$ *de la manière suivante : Le domaine de définition de G est* $X_{Y \cup S \cup Z}$, *et son expression est :*

$$\begin{aligned} \forall y \in X_Y \forall s \in X_S \forall z \in X_Z \; G(y, s, z) &= E(y, s) \boxplus F(s, z) \\ &= \{v + w / v \in E(y, s) \text{ et } w \in F(s, z)\} \end{aligned}$$

L'ensemble $\mathcal{U}$ *peut donc se réécrire en exploitant la GAI-décomposabilité du réseau vectoriel :*

$$\mathcal{U} = \bigcup_{x \in \mathcal{X}} \boxplus_{C_j \in C} \mathcal{U}_j(x_{C_j})$$
$$avec \quad \forall C_j \in C \forall x_{C_j} \in X_{C_j}\ \mathcal{U}_j(x_{C_j}) = \{u_j(x_{C_j})\}.$$

De plus, nous pouvons remarquer que les opérateurs $\cup$ *et* $\boxplus$ *vérifient les mêmes propriétés que les opérateurs* max *et* $+$*, utilisés dans l'algorithme de détermination du choix optimal (voir les axiomes présentés dans Shenoy et Shafer (1990)). Donc, en appliquant un raisonnement similaire à cet algorithme, il est possible d'en déduire une réécriture de* $\mathcal{U}$ *propice à un algorithme de collecte :*

$$\begin{cases} \psi_1(x_1, x_2) & = \mathcal{U}_1(x_1, x_2) \\ \phi_{1,3}(x_2) & = \bigcup_{x_1 \in X_1} \psi_1(x_1, x_2) \\ \psi_1(x_3, x_5) & = \mathcal{U}_2(x_3, x_5) \\ \phi_{2,3}(x_3) & = \bigcup_{x_5 \in X_5} \psi_2(x_3, x_5) \\ \psi_3(x_2, x_3, x_4) & = \phi_{1,3}(x_2) \boxplus \phi_{2,3}(x_3) \boxplus \mathcal{U}_3(x_2, x_3, x_4) \\ \phi_{3,4}(x_4) & = \bigcup_{(x_2, x_3) \in X_2 \times X_3} \psi_3(x_2, x_3, x_4) \\ \psi_4(x_4, x_6) & = \phi_{3,4}(x_4) \boxplus \mathcal{U}_4(x_4, x_6) \\ \mathcal{U} & = \bigcup_{(x_4, x_6) \in X_4 \times X_6} \psi_4(x_4, x_6) \end{cases}$$

4.1.3 Collecte vectorielle pour le calcul de la frontière de Pareto

Dans l'exemple 30, nous avons vu que la détermination de $\mathcal{U}$ avait une forme propice à un algorithme de collecte. Nous allons maintenant aborder la détermination de la frontière de Pareto en se basant sur cette idée d'algorithme à collecte vectorielle.

Une différence que nous allons introduire par rapport à l'exemple précédent est que nous ne sommes pas intéressés uniquement par des vecteurs d'utilité, mais aussi par les instanciations les ayant engendrés. Nous ne ferons donc plus transiter de vecteurs par message, mais des associations $\langle v, x \rangle$ où v sera un vecteur, et x l'instanciation d'attributs ayant engendré ce vecteur. Les messages transitant sur les séparateurs seront donc des hypermatrices contenant des ensembles de ces associations. Il est donc nécessaire de modifier l'opérateur $\boxplus$:

Définition 75 (Combinaison ensembliste)
Soit $X_1, \ldots, X_n$ *des attributs,* $Y \subseteq \{1, \ldots, n\}$, $Y' \subseteq \{1, \ldots, n\}$, $S = Y \cap Y'$, E *un ensemble d'associations* $\langle v, x \rangle$ *avec* $v \in \mathbb{Z}_+^m$ *et* $x \in X_Y$, F *un ensemble d'associations* $\langle v', x' \rangle$ *avec* $v' \in \mathbb{Z}_+^m$ *et* $x' \in X_{Y'}$*. L'opérateur de combinaison ensembliste* $\oplus$ *est défini uniquement dans le cas où* $\forall \langle v, x \rangle \in E \forall \langle v', x' \rangle \in F\ x_S = x'_S$ *par :*

$$E \oplus F = \{\langle v + v', x'' \rangle / \langle v, x \rangle \in E \text{ et } \langle v', x' \rangle \in F \text{ et } x'' = (x_Y, x_{Y' \setminus S})\}.$$

Il est à noter qu'en respectant le séquençage de calculs induit par un algorithme de collecte, l'opérateur $\oplus$ est toujours bien défini : les opérations ne se font qu'à valeur de séparateur fixée pour les instanciations. Pour reprendre le parallèle avec l'exemple 30, il faudrait modifier la définition des fonctions $\mathcal{U}_j$ par $\mathcal{U}_j(x_{C_j}) = \{\langle u_j(x_{C_j}), x_{C_j} \rangle\}$.

L'idée de notre algorithme de détermination de la frontière de Pareto est d'exploiter les propriétés de la dominance de Pareto afin de pouvoir supprimer au fur et à mesure de la collecte vectorielle des vecteurs dominés. Pour cela, nous utiliserons la propriété 28.

Propriété 28 *La dominance de Pareto vérifie :*

- *La transitivité :* $\forall v \in \mathbb{R}^m \ \forall w \in \mathbb{R}^m \ \forall z \in \mathbb{R}^m \ v \succ_P w$ *et* $w \succ_P z \implies v \succ_P z$
- *L'additivité :* $\forall v \in \mathbb{R}^m \ \forall w \in \mathbb{R}^m \ \forall z \in \mathbb{R}^m \ v \succ_P w \implies v + z \succ_P w + z$

Exemple 31 *La propriété d'additivé permet d'effectuer des calculs de non-dominance au cours de la phase de collecte : Considérons la fonction d'utilité* $u(x_1, x_2, x_3) = u_1(x_1, x_2) + u_2(x_2, x_3)$*, et supposons qu'il existe des instanciations* x_{11}*,* x_{12} *et* x_{21} *telles que* $u_1(x_{11}, x_{21}) \succ_P u_1(x_{12}, x_{21})$*, alors il n'existera aucune valeur* $x_3 \in X_3$ *telle que* (x_{12}, x_{21}, x_3) *appartienne à la frontière de Pareto. Pour s'en rendre compte, il suffit de réécrire la propriété :*

$$\forall x_3 \in X_3 \ u_1(x_{11}, x_{21}) \succ_P u_1(x_{12}, x_{21}) \implies u_1(x_{11}, x_{21}) + u_2(x_{21}, x_3) \succ_P u_1(x_{12}, x_{21}) + u_2(x_{21}, x_3).$$

Cette réécriture de manière formelle fait clairement apparaître l'utilisation de l'additivité de la dominance de Pareto. L'importance de la transitivité s'exprime par le fait de ne jamais remettre en cause des solutions que nous avons écartées de la frontière de Pareto : Prenons trois vecteurs v*,* w *et* z*. Supposons que* $v \succ_P w$*, on aimerait supprimer* w *de la frontière de Pareto. Cette suppression serait-elle valide si par la suite on a* $z \succ_P v$ *? Par transitivité, on a* $z \succ_P w$*, et donc* w *sera toujours dominé par un vecteur. Il est donc possible d'écrire un algorithme exploitant l'additivité et la transitivté pour effectuer des calculs de dominance au fur et à mesure d'une phase de collecte.*

Les calculs de non-dominance pourront donc s'appliquer directement après une exécution de l'opérateur $\oplus$, ou après une union ensembliste. On en déduit donc directement l'algorithme 16.

Remarque 21 *Pour simplifier la lecture de l'algorithme, nous supposerons l'existence de deux fonctions :*

- `NonDominés` *: Cette fonction prend en argument un ensemble* E *d'associations* $\langle v, x \rangle$ *et a pour valeur de retour un sous-ensemble* $F \subseteq E$ *contenant les associations composant la frontière de Pareto de* E*. Par exemple, l'appel de la fonction* `NonDominés(`$\{\langle (42, 42), x \rangle, \langle (51, 1664), z \rangle, \langle (1664, 51), y \rangle\}$`)` *s'évalue en l'ensemble* $\{\langle (51, 1664), z \rangle, \langle (1664, 51), y \rangle\}$*.*
- `NonDominésParCase` *: Cette fonction prend en argument une hypermatrice* H *d'ensembles d'associations et a pour valeur de retour une hypermatrice correspondant à un appel de* `NonDominés` *sur tous les ensembles que contient* H*. Par exemple, étant donné une fonction* u_i *définie sur* X_{C_i}*, un appel à* `NonDominésParCase(`u_i`)` *s'évalue en une fonction* f *définie sur* X_{C_i} *et vérifiant :* $\forall x \in X_{C_i} \ f(x) =$`NonDominés(`$u_i(x)$`)`*.*

Pour montrer la validité de l'algorithme 16, nous allons dans un premier temps montrer la validité des combinaisons ensemblistes dans le cadre des utilités additives (proposition 6), afin de pouvoir ensuite étendre ce résultat dans le cas d'une utilité GAI-décomposable en deux sous-fonctions (proposition 7). Cette proposition va elle aussi être généralisée au cas où le nombre de cliques est quelconque (proposition 8), pour finalement être exploitée pour montrer la validité de l'algorithme 16 (proposition 9).

Proposition 6 (Combinaison pour des utilités additives) *Soit* (D, E) *une partition de* $\{1, \ldots, n\}$*, et* u *une fonction d'utilité vectorielle additivement décomposable en* $\forall x \in \mathcal{X} \ u(x) = u_1(x_D) + u_2(x_E)$*. Alors, on a :*

$$ND_P(\mathcal{U}) \subseteq ND_P(\{v / \exists x_D \in X_D \ u_1(x_D) = v\}) \boxplus ND_P(\{v / \exists x_E \in X_E \ u_2(x_E) = v\}).$$

Algorithme 16 : CollecteVectorielle

Entrées :
C_a : clique réceptionnant les messages
C_p : clique destinataire du message à créer

$\psi_a \leftarrow$ **créer** hypermatrice vérifiant $\forall x_{C_a} \in X_{C_a}\ \psi_a(x) = \{\langle u_a(x_{C_a}), x_{C_a}\rangle\}$;
pour chaque C_b *clique voisine de* C_a **faire**
  si $C_b \neq C_p$ **alors**
    $\phi_{b,a} \leftarrow$ `CollecteVectorielle`(C_b, C_a);
    $\psi_a \leftarrow$ `NonDominésParCase`$(\psi_a \oplus \phi_{b,a})$;
  fin
fin
si $C_a \neq C_p$ **alors**
  $\phi_{a,p} \leftarrow$ **créer** hypermatrice définie sur $X_{S_{ap}}$;
  pour tous les $x_{S_{ap}} \in X_{S_{ap}}$ **faire**
    $\phi_{a,p}(x_{S_{ap}}) \leftarrow$ `NonDominés`$(\bigcup_{y_{C_a \setminus S_{ap}} \in X_{C_a \setminus S_{ap}}} \psi_a(x_{S_{ap}}, y_{C_a \setminus S_{ap}}))$;
  fin
  retourner $\phi_{a,p}$;
sinon
  retourner `NonDominés`$(\bigcup_{x_{C_a} \in X_{C_a}} \psi_a(x_{C_a}))$;
fin

Preuve. Considérons deux instanciations x_D et y_D de X_D, telles que $u_1(x_D) \succ_P u_2(y_D)$. Par additivité de la dominance de Pareto, on aura donc $\forall x_E \in X_E\ u_1(x_D) + u_2(x_E) \succ_P u_1(y_D) + u_2(x_E)$. Il n'existe donc pas de vecteur dans $\mathrm{ND}_P(\mathcal{U})$ résultant d'une addition vectorielle entre $u_1(y_D)$ et un vecteur $u_2(x_E)$. Pour la même raison, il ne peut exister de vecteur dans $\mathrm{ND}_P(\mathcal{U})$ résultant de l'addition de $u_1(x_D)$ avec un vecteur dominé $u_2(y_E)$. Étant donné que $\mathcal{U} = \{v/\exists x_D \in X_D\ u_1(x_D) = v\} \boxplus \{v/\exists x_E \in X_E\ u_2(x_E) = v\}$ et $D \cap E = \varnothing$, on obtient donc $\mathrm{ND}_P(\mathcal{U}) \subseteq \mathrm{ND}_P(\{v/\exists x_D \in X_D\ u_1(x_D) = v\}) \boxplus \mathrm{ND}_P(\{v/\exists x_E \in X_E\ u_2(x_E) = v\})$. ■

Proposition 7 (Combinaison pour une utilité GAI-décomposable) *Considérons un réseau GAI vectoriel décomposable en exactement deux cliques : X_{C_1} et X_{C_2}, avec leur séparateur $S_{12} = C_1 \cap C_2$. Posons $D_1 = C_1 \setminus S_{12}$ et $D_2 = C_2 \setminus S_{12}$. Alors, on a :*

$$\begin{array}{rl} & ND_P(\mathcal{U}) \subseteq \bigcup_{x_{S_{12}} \in X_{S_{12}}} \big(N_1(x_{S_{12}}) \boxplus N_2(x_{S_{12}})\big) \\ \textit{avec} & N_1(x_{S_{12}}) = ND_P\big(\{v/\exists x_{D_1} \in X_{D_1}\ u_1(x_{D_1}, x_{S_{12}}) = v\}\big) \\ \textit{et} & N_2(x_{S_{12}}) = ND_P\big(\{v/\exists x_{D_2} \in X_{D_2}\ u_2(x_{D_2}, x_{S_{12}}) = v\}\big). \end{array}$$

Preuve. Cette démonstration est une application directe de la proposition 6 : La propriété d'intersection courante permet d'affirmer que $D_1 \cap D_2 = \varnothing$. Donc, à valeur de $X_{S_{12}}$ fixée, u_1 et u_2 sont des utilités additives. D'où $\mathrm{ND}_P(\mathcal{U}) \subseteq \bigcup_{x_{S_{12}} \in X_{S_{12}}} \big(N_1(x_{S_{12}}) \boxplus N_2(x_{S_{12}})\big)$. ■

Proposition 8 (Généralisation de la proposition 7) *Considérons un réseau GAI vectoriel décomposable selon $C = \{C_1, \ldots, C_q\}$. Soit X_S un séparateur partitionnant l'ensemble des cliques en $\mathcal{C}_1$ et $\mathcal{C}_2$ (de chaque côté du séparateur). Posons $D_1 = \bigcup_{C_i \in \mathcal{C}_1} C_i \backslash S$ et $D_2 = \bigcup_{C_j \in \mathcal{C}_2} C_j \backslash S$. Alors, on a :*

$$\begin{array}{rl} & ND_P(\mathcal{U}) \subseteq \bigcup_{x_S \in X_S} \big(N_1(x_S) \boxplus N_2(x_S)\big) \\ \textit{avec} & N_1(x_S) = ND_P\big(\{v / \exists x_{D_1} \in X_{D_1} \ \sum_{C_i \in \mathcal{C}_1} u_i((x_{D_1}, x_S)_{C_i}) = v\}\big) \\ \textit{et} & N_2(x_S) = ND_P\big(\{v / \exists x_{D_2} \in X_{D_2} \ \sum_{C_j \in \mathcal{C}_2} u_j((x_{D_2}, x_S)_{C_j}) = v\}\big). \end{array}$$

Preuve. Cette proposition est une généralisation de la proposition 7 : les ensembles de cliques $\mathcal{C}_1$ et $\mathcal{C}_2$ permettent de définir un réseau GAI à deux cliques : $\bar{C}_1 = \bigcup_{C_i \in \mathcal{C}_1} C_i$ et $\bar{C}_2 = \bigcup_{C_j \in \mathcal{C}_2} C_j$, de sous-fonctions d'utilité respectives $u_1 = \sum_{C_i \in \mathcal{C}_i} u_i$ et $u_2 = \sum_{C_j \in \mathcal{C}_2} u_j$. Ce réseau a deux cliques représente bien la même utilité vectorielle u que le réseau initial, la proposition est donc une réécriture de la proposition 7. ■

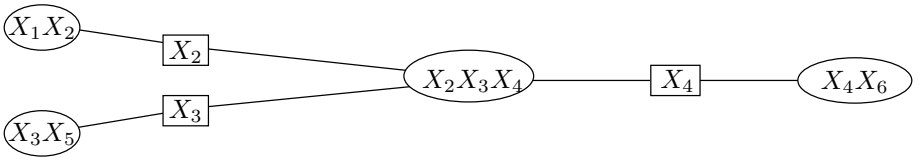
Proposition 9 (Validité de l'algorithme) *L'algorithme* `CollecteVectorielle`*(X_{C_i}, X_{C_i}) se termine en renvoyant un ensemble $E = \{\langle v_i, x_i \rangle\}$ correspondant aux associations de vecteurs et d'instanciations composant la frontière de Pareto du problème.*

Preuve. Montrons cette proposition par récurrence sur les cliques de l'arbre de jonction, ordonné en fonction des messages de la collecte vectorielle.

- Considérons une feuille X_{C_i} de l'arbre de jonction, reliée à la clique X_{C_j} par le séparateur S_{ij}. Considérons un appel à `CollecteVectorielle`(C_i, C_j). La ligne 01 va former une fonction ψ_i contenant l'intégralité des associations entre instanciation de $x_{C_i} \in X_{C_i}$ et $u_i(x_{C_i})$. Puisque la seule clique voisine de C_i est C_j, la boucle Pour (lignes 02 à 07) ne modifiera pas ψ_i. Puis, les lignes 09 à 13 vont construire une fonction $\phi_{i,j}$ qui, à chaque valeur de séparateur $x_{S_{ij}}$ fixé, associe une frontière de Pareto (sous forme d'association) locale à la clique C_i. La propriété 8 (en posant $\mathcal{C}_1 = \{C_i\}$) nous permet d'affirmer que toute solution écartée lors de ces calculs ne peut être complétée pour appartenir à la frontière de Pareto globale du réseau GAI.
- Soit maintenant une clique C_i qui n'est pas une feuille, et considérons un appel à `CollecteVectorielle`(C_i, C_j). Les lignes 02 à 07 de l'algorithme permettent de combiner (avec l'opérateur $\oplus$, correspondant à l'opérateur $\boxplus$ prenant en compte les associations) les messages $\phi_{z,i}$ reçus avec les associations contenues dans la fonction ψ_i. À la fin de l'exécution de la boucle Pour, la fonction ψ_i contient donc l'intégralité des associations possibles issues de tout le sous-réseau déjà parcouru, à laquelle on a supprimé des associations qui ne peuvent appartenir à la frontière de Pareto du réseau GAI vectoriel. De la même façon que précédemment, les calculs de non-dominance (lignes 09 à 13) à valeur de séparateur fixé suppriment des associations qui ne pourront appartenir à la frontière de Pareto (proposition 8 avec $S = S_{ij}$).
- Au niveau de la clique racine C_r, on aura donc (à la fin de la boucle Pour) une fonction ϕ_r contenant un sur-ensemble de la frontière de Pareto du problème. La ligne 15 applique un dernier calcul de non-dominance permettant d'obtenir exactement la frontière de Pareto du problème.

Donc, par récurrence sur les cliques du réseau, un appel à `CollecteVectorielle`(X_{C_i}, X_{C_i}) renverra bien la frontière de Pareto du problème. ■

Exemple 32 *Appliquons l'algorithme de collecte vectorielle sur le réseau GAI vectoriel de la figure 4.2.*



(a) Arbre de jonction

X_1	X_2 : x_{21}	X_2 : x_{22}
x_{11}	$(1,5)$	$(2,3)$
x_{12}	$(8,2)$	$(3,4)$
x_{13}	$(7,1)$	$(6,2)$

(b) $u_1(x_1, x_2)$

X_5	X_3 : x_{31}	X_3 : x_{32}
x_{51}	$(2,9)$	$(4,3)$
x_{52}	$(1,7)$	$(4,9)$
x_{53}	$(4,1)$	$(4,8)$
x_{54}	$(3,1)$	$(5,3)$

(c) $u_2(x_3, x_5)$

	X_3 : x_{31}		X_3 : x_{32}	
	X_2		X_2	
X_4	x_{21}	x_{22}	x_{21}	x_{22}
x_{41}	$(3,3)$	$(5,3)$	$(1,2)$	$(4,4)$
x_{42}	$(2,4)$	$(1,4)$	$(6,2)$	$(2,5)$

(d) $u_3(x_2, x_3, x_4)$

X_4	X_6 : x_{61}	X_6 : x_{62}
x_{41}	$(9,1)$	$(6,7)$
x_{42}	$(3,2)$	$(2,6)$

(e) $u_4(x_4, x_6)$

FIGURE 4.2 – Exemple de réseau GAI vectoriel (bicritère)

Supposons que l'appel de la fonction se fasse sur la clique contenant u_4. L'algorithme effectuera l'enchaînement des appels récursifs (voir la figure 4.3 pour une représentation sous forme de diffusion de messages), puis commencera à construire les premiers messages.

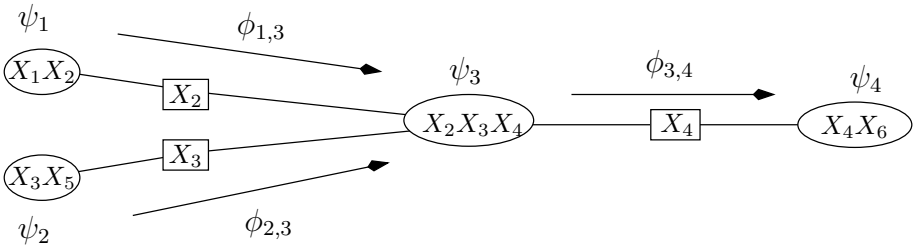


FIGURE 4.3 – Ordre des messages dans l'exemple 32

Dans un premier temps, les sous-fonctions d'utilité u_1 et u_2 seront converties en fonctions renvoyant des ensembles d'associations ψ_1 et ψ_2.

$\psi_1(x_1, x_2)$	X_2	
X_1	x_{21}	x_{22}
x_{11}	$\{\langle(1,5),(x_{11},x_{21})\rangle\}$	$\{\langle(2,3),(x_{11},x_{22})\rangle\}$
x_{12}	$\{\langle(8,2),(x_{12},x_{21})\rangle\}$	$\{\langle(3,4),(x_{12},x_{22})\rangle\}$
x_{13}	$\{\langle(7,1),(x_{13},x_{21})\rangle\}$	$\{\langle(6,2),(x_{13},x_{22})\rangle\}$

$\psi_2(x_3, x_5)$	X_3	
X_5	x_{31}	x_{32}
x_{51}	$\{\langle(2,9),(x_{51},x_{31})\rangle\}$	$\{\langle(4,3),(x_{51},x_{32})\rangle\}$
x_{52}	$\{\langle(1,7),(x_{52},x_{31})\rangle\}$	$\{\langle(4,9),(x_{52},x_{32})\rangle\}$
x_{53}	$\{\langle(4,1),(x_{53},x_{31})\rangle\}$	$\{\langle(4,8),(x_{53},x_{32})\rangle\}$
x_{54}	$\{\langle(3,1),(x_{54},x_{31})\rangle\}$	$\{\langle(5,3),(x_{54},x_{32})\rangle\}$

Puis le message $\phi_{1,3}$ sera construit en calculant pour chaque valeur $x_2 \in X_2$, `NonDominés`$(\bigcup_{x_1 \in X_1} \psi_1(x_1, x_2))$*. De la même façon, pour $\phi_{2,3}$, on calculera pour chaque valeur de $x_3 \in X_3$* `NonDominés`$(\bigcup_{x_5 \in X_5} \psi_2(x_3, x_5))$*.*

X_2	$\phi_{1,3}(x_2)$
x_{21}	$\{\langle(1,5),(x_{11},x_{21})\rangle, \langle(8,2),(x_{12},x_{21})\rangle\}$
x_{22}	$\{\langle(3,4),(x_{12},x_{22})\rangle, \langle(6,2),(x_{13},x_{22})\rangle\}$

X_3	$\phi_{2,3}(x_3)$
x_{31}	$\{\langle(2,9),(x_{51},x_{31})\rangle, \langle(4,1),(x_{53},x_{31})\rangle\}$
x_{32}	$\{\langle(5,3),(x_{54},x_{32})\rangle, \langle(4,9),(x_{52},x_{32})\rangle\}$

Ensuite, la fonction ψ_3 va être mise à jour en effectuant cette succession d'opérations :
$\psi_3 \leftarrow$ `NonDominésParCase`$(\psi_3 \oplus \phi_{1,3})$
$\psi_3 \leftarrow$ `NonDominésParCase`$(\psi_3 \oplus \phi_{2,3})$
L'état final de la fonction ψ_3 est représenté par :

X_3	X_2	X_4	$\psi_3(x_2, x_3, x_4)$
x_{31}	x_{21}	x_{41}	$\{\langle(6,17),(x_{11},x_{21},x_{31},x_{41},x_{51})\rangle, \langle(13,14),(x_{12},x_{21},x_{31},x_{41},x_{51})\rangle, \langle(15,6),(x_{12},x_{21},x_{31},x_{41},x_{53})\rangle\}$
x_{31}	x_{22}	x_{41}	$\{\langle(10,16),(x_{12},x_{22},x_{31},x_{41},x_{51})\rangle, \langle(13,14),(x_{13},x_{22},x_{31},x_{41},x_{51})\rangle, \langle(15,6),(x_{13},x_{22},x_{31},x_{41},x_{53})\rangle\}$
x_{32}	x_{21}	x_{41}	$\{\langle(14,7),(x_{12},x_{21},x_{32},x_{41},x_{54})\rangle, \langle(6,16),(x_{11},x_{21},x_{32},x_{41},x_{52})\rangle, \langle(13,13),(x_{12},x_{21},x_{32},x_{41},x_{52})\rangle\}$
x_{32}	x_{22}	x_{41}	$\{\langle(15,9),(x_{13},x_{22},x_{32},x_{41},x_{54})\rangle, \langle(11,17),(x_{12},x_{22},x_{32},x_{41},x_{52})\rangle, \langle(14,15),(x_{13},x_{22},x_{32},x_{41},x_{52})\rangle\}$
x_{31}	x_{21}	x_{42}	$\{\langle(5,18),(x_{11},x_{21},x_{31},x_{42},x_{51})\rangle, \langle(12,15),(x_{12},x_{21},x_{31},x_{42},x_{51})\rangle, \langle(14,7),(x_{12},x_{21},x_{31},x_{42},x_{53})\rangle\}$
x_{31}	x_{22}	x_{42}	$\{\langle(6,17),(x_{12},x_{22},x_{31},x_{42},x_{51})\rangle, \langle(9,15),(x_{13},x_{22},x_{31},x_{42},x_{51})\rangle, \langle(11,7),(x_{13},x_{22},x_{31},x_{42},x_{53})\rangle\}$
x_{32}	x_{21}	x_{42}	$\{\langle(19,7),(x_{12},x_{21},x_{32},x_{42},x_{54})\rangle, \langle(11,16),(x_{11},x_{21},x_{32},x_{42},x_{52})\rangle, \langle(18,13),(x_{12},x_{21},x_{32},x_{42},x_{52})\rangle\}$
x_{32}	x_{22}	x_{42}	$\{\langle(13,10),(x_{13},x_{22},x_{32},x_{42},x_{54})\rangle, \langle(9,18),(x_{12},x_{22},x_{32},x_{42},x_{52})\rangle, \langle(12,16),(x_{13},x_{22},x_{32},x_{42},x_{52})\rangle\}$

Finalement, le message $\psi_{3,4}$ est obtenu en calculant pour chaque valeur de $x_4 \in X_4$

$\texttt{NonDominés}(\bigcup_{(x_2,x_3)\in X_2\times X_3}\psi_3(x_2,x_3,x_4))$.

X_4	$\phi_{3,4}(x_4)$
x_{41}	$\{\langle(15,9),(x_{13},x_{22},x_{32},x_{41},x_{54})\rangle,\langle(11,17),(x_{12},x_{22},x_{32},x_{41},x_{52})\rangle,$ $\langle(14,15),(x_{13},x_{22},x_{32},x_{41},x_{52})\rangle\}$
x_{42}	$\{\langle(19,7),(x_{12},x_{21},x_{32},x_{42},x_{54})\rangle,\langle(18,13),(x_{12},x_{21},x_{32},x_{42},x_{52})\rangle,$ $\langle(12,16),(x_{13},x_{22},x_{32},x_{42},x_{52})\rangle,\langle(9,18),(x_{12},x_{22},x_{32},x_{42},x_{52})\rangle\}$

Il ne reste qu'à sommer le dernier message avec la fonction ψ_4 et à effectuer des calculs de non-dominance pour chaque instanciation de $X_4 \times X_6$, pour obtenir l'état final de ψ_4 :

X_4	X_6	$\psi_4(x_4,x_6)$
x_{41}	x_{61}	$\{\langle(24,10),(x_{13},x_{22},x_{32},x_{41},x_{54},x_{61})\rangle,$ $\langle(20,18),(x_{12},x_{22},x_{32},x_{41},x_{52},x_{61})\rangle,$ $\langle(23,16),(x_{13},x_{22},x_{32},x_{41},x_{52},x_{61})\rangle\}$
x_{41}	x_{62}	$\{\langle(21,16),(x_{13},x_{22},x_{32},x_{41},x_{54},x_{62})\rangle,$ $\langle(17,24),(x_{12},x_{22},x_{32},x_{41},x_{52},x_{62})\rangle,$ $\langle(20,22),(x_{13},x_{22},x_{32},x_{41},x_{52},x_{62})\rangle\}$
x_{42}	x_{61}	$\{\langle(22,9),(x_{12},x_{21},x_{32},x_{42},x_{54},x_{61})\rangle,$ $\langle(21,15),(x_{12},x_{21},x_{32},x_{42},x_{52},x_{61})\rangle,$ $\langle(15,18),(x_{13},x_{22},x_{32},x_{42},x_{52},x_{61})\rangle,$ $\langle(12,20),(x_{12},x_{22},x_{32},x_{42},x_{52},x_{61})\rangle\}$
x_{42}	x_{62}	$\{\langle(21,13),(x_{12},x_{21},x_{32},x_{42},x_{54},x_{62})\rangle,$ $\langle(20,19),(x_{12},x_{21},x_{32},x_{42},x_{52},x_{62})\rangle,$ $\langle(14,22),(x_{13},x_{22},x_{32},x_{42},x_{52},x_{62})\rangle,$ $\langle(11,24),(x_{12},x_{22},x_{32},x_{42},x_{52},x_{62})\rangle\}$

La dernière étape de l'algorithme consiste à faire l'union de tous les ensembles contenus dans ψ_4 et d'effectuer le calcul de non-dominance final pour obtenir la frontière de Pareto du problème :

$$\{\langle(24,10),(x_{13},x_{22},x_{32},x_{41},x_{54},x_{61})\rangle,$$
$$\langle(23,16),(x_{13},x_{22},x_{32},x_{41},x_{52},x_{61})\rangle,$$
$$\langle(20,22),(x_{13},x_{22},x_{32},x_{41},x_{52},x_{62})\rangle,$$
$$\langle(17,24),(x_{12},x_{22},x_{32},x_{41},x_{52},x_{62})\rangle\}.$$

4.2 Un algorithme approché avec garantie de performance

Dans la section précédente, nous avons montré que le problème de la détermination de la frontière de Pareto dans le modèle GAI est NP-complet et nous avons présenté un algorithme exploitant la structure des dépendances entre attributs pour résoudre le problème par l'intermédiaire du réseau GAI vectoriel. Toutefois, lorsque nous avions présenté la notion de frontière de Pareto dans le chapitre 1, nous avions montré qu'il était possible que sa taille soit équivalente à la taille de $\mathcal{X}$, donc une taille de frontière pouvant

augmenter exponentiellement avec le nombre d'attributs du réseau, ce qui implique de graves problèmes computationels.

Cette section a pour but d'adapter l'algorithme présenté précédemment pour produire un algorithme approchant la frontière de Pareto et possédant une garantie de performance. Dans un problème d'optimisation monocritère cherchant à maximiser une valeur scalaire, on dit qu'un algorithme est $(1+\epsilon)$-approché (avec $\epsilon > 0$), si au lieu de renvoyer la solution optimale x^*, l'algorithme renvoie une solution x vérifiant $x^* \leqslant (1+\epsilon)x$ (Vazirani, 2001). Dans le cadre multicritère, nous allons nous baser sur la notion d'ϵ-dominance (Papadimitriou et Yannakakis, 2000; Laumanns et al., 2002; Perny et Spanjaard, 2008) qui sera présentée en 4.2.1, pour finalement déduire un algorithme approché avec garantie de performance, présenté en 4.2.2.

4.2.1 ϵ-dominance

Comment adapter la notion de $(1+\epsilon)$-approximation à un algorithme renvoyant un ensemble d'alternatives et de vecteurs ? Une frontière de Pareto est basée sur la notion de vecteurs non dominés au sens de la dominance de Pareto, il semble donc logique de prendre en compte l'approximation au sein de la relation de dominance en elle-même. L'ϵ-dominance est une transformation de la dominance de Pareto (voir la définition 76) et s'interprète de la façon suivante : « un vecteur v ϵ-domine un vecteur v' s'il domine v' au sens de Pareto à un facteur $1+\epsilon$ près ».

Définition 76 (ϵ-dominance) *Soient $v \in \mathbb{Z}_+^m$, $v' \in \mathbb{Z}_+^m$ et $\epsilon > 0$, on dit que v ϵ-domine v' (et on note $v \succ_\epsilon v'$) si, et seulement si, on a :*

$$v \succ_\epsilon v' \Leftrightarrow (1+\epsilon)v \succ_P v'.$$

Cette relation de dominance possède des propriétés la rendant très intéressante d'un point de vue computationnel et que nous présenterons lors de l'analyse théorique de nos algorithmes (dans la section 4.3.1). Dans le chapitre 1, nous avions donné un vision schématique de la dominance de Pareto par l'intermédiaire des cônes de dominance. L'ϵ-dominance étant basée sur la dominance de Pareto, une telle vision est toujours valable, l'introduction du facteur multiplicatif peut être vu comme un déplacement de la zone de dominance (voir la figure 4.4).

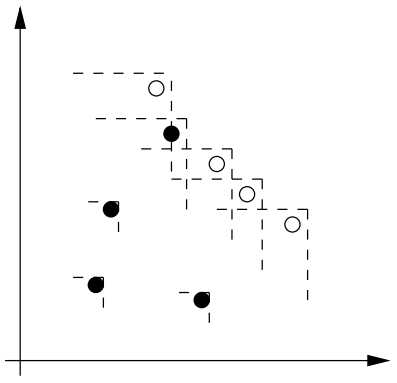


FIGURE 4.4 – Exemple de représentation des cônes d'ϵ-dominance

Un algorithme $(1+\epsilon)$-approché de la détermination de la frontière de Pareto consistera donc à déterminer l'ensemble des alternatives engendrant l'ensemble des vecteurs qui ne sont pas ϵ-dominés, pour une valeur de $\epsilon > 0$ fixée, ce qui est aussi connu dans la littérature sous le nom d'ϵ-couverture.

Définition 77 (ϵ-couverture) *Soit $\mathcal{X}$ un espace d'alternatives, u une fonction d'utilité multicritère, $\mathcal{U}$ l'espace des critères engendré par u et ϵ un réel strictement positif. On appelle ϵ-couverture de $\mathcal{U}$, et on note $ND_\epsilon(\mathcal{U})$ l'ensemble de vecteurs défini par $ND_\epsilon(\mathcal{U}) = \{v \in \mathcal{U} / \nexists v' \in \mathcal{U}\ v' \succ_\epsilon v\}$.*
On introduit de la même façon l'ensemble des alternatives engendrant une ϵ-couverture par : $ArgND_\epsilon(\mathcal{X}, u) = \{x \in \mathcal{X} / \nexists x' \in \mathcal{X}\ u(x') \succ_\epsilon u(x)\}$.

Papadimitriou et Yannakakis (2000) ont montré une propriété intéressante permettant de créer une autre forme de dominance impliquant l'ϵ-dominance. Cette nouvelle dominance, appelée log-dominance (voir définition 78), permet d'associer à chaque vecteur d'utilité un vecteur d'entiers naturels et d'effectuer une dominance de Pareto sur les entiers.

Définition 78 (log-dominance) *Soit $\epsilon > 0$, et $\Phi : \mathbb{Z}_+ \to \mathbb{N}$ une fonction définie par $\forall v^i \in \mathbb{Z}_+\ \Phi(v^i) = \lfloor \frac{\log(v^i)}{\log(1+\epsilon)} \rfloor$. Par abus de notation, nous noterons aussi Φ l'extension de cette fonction aux vecteurs à m dimensions : $\Phi(v^1, \ldots, v^m) = (\Phi(v^1), \ldots, \Phi(v^m))$.*
Soit $v \in \mathbb{Z}_+^m$, $v' \in \mathbb{Z}_+^m$. On dit que v log-domine v' (et on note $v \succ_{\log} v'$) si, et seulement si, on a :

$$v \succ_{\log} v' \Leftrightarrow \Phi(v) \succ_P \Phi(v').$$

Propriété 29 *Soit $\epsilon > 0$, quels que soient les vecteurs $v \in \mathbb{Z}_+^m$ et $v' \in \mathbb{Z}_+^m$, on a :*

$$v \succ_{\log} v' \implies v \succ_\epsilon v'.$$

Remarque 22 *Cette approche s'interprète facilement graphiquement : Considérons un vecteur $(v^1, v^2) \in \mathbb{Z}_+^2$, ainsi que deux entiers naturels z et z' vérifiant $(1+\epsilon)^z \leqslant v^1 < (1+\epsilon)^{z+1}$ et $(1+\epsilon)^{z'} \leqslant v^2 < (1+\epsilon)^{z'+1}$. On aura donc : $z \leqslant \log_{1+\epsilon}(v^1) < z+1$ et $z' \leqslant \log_{1+\epsilon}(v^2) < z'+1$, d'où $(z, z') = \Phi(v^1, v^2)$. La fonction Φ associe donc à chaque vecteur d'utilité un vecteur d'entiers naturels correspondant à l'indice du coin inférieur gauche de la région à laquelle il appartient sur la figure 4.5.*

La propriété 29 s'interprète en considérant que s'il existe une dominance de Pareto entre les indices des régions obtenus par application de la fonction Φ, alors par construction des régions, il existe forcément une ϵ-dominance entre les vecteurs. De plus, chaque vecteur d'une même région s'ϵ-dominant mutuellement, il semble naturel de remarquer que l'ensemble des non-dominés au sens de la log*-dominance est une ϵ-couverture du problème.*

4.2.2 Vers un algorithme approché avec garantie de performance

Nous avons vu de nouveaux types de dominance permettant d'approcher la frontière de Pareto. Nous allons maintenant étudier comment l'algorithme exact présenté dans la section 4.1.3 peut s'adapter pour donner un algorithme approché avec garantie de

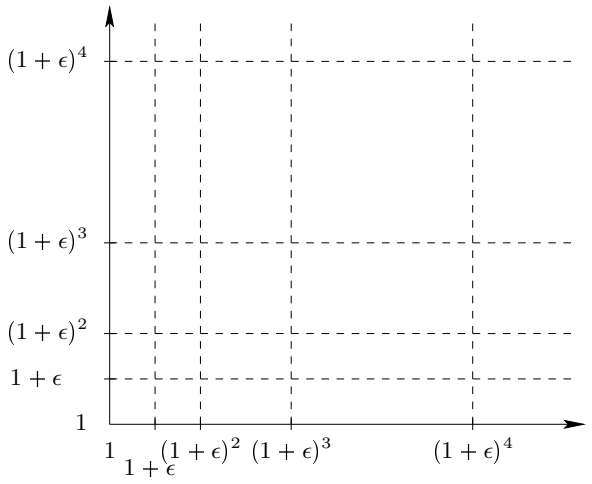


FIGURE 4.5 – Exemple de représentation de la log-dominance

performance. La première idée venant à l'esprit est d'avoir la même démarche que précédemment et de garder le même algorithme en remplaçant les fonctions `NonDominés` et `NonDominésParCase` par un calcul de vecteurs non dominés au sens de la log-dominance ou de l'ϵ-dominance. En pratique, un tel algorithme ne peut s'écrire tel quel, car autant l'ϵ-dominance vérifie la propriété d'additivité (voir la propriété 30), autant la transitivité n'est pas vérifiée (voir l'exemple 33).

Propriété 30 $\forall v \in \mathbb{Z}_+^m \ \forall v' \in \mathbb{Z}_+^m \ \forall z \in \mathbb{Z}_+^m \ v \succ_\epsilon v' \implies v + z \succ_\epsilon v' + z.$

Preuve. Supposons que $v \succ_\epsilon v'$, alors on a $(1+\epsilon)v \succ_P v'$ par définition, et donc $(1+\epsilon)v + z \succ_P v' + z$. Or, étant donné que $(1+\epsilon)(v+z) \succ_P (1+\epsilon)v + z$, on aura donc $(1+\epsilon)(v+z) \succ_P v'+z$ (par transitivité de la dominance de Pareto), d'où : $v+z \succ_\epsilon v'+z$. ■

Exemple 33 *Posons* $\epsilon = 0.1$, *et considérons les vecteurs* $v = (100, 100)$, $v' = (105, 105)$ *et* $v'' = (112, 112)$. *On aura* $1,1 \times 100 = 110 > 105$ *donc* $v \succ_\epsilon v'$, *et* $1,1 \times 105 = 115,5 > 112$ *donc* $v' \succ_\epsilon v''$. *Si* $\succ_\epsilon$ *était une relation transitive, on devrait donc avoir* $v \succ_\epsilon v''$, *pourtant* $112 > 110$. *La relation* $\succ_\epsilon$ *n'est donc pas transitive. D'un point de vue plus général, cette non-transitivité s'explique facilement en reprenant la définition de l'*ϵ*-dominance : Si* $v \succ_\epsilon v'$, *on a* $(1+\epsilon)v \succ_P v'$. *Si* $v' \succ_\epsilon v''$, *on a* $(1+\epsilon)v' \succ_P v''$. *On peut donc seulement garantir que* $(1+\epsilon)^2 v \succ_P v''$.

Pour résoudre le problème de transitivité de l'ϵ-dominance, nous allons bâtir une nouvelle relation qui nous assurera une plus grande maîtrise de l'exponentiation du facteur $1+\epsilon$: l'(ϵ, w)-dominance.

Définition 79 ((ϵ, w)-dominance) *Soit* $v \in \mathbb{Z}_+^m$, $v' \in \mathbb{Z}_+^m$, $\epsilon > 0$ *et* $w > 0$. *On dit que* v (ϵ, w)*-domine* v' *(et on note* $v \succ_\epsilon^w v'$*) si, et seulement si, on a :*

$$v \succ_\epsilon^w v' \Leftrightarrow (1+\epsilon)^w v \succ_P v'.$$

De façon triviale, on remarque que l'(ϵ, w)-dominance est une ϵ-dominance si (et seulement si) $w = 1$. L'intérêt de cette définition est que la relation est additive, et qu'elle possède une forme de transitivité que nous appellerons « transitivité additive », c'est-à-dire que sa transitivité s'exprimera sous la forme d'une relation additive sur l'exposant w (voir les propositions 10 et 11).

Proposition 10 (Additivité) $\forall v \in \mathbb{Z}_+^m \; \forall v' \in \mathbb{Z}_+^m \; \forall z \in \mathbb{Z}_+^m \; v \succ_\epsilon^w v' \implies v+z \succ_\epsilon^w v'+z.$

Preuve. Supposons que $v \succ_\epsilon^w v'$, on a donc $\forall j \in \{1, \ldots, m\}$ $(1+\epsilon)^w v^j > v'^j$. Donc $(1+\epsilon)^w v^j + z^j > v'^j + z^j$. Or, étant donné que $\epsilon > 0$ et $w > 0$, on aura donc $(1+\epsilon)^w > 1$, donc $(1+\epsilon)^w(v^j + z^j) > (1+\epsilon)^w v^j + z^j$. D'où $(1+\epsilon)^w(v^j + z^j) > v'^j + z^j$. On a donc :

$$v + z \succ_\epsilon^w v' + z.$$

■

Proposition 11 (Transitivité additive)

$$\forall v \in \mathbb{Z}_+^m \; \forall v' \in \mathbb{Z}_+^m \; \forall v'' \in \mathbb{Z}_+^m \; \forall w > 0 \; \forall w' > 0 \; v \succ_\epsilon^w v' \text{ et } v' \succ_\epsilon^{w'} v'' \implies v \succ_\epsilon^{w+w'} v''.$$

Preuve. On a $v \succ_\epsilon^w v' \Leftrightarrow \forall j \in \{1, \ldots, m\}(1+\epsilon)^w v^j > v'^j$, donc $(1+\epsilon)^{w+w'} v^j > (1+\epsilon)^{w'} v'^j$. De plus, $v' \succ_\epsilon^w v'' \Leftrightarrow \forall j \in \{1, \ldots, m\}(1+\epsilon)^w v'^j > v''^j$. Donc, $(1+\epsilon)^{w+w'} v^j > v''^j$. ■

En remarquant que $\log\left((1+\epsilon)^w\right) = w\log(1+\epsilon)$, il est possible de recréer une relation de log-dominance, paramétrée par w, de manière à pouvoir utiliser une comparaison entre entiers naturels.

Définition 80 ($(\log, w)$-dominance) *Soit $\epsilon > 0$, $w > 0$, et $\Phi_w : \mathbb{Z}_+ \to \mathbb{N}$ une fonction définie par $\forall v^i \in \mathbb{Z}_+ \; \Phi_w(v^i) = \lfloor \frac{\log(v^i)}{w\log(1+\epsilon)} \rfloor$. Par abus de notation, nous noterons aussi Φ_w l'extension de cette fonction aux vecteurs à m dimensions : $\Phi_w(v^1, \ldots, v^m) = (\Phi_w(v^1), \ldots, \Phi_w(v^m))$.*

Soit $v \in \mathbb{Z}_+^m$, $v' \in \mathbb{Z}_+^m$, on dit que v $(\log, w)$-domine v' (et on note $v \succ_{\log}^w v'$) si, et seulement si, on a :

$$v \succ_{\log}^w v' \Leftrightarrow \Phi_w(v) \succ_P \Phi_w(v')$$

L'idée de l'algorithme approché va être de paramétrer la fonction `NonDominés` (ainsi que son application aux hypermatrices : `NonDominésParCase`) en fonction de w (on suppose que la valeur de ϵ est déjà définie par l'utilisateur de l'algorithme en fonction de la qualité d'approximation désirée), de manière à ce que le résultat corresponde à l'ensemble des vecteurs qui ne sont pas $(\log, w)$-dominés. Au fur et à mesure de la collecte vectorielle, le paramètre w va évoluer par additivité de façon à ce qu'en fin d'algorithme, nous obtenions $w = 1$. Dans l'algorithme 17, nous avons associé à chaque clique une valeur $w = \frac{1}{|C|}$, nous aurons donc bien à la fin de l'éxécution de l'algorithme un dernier test de non-dominance effectué avec $w = \sum_{C_j \in C} \frac{1}{|C|} = \frac{|C|}{|C|} = 1$.

Algorithme 17 : CollecteVectorielleApprochee

Entrées :
C_a : clique réceptionnant les messages
C_p : clique destinataire du message à créer

$\psi_a \leftarrow$ **créer** hypermatrice vérifiant $\forall x_{C_a} \in X_{C_a}\ \psi_a(x_{C_a}) = \{\langle u_a(x_{C_a}), x_{C_a}\rangle\}$;
$w \leftarrow \frac{1}{|C|}$;
pour chaque C_b *clique voisine de* C_a **faire**
  si $C_b \neq C_p$ **alors**
    $(\phi_{b,a}, w') \leftarrow$ `CollecteVectorielleApprochee`(C_b, C_a);
    $w \leftarrow w + w'$;
    $\psi_a \leftarrow$ `NonDominésParCase`$(\psi_a \oplus \phi_{b,a}, w)$;
  fin
fin
si $C_a \neq C_p$ **alors**
  $\phi_{a,p} \leftarrow$ **créer** hypermatrice définie sur $X_{S_{ap}}$;
  pour tous les $x_{S_{ap}} \in X_{S_{ap}}$ **faire**
    $\phi_{a,p}(x_{S_{ap}}) \leftarrow$ `NonDominés`$(\bigcup_{y_{C_a \setminus S_{ap}} \in X_{C_a \setminus S_{ap}}} \psi_a(x_{S_{ap}}, y_{C_a \setminus S_{ap}}), w)$;
  fin
  retourner $(\phi_{a,p}, w)$;
sinon
  retourner `NonDominés`$(\bigcup_{x_{C_a} \in X_{C_a}} \psi_a(x_{C_a}), w)$;
fin

Remarque 23 *Il est à noter que dans le cas où le réseau GAI a une structure de forêt de jonction contenant plusieurs arbres, il est possible d'adapter les algorithmes présentés. Dans un premier temps, il faudra exécuter l'algorithme sur chaque composante connexe de la forêt. Ainsi, si le réseau GAI contient k arbres de jonction, nous obtiendrons k frontières de Pareto (ou approximations de ces frontières), il ne restera plus qu'à les fusionner entre elles et à effectuer une recherche globale de non-dominance.*

4.3 Analyse et conclusion

Nous avons présenté deux algorithmes basés sur les notions de collecte vectorielle et de calcul de non-dominance. Cette section a pour but d'évaluer ces algorithmes, aussi bien sur le plan théorique (section 4.3.1) que pratique (section 4.3.2), pour finalement conclure sur ces algorithmes. Par abus de notation, le cas de l'algorithme exact sera désigné comme étant l'algorithme approché avec un paramètre $\epsilon = 0$. Cela n'a aucun sens du point de vue mathématique, étant donné que nous avons posé $\epsilon > 0$, mais du point de vue humain, il est aisé de comprendre que $\epsilon = 0$ signifie qu'aucune approximation n'est effectuée.

4.3.1 Analyse théorique

Étant donné que la frontière de Pareto peut être de taille équivalente à $\mathcal{X}$, il est nécessaire de prendre en compte la forme de l'ensemble des solutions potentielles dans l'espace des critères. Pour cela, nous allons exploiter le choix de $\mathbb{Z}_+^m$ afin d'utiliser un indicateur : Nous noterons $K \in \mathbb{Z}_+$ le plus petit entier vérifiant $\forall x \in \mathcal{X}\ \forall j \in \{1, \ldots, m\}\ u^j(x) \leqslant K$. Ainsi, pour un critère donné, les vecteurs peuvent avoir K valeurs différentes, et donc $|\mathcal{U}| \in O(K^m)$.

Proposition 12 (Complexité de `NonDominés`) *Le temps de calcul d'un appel à la fonction* `NonDominés` *sur un ensemble E de vecteurs à m dimensions est en $O(m|E|^2)$*

Preuve. De façon simple, pour déterminer si un vecteur en domine un autre, il est nécessaire de parcourir toutes leurs composantes, donc un temps de calcul en $O(m)$. Pour déterminer si un vecteur est non dominé, il faudra parcourir tous les autres vecteurs de E, donc du $O(m|E|)$. En éxécutant le test de non-dominance pour tous les vecteurs de E, on aura un temps de calcul en $O(m|E|^2)$. ■

Pour simplifier l'expression des complexités qui suivront, nous noterons $|X_{C_{\max}}|$ la taille de la clique maximale, c'est-à-dire : $|X_{C_{\max}}| = \max_{C_i \in C} \prod_{X_i \in C_j} |X_i|$.

Proposition 13 (Complexité de l'algorithme exact) *Le temps de calcul de l'algorithme exact (algorithme 16) est en $O(|C||X_{C_{\max}}|mK^{2m})$.*

Preuve. Considérons les opérations se déroulant au sein d'une clique sans se soucier du temps de calcul des appels récursifs dans un premier temps.

Les premières opérations consistent à recomposer des ensembles (avec l'opérateur $\oplus$) par l'intermédiaire des messages transitant le long des séparateurs et des associations contenues dans la fonction ψ associée à la clique. En remarquant que l'ensemble des vecteurs composés ne pourra être de taille supérieure à $|\mathcal{U}| \in O(K^m)$ et que le parcours

d'un sous-ensemble des instanciations des attributs d'une clique ne pourra parcourir plus d'éléments que $|X_{C_{\max}}|$, on en déduit qu'à la fin de la boucle Pour de l'algorithme 16, le temps de calcul est de l'ordre de $O(s|X_{C_{\max}}|mK^{2m})$, avec s le nombre de messages $\phi_{i,j}$ parcourus dans la boucle.

Après la boucle Pour, soit tous les ensembles contenus dans ϕ seront unis pour effectuer un dernier test de non-dominance (cas de la clique racine), soit seulement une partie de ces ensembles subiront cette opération. Dans tous les cas, de la même façon que précédemment, le temps de calcul sera dans le pire des cas $O(|X_{C_{\max}}|mK^{2m})$. Donc le temps de calcul global de `CollecteVectorielle` en exceptant le temps de calcul des appels récursifs sera en $O(s|X_{C_{\max}}|mK^{2m})$.

D'un point de vue plus général, l'algorithme éxécutera autant d'appels récursifs que de cliques dans le graphe. Notons s_i le nombre de messages ayant pour destination la clique C_i. En remarquant que le réseau GAI est un arbre, son nombre de séparateurs est lié linéairement au nombre de cliques, donc $\sum_{C_i \in C} s_i \in O(|C|)$. Il ne reste donc plus qu'à sommer les temps d'éxécutions pour chaque clique, on aura donc :

$$\sum_{C_i \in C} s_i |X_{C_{\max}}| mK^{2m} \in O(|C||X_{C_{\max}}|mK^{2m})$$

■

Bien que l'expression de $|X_{C_{\max}}|$ soit une exponentielle de la treewidth du réseau GAI, nous pouvons remarquer qu'en bornant cette valeur et en supposant le nombre de critères constant, l'algorithme s'éxécute en un temps pseudo-polynômial.

La démonstration de la complexité de l'algorithme approché suit un raisonement similaire, et applique directement un résultat de Papadimitriou et Yannakakis (2000) : Si K est la borne supérieure des valeurs d'utilité sur chaque critère, alors après la transformation Φ, chaque valeur sur un critère sera bornée par $\lceil \frac{\log(K)}{\log(1+\epsilon)} \rceil$. On en déduit donc directement que nous ne pourrons manipuler un ensemble de vecteurs de taille supérieure à $\lceil \frac{\log(K)}{\log(1+\epsilon)} \rceil^m$.

Proposition 14 (Complexité de l'algorithme approché) *Le temps de calcul de l'algorithme approché (algorithme 17) est en* $O\big(|C||X_{C_{\max}}|m(\frac{|C|\log(K)}{\epsilon})^{2m}\big)$.

Preuve. La démonstration est similaire à celle de l'algorithme exact. Analysons la différence entre les deux algorithmes. L'affectation de la valeur initiale de w et sa modification au sein de la boucle Pour sont des opérations en $O(1)$ et ne changent rien à la complexité globale de l'algorithme. La différence majeure est contenue au sein des appels à la fonction `NonDominés` (et `NonDominésParCase`). Étant donné que la relation de dominance n'est pas la même que celle de l'algorithme exact, la borne supérieure sur la taille des ensembles de solutions diffère de la précédente démonstration.

La somme des cardinaux des ensembles transitant le long d'un séparateur ne peut être supérieure à $\lceil \frac{\log(K)}{w_i \log(1+\epsilon)} \rceil^m$ où w_i est la valeur utilisée pour appliquer l'(ϵ, w)-dominance à un certain niveau de récursion. Or, on peut remarquer que nous nous intéressons essentiellement à de petites valeurs de ϵ, le dévelopement limité à l'ordre p de $\ln(1+\epsilon)$ est :

$$\ln(1+\epsilon) = \sum_{i=1}^{p} (-1)^{i-1} \frac{\epsilon^i}{i} + o(\epsilon^p)$$

On aura donc :

$$\ln(1+\epsilon) \geqslant \epsilon - \frac{\epsilon}{2} = \frac{\epsilon}{2}$$

Étant donné que nous n'appliquons pas directement l'ϵ-dominance, mais l'(ϵ, w)-dominance, avec une valeur initiale de w_i pour chaque clique fixée à $\frac{1}{|C|}$, on en déduit que la taille des ensembles est bornée par :

$$\lceil \frac{\log(K)}{w_i \log(1+\epsilon)} \rceil^m \leqslant \lceil \frac{2|C| \ln(K)}{\epsilon} \rceil^m$$

On en déduit que les appels à `NonDominés` et `NonDominésParCase` ont une complexité temporelle en $O(m(\frac{|C|\log(K)}{\epsilon})^{2m})$. En appliquant le même raisonnement que précédemment, la complexité globale de l'algorithme sera donc en $O\big(|C||X_{C_{\max}}|m(\frac{|C|\log(K)}{\epsilon})^{2m}\big)$. ■

4.3.2 Analyse expérimentale

Nous allons maintenant analyser expérimentalement nos deux algorithmes. Dans un premier temps, nous allons montrer que la structure de l'algorithme exact (et donc de l'algorithme approché) exploite pleinement la GAI-décomposabilité du problème. Pour cela, nous avons généré un réseau GAI en forme de chaîne contenant 17 attributs $X_1, \ldots, X_{17}$ et 16 cliques vérifiant $\forall j \in \{1, \ldots, 16\}\ C_j = \{j, j+1\}$. Les séparateurs sont placés entre chaque clique C_j et C_{j+1} (un tel séparateur sera donc représenté avec l'attribut X_{j+1} à l'intérieur du rectangle). Les tables d'utilité sont générées aléatoirement. Après avoir exécuté l'algorithme exact sur une instance, nous fusionnons les cliques reliées par un séparateur deux à deux (voir figure 4.6), pour obtenir un nouveau réseau GAI représentant la même relation de préférence. Nous exécutons l'algorithme sur cette nouvelle instance, puis nous refusionnons les cliques, et ainsi de suite jusqu'à obtenir un réseau GAI réduit à une unique clique.

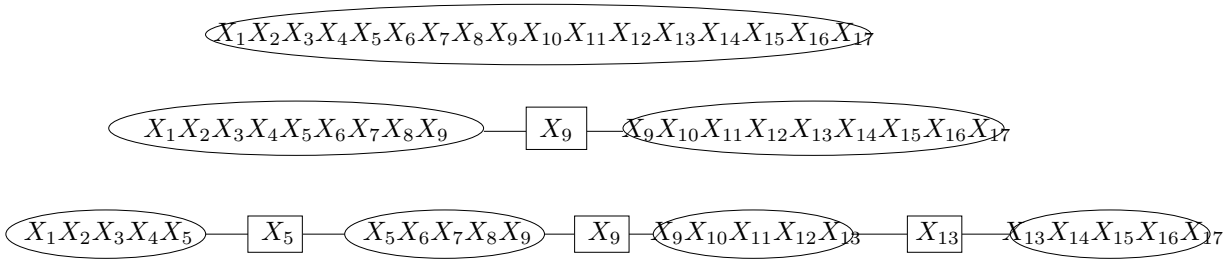


FIGURE 4.6 – Exemple de structures des réseaux générés dans la première expérimentation

En procédant ainsi, nous avons dressé le tableau suivant pour un problème à 2 et à 5 critères, en représentant les treewidths des différents réseaux générés. Le contenu de ce tableau représente les temps de calcul (en secondes) obtenus par l'éxécution de l'algorithme exact :

	Treewidth				
	2	3	5	9	17
2 critères	0.055	0.141	1.486	895.556	> 3600
5 critères	31.316	125.726	2467.78	> 3600	> 3600

Nous pouvons nous apercevoir que l'algorithme de collecte vectorielle exploite bien la GAI-décomposabilité du problème : plus la treewidth est faible, plus la fonction d'utilité vectorielle sera GAI-décomposable, et plus les temps de calcul sont réduits.

Nous avons ensuite considéré le pire cas possible de l'algorithme pour évaluer le potentiel de réduction de l'algorithme approché : l'adaptation des instances de Hansen (1980) aux réseaux GAI. Nous avons généré des réseaux GAI à n attributs binaires et n sous-fonctions d'utilité vectorielles en posant $\forall i \in \{1, \ldots, n-1\}$ $u_i(x_i, x_{i+1}) = (2^i x_i, 2^i(1-x_i))$ et $u_n(x_n) = (2^n x_n, 2^n(1-x_n))$. En procédant ainsi, nous obtenons la même configuration de solutions que celle décrite dans l'exemple 7 à la page 24, et nous aurons donc 2^n vecteurs dans la frontière de Pareto. Nous avons recensé les temps de calculs obtenus pour l'éxécution de l'algorithme exact ($\epsilon = 0$), et différentes valeurs de ϵ pour l'algorithme approché. Le tableau ci-dessous présente ces temps de calcul (en secondes) et le nombre de vecteurs composant la frontière de Pareto (# sol) en fonction du nombre d'attributs et de la valeur de ϵ.

	Nombre d'attributs					
	10		15		20	
ϵ	# sol	temps	# sol	temps	# sol	temps
0	1024	0.008	32768	0.257	10^6	8.9
0.01	407	0.008	644	0.014	782	0.206
0.05	117	0.002	149	0.004	193	0.010
0.1	58	0.002	72	0.002	109	0.005

Nous pouvons observer l'intérêt de l'algorithme approché : même pour de faibles valeurs de ϵ (0.01 par exemple), les temps de calcul sont drastiquement réduits, de même que le nombre de solutions composant la frontière.

Nous avons ensuite réalisé une nouvelle série d'expériences basées sur le même principe, mais en s'intéressant non plus à des instances pathologiques mais des instances aléatoires. Les réseaux GAI sont cette fois-ci générés à partir de leur graphe markovien : Chaque attribut peut prendre entre 3 et 5 valeurs. Pour chaque attribut, deux arêtes sont ajoutées aléatoirement entre cet attribut et deux autres déterminés aléatoirement. Le graphe markovien est ensuite triangulé en une forêt de jonction, et le réseau GAI est obtenu en associant à chaque clique de la forêt générée une sous-fonction d'utilité bicritère emplie de valeurs tirées entre 1 et 100. Nous obtenons le tableau suivant (chaque case est une moyenne de 100 expériences) :

	Nombre d'attributs							
	15		20		25		30	
ϵ	# sol	temps	# sol	temps	# sol	temps	# sol	temps
0	94	0.21	390	5.05	537	53.87	1595	103.02
0.01	37	0.029	102	0.064	123	0.30	147	0.64
0.05	9	0.012	14	0.017	17	0.060	18	0.13
0.1	3	0.006	3	0.009	4	0.027	4	0.059

La dernière série d'expérimentations a pour but d'appréhender l'impact de l'augmentation du nombre de critères sur l'algorithme. Pour cela nous avons fixé $\epsilon = 0.1$, nous avons généré de la même façon que précédemment des réseaux GAI mais pouvant cette fois-ci englober 2, 5 ou 10 critères. Le tableau suivant fournit les temps de calculs moyens obtenus (en secondes, chaque case est une moyenne de 100 expériences).

	Nombre d'attributs		
# critères	5	10	15
2	0.001	0.002	0.003
5	0.003	1.07	32.9
10	0.084	189	589

4.3.3 Conclusion

Nous avons pu voir dans ce chapitre comment utiliser les réseaux GAI vectoriels pour déterminer la frontière exacte ou approchée de Pareto (et les alternatives engendrant cette frontière). Ces algorithmes donnent des temps de réponse satisfaisants théoriquement et expérimentalement, même sur des instances pathologiques.

On peut toutefois faire une remarque : nous avons choisi de nous baser sur la notion d'(ϵ, w)-dominance pour construire la frontière approchée de Pareto, et nous avons posé une valeur $w_i = 1/|C|$ associée à la clique C_i. Toutefois, on peut se demander si la modification de ces valeurs peuvent améliorer l'algorithme, une condition suffisante sur les w_i pour que l'algorithme reste valide étant que $w_i \geqslant 0$ et $\sum_{C_i \in C} w_i = 1$ (de manière à garantir l'ϵ-couverture à la sortie de l'algorithme).

Chapitre 5

Approche filaire pour les réseaux GAI vectoriels [1]

Résumé. *Ce chapitre a pour but d'élaborer un algorithme général reposant sur un concept proche de A*. Il est général dans le sens où le même principe résoudra à la fois :*
- *La détermination de la frontière de Pareto,*
- *La détermination de la frontière de Lorenz,*
- *La détermination de la solution agrégée optimale (pour des agrégateurs P-monotones).*

Cet algorithme reposera sur des heuristiques majorantes directement liées aux réseaux GAI vectoriels : Dans une approche de type collecte dans un arbre de jonction, l'heuristique permettra d'obtenir sur chaque séparateur un vecteur qui sera une estimation majorante du gain qu'une solution potentielle peut obtenir en poursuivant sa phase de collecte. Il en résultera un algorithme propageant localement des solutions potentielles entre les cliques et capable de déterminer des situations dans lesquelles il n'est plus nécessaire de poursuivre la propagation d'une solution jusqu'à la racine de la collecte. Il est à noter qu'un algorithme de génération d'heuristique sera fourni et servira en tant que pré-traitement pour l'algorithme d'inférence.

1. Travaux publiés dans Dubus et al. (2009b) et à paraître dans Artificial Intelligence Journal

Ce chapitre a pour but de présenter une nouvelle approche pour la résolution de problèmes multicritères. Dans les chapitres précédents, nous avons présenté un algorithme de l'état de l'art pour déterminer les solutions agrégées optimales (chapitre 2). Nous avons étendu cet algorithme aux cas des réseaux GAI de forte treewidth (chapitre 3), et nous avons présenté un algorithme exact et un algorithme approché avec garantie de performance pour déterminer la frontière de Pareto (chapitre 4). Toutefois deux problèmes ont été laissés de côté :

- Comment déterminer efficacement la frontière de Lorenz d'un problème multicritère ?
- Comment déterminer efficacement la solution agrégée optimale sans posséder une borne supérieure linéairement décomposable de l'agrégateur choisi ? (par exemple, lorsque nous avons une intégrale de Choquet avec une capacité non convexe, ou un OWA avec un jeu de poids non décroissant).

Nous allons présenter une nouvelle approche qui sera à la fois exacte et générale pour résoudre ces problèmes : elle permettra de déterminer les frontières de Pareto, de Lorenz, et les solutions agrégées par n'importe quel type d'agrégateur P-monotone. Pour ce faire, nous allons nous baser sur une heuristique et formuler un algorithme proche de A* mais adapté aux réseaux GAI. Dans le chapitre 4, nous avions introduit un calcul exact basé sur une phase de collecte, les messages transitant le long de la structure d'arbre GAI étant des ensembles contenant des instanciations partielles des attributs et des vecteurs d'utilité. Les différents messages étaient combinés par un opérateur $\oplus$, et épurés par un opérateur de non-dominance. L'algorithme que nous allons présenter diffère de l'algorithme précédent sur deux points :

- Les associations entre instanciation et vecteur d'utilité ne transitent plus par « paquets » le long de l'arbre de jonction, mais les vecteurs d'utilité de n'importe quelle instanciation peuvent être envoyés d'une clique vers une autre indépendamment les uns des autres (en respectant l'ordre de collecte). Ainsi, des associations n'ayant pas atteint la clique racine de l'arbre de jonction seront des « ouverts » au sens de A*, c'est-à-dire qu'elles devront continuer à parcourir l'arbre pour que l'instanciation partielle des attributs devienne l'instanciation de tous les attributs.
- Au fur et à mesure de l'exécution de l'algorithme, certaines instanciations vont atteindre la clique racine avant les autres et composer une frontière de Pareto temporaire. L'algorithme dispose d'une heuristique sur chaque séparateur du réseau. Cette heuristique est une **estimation majorante** du gain sur les valeurs d'utilité que pourrait prendre un vecteur si on continue à le faire transiter le long de l'arbre de jonction. En procédant ainsi, si une association n'a pas fini de parcourir le réseau, mais que la somme (vectorielle) de son vecteur d'utilité et de son heuristique est dominée par un vecteur d'utilité ayant déjà achevé son parcours, nous pouvons déduire qu'il sera inutile de la faire transiter plus loin car elle aboutira nécessairement à une solution dominée. Elle pourra donc être supprimée à ce stade de l'agorithme.

La section 5.1 présentera ce que sont des heuristiques dans le contexte des réseaux GAI et founira un algorithme générant automatiquement ces heuristiques. La section 5.2 présentera une première version de l'algorithme pour résoudre le problème de la détermination de la frontière de Pareto. Ce premier algorithme doit être vu comme le socle sur lequel s'appuie cette nouvelle approche. En effet, la section 5.3.1 présentera une adapta-

tion de l'algorithme pour déterminer la frontière de Lorenz, tandis que la section 5.3.2 fera de même pour déterminer la solution agrégée optimale. Finalement, la section 5.4 présentera les résultats obtenus expérimentalement sur des adaptations de problèmes concrets, ainsi que sur des jeux d'essais aléatoires.

5.1 Fonctions heuristiques pour un réseau GAI vectoriel

Quel que soit le problème auquel nous nous intéresserons dans ce chapitre, nous aurons besoin d'une heuristique majorante définie sur les séparateurs du réseau GAI vectoriel. À la différence de l'algorithme A*, il semble difficile de demander au décideur une heuristique majorante sur ses préférences. Nous allons présenter dans cette section un algorithme pour générer automatiquement une telle heuristique. Celui-ci doit être vu comme un prétraitement du réseau GAI vectoriel, exécuté avant les différents algorithmes que nous présenterons par la suite.

5.1.1 Algorithme de détermination des heuristiques majorantes

Avant de présenter l'algorithme de génération automatique d'heuristique, nous allons dans un premier temps définir ce qu'est une heuristique dans le sens où nous allons l'exploiter.

Définition 81 (Fonction heuristique) *Soit $\mathcal{X} = \prod_{i=1}^{n} X_i$, u une fonction vectorielle GAI-décomposable selon $C = \{C_1, \ldots, C_q\}$, en $u(x) = \sum_{j=1}^{q} u_j(x_{C_j})$. Soit X_S un séparateur du réseau GAI vectoriel représentant u, partitionnant l'ensemble des cliques en $\mathcal{C}_1$ et $\mathcal{C}_2$ (cliques de part et d'autre du séparateur). Soit $D = \bigcup_{C_i \in \mathcal{C}_1} C_i$ et $E = \bigcup_{c_j \in \mathcal{C}_2} C_j$, on aura donc $D \cup E = \{1, \ldots, n\}$ et $D \cap E = S$. On dit que $\mathcal{H}$ est une fonction heuristique majorante définie sur X_S et estimant X_E si, et seulement si, on a :*

- $\mathcal{H} : X_S \to \mathbb{R}^m$.
- $\forall x_E \in X_E \ \mathcal{H}((x_E)_S) \geqslant \sum_{C_j \in C : C_j \subseteq E} u_j((x_E)_{C_j})$.

L'intérêt de cette définition est d'exploiter directement la GAI-décomposabilité (voir figure 5.1) : Si on considère l'ensemble des sous-fonctions d'utilité, la définition partitionne cet ensemble en deux parties : Chaque sous-fonction est définie sur un ensemble d'attributs dont les indices appartiennent soit à E soit à D, S étant l'ensemble des indices en commun entre E et D. D'un point de vue graphique, on peut interpréter S comme un séparateur, et E et D comme deux parties de l'arbre de jonction. Une fonction heuristique est donc une fonction qui à chaque instanciation du séparateur renverra une estimation majorante de l'utilité vectorielle qu'il est possible d'obtenir sur n'importe quelle instanciation de X_E à valeur de X_S fixée. Une conséquence directe de cette définition est donnée par la proposition 15.

Proposition 15 (Majoration de l'utilité globale) *Soit $\mathcal{X} = \prod_{i=1}^{n} X_i$, u une fonction d'utilité vectorielle sur $\mathcal{X}$ GAI-décomposable selon $C = \{C_1, \ldots, C_q\}$. Considérons un séparateur X_S du réseau GAI représentant u, et partitionnant l'ensemble des cliques en $\mathcal{C}_1$ et $\mathcal{C}_2$ (de la même façon que dans la définition 81). Posons $D = \bigcup_{C_i \in \mathcal{C}_1} C_i$ et $E = \bigcup_{C_j \in \mathcal{C}_2} C_j$. Soit $\mathcal{H}$ une fonction heuristique définie sur X_S et estimant X_E et $w = \sum_{C_j \in \mathcal{C}_1} u_j$. On a :*

$$\forall x \in \mathcal{X} \ w(x_D) + \mathcal{H}(x_S) \geqslant u(x)$$

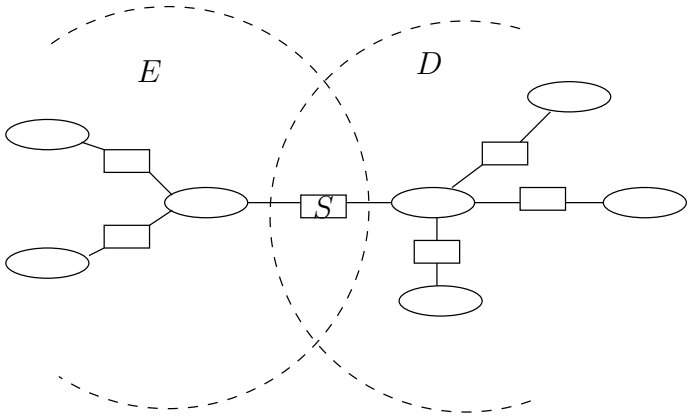


FIGURE 5.1 – Exemple de positionnement des ensembles D, E et S

Preuve. Le partitionnement de l'ensemble C en $\mathcal{C}_1$ et $\mathcal{C}_2$ induit un partitionnement de l'ensemble des sous-fonctions d'utilité en

$$\forall x \in \mathcal{X}\ u(x) = \sum_{C_i \in C} u_i(x_{C_i}) = \sum_{C_i \in \mathcal{C}_1} u_i(x_{C_i}) + \sum_{C_j \in \mathcal{C}_2} u_j(x_{C_j}) = w(x_D) + \sum_{C_j \in \mathcal{C}_2} u_j(x_{C_j}).$$

Par définition de la fonction heuristique (deuxième tiret), on aura donc bien $w(x_D) + \mathcal{H}(x_S) \geqslant u(x)$. ■

Cette proposition introduit donc le fait qu'au fur et à mesure d'une collecte vectorielle, si, sur chaque séparateur, il existe une fonction heuristique estimant les parties non parcourues sur réseau, il sera possible à chaque étape de calculer une borne supérieure de l'utilité qui sera obtenue en fin de collecte. Cette borne pourra donc être exploitée pour supprimer les solutions partielles qui, combinées à l'heuristique, fourniraient des solutions dominées à la racine de la collecte.

Nous allons maintenant nous intéresser à la génération de fonctions heuristiques sur chaque séparateur d'un réseau GAI. Une telle génération sera basée sur un algorithme de type collecte/diffusion.

Lors de la phase de collecte sur un réseau GAI vectoriel, le but sera d'obtenir sur la clique collectant les derniers messages (nommée clique racine de la collecte, nous la noterons C_{root}), les vecteurs d'utilité dont chaque composante correspondra à la plus forte valeur d'utilité qu'il est possible d'obtenir dans le réseau. Ce problème est donc très proche de la phase de collecte de l'algorithme de choix optimal, à la différence que la somme sera une somme vectorielle, et que la maximisation devra se faire composante par composante sur les vecteurs. De la même façon que les algorithmes de collecte définis auparavant, nous noterons $\phi_{i,j}$ le message transitant sur le séparateur reliant la clique C_i à la clique C_j. Nous noterons ψ_i la sommation de la fonction d'utilité u_i avec l'ensemble des messages reçus dans la collecte (voir algorithme 18).

Remarque 24 *De manière à lever toute ambiguïté, on notera maxparcomp la maximisation vectorielle (composante par composante), par exemple, maxparcomp*$\{(1,42),(3,14)\} = (3,42)$. *De la même manière que pour la maximisation scalaire, nous utiliserons la possibilité d'effectuer les maximisations directement sur une table. Ainsi, si on pose* $u : X_1 \times$

$X_2 \to \mathbb{R}^2$, *avec* $X_1 = \{x_{11}, x_{12}\}$ *et* $X_2 = \{x_{21}, x_{22}\}$, *et* $u(x_{11}, x_{21}) = (10, 0)$, $u(x_{11}, x_{22}) = (0, 10)$, $u(x_{12}, x_{21}) = (8, 9)$, $u(x_{12}, x_{22}) = (15, 1)$. *Alors,* $v = maxparcomp_{X_2} u$ *est une fonction définie sur* X_1 *et vérifiant* $v(x_{11}) = maxparcomp\{(10, 0), (0, 10)\} = (10, 10)$ *et* $v(x_{12}) = maxparcomp\{(8, 9), (15, 1)\} = (15, 9)$.

Algorithme 18 : CollecteHeuristique

Entrées :
act : indice de la clique actuelle
succ : indice de la clique appelante

$\psi_{act} \leftarrow u_{act}$;
pour chaque *clique* C_j *voisine de* C_{act} **faire**
  si $j \neq succ$ **alors**
    $\phi_{j,act} \leftarrow$ `CollecteHeuristique(`j, act`)`;
    $\psi_{act} \leftarrow \psi_{act} + \phi_{j,act}$;
  fin
fin
si $act \neq succ$ **alors**
  retourner $\phi_{act,succ} \leftarrow \text{maxparcomp}_{C_{act} \setminus S_{act,succ}} \psi_{act}$;
fin

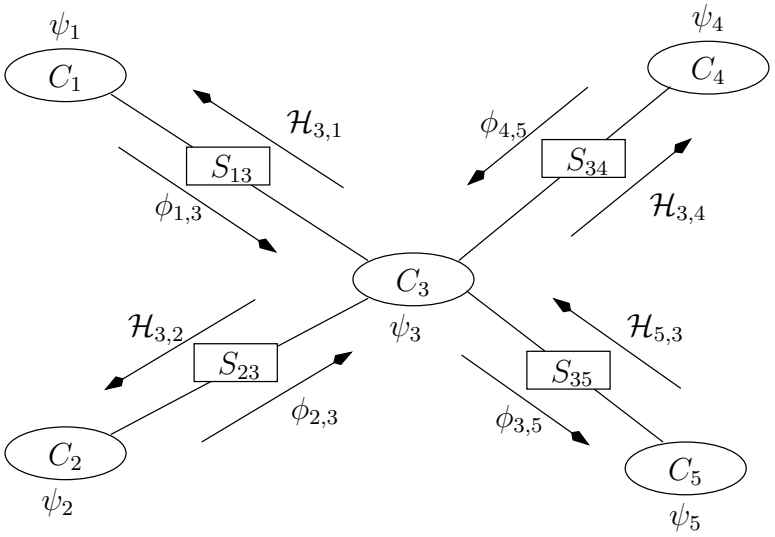


FIGURE 5.2 – Représentation des variables de l'algorithme de détermination des fonctions heuristiques ($C_{root} = C_5$)

La phase de diffusion aura pour but de déterminer les valeurs heuristiques associées à chaque valeur des séparateurs. Le réseau sera parcouru en sens inverse de la collecte. Nous noterons $\mathcal{H}_{j,i}$ l'heuristique obtenue ainsi. Etant donné que les heuristiques transiteront en sens inverse de la collecte, nous aurons donc en fin de phase de diffusion, pour chaque message $\phi_{i,j}$ une heuristique $\mathcal{H}_{j,i}$. La première fonction heuristique déterminée sera celle portant sur le séparateur conduisant à la clique racine, il suffira de maximiser sur les attributs du séparateur le contenu de ψ_{root}. Ensuite, pour chaque clique C_j recevant une

heuristique $\mathcal{H}_{z,j}$, on pourra affirmer que $\psi_j + \mathcal{H}_{z,j}$ sera une estimation majorante des plus fortes valeurs d'utilité vectorielle qu'il est possible d'obtenir. Donc, la fonction heuristique $\mathcal{H}_{j,i}$ se déduira facilement en maximisant $\psi_j + \mathcal{H}_{j,i} - \phi_{i,j}$. La figure 5.2 symbolise les différentes variables utilisées dans l'algorithme, l'algorithme de diffusion correspond à l'algorithme 19.

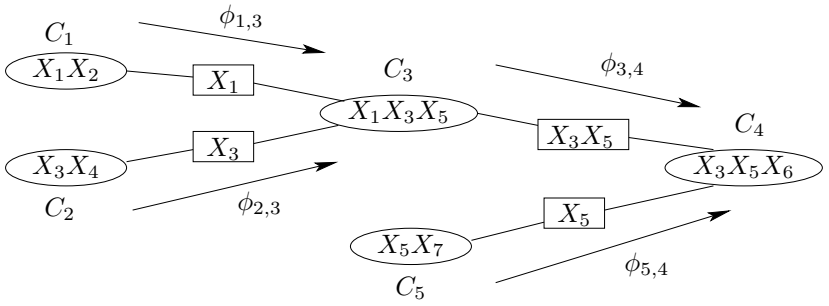
Algorithme 19 : DiffusionHeuristique

```
1  pour chaque clique C_act voisine de C_root faire
2  |  H_{root,act} ← maxparcomp_{C_root \ S_{root,act}}(ψ_root − φ_{act,root});
3  fin
4  pour chaque clique C_act ≠ C_root en ordre inverse de la collecte faire
5  |  succ ← Succ(act);
6  |  pour chaque clique C_j voisine de C_act faire
7  |  |  si j ≠ succ alors
8  |  |  |  H_{act,j} ← maxparcomp_{C_act \ S_{act,j}}(ψ_act + H_{succ,act} − φ_{j,act});
9  |  |  fin
10 |  fin
11 fin
```

Exemple 34 *Considérons le réseau GAI vectoriel représenté par la figure 5.3, et effectuons un calcul des fonctions heuristiques en considérant une collecte/diffusion dont la clique racine sera $X_3X_5X_6$ ($C_4 = C_{root}$). On aura donc comme ordre de collecte possible : $X_1X_2(C_1)$, $X_3X_4(C_2)$, $X_1X_3X_5(C_3)$, $X_5X_7(C_5)$, $X_3X_5X_6(C_4)$. La phase de collecte est initiée par l'appel de la fonction* `CollecteHeuristique`*(4, 4). Les premières tables à être générées seront $\psi_1 = u_1$ servant à produire le message $\phi_{1,3} = maxparcomp_{x_2 \in X_2}\psi_1$. La maximisation se faisant composante de vecteur par composante de vecteur, on aura donc $\phi_{1,3}(x_1) = (v^1, v^2)$, $v^1 = \max_{x_2 \in X_2} \psi_1(x_1, x_2)^1$ et $v^2 = \max_{x_2 \in X_2} \psi_1(x_1, x_2)^2$. La fonction $\phi_{1,3}$ ainsi obtenue est représentée au sein de la figure 5.4(a). De la même façon, on aura $\psi_2 = u_2$ et $\phi_{2,3} = maxparcomp_{x_4 \in X_4}\psi_2$ (représentée par la figure 5.4(b)).*

Les messages $\phi_{1,3}$ et $\phi_{2,3}$ transitent le long de leur séparateur pour atteindre la clique C_3. Ils sont ensuite additionnés avec la sous-fonction d'utilité vectorielle u_3 pour former $\psi_3 = u_3 + \phi_{1,3} + \phi_{2,3}$, qui est décrit sur la figure 5.4(c). Cette nouvelle fonction va ensuite être réduite en $\phi_{3,4} = maxparcomp_{x_1 \in X_1}\psi_3$ (voir figure 5.4(d)). Pour la feuille correspondant à la clique C_5, on aura $\psi_4 = u_4$ et $\phi_{5,4} = maxparcomp_{x_7 \in X_7}\psi_5$. La dernière opération effectuée par l'algorithme correspondra donc à traiter les messages au niveau de la clique racine : $\psi_4 = u_4 + \phi_{3,4} + \phi_{5,4}$ (figure 5.4(f)), ψ_4 correspond donc à une majoration des vecteurs d'utilité possibles sur $\mathcal{X}$.

Il ne reste plus qu'à effectuer la phase de diffusion pour déterminer les fonctions heuristiques associées à chaque séparateur. La première étape consiste à déterminer les heuristiques à proximité de la clique racine. Étant donné que ψ_4 contient une estimation majorante des vecteurs d'utilité qu'il est possible d'obtenir dans le réseau GAI, $\psi_4 - \phi_{3,4}$ correspondra donc à $u_4 + \phi_{5,4}$, c'est-à-dire à une estimation majorante du vecteur d'utilité qu'il est possible d'obtenir dans le sous-réseau GAI composé de C_4 et C_5 ($u_4 + u_5$). Donc, $\mathcal{H}_{4,3} = maxparcomp_{x_6 \in X_6}(\psi_4 - \phi_{3,4})$ restera une estimation majorante de ce sous-réseau, sous forme d'une fonction portant sur le produit cartésien des attributs en commun entre



(a) Arbre de jonction

	X_1	
X_2	x_{11}	x_{12}
x_{21}	$(1,5)$	$(2,3)$
x_{22}	$(8,2)$	$(3,4)$
x_{23}	$(7,1)$	$(6,2)$

(b) $u_1(x_1, x_2)$

	X_3	
X_4	x_{31}	x_{32}
x_{41}	$(2,9)$	$(5,2)$
x_{42}	$(2,2)$	$(4,9)$
x_{43}	$(4,1)$	$(5,3)$

(c) $u_2(x_3, x_4)$

	X_3			
	x_{31}		x_{32}	
	X_1		X_1	
X_5	x_{11}	x_{12}	x_{11}	x_{12}
x_{51}	$(3,3)$	$(5,3)$	$(1,2)$	$(4,4)$
x_{52}	$(2,4)$	$(1,4)$	$(6,2)$	$(2,5)$

(d) $u_3(x_1, x_3, x_5)$

	X_7	
X_5	x_{71}	x_{72}
x_{51}	$(1,0)$	$(1,1)$
x_{52}	$(1,1)$	$(2,1)$

(e) $u_5(x_5, x_7)$

	X_3			
	x_{31}		x_{32}	
	X_6		X_6	
X_5	x_{61}	x_{62}	x_{61}	x_{62}
x_{51}	$(8,1)$	$(5,6)$	$(2,1)$	$(1,5)$
x_{52}	$(7,0)$	$(8,6)$	$(1,1)$	$(0,5)$

(f) $u_4(x_3, x_5, x_6)$

FIGURE 5.3 – Réseau GAI vectoriel de l'exemple 34

le sous-réseau et le reste du réseau. Cette table est représentée sur la figure 5.5(c). De la même façon, on aura $\mathcal{H}_{4,5} = maxparcomp_{(x_3,x_6)\in X_3\times X_6}(\psi_4 - \phi_{5,4})$ *(voir la figure 5.5(d)).*

Il ne reste plus qu'à parcourir les cliques dans l'ordre inverse de la collecte : C_5 *étant une feuille, aucune nouvelle opération ne sera effectuée. Pour* C_3, $\psi_3 + \mathcal{H}_{4,5}$ *correspondra à une estimation majorante de u (sur les attributs formant la clique). Donc,* $\psi_3 + \mathcal{H}_{4,5} - \phi_{1,3}$ *sera une estimation majorante de* $u_2 + u_3 + u_4 + u_5$. *On calcule donc* $\mathcal{H}_{3,1} = maxparcomp_{(x_3,x_5)\in X_3\times X_5}(\psi_3 + \mathcal{H}_{4,5} - \phi_{1,3})$ *(figure 5.5(a)). De la même façon,* $\psi_3 + \mathcal{H}_{4,5} - \phi_{2,3}$ *sera une estimation majorante de* $u_1 + u_3 + u_4 + u_5$, *d'où* $\mathcal{H}_{3,2} = maxparcomp_{(x_1,x_5)\in X_1\times X_5}\psi_3 + \mathcal{H}_{4,5} - \phi_{2,3}$ *(figure 5.5(b)). Les cliques* C_2 *et* C_1 *étant des feuilles, aucune nouvelle opération ne sera effectuée, et l'algorithme se termine.*

5.1.2 Validité et complexité de l'algorithme

Nous allons maintenant montrer que l'algorithme présenté, de type collecte/diffusion, calcule bien des fonctions heuristiques du réseau GAI vectoriel. Dans un premier temps, la proposition 16 montre la propriété fondamentale de l'algorithme `CollecteHeuristique` : le fait que toute fonction ψ_i soit une borne supérieure de la somme des sous-fonctions d'utilité associées à C_i, aux prédecesseurs de C_i, aux prédecesseurs des prédecesseurs, etc... (voir la notion de prédecesseur dans un ordre de collecte, définie page 63). De

X_1	
x_{11}	$(8,5)$
x_{12}	$(6,4)$

(a) $\phi_{1,3}(x_1)$

X_3	
x_{31}	$(4,9)$
x_{32}	$(5,9)$

(b) $\phi_{2,3}(x_3)$

	X_3			
	x_{31}		x_{32}	
	X_1		X_1	
X_5	x_{11}	x_{12}	x_{11}	x_{12}
x_{51}	$(15,17)$	$(15,16)$	$(14,16)$	$(15,17)$
x_{52}	$(14,18)$	$(11,17)$	$(19,16)$	$(13,18)$

(c) $\psi_3(x_1, x_3, x_5)$

X_5	X_3	
	x_{31}	x_{32}
x_{51}	$(15,17)$	$(15,17)$
x_{52}	$(14,18)$	$(19,18)$

(d) $\phi_{3,4}(x_3, x_5)$

X_5	
x_{51}	$(1,1)$
x_{52}	$(2,1)$

(e) $\phi_{5,4}(x_5)$

	X_3			
	x_{31}		x_{32}	
	X_6		X_6	
X_5	x_{61}	x_{62}	x_{61}	x_{62}
x_{51}	$(24,19)$	$(21,24)$	$(18,19)$	$(17,23)$
x_{52}	$(23,19)$	$(24,25)$	$(22,20)$	$(21,24)$

(f) $\psi_4(x_3, x_5, x_6)$

FIGURE 5.4 – Tables calculées lors de la collecte dans l'exemple 34

X_1	
x_{11}	$(16,20)$
x_{12}	$(18,20)$

(a) $\mathcal{H}_{3,1}(x_1)$

X_3	
x_{31}	$(20,16)$
x_{32}	$(17,20)$

(b) $\mathcal{H}_{3,2}(x_3)$

X_5	X_3	
	x_{31}	x_{32}
x_{51}	$(9,7)$	$(3,6)$
x_{52}	$(10,7)$	$(3,6)$

(c) $\mathcal{H}_{4,3}(x_3, x_5)$

X_5	
x_{51}	$(23,23)$
x_{52}	$(22,24)$

(d) $\mathcal{H}_{4,5}(x_5)$

FIGURE 5.5 – Tables calculées lors de la diffusion dans l'exemple 34

manière à démontrer simplement ces propositions, nous allons introduire préalablement la notion d'antécédents d'une clique (définition 82).

Définition 82 (Antécédents d'une clique) *Soit $C = \{C_1, \ldots, C_q\}$ les cliques d'un réseau GAI. On définit la fonction « Antécédents » (notée Ant) de la façon suivante :*

- $Ant : \{1, \ldots, q\} \to \mathcal{P}(\{1, \ldots, q\})$.
- $\forall i \in \{1, \ldots, q\}\ Ant(i) = \{i\} \cup \bigcup_{j \in Pred(i)} Ant(j)$.

Exemple 35 *Dans l'exemple 34, on a $Ant(3) = \{1, 2, 3\}$ et $Ant(4) = \{4\}$.*

Cette définition récursive permet de définir par leur indice l'ensemble des cliques précédant une clique C_i durant la phase de collecte. Ainsi, il est nécessaire de vérifier que chaque fonction ψ_i de l'algorithme 18 correspond bien à une estimation majorante de la somme de chaque sous-fonction d'utilité contenue dans une clique C_j avec $j \in Ant(i)$.

Proposition 16 *À la fin de l'exécution de* `CollecteHeuristique(root, root)`*, on a :*

$$\forall x \in \mathcal{X}\ u(x) \leqslant \psi_{root}(x_{C_{root}}).$$

Preuve. Cette propriété se démontre facilement par récurrence sur les sous-réseaux GAI (représentés par la fonction Ant) engendrés par la phase de collecte. Il suffit de montrer que toute fonction ψ_i formée dans l'algorithme 18 majore $\sum_{j \in Ant(i)} u_j$, de même que toute fonction $\phi_{i,j}$.

- Si C_{act} est une feuille alors $Ant(act) = \{act\}$, on aura $\psi_{act} = u_{act}$ et donc, on aura bien $\forall x_{C_{act}} \in X_{C_{act}}\ \psi_{act}(x_{C_{act}}) \geqslant u_{act}(x_{C_{act}})$. De plus, en considérant le séparateur $S_{act,succ} \subset C_{act}$, si $\phi_{act,succ} = \text{maxparcomp}_{C_{act} \setminus S_{act,succ}} \psi_{act}$, alors on aura clairement $\forall x \in X_{C_{act}}\ \phi_{act,succ}(x_{S_{act,succ}}) \geqslant \psi_{act}(x_{C_{act}}) \geqslant u_{act}(x_{C_{act}})$.
- Considérons une clique C_{act} qui ne soit pas une feuille. On aura donc $\psi_{act} = u_{act} + \sum_{i \in Pred(act)} \phi_{i,act}$. Or, étant donné la structure de l'arbre de jonction et l'ordre de collecte, on aura $Ant(act) = \{act\} \cup \bigcup_{j \in Pred(act)} Ant(j)$. Par hypothèse de récurrence, les $\phi_{j,act}$ sont des bornes supérieures de $\sum_{k \in Ant(j)} u_k$, donc par construction, $\psi_{act} = u_{act} + \sum_{k \in Pred(act)} \phi_{k,act}$ est une borne supérieure de $\sum_{i \in Ant(act)} u_i$. Dans le cadre d'un appel à `CollecteHeuristique(`act, $succ$`)`, la fonction $\phi_{act,succ}$ produite à la fin de l'appel récursif étant définie comme une maximisation de ψ_{act}, elle est donc aussi une borne supérieure de $\sum_{i \in Ant(act)} u_i$.

Donc, un appel à `CollecteHeuristique(`$root$, $root$`)` produit bien une fonction ψ_{root} vérifiant $\forall x \in \mathcal{X}\ u(x) \leqslant \psi_{root}(x_{C_{root}})$, étant donné que $\sum_{i \in Ant(root)} u_i = \sum_{C_i \in C} u_i = u$. ■

À partir de cette propriété et de la construction des fonctions ψ et ϕ, il est ensuite possible de montrer que les fonctions $\mathcal{H}$ calculées par `DiffusionHeuristique` sont des estimations majorantes d'une partie du réseau GAI liée à l'ordre de collecte (proposition 17).

Proposition 17 *Les fonctions $\mathcal{H}_{act,j}$ produites par* `DiffusionHeuristique` *sont des fonctions heuristiques du réseau GAI vectoriel définies sur $X_{S_{act,j}}$ et estimant X_E avec $E = \bigcup_{i \in \{1,\ldots,q\} \setminus Ant(j)} C_i$.*

Preuve. Première boucle : ψ_{root} est une estimation majorante de l'intégralité du réseau GAI vectoriel. Par construction (pendant la collecte de l'heuristique), on a $\psi_{root} = u_{root} + \sum_{i \in Pred(root)} \phi_{i,root}$. Donc, $\psi_{root} - \phi_{act,root} = u_{root} + \sum_{i \in Pred(root)} \phi_{i,root} - \phi_{act,root}$. Étant donné que $\forall i \in Pred(root)\ \phi_{i,root}$ est une estimation majorante de $\sum_{j \in Ant(i)} u_j$, on a donc que $\psi_{root} - \phi_{act,root}$ est une estimation majorante de $\sum_{i \in \{1,\ldots,q\} \setminus Ant(act)} u_i$. Donc $\mathcal{H}_{root,act} = \text{maxparcomp}_{C_{act} \setminus S_{act,root}}(\psi_{root} - \phi_{act,root})$ est une estimation majorante de $\sum_{i \in \{1,\ldots,q\} \setminus Ant(act)} u_i$ définie sur $X_{S_{act,root}}$. $\mathcal{H}_{root,act}$ est donc bien une fonction heuristique définie sur $X_{S_{act,root}}$ et estimant X_E avec $E = \bigcup_{i \in \{1,\ldots,q\} \setminus Ant(act)} C_i$.

Seconde boucle : La seconde boucle reprend l'idée de la première, en y introduisant une fonction heuristique. Supposons que la propriété soit vraie jusqu'à l'itération sur C_{act}, les fonctions sont construites par l'expression $\mathcal{H}_{act,j} \leftarrow \text{maxparcomp}_{C_{act} \setminus S_{act,j}}(\psi_{act} + \mathcal{H}_{succ,act} - \phi_{j,act})$. Or (voir démonstration 16), on a $\psi_{act} = u_{act} + \sum_{k \in Pred(act)} \phi_{k,act}$ donc $\psi_{act} + \mathcal{H}_{succ,act} - \phi_{j,act} = u_{act} + \mathcal{H}_{succ,act} + \sum_{k \in Pred(act)/k \neq j} \phi_{k,act}$. De la même façon que pour la première boucle, on aura donc une estimation majorante de $\sum_{i \in \{1,\ldots,q\} \setminus Ant(j)} u_i$. ■

Finalement, il ne reste plus qu'à évaluer la complexité globale des algorithmes (proposition 18). Cette complexité est proche de la complexité obtenue pour le choix optimal, à un facteur m près. Cela n'a rien d'étonnant : Le principe est exactement le même, seul le contenu des messages et des sous-fonctions change. Les opérations finales portant sur des

vecteurs de dimension m, l'algorithme de collecte effectue m fois plus d'opérations. L'algorithme de diffusion peut être vu comme une collecte en sens inverse (une redistribution des messages) et effectue des opérations similaires à la phase de collecte (maximisation, opérations sur des hypermatrices).

Proposition 18 *Les fonctions* `CollecteHeuristique` *et* `DiffusionHeuristique` *s'exécutent en* $O(|C|m \max_{C_i \in C} \prod_{j \in C_i} |X_j|)$.

Preuve. On remarque le nombre nombre de valeurs contenues dans une fonction (sous-utilité, messages, heuristiques) est inférieur ou égal à $m \max_{C_i \in C} \prod_{j \in C_i} |X_j| = K$, les opérations d'addition, de maximisation ou de soustraction seront donc en $O(mK)$.

Dans la fonction `CollecteHeuristique`, on aura un nombre d'additions et de maximisations de fonctions proportionnel au nombre de séparateurs du réseau GAI vectoriel. Or, ce réseau a une structure arborescente, on aura donc $O(|C|)$ opérations, et donc la complexité temporelle globale de l'algorithme sera en $O(|C|K)$.

Dans la fonction `DiffusionHeuristique`, on aura un nombre d'additions, de maximisations et de soustractions proportionnel au nombre de séparateurs du réseau. Donc, pour les mêmes raisons que précédemment, la complexité temporelle globale de l'algorithme sera en $O(|C|K)$. ■

5.2 Détermination de la frontière de Pareto

Nous disposons maintenant d'un algorithme permettant de déterminer une fonction heuristique sur chaque séparateur du graphe étant donné un ordre de collecte. Dans cette section, nous allons montrer comment exploiter une telle heuristique pour déterminer la frontière de Pareto d'un problème. Dans un premier temps, afin de bien comprendre le fonctionnement de l'algorithme, nous n'exploiterons pas l'heuristique et nous présenterons un algorithme inefficace (mais pédagogique), pour comprendre les méthodes de déplacement de vecteurs utilisés, puis (en 5.2.2), nous adapterons cet algorithme pour intégrer des principes de coupe par heuristique.

5.2.1 Approche filaire sans heuristique

Dans le chapitre 4, le principe de détermination de la frontière de Pareto pouvait être grossièrement vu comme une phase de collecte faisant transiter des vecteurs par « paquets » le long des séparateurs, et effectuant des tests de non-dominance à chaque « mouvement » de ces paquets. L'idée de l'approche filaire est de faire transiter des vecteurs un par un, et par petits mouvements, c'est-à-dire uniquement en effectuant « un pas » vers la clique suivante.

Pour formaliser ces notions, nous allons d'abord définir le contenu qui transitera le long de l'arbre de jonction. Nous sommes intéressés par trois informations :

- La valeur vectorielle d'une solution partielle.
- L'instanciation partielle des attributs engendrant cette valeur vectorielle.
- La position actuelle du message dans l'arbre de jonction.

Nous appellerons ces associations des labels, la définition formelle associée à cette idée sera donc :

Définition 83 (Label) *Un label est un triplet $\langle v, x_D, i\rangle$ où v est un vecteur de $\mathbb{R}^m$, x_D est une instanciation des attributs de X_D, et i est l'indice d'une clique $C_i \in C$.*

De façon à convertir une sous-fonction d'utilité en un ensemble de labels, nous utiliserons l'opération de « labelisation », dont le principe consiste simplement à parcourir toutes les valeurs des sous-fonctions d'utilité, comme le montre l'algorithme 20.

Algorithme 20 : Labeliser

Entrée : $u_i : X_{C_i} \rightarrow \mathbb{R}^m$

$\mathcal{V} \leftarrow \varnothing$;
pour chaque $x_{C_i} \in X_{C_i}$ **faire**
  $\mathcal{V} \leftarrow \mathcal{V} \cup \{\langle u_i(x_{C_i}), x_{C_i}, i\rangle\}$;
fin
retourner $\mathcal{V}$;

Nous sommes donc capables de générer des labels à faire transiter à partir d'une clique du graphe. Nous allons maintenant définir en quoi consiste un pas de notre approche filaire : il s'agit de sélectionner un label, de le faire transiter le long d'un séparateur pour atteindre la prochaine clique de la phase de collecte. Pour effectuer un tel déplacement, nous pouvons noter certains aspects importants :

- Le déplacement d'un label vers une clique engendre plusieurs labels sur cette clique : il doit en effet être combiné avec tous les labels de cette clique qui lui sont compatibles (à valeur de séparateur fixée).
- Ce phénomène de démultiplication du vecteur par combinaison n'est pas uniquement dû à l'arrivée sur une clique, d'autres vecteurs sont peut-être déjà « venus » sur cette clique à partir d'une autre clique via des séparateurs adjacents. Il est donc non seulement nécessaire de stocker le label transitant sur le séparateur (nous noterons $\mathcal{M}_{ij}$ les messages stockés sur le séparateur S_{ij}), mais aussi de le combiner avec tous les autres vecteurs compatibles ayant déjà transité (et qui sont donc stockés) sur les séparateurs adjacents.

De manière à simplifier le pseudo-code et les notations utilisées, nous allons introduire une série d'opérateurs :

- L'opérateur $\otimes$ effectue une combinaison de deux ensembles de labels : si $\mathcal{V}$ et $\mathcal{W}$ sont deux ensembles de labels, alors $\mathcal{V} \otimes \mathcal{W} = \{\langle v + w, x_{D\cup E}, i\rangle \,/ \langle v, x_D, i\rangle \in \mathcal{V}$ et $\langle w, x_E, j\rangle \in \mathcal{W}\}$. Cet opérateur génère donc l'ensemble des labels qu'il est possible d'obtenir par une somme vectorielle des vecteurs contenus dans ces labels. Notons que les valeurs des attributs de x_D et x_E doivent être compatibles : $(x_D)_{D\cap E} = (x_E)_{D\cap E}$. Les instanciations des attributs sont regroupées entre elles. Il est à noter que cet opérateur a un effet absorbant sur l'indice de la clique contenu dans les labels (i est toujours conservé).
- Un opérateur ($\rightarrow$) permettant de modifier l'indice de clique contenu dans un ensemble de labels : $\mathcal{V}_{\rightarrow j} = \{\langle v, x_D, j\rangle / \langle v, x_D, i\rangle \in \mathcal{V}\}$.
- Tous les labels d'un ensemble contiennent une instanciation du même ensemble d'attributs, l'opérateur [] vise à extraire l'ensemble des labels compatibles avec une instanciation partielle de ces attributs : $\mathcal{V}[y_S] = \{\langle v, x_D, i\rangle \in \mathcal{V} / (x_D)_S = y_S\}$.

– $\mathcal{V}_{\Downarrow X_E} = \bigcup_{x_E \in X_E} \mathrm{ND}_P(\mathcal{V}[x_E])$, l'opérateur vise à effectuer un test de non-dominance de Pareto uniquement sur les labels compatibles entre eux selon certains attributs (X_E), puis à reformer l'ensemble des labels.

Algorithme 21 : UnPas

Entrée : $\langle v, x_D, i \rangle$: un label

```
1  j ← Succ(i);
2  M_ij ← M_ij ∪ {⟨v, x_D, i⟩};
3  V ← Labeliser(u_j) ⊗{⟨v, x_D, i⟩};
4  pour chaque z ∈ Pred(j) faire
5  |  si z ≠ i alors
6  |  |  V ← (V ⊗ M_zj);
7  |  fin
8  fin
9  si j = root alors
10 |  V ← NonDominés(V);
11 sinon
12 |  succ ← Succ(j);
13 |  V ← V_⇓X_{S_{j,succ}};
14 fin
15 V ← V_→j;
16 retourner V;
```

L'algorithme 21 implante cette notion de mouvement et s'interprète de la façon suivante : j est initialisé comme étant le successeur de i dans l'ordre de collecte. Comme le label va transiter de la clique C_i vers la clique C_j, il est stocké dans la liste des messages transitant sur ce séparateur : $\mathcal{M}_{ij}$. Le label va ensuite être combiné au contenu de l'utilité u_j pour former l'ensemble de labels $\mathcal{V}$, qui va lui même être combiné aux messages compatibles ayant déjà transité sur les séparateurs adjacents (les $\mathcal{M}_{zj}$). Finalement, si le label a atteint la clique racine, un test de non-dominance est exécuté globalement sur $\mathcal{V}$. Sinon, un test de non-dominance sera effectué uniquement à valeur de séparateur $X_{S_{j,succ}}$ fixée, puis les labels seront modifiés pour prendre en compte le changement d'indice de leur position actuelle (ils auront transité de C_i vers C_j).

On peut toutefois objecter à cette notion de mouvement qu'il est nécessaire que les ensembles de labels ayant déjà transité sur des séparateurs adjacents à la clique de destination (les $\mathcal{M}_{zj}$) ne soient pas vides. En effet, si un label transite de la clique C_i vers la clique C_j, et s'il existe une clique C_z telle que $j = \texttt{Succ}(z)$ avec $\mathcal{M}_{zj} = \varnothing$, alors il sera impossible d'obtenir une combinaison de labels cohérents à stocker dans C_j. Pour résoudre ce problème, il est nécessaire d'élaborer un précalcul initialisant les $\mathcal{M}_{zj}$ sur tous les séparateurs du réseau GAI.

L'idée de l'algorithme 22 est d'initier un déplacement à partir des feuilles du réseau GAI vers la clique racine. Seul un label pour chaque instanciation possible de séparateur se déplacera à partir des feuilles. Puis les combinaisons de labels seront effectuées, formant ainsi un ensemble de labels ayant déjà transité vers une clique. De nouveau, un label pour chaque valeur de séparateur sera choisi pour transiter sur la clique suivante et ainsi de

suite jusqu'à la clique racine.

Remarque 25 *Pour simplifier l'écriture et la lecture de l'algorithme, nous allons nous servir d'un « faux » séparateur :* $S_{root,root} = C_{root}$.

Algorithme 22 : InitialiserMouvement

Entrées :
act : indice de clique
succ : indice de l'appelant

$\mathcal{V} \leftarrow$ `Labeliser`(u_i);
pour chaque *clique* C_i *voisine de* C_{act} **faire**
  si $i \neq succ$ **alors**
    `InitialiserMouvement`(i, act);
    $\mathcal{V} \leftarrow \mathcal{V} \otimes \mathcal{M}_{i,act}$;
  fin
fin
$\mathcal{V} \leftarrow \mathcal{V}_{\Downarrow X_{S_{act,succ}}}$;
$\mathcal{V} \leftarrow \mathcal{V}_{\rightarrow act}$;
$\mathcal{M}_{act,succ} \leftarrow \bigcup_{x_{S_{act,succ}} \in X_{S_{act,succ}}}$ Plus prometteur de $\mathcal{V}[x_{S_{act,succ}}]$;
$\mathcal{L}^{open} \leftarrow \mathcal{L}^{open} \cup (\mathcal{V} \backslash \mathcal{M}_{act,succ})$;

Il est important de noter deux conséquences de ce précalcul :

- Au fur et à mesure de la propogation des labels jusqu'à la racine, des combinaisons des labels sont formées sans atteindre la racine, toutes ces combinaisons sont stockées dans un ensemble noté $\mathcal{L}^{open}$. Nous dirons que cet ensemble est l'ensemble des labels « ouverts » de l'algorithme.
- Un des ensembles $\mathcal{M}$ ne correspond pas à un séparateur : il s'agit de $\mathcal{M}_{root,root}$. Cet ensemble correspond aux labels qui ne sont pas encore dominés ayant transité jusqu'à la clique racine.

Nous disposons maintenant de toutes les briques de base pour définir une première version de l'approche filaire de l'algorithme de détermination de la frontière de Pareto (algorithme 23). Le fonctionnement de l'algorithme va consister en un lancement de `InitialiserMouvement` dans un premier temps. Le résultat de ce précalcul va ensuite être exploité : $\mathcal{M}_{root,root}$ va être stocké dans une variable nommée $\mathcal{L}^{pareto}$, et tant que $\mathcal{L}^{open}$ ne sera pas vide, on choisira un label de $\mathcal{L}^{open}$ pour lui faire effectuer un pas de déplacement vers la racine. Chaque fois qu'un ensemble de labels atteint la clique racine, $\mathcal{L}^{pareto}$ est mis à jour pour toujours contenir une liste de labels non dominés ayant achevé leur déplacement. Lorsque des labels sont déplacés mais qu'ils n'atteignent pas encore la racine, $\mathcal{L}^{open}$ est mis à jour pour prendre en compte ces nouveaux labels (ayant besoin d'effectuer de nouveaux déplacements pour atteindre la clique racine).

Remarque 26 *L'algorithme 23 a uniquement pour but d'introduire la notion de déplacement de labels, toutefois on peut faire un parallèle avec l'algorithme exact présenté dans le chapitre 4 : Le principe de ces deux algorithme est exactement le même, à savoir effectuer des tests de non-dominance locaux et des combinaisons de labels pour déterminer la frontière de Pareto. La grande différence entre ces deux algorithmes est que l'un fait*

Algorithme 23 : ParetoSansHeuristique

```
1  DeterminerOrdreCollecte();
2  InitialiserMouvement(C_root, C_root);
3  L^Pareto ← NonDominés(M_root,root);
4  tant que L^open ≠ ∅ faire
5  |  ⟨v, x_D, i⟩ ← Choisir(L^open);
6  |  L^open ← L^open \ {⟨v, x_D, i⟩};
7  |  V ← UnPas(⟨v, x_D, i⟩);
8  |  si Succ(i) = root alors
9  |  |  L^Pareto ← NonDominés(L^Pareto ∪ V);
10 |  sinon
11 |  |  L^open ← L^open ∪ V;
12 |  fin
13 fin
```

transiter directement l'intégralité des labels (et donc, après extraction des labels d'une clique, il ne s'intéresse jamais plus à cette clique), tandis que l'autre n'effectue que des petits déplacements d'une clique vers une autre. Cette différence à un impact direct sur les temps de calculs : Dans le cas de `ParetoSansHeuristique`*, il est nécessaire de gérer de nombreux ensembles qui s'avèrent plus coûteux que la première approche.*

5.2.2 Intégration de l'heuristique

Nous avons précédemment vu comment générer des fonctions heuristiques à partir d'un réseau GAI vectoriel, et un premier algorithme déterminant la frontière de Pareto par transition de labels. Nous allons maintenant assembler ces deux notions pour bâtir un algorithme capable d'exploiter ces heuristiques. Dans `ParetoSansHeuristique`, à chaque itération, un label de $\langle v, x_D, i\rangle \in \mathcal{L}^{open}$ est choisi pour effectuer un déplacement de la clique C_i vers la clique C_j. Étant donné que $S_{ij} \subset D$, $(x_D)_{S_{ij}}$ est une projection valide d'une instanciation de X_D sur les attributs de $X_{S_{ij}}$. On a donc que $v + \mathcal{H}_{ji}((x_D)_{S_{ij}})$ est une estimation majorante des vecteurs d'utilité qu'il est possible de produire en faisant transiter le label jusqu'à la clique racine. Donc, si $v + \mathcal{H}_{ji}((x_D)_{S_{ij}})$ est dominé par un vecteur de $\mathcal{L}^{pareto}$, il n'est plus nécessaire de continuer le déplacement, car n'importe quel vecteur d'utilité obtenu en fin de déplacement sera lui aussi dominé par un vecteur de $\mathcal{L}^{pareto}$. En appliquant cette idée, on obtient donc directement l'algorithme 24.

5.2.3 Validité et complexité de l'algorithme

Nous allons maintenant établir les preuves de validité et de complexité de l'algorithme 24. Pour la validité, il est important de noter qu'il suffit de démontrer la validité de `ParetoSansHeuristique` (proposition 19), puis de montrer que les coupes dûes à l'heuristique sont valides (proposition 20).

Proposition 19 (Validité de `ParetoSansHeuristique`)
L'algorithme `ParetoSansHeuristique` *calcule la frontière de Pareto ($\mathcal{L}^{pareto}$) d'un réseau GAI vectoriel.*

Algorithme 24 : ParetoHeuristique

```
1  DeterminerOrdreCollecte();
2  CollecteHeuristique(C_root, C_root);
3  DiffusionHeuristique();
4  InitialiserMouvement(C_root, C_root);
5  L^Pareto ← NonDominés(M_root,root);
6  tant que L^open ≠ ∅ faire
7  |  ⟨v, x_D, i⟩ ← Choisir(L^open);
8  |  L^open ← L^open \ {⟨v, x_D, i⟩};
9  |  j ← Succ(i);
10 |  si v + H_ji((x_D)_S_ij) non dominé dans L^Pareto alors
11 |  |  V ← UnPas(⟨v, x_D, i⟩);
12 |  |  si Succ(i) = root alors
13 |  |  |  L^Pareto ← NonDominés(L^Pareto ∪ V);
14 |  |  sinon
15 |  |  |  L^open ← L^open ∪ V;
16 |  |  fin
17 |  fin
18 fin
```

Preuve. Effectuons une récurrence sur les cliques du réseau GAI vectoriel pour montrer la validité des calculs et le fait qu'il n'y ait pas de « perte » de labels en cours d'exécution de l'algorithme :

- Si le réseau est réduit à une unique clique, alors `InitialiserMouvement` effectue un test de non-dominance sur tous les labels composant la clique (étant donné que $S_{root,root} = C_{root}$), le résultat sera stocké dans $\mathcal{L}^{pareto}$, on aura $\mathcal{L}^{open} = \varnothing$, donc la boucle « tant que » ne sera pas exécutée. À la fin de l'exécution de l'algorithme, $\mathcal{L}^{pareto}$ contiendra donc tous les labels composant la frontière de Pareto.
- Sur les feuilles de l'arbre de jonction (en considérant C_{root} comme la racine de l'arbre), `InitialiserMouvement` effectue un test de non-dominance à valeur de séparateur fixé (voir le chapitre 4 pour la validité de ce test), puis fait transiter un label pour chaque instanciation de séparateur possible, il n'existera donc pas de message vide pour une instanciation donné du séparateur. De plus, les messages n'ayant pas transité seront stockés dans $\mathcal{L}^{open}$, il n'y aura donc pas de perte de labels : Soit un label appartient au message du séparateur, soit il appartient à $\mathcal{L}^{open}$. Dans `ParetoSansHeuristique`, tout label transitant par la suite le long du séparateur est bien supprimé de $\mathcal{L}^{open}$, et stocké dans $\mathcal{M}_{ij}$ (j successeur de i dans l'ordre de collecte).
- Sur une clique C_i n'étant ni une feuille de l'arbre de jonction, ni la clique racine, `InitialiserMouvement` combinera les labels envoyés sur les séparateurs par les prédecesseurs de i. Ces combinaisons sont valides car les messages $\mathcal{M}_{ij}$ (j prédecesseur de i) ne sont pas vides par hypothèse de récurrence. Puis, la fonction effectuera un test de non-dominance à valeur du séparateur $S_{i,succ}$ fixé (avec $succ$ successeur de i dans la phase de collecte). De la même façon que pour les feuilles, un label sera

choisi pour chaque instanciation de $X_{S_{i,succ}}$ pour composer un nouveau message vers C_{succ}. Les autres labels formés sont insérés dans $\mathcal{L}^{open}$, il n'y a donc pas de perte de labels : tout label calculé appartient soit à un séparateur, soit à $\mathcal{L}^{open}$. Dans `ParetoSansHeuristique`, tout label transitant par la suite sur cette clique se combine avec les labels compatibles déjà créés pour former un ensemble $\mathcal{V}$ qui est bien stocké dans $\mathcal{L}^{open}$ et affecté à la clique i.

- Sur la clique racine C_{root}, pour les mêmes raisons que pour les cliques qui ne sont pas des feuilles, les combinaisons de labels sont valides. Toutefois, `InitialiserMouvement` n'effectue pas un choix de labels, mais effectue un test de non-dominance pour composer $\mathcal{L}^{pareto}$ (étant donné que $S_{root,root} = C_{root}$), qui correspond bien à la frontière de Pareto des labels ayant atteint la clique. De plus, pendant l'exécution de la boucle « tant que », chaque ensemble $\mathcal{V}$ de combinaison de labels arrivant jusqu'à C_{root} contient bien la somme de toutes les sous-fonctions d'utilité vectorielles et une instanciation complète des attributs ayant engendrés ces valeurs. La frontière de Pareto $\mathcal{L}^{pareto}$ est bien mise à jour et, à la fin d'une itération, contient bien la frontière de Pareto de tous les labels ayant transité jusqu'à C_{root}.

Par récurrence, puisque les calculs sont justes, et qu'aucune information ne se perd (tous les labels transitant se retrouvent bien dans $\mathcal{L}^{pareto}$ ou dans $\mathcal{L}^{open}$ avec le bon indice de clique), dès lors que $\mathcal{L}^{open} = \varnothing$, $\mathcal{L}^{pareto}$ correspond donc bien aux labels de la frontière de Pareto du problème. ■

Proposition 20 (Validité de l'heuristique) *Soit v, v', w et z quatre vecteurs de $\mathbb{R}^m$ vérifiant $v' \succsim_P v$, alors on a :*

$$z \succ_P w + v' \implies z \succ_P w + v$$

Preuve. Cette preuve est automatique : par définition on a $v' \succsim_P v$, donc $w+v' \succsim_P w+v$, donc si $z \succ_P w + v'$, on aura forcément $z \succ_P w + v$. ■

La proposition 20 permet d'affirmer que les coupes effectuées par l'algorithme `ParetoHeuristique` sont valides : v' correspond à l'estimation majorante obtenue lors de l'extraction du label, et v à la véritable valeur que l'on pourrait obtenir en poursuivant le mouvement du label jusqu'à la racine, z est un vecteur contenu dans $\mathcal{L}^{pareto}$ et w le vecteur d'utilité actuellement contenu dans le label. Donc si l'estimation de la valeur finale du label $(w + v')$ est dominée au sens de Pareto par un vecteur composant la frontière de Pareto courante, nous pouvons être sûrs qu'il ne sert à rien d'effectuer son déplacement : il sera de toutes façons dominé par un vecteur de la frontière de Pareto lorsqu'il atteindra la clique racine.

Nous allons maintenant établir la complexité de l'algorithme (proposition 21). Il est toutefois à noter que cette complexité n'est qu'une borne théorique, car il n'est pas possible de prendre en compte les coupes dûes aux fonctions heuristiques dans ce résultat. En pratique, l'algorithme est bien meilleur, comme nous le verrons dans les expérimentations (section 5.4).

Proposition 21 (Complexité) *L'algorithme* `ParetoSansHeuristique` *s'exécute en un temps en $O(m|C||X_{C_{max}}|^2 K^{3m})$, avec K une borne supérieure sur la valeur des utilités w^j et $|X_{C_{max}}| = \max_{C_i \in C} \prod_{j \in C_i} |X_j|$.*

Preuve. Pour démontrer finement la complexité de cet algorithme, nous n'allons pas suivre exactement l'ordre des instructions présentées dans le pseudo-code, mais un ordre différent correspondant néanmoins aux mêmes calculs. L'algorithme fait des appels à la fonction `UnPas`. Cette fonction prend en argument des labels $\langle v, x_D, i\rangle$ et les fait transiter d'une clique (C_i) vers une clique adjacente (C_j). Pour cela, elle effectue toutes les combinaisons possibles du label $\langle v, x_D, i\rangle$ avec les labels compatibles de C_j (ligne 03) ainsi qu'avec tous les labels compatibles envoyés par messages sur les séparateurs adjacents à C_j (les messages $\mathcal{M}_{zj}$, ligne 06). Pour un label donné $\langle v, x_D, i\rangle$ transitant sur le message $\mathcal{M}_{ij}$, la complexité de calcul de la ligne 03 de `UnPas` est en $O(m|X_{C_{max}}|)$. Par hypothèse, il ne peut exister plus de K^m labels différents transitant sur le message $\mathcal{M}_{ij}$ pour une valeur $x_{S_{ij}} \in X_{S_{ij}}$ donnée. Notons que, pour deux valeurs différentes $x_{S_{ij}}, y_{S_{ij}} \in X_{S_{ij}}$, les combinaisons effectuées sur la ligne 03 vont faire intervenir des sous-parties distinctes de la table u_j. Par conséquent, la totalité des combinaisons effectuées sur la ligne 03 de `UnPas` pour l'ensemble des labels du message $\mathcal{M}_{ij}$ est calculé en $O(mK^m|X_{C_{max}}|)$. À la fin de l'algorithme `ParetoSansHeuristique`, on aura appelé `UnPas` avec les messages de tous les séparateurs. Donc, globalement, à la fin de `ParetoSansHeuristique`, l'ensemble des appels à la ligne 03 de `UnPas` aura été réalisé en $O(m|C|K^m|X_{C_{max}}|)$.

La boucle des lignes 04–08 de `UnPas` effectue toutes les combinaisons possibles des labels de $\mathcal{V}$ avec ceux des messages $\mathcal{M}_{zj}$. Par hypothèse, il ne peut exister plus de K^m labels différents transitant sur les messages $\mathcal{M}_{zj}$ pour une valeur $x_{S_{zj}} \in X_{S_{zj}}$ donnée. Donc, lorsque l'on effectue la première fois la ligne 06 de `UnPas` pour l'ensemble des labels de $\mathcal{M}_{ij}$, on réalise $O(K^{2m}|X_{C_{max}}|)$ combinaisons car, pour chaque $x_{C_j} \in X_{C_j}$, on est potentiellement amené à combiner K^m labels par K^m labels. Par hypothèse, on ne pourra obtenir plus de K^m vecteurs différents (donc labels différents) pour chaque valeur de x_{C_j}. Donc, globalement, sur une clique donnée, le nombre de combinaisons de labels réalisées sur les lignes 04–08 est en $O(sK^{2m}|X_{C_{max}}|)$, où s est le nombre de séparateurs adjacents à la clique (moins 1). Donc, globalement, sur tout l'arbre de jonction, et donc à la fin de `ParetoSansHeuristique`, les lignes 04–08 auront engendré $O(|C|K^{2m}|X_{C_{max}}|)$ combinaisons. Chaque combinaison étant réalisée en $O(m)$ opérations, on aura donc effectué $O(m|C|K^{2m}|X_{C_{max}}|)$ opérations.

Nous avons vu que, dans la fonction `UnPas`, $\mathcal{V}$ ne pouvait pas contenir plus de $K^m|X_{C_{max}}|$ éléments. Avec une structure de données appropriée, on peut supprimer en $O(mK^m|X_{C_{max}}|)$ les vecteurs d'utilité identiques, permettant de restreindre $|\mathcal{V}|$ à, au plus, K^m vecteurs/labels différents. Ensuite, il ne faut que mK^{2m} opérations pour calculer les non-dominés de la ligne 10. Globalement, dans `ParetoSansHeuristique`, on est amené à appeler la ligne 10 de `UnPas` pour chaque label possible de chaque séparateur adjacent à `root`. Autrement dit, on a potentiellement $O(|C|K^m|X_{C_{max}}|)$ appels possibles. Donc, sur l'ensemble des appels de la ligne 10 de `UnPas`, on a une complexité en $O(m|C|K^{3m}|X_{C_{max}}|)$ opérations. Enfin, la ligne 13 de `UnPas` réalise des projections de $\mathcal{V}$ sur des séparateurs. Chaque appel à cette ligne est réalisée en $O(m|X_{C_{max}}|K^{2m})$ opérations. Étant donné que `UnPas` est potentiellement appelé pour chaque label possible de chaque séparateur, la ligne 13 est appelée $O(|C||X_{C_{max}}|K^m)$ fois. Donc, globalement, l'ensemble des appels à la ligne 13, et plus généralement, l'ensemble des appels de `UnPas`, est réalisé en $O(m|C||X_{C_{max}}|^2K^{3m})$ opérations.

Pour terminer cette démonstration, notons qu'outre les appels à `UnPas`, la complexité de `ParetoSansHeuristique` provient aussi de la ligne 9 de `ParetoSansHeuristique` (les

lignes 5, 6, 11 pouvant chacune être effectuée en $O(1)$). Chaque appel à la ligne 9 calcule des non-dominés sur un ensemble d'au plus K^m vecteurs différents. Comme nous l'avons vu dans le paragraphe précédent, cette ligne peut être appelée au plus $O(|C|K^m|X_{C_{max}}|)$ fois. On a donc une complexité en $O(|C|K^{3m}|X_{C_{max}}|)$ pour les appels à la ligne 9. En conclusion, globalement, l'algorithme `ParetoSansHeuristique` a une complexité en $O(m|C||X_{C_{max}}|^2K^{3m})$. ■

5.3 Extension de l'algorithme à de nouveaux problèmes

Nous avons vu dans la section précédente une approche heuristique pour déterminer la frontière exacte de Pareto. Dans les chapitres précédents, nous avons présenté différentes problématiques de décision multicritère. Nous allons brosser un rapide résumé de ces problématiques puis présenter comment il est possible d'adapter notre nouvel algorithme pour résoudre les problèmes laissés en suspens :

- La détermination de la **frontière de Pareto** : Ce problème a fait l'objet du chapitre 4 et de la section précédente.
- La détermination de la **frontière de Lorenz** : Nous n'avons pas, pour l'instant, présenté d'algorithmes résolvant ce problème. On peut toutefois imaginer une approche consistant à déterminer la frontière de Pareto, puis calculant la frontière de Lorenz par non-dominance de Lorenz. Toutefois, il serait sûrement plus intéressant de concevoir un algorithme exploitant directement la dominance de Lorenz. Nous présenterons dans la section 5.3.1, une adaptation de l'approche filaire pour résoudre ce problème.
- La détermination de la **solution agrégée optimale** : Nous avons présenté dans le chapitre 2 un algorithme de l'état de l'art résolvant ce problème, et en guise de conclusion du chapitre 3, nous avons montré qu'il était possible d'utiliser une approche par approximations pour déterminer la solution agrégée exacte du problème. Toutefois, ces deux approches se basent sur une hypothèse importante : l'existence d'une approximation de l'agrégateur qui soit à la fois une borne supérieure de la valeur agrégée quel que soit le vecteur d'utilité, et une fonction linéairement séparable. Ainsi, un OWA dont les poids ne sont pas décroissants ou une intégrale de Choquet dont la capacité n'est pas convexe ne peuvent être résolus par ces algorithmes. Nous montrerons dans la section 5.3.2 comment résoudre ces problèmes en utilisant l'approche filaire.

5.3.1 Détermination de la frontière de Lorenz

Avant d'aborder le problème de la détermination de la frontière de Lorenz, il est important de déterminer la classe de complexité de ce problème. Le problème de décision correspondant est décrit dans la définition 84 et la proposition 22 montre que ce problème est NP-complet. Il ne peut donc exister d'algorithme polynômial pour le résoudre (à moins que P = NP). Il est donc intéressant d'adapter notre approche heuristique pour calculer la frontière de Lorenz.

Définition 84 (L_v : problème de décision (frontière de Lorenz)) *On notera L_v le problème de décision (au sens de la théorie de la complexité) associé à la détermination*

de la frontière de Pareto dont la formulation est :

- **Données :** *Un ensemble de n attributs X_i avec $\forall i \in \{1,\ldots,n\}|X_i| \geqslant 2$, m ensembles C^j vérifiant $\forall C_i^j \in C^j\ C_i^j \subseteq \{1,\ldots,n\}$, auxquels on associe des fonctions $u_i^j : \prod_{z\in C_i^j} X_z \to \mathbb{Z}_+$, et un vecteur $v \in \mathbb{Z}_+^m$. On notera $u^j = \sum_{C_i^j \in C^j} u_i^j$.*
- **Question :** *Existe-t-il une instanciation x des attributs formant $\mathcal{X}$ dont le vecteur d'utilité associé $(u^1(x),\ldots,u^m(x))$ domine au sens de Lorenz le vecteur v ?*

Proposition 22 *L_v est un problème NP-complet.*

Preuve. Nous allons établir la preuve pour un problème bicritère ($m = 2$). Pour ce faire, nous allons considérer la version décisionnelle du problème Partition (que nous noterons PART), dont la formulation est la suivante :

- **Données :** Un ensemble fini $\mathcal{A} = \{a_1,\ldots,a_n\}$, et une mesure $s : \mathcal{A} \to \mathbb{N}$.
- **Question :** Est-il possible de partitionner $\mathcal{A}$ en deux ensembles $\mathcal{A}_1$ et $\mathcal{A}_2$ vérifiant $\sum_{a_i\in\mathcal{A}_1} s(a_i) = \sum_{a_j\in\mathcal{A}_2} s(a_j)$?

Construisons une instance de L_v à partir d'une instance de PART. Soit $v = (\beta,\beta)$ avec $\beta = \sum_{a_i\in\mathcal{A}} s(a_i)/2$, et considérons n attributs booléens $X_i = \{0,1\}$. Posons $C_i^1 = C_i^2 = i\ \forall i \in \{1,\ldots,n\}$. Ainsi, à chaque attribut X_i, nous associons les sous-fonctions d'utilité vérifiant $\forall x_i \in X_i\ u_i^1(x_i) = s(a_i)x_i$ et $u_i^2(x_i) = s(a_i)(1-x_i)$. On aura donc

$$\forall x \in \mathcal{X}\ u^1(x) = \sum_{i=1}^{n} s(a_i)x_i \text{ et } u^2(x) = \sum_{i=1}^{n} s(a_i)(1-x_i)$$

Ainsi, pour toute partition $(B,\mathcal{A}\backslash B)$ de $\mathcal{A}$, il est possible d'associer le vecteur de booléens $x^B \in \mathcal{X}$ vérifiant :

$$\forall i \in \{1,\ldots,n\}\ x_i^B = \begin{cases} 1 & \text{si } a_i \in B \\ 0 & \text{sinon} \end{cases}$$

L'utilité associée à x^B sera donc $u(x^B) = (\sum_{a_i\in B} a_i, \sum_{a_j\in\mathcal{A}\backslash B} a_j)$. Donc, s'il existe un partitionnement de $\mathcal{A}$ de valeur (β,β), alors l'instanciation x^B est solution de L_v. De plus, s'il n'existe pas de partitionnement de $\mathcal{A}$ de valeur (β,β), alors l'utilité associée à x^B sera soit de la forme $u(x^B) = (\beta - z, \beta + z)$, soit $u(x^B) = (\beta + z, \beta - z)$ avec $0 < z \leqslant \beta$. Dans tous les cas, la transformée de Lorenz du vecteur sera $(\beta - z, 2\beta)$, alors que la transformée de Lorenz de v sera $(\beta, 2\beta)$, on aura donc $v \succ_L u(x^B)$. Donc, $\exists x^B \in \mathcal{X}$ tel que $u(x^B) \succsim_L v$ si, et seulement si, il existe une solution à PART. Il en résulte que PART se réduit polynômialement à L_v, PART étant NP-complet (Garey et Johnson, 1979), L_v est donc NP-difficile.

De plus, il est simple de tester polynômialement si, pour une instanciation x donnée, on a $u(x) \succ_L v$, donc L_v est aussi un problème de la classe NP. Donc L_v est un problème NP-complet. ■

Une première idée pour adapter l'algorithme 24 afin de déterminer la frontière de Lorenz est de remplacer partout dans cet algorithme la dominance de Pareto par la dominance de Lorenz. Cette idée est fausse, comme le montre l'exemple 36.

Exemple 36 *Considérons un réseau GAI vectoriel comportant trois attributs binaires : X_1, X_2, X_3 et 2 critères. Supposons les utilités décomposables en $u(x_1,x_2,x_3) = u_1(x_1,x_2)+$*

$u_2(x_2, x_3)$. *Supposons que la clique* X_1X_2 *comporte deux vecteurs Pareto optimaux :* $u_1(1,1) = (2,2)$ *et* $u_1(0,1) = (3,1)$. *La transformée de Lorenz de ces vecteurs sera donc* $L(2,2) = (2,4)$ *et* $L(3,1) = (1,4)$. *On serait tenté d'éliminer le vecteur* $(3,1)$ *par Lorenz-dominance (étant donné que* $(2,4) \succ_P (1,4)$ *et donc* $(2,2) \succ_L (1,4)$*) et d'envoyer uniquement le label* $\langle (1,1),(2,2),1\rangle$ *à la clique* X_2X_3 *(pour une valeur* $X_2 = 1$ *fixée). Néanmoins, ce serait une erreur : Supposons que* $u_2(1,0) = (1,3)$, *alors on a* $u(1,1,0) = (2,2)+(1,3) = (3,5)$ *et* $u(0,1,0) = (3,1)+(1,3) = (4,4)$. *Or,* $L(3,5) = (3,8)$ *et* $L(4,4) = (4,8)$. *On a donc* $(4,4) \succ_L (3,5)$. *On ne pouvait donc pas se permettre de couper l'envoi du message* $\langle (0,1),(3,1),1\rangle$.

Néanmoins, la proposition 23 montre qu'il est toujours possible d'utiliser les fonctions heuristiques majorantes pour effectuer des coupes en utilisant la dominance de Lorenz à la place de la dominance de Pareto.

Proposition 23 *Soit* v, v', w *et* z *quatre vecteurs de* $\mathbb{R}^m$ *vérifiant* $v' \succsim_P v$, *alors on a :*

$$z \succ_L w + v' \implies z \succ_L w + v$$

Preuve. Étant donné que $v' \succsim_P v$, on a donc $w + v' \succsim_P w + v$. Or, la dominance de Lorenz est P-monotone. On aura donc $w + v' \succsim_L w + v$. De plus, la dominance de Lorenz est transitive, donc si $z \succ_L w + v'$ et $w + v' \succsim_L w + v$, alors $z \succ_L w + v$. ■

Il résulte de ces constations que, pour adapter l'algorithme filaire de détermination de la frontière de Pareto, l'heuristique à utiliser peut être la dominance de Lorenz, mais les tests de non-dominance utilisés localement dans les fonctions `InitialiserMouvement` et `UnPas` doivent se faire au sens de Pareto (la dominance de Lorenz étant P-monotone). On obtient donc l'algorithme 25, utilisant les mêmes fonctions `InitialiserMouvement` et `UnPas` que pour la détermination de la frontière de Pareto.

5.3.2 Détermination de la solution agrégée optimale

Nous allons maintenant aborder le problème de la détermination de la solution agrégée optimale. Les expérimentations finales porteront sur l'agrégateur OWA dans son cas général (les poids ne seront pas nécessairement décroissants). Nous allons donc présenter l'algorithme appliqué à cet agrégateur, mais il sera généralisable à de nombreux autres agrégateurs couramment utilisés (s'ils sont P-monotones, donc pour un agrégateur A, si $v \succsim_P v'$ alors $A(v) \geqslant A(v')$, voir remarque 27). Dans le cas d'OWA, une première constatation est la NP-complétude du problème de décision qui lui est associé (définition 85 et proposition 24).

Définition 85 (O_α : problème de décision (maximisation OWA)) *On notera* O_α *le problème de décision (au sens de la théorie de la complexité) associé à la maximisation d'un agrégateur OWA dont la formulation est :*

- **Données :** *Un ensemble de* n *attributs* X_i *avec* $\forall i \in \{1,\ldots,n\}|X_i| \geqslant 2$, m *ensembles* C^j *vérifiant* $\forall C_i^j \in C^j\ C_i^j \subseteq \{1,\ldots,n\}$, *auxquels on associe des fonctions* $u_i^j : \prod_{z\in C_i^j} X_z \to \mathbb{Z}_+$, *et un scalaire* α. *On notera* $u^j = \sum_{C_i^j \in C^j} u_i^j$.
- **Question :** *Existe-t-il une instanciation* $x \in \mathcal{X}$ *tel que* $OWA(u(x)) \geqslant \alpha$ *?*

Algorithme 25 : LorenzHeuristique

```
1  DeterminerOrdreCollecte();
2  CollecteHeuristique(C_root, C_root);
3  DiffusionHeuristique();
4  InitialiserMouvement(C_root, C_root);
5  L^Lorenz ← NonDominés_lorenz(M_root,root);
6  tant que L^open ≠ ∅ faire
7  |  ⟨v, x_D, i⟩ ← Choisir(L^open);
8  |  L^open ← L^open \ {⟨v, x_D, i⟩};
9  |  j ← Succ(i);
10 |  si v + H_ji((x_D)_S_ij) non dominé (au sens de Lorenz) dans L^Lorenz alors
11 |  |  V ← UnPas(⟨v, x_D, i⟩);
12 |  |  si Succ(i) = root alors
13 |  |  |  L^Lorenz ← NonDominés_lorenz(L^Lorenz ∪ V);
14 |  |  sinon
15 |  |  |  L^open ← L^open ∪ V;
16 |  |  fin
17 |  fin
18 fin
```

Proposition 24 *O_α est un problème NP-complet.*

Preuve. Nous allons établir la preuve pour un problème bicritère ($m = 2$). Pour ce faire, de la même façon que pour la propriété 22, nous allons considérer la version décisionnelle du problème de partitionnement (notée PART), dont la formulation est la suivante :

- **Données :** Un ensemble fini $\mathcal{A} = \{a_1, \ldots, a_n\}$, et une mesure $s : \mathcal{A} \to \mathbb{N}$.
- **Question :** Est-il possible de partitionner $\mathcal{A}$ en deux ensembles $\mathcal{A}_1$ et $\mathcal{A}_2$ vérifiant $\sum_{a_i \in \mathcal{A}_1} s(a_i) = \sum_{a_j \in \mathcal{A}_2} s(a_j)$?

Construisons un problème O_α à partir d'une instance de PART. Soit w_1 et w_2 les coefficients de l'agrégation OWA, avec $w_1 > w_2$, et $\alpha = (w_1 + w_2)\beta$ avec $\beta = \sum_{a_i \in \mathcal{A}} s(a_i)/2$. Posons n attributs booléens $X_1, \ldots, X_n$ tels que $C_i^1 = C_i^2 = i \ \forall i \in \{1, \ldots, n\}$, et $u_i^1(x_i) = s(a_i)x_i$ et $u_i^2(x_i) = s(a_i)(1 - x_i)$. Ainsi, pour chaque partition de $\mathcal{A}$ de type $(B, \mathcal{A}\backslash B)$ (avec $B \subseteq A$), on peut associer une instanciation des attributs $x^B \in \mathcal{X}$ vérifiant $x_i^B = 1 \iff a_i \in B$. Par construction, on a $u(x^B) = (\sum_{i=1}^n s(a_i)x_i^B, \sum_{i=1}^n s(a_i)(1-x_i^B)) = (\sum_{a_i \in B} s(a_i), \sum_{a_i \in \mathcal{A}\backslash B} s(a_i))$.

En procédant ainsi, on a $\mathrm{OWA}(\beta, \beta) = \alpha$. Donc, dire que la réponse à O_α est « oui » est équivalent à dire que la réponse à PART est « oui ». De plus, si la réponse à PART est « non », alors tout partitionnement est déséquilibré, et correspondra à une utilité de type $(\beta - k, \beta + k)$ ou $(\beta + k, \beta - k)$ avec $0 < k \leqslant \beta$. On aura donc $\mathrm{OWA}(\beta - k, \beta + k) = w_1(\beta - k) + w_2(\beta + k) = \alpha - k(w_1 - w_2) < \alpha$. Donc la réponse de O_α sera « non ».

La transformation de problème étant polynômiale, et PART étant NP-complet (Garey et Johnson, 1979), O_α est donc NP-difficile. De plus, comme O_α est un aussi un problème de la classe NP, O_α est donc NP-complet. ■

Toutefois, de la même façon que pour la dominance de Lorenz, l'agrégateur OWA possède une propriété nous permettant d'exploiter l'approche filaire pour résoudre le problème de la détermination de la solution OWA-optimale (proposition 25).

Proposition 25 *Soit v, v', w et z des vecteurs de $\mathbb{R}^m$ tels que $v' \gtrsim_P v$, alors on a :*

$$OWA(z) > OWA(w + v') \implies OWA(z) > OWA(w + v)$$

Preuve. Cette proposition se montre de la même façon que pour la dominance de Lorenz : On a $w + v' \gtrsim_P w + v$. Par P-monotonie d'OWA, on aura donc $\text{OWA}(w + v') \geqslant \text{OWA}(w + v)$. Or, $\text{OWA}(z) > \text{OWA}(w + v')$ donc $\text{OWA}(z) > \text{OWA}(w + v)$. ■

Remarque 27 *Cette propriété est généralisable à tous les agrégateurs A qui sont P-monotones. En effet, si $w + v' \gtrsim_P w + v$, par P-monotonie on aura $A(w + v') \geqslant A(w + v)$ et donc $A(z) > A(w + v') \implies A(z) > A(w + v)$.*

L'adaptation de l'approche heuristique pour déterminer la solution OWA-optimale va donc être semblable à l'algorithme pour déterminer la frontière de Lorenz :

- Les fonctions `InitialiserMouvement` et `UnPas` doivent exploiter la dominance de Pareto pour réaliser leurs coupes. Par P-monotonie, les coupes réalisées seront compatibles avec OWA.
- La solution désirée n'est plus une frontière mais une solution unique, les coupes réalisées par l'heuristique correspondent donc au cas où la valeur agrégée de la mailleure solution ayant transité jusqu'à C_{root} est supérieure à la valeur agrégée de la solution choisie additionnée à l'estimation majorante de ce qu'elle peut obtenir en continuant son parcours.

En appliquant ce principe, on obtient finalement l'algorithme 26.

5.4 Analyse expérimentale

De manière à évaluer les performances des algorithmes présentés dans ce chapitre, nous avons choisi d'utiliser des données concrètes disponibles sur internet [2]. Toutefois, étant donné que les approches que nous avons proposées sont exactes, nous avons choisi de nous restreindre aux structures d'arbres de jonction dont la treewidth t est inférieure ou égale à 15. Les expérimentations proposées sur internet étant monocritères, nous les avons adaptées de manière à concorder avec les problèmes qui nous intéressent. Ainsi, la structure de dépendance entre chaque attribut reste inchangée, mais chaque sous-fonction d'utilité monocritère (contenant une valeur minimale $u^\perp$ et une valeur maximale $u^\top$) est transformée en une sous-fonction bicritère. Donc, toutes les composantes de tous les vecteurs sont générés aléatoirement avec des valeurs comprises entre $u^\perp$ et $u^\top$. En procédant ainsi, nous avons mesuré les temps de réponses des algorithmes calculant la frontière de Pareto (noté $\text{Par}^*_{\mathcal{H}}$), la frontière de Lorenz (noté $\text{Lor}^*_{\mathcal{H}}$), et une solution agrégée (nous avons choisi d'utiliser un OWA). Nous avons retranscrit les temps de réponse moyens obtenus sur 100 expériences pour chaque instance dans le tableau de la figure 5.6, et nous avons indiqué le nombre d'attributs du problème (n), la treewidth du problème (t), le nombre moyen de solutions composant les frontières de Lorenz (#L) et de Pareto (#P).

2. `http://carlit.toulouse.inra.fr/cflibtars`

Algorithme 26 : AgregHeuristique

```
1  DeterminerOrdreCollecte();
2  CollecteHeuristique(C_root, C_root);
3  DiffusionHeuristique();
4  InitialiserMouvement(C_root, C_root);
5  SolutionAgregOpt ← arg max_{label∈M_{root,root}} Agréger(label);
6  tant que L^open ≠ ∅ faire
7  |  ⟨v, x_D, i⟩ ← Choisir(L^open);
8  |  L^open ← L^open \ {⟨v, x_D, i⟩};
9  |  j ← Succ(i);
10 |  si Agréger(v + H_ji((x_D)_{S_ij})) > Agréger(SolutionAgregOpt) alors
11 |  |  V ← UnPas(⟨v, x_D, i⟩);
12 |  |  si Succ(i) = root alors
13 |  |  |  MeilleurDeV ← arg max_{label∈V} Agréger(label);
14 |  |  |  si Agréger(MeilleurDeV) > Agréger(SolutionAgregOpt) alors
15 |  |  |  |  SolutionAgregOpt ← MeilleurDeV ;
16 |  |  |  fin
17 |  |  sinon
18 |  |  |  L^open ← L^open ∪ V;
19 |  |  fin
20 |  fin
21 fin
```

Fichier	n	t	$\text{Par}^*_{\mathcal{H}}$	$\text{Lor}^*_{\mathcal{H}}$	OWA	#L	#Par
GEOM30a_4	30	6	0.315	0.317	0.286	6	16
GEOM40_2	40	5	0.019	0.008	0.007	2	18
dubois30	90	3	0.054	0.046	0.041	1	38
dubois50	150	3	0.152	0.121	0.106	4	66
dubois100	300	3	0.751	0.637	0.568	1	115
pret150_25	150	8	1.032	0.998	0.823	1	55
pret150_40	150	8	0.750	0.705	0.597	6	56
pret150_75	150	9	2.048	1.842	1.644	2	57
hailfinder	56	4	29.302	28.553	27.808	34	169
insurance	27	8	28.279	28.271	28.151	1	58
kbtree5_2_4_5_10_1	62	5	4.335	3.863	3.196	3	46
kbtree5_2_4_5_30_1	62	5	5.109	4.607	4.203	3	47
kbtree5_2_4_5_50_1	62	5	3.530	3.136	2.783	5	42
kbtree5_2_4_5_70_1	62	5	3.026	2.717	2.393	3	42
kbtree5_2_4_5_90_1	62	5	3.692	3.322	3.072	4	49
cnf2.40.100.730621	40	11	0.746	0.730	0.672	3	16
cnf2.40.100.730623	40	12	1.630	1.622	1.433	5	10
cnf2.80.100.735545	80	6	0.050	0.044	0.038	1	16
cnf2.80.100.735549	80	6	0.038	0.034	0.030	2	17
alarm	37	4	0.172	0.150	0.128	23	92

FIGURE 5.6 – Expérimentations sur les instances classiques transformées en problèmes bicritères

Comme nous pouvions nous y attendre, les temps de réponse de la détermination de la frontière de Lorenz et de la solution agrégée optimale sont meilleures que celles du calcul de la frontière de Pareto. Mais il est toutefois à noter que les attributs des problèmes précédents sont booléens (à l'exception de hailfinder et insurance, dont les attributs peuvent prendre jusqu'à 11 valeurs différentes), ce qui reste un cas très particulier dans le cadre d'une prise de décision.

De manière à tester nos algorithmes dans un cadre plus proche de notre domaine d'application, nous avons relancé les mêmes expériences, mais en modifiant les modalités des attributs. Ainsi, chaque attribut pouvait avoir 4 valeurs différentes, et les sous-fonctions d'utilité étaient remplies par des vecteurs à deux dimensions contenant des nombres tirés aléatoirement entre 0 et 20. Certaines instances ont engendré des dépassements de la capacité mémorielle de l'ordinateur faisant tourner ces expérimentations (2 GB). Dans ce cas de figure, nous avons remplacé le temps de calcul obtenu par un « – ». Les résultats sont présentés sur le tableau de la figure 5.7.

Finalement, nous avons décidé d'observer le comportement de nos algorithmes sur des problèmes comportant 5 critères conflictuels. Nous avons choisi d'utiliser des problèmes générés aléatoirement plutôt que les instances classiques, car l'augmentation du nombre de critères rend les problèmes bien plus difficiles à résoudre. Ainsi, nous avons généré aléatoirement des réseaux GAI dont toutes les cliques contiennent 3 attributs et tous les séparateurs en contiennent 2. Chaque attribut peut prendre 4 valeurs différentes. Pour chaque clique, nous avons généré aléatoirement 5 hypermatrices différentes (chacune

Fichier	n	t	$\text{Par}^*_{\mathcal{H}}$	$\text{Lor}^*_{\mathcal{H}}$	OWA	#L	#Par
GEOM30a_4	30	6	1.001	1.037	0.936	4	65
GEOM40_2	40	5	0.394	0.423	0.352	4	64
dubois30	90	3	11.616	10.911	9.904	4	310
dubois50	150	3	54.627	51.974	48.046	15	530
dubois100	300	3	1047.16	949.368	905.818	10	1451
pret150_25	150	8	–	–	–	–	–
pret150_40	150	8	–	–	–	–	–
pret150_75	150	9	–	–	–	–	–
hailfinder	56	4	25.888	23.067	21.095	5	178
insurance	27	8	–	–	–	–	–
kbtree5_2_4_5_10_1	62	5	11.955	10.188	9.471	8	269
kbtree5_2_4_5_30_1	62	5	11.781	10.274	7.575	5	199
kbtree5_2_4_5_50_1	62	5	9.099	7.516	7.035	8	304
kbtree5_2_4_5_70_1	62	5	12.299	10.502	9.492	8	243
kbtree5_2_4_5_90_1	62	5	9.955	8.169	7.074	3	254
cnf2.40.100.730621	40	11	–	–	–	–	–
cnf2.40.100.730623	40	12	–	–	–	–	–
cnf2.80.100.735545	80	6	–	–	–	–	–
cnf2.80.100.735549	80	6	–	–	–	–	–
alarm	37	4	2.734	2.643	2.416	7	103

FIGURE 5.7 – Expérimentations sur les instances classiques dont les attributs peuvent prendre 4 valeurs différentes

représentant un des critères) emplies de valeurs tirées entre 0 et 20. Les temps de réponse obtenus sont présentés dans le tableau de la figure 5.8. Il est à noter que « – » signifie un temps de calcul supérieur à 2400 secondes, et que la colonne « Par » correspond à l'algorithme exact de détermination de la frontière de Pareto présenté dans le chapitre 4.

Nous pouvons remarquer que l'approche filaire améliore considérablement les temps de calcul et permet d'atteindre des tailles de problèmes que l'algorithme par « paquets » ne peut atteindre. De plus, même si les fonctions `InitialiserMouvement` sont les mêmes dans chaque algorithme exploitant l'approche filaire, les coupes liées aux fonctions heuristiques (différentes selon le problème) entrent en jeu pour permettre à OWA de posséder des temps de calculs légèrement inférieurs à $\text{Lor}^*_{\mathcal{H}}$ qui sont eux-même très clairement inférieurs à $\text{Par}^*_{\mathcal{H}}$.

n	Par	$\text{Par}^*_{\mathcal{H}}$	$\text{Lor}^*_{\mathcal{H}}$	OWA	#L	#Par
10	1.519	0.451	0.237	0.196	7	2957
11	8.199	3.927	0.398	0.301	21	7134
12	31.512	11.995	7.506	7.143	5	8891
13	55.833	23.389	7.903	7.322	9	11484
14	162.425	44.526	6.055	5.022	22	16928
15	427.137	104.028	76.941	73.707	9	22676
16	2050.512	105.577	60.467	56.931	5	33334
17	—	264.504	70.189	64.836	15	44621
18	—	552.296	190.949	179.853	9	37788
19	—	1620.304	392.702	359.253	11	42655
20	—	—	512.344	484.233	13	45245

FIGURE 5.8 – Expérimentations sur des problèmes aléatoires à 5 critères

Conclusion

Synthèse des travaux effectués. Un réseau GAI permet de représenter de manière compacte une relation de préférence et en tirant partie des indépendances préférentielles (conditionnelles) entre les attributs servant à caractériser les alternatives sur lesquelles porte la relation de préférence. Les algorithmes développés dans cette thèse ont permis d'étendre l'exploitation des réseaux GAI pour résoudre les problèmes suivants :

- **Choix optimal sous forte treewidth** : L'approche proposée dans le chapitre 3 analyse les problèmes de dépendances entre attributs induisant des temps de calcul prohibitifs lors de la résolution des problèmes de choix (détermination de l'alternative préférée). Suite à cette analyse, nous avons proposé une nouvelle méthode de résolution exacte des problèmes de choix en résolvant un problème de rangement avec un réseau GAI approché de faible treewidth déduit du réseau GAI d'origine.
- **Rangement sous forte treewidth** : La méthode du chapitre 3 a été étendue au problème du rangement des k meilleures alternatives pour un réseau GAI de forte treewidth en implémentant une simple structure de données à insérer dans l'algorithme de choix optimal.
- **Solution agrégée optimale sous forte treewidth** : Au sein du chapitre 2, nous avions présenté un algorithme de l'état de l'art pour résoudre la détermination de l'alternative d'utilité agrégée optimale (sous l'hypothèse de l'existence d'une borne supérieure linéairement décomposable de l'agrégateur). Cette approche se basait sur un réseau GAI dit scalarisé obtenu par agrégation (triangulation) des réseaux GAI représentant chaque critère, ainsi que sur l'algorithme de rangement. Or, nous avons pu voir que la triangulation des forêts de jonction pouvait générer une structure de forêt ayant une forte treewidth induite. L'algorithme du chapitre 3 pouvant être étendu au problème de rangement sous forte treewidth, il a donc été naturellement utilisé pour résoudre le problème de la détermination de la solution agrégée optimale sous les même hypothèses sur les agrégateurs.
- **Frontière de Pareto exacte** : La notion de réseau GAI vectoriel introduite dans le chapitre 4 a permis d'élaborer deux algorithmes pour déterminer l'ensemble des alternatives engendrant la frontière de Pareto d'un problème multiattribut et multicritère. Le premier algorithme (présenté dans le chapitre 4) exploitait la programmation dynamique non sérielle avec des opérateurs combinant des ensembles de vecteurs. Le second (introduit dans le chapitre 5) se basait sur une approche filaire et l'utilisation d'une heuristique générée lors d'une phase de précalcul.
- **Frontière de Pareto approchée (avec garantie de performance)** : L'agorithme exact du chapitre 4 mêle aux opérateurs de combinaison une notion de coupe basée sur des tests de non-dominance. Nous avons présenté une adaptation de l'algorithme effectuant des tests de non-dominance selon une relation d'(ϵ, w)-

dominance de manière à générer une $(1+\epsilon)$-approximation de la frontière de Pareto (pour un paramètre $\epsilon > 0$ défini par le décideur).

- **Frontière de Lorenz** : L'approche heuristique développée dans le chapitre 5 peut se généraliser à de nombreux problèmes de décision multiattribut et multicritère. Parmi ces problèmes, nous avons montré que la détermination de la frontière de Lorenz se résoud aisément en utilisant une heuristique majorante adéquate.
- **Solution agrégée optimale (généralisée)** : Nous avons montré dans le chapitre 5 que l'approche heuristique permet également de résoudre le problème de la détermination de la solution agrégée optimale. Cette nouvelle approche permet de repousser les limites posées par les hypothèses de l'algorithme de l'état de l'art. Ainsi, il n'est plus nécessaire de déterminer une borne supérieure linéairement décomposable de l'agrégateur que l'on souhaite utiliser. L'approche filaire que nous avons développée peut déterminer la solution agrégé optimale de n'importe quel agrégateur à partir du moment où celui-ci est P-monotone (ce qui est la cas des agrégateurs usuellement utilisés). Parmi les agrégateurs présentés dans le chapitre 1, cette approche permet de résoudre le cas où les coefficients de OWA ne sont pas décroissants, ou encore le cas où l'intégrale de Choquet est définie avec une capacité pas nécessairement convexe.

Perspectives de recherche. Nous allons présenter maintenant quelques perspectives de recherche qu'ouvrent cette thèse et les travaux qu'il serait intéressant d'entreprendre dans le futur.

- **Améliorations de l'approche par approximation** : Le chapitre 3 se base sur la génération d'un réseau approximé pour résoudre exactement des problèmes de choix, ainsi que leurs dérivés (rangement, etc...). L'algorithme est basé sur différents moteurs qui s'appellent mutuellement (le rangement, l'analyse graphique et l'approximation). Il est intéressant de tester de nouveaux moteurs d'analyse graphique et d'approximation pour étudier leur impact sur les temps de calcul de l'algorithme. Notamment, nous avons choisi d'utiliser un moteur d'approximation exploitant la programmation linéaire, mais il est possible de générer de nouvelles formulations du problème (sous forme de programmes quadratiques par exemple). De plus, l'algorithme tel qu'il a été présenté, exploite implicitement un quatrième moteur (évoqué dans les conclusions du chapitre), évaluant des distances d'approximation. Ces distances sont calculées en fonction de la valeur de la fonction objectif d'un programme linéaire (lié aux approximations) à l'optimum. Il peut être intéressant de déterminer une fonction de distance liée aux utilités, au même titre que la divergence de Kullback-Leibler (Kullback et Leibler, 1951) pour les probabilités. Cette fonction de distance devrait idéalement :
  - Renvoyer 0 si une utilité correspond à une transformée linéaire (avec un coefficient multiplicatif strictement positif) de l'autre utilité.
  - Être symétrique.
  - Vérifier l'inégalité triangulaire.
- **Une nouvelle approche de la triangulation** : L'algorithme du chapitre 3 nous donne la capacité d'exploiter un réseau GAI obtenu par triangulation d'un graphe markovien auquel il est possible de supprimer des arêtes. Dans la présentation que nous avons faite du moteur d'analyse graphique, nous avons supprimé

les arêtes après avoir effectué la triangulation. Mais si on intègre la notion de suppression d'arêtes au sein même du processus de triangulation, on obtient un nouveau problème dont la formulation est : Étant donné un graphe non orienté $G = (V, E)$, comment déterminer des ensembles $S \subseteq E$ et $F \subseteq \mathcal{P}_2(V) \backslash E$, tels que $G' = (V, (E \backslash S) \cup F)$ soit cordal ? En insérant un objectif à optimiser pour les ensembles E et F (minimiser la taille des cliques produites, etc...), il est intéressant d'étudier ce nouveau concept de triangulation et de déterminer l'ensemble des problèmes qu'une telle approche peut résoudre (dans le domaine des réseaux bayésiens, des CSP, des diagrammes d'influence, etc...).

- **Généralisation des algorithmes aux autres modèles graphiquement décomposables** : Les réseaux GAI entretiennent un rapport fort avec d'autres modèles graphiquement décomposables. Par exemple, un réseau bayésien (Pearl, 1988) est une représentation graphique d'une probabilité jointe sur un ensemble de n variables aléatoires. Il exploite les indépendances conditionnelles entre variables aléatoires pour réduire l'espace mémoire requis pour représenter cette probabilité jointe. On a donc une décomposition de la forme : $P(X_1, \ldots, X_n) = \prod_{i=1}^{n} P(X_i|Pa(X_i))$ (avec $Pa(X_i)$ un ensemble de variables conditionnantes). Un réseau bayésien peut se transformer aisément en réseau GAI, il suffit de définir $X_1, \ldots, X_n$ comme étant des attributs, alors le logarithme de la loi de probabilité devient :

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

  En posant $u_i(X_i, Pa(X_i)) = \log(P(X_i|Pa(X_i)))$, on obtient exactement une GAI-décomposition. En particulier, le problème MPE [3] consiste à déterminer l'instanciation des variables x^* vérifiant $x^* = \arg\max P(X_1, \ldots, X_n)$. Or, si x^* maximise la loi de probabilité, il maximise aussi le logarithme de cette loi. MPE peut donc se ramener à un problème de détermination du choix optimal. Il serait donc intéressant d'étudier les parallèles entre les différentes problématiques des modèles graphiquement décomposables pour voir comment certaines approches développées dans cette thèse peuvent s'adapter pour résoudre d'autres problèmes.
- **Vers la décision collective et les systèmes multi-agents** : Nous avons évoqué dans l'introduction de cette thèse qu'il existe un rapport entre décision multicritère et décision collective. En effet, un problème comportant un ensemble de m décideurs devant prendre une décision commune peut être traité comme un problème comportant un unique décideur et m critères, les relations de préférence de chaque critère correspondant aux relations de préférence de chaque agent. La principale différence entre décision collective et décision multicritère porte ici plutôt sur le nombre attendu de relations. En décision multicritère, il est rare d'atteindre 20 ou 25 critères, tandis que dans les problèmes de systèmes multi-agents, le nombre d'agents (les décideurs) peut être de l'ordre de la centaine, voir bien plus. Il serait intéressant de mélanger les différentes techniques proposées pour étudier les temps de calcul obtenus suite au passage à l'échelle sur des systèmes multi-agents (par exemple, élaborer un algorithme approché avec garantie de performance pour déterminer la solution optimale d'une intégrale de Choquet non convexe pour un problème de décision collective).

3. Most Probable Explanation

– **Algorithmique parallèle et répartie** : Le systèmes multi-processeurs se démocratisent de plus en plus avec l'émergence des dual-core et quad-core sur les ordinateurs personnels, la possible présence d'un cluster de calculs au sein des entreprises, ou tout simplement la mise en réseau des ordinateurs. La structure des réseaux GAI peut être utilisée pour répartir les calculs. Dans un algorithme classique de collecte/diffusion, on peut aisément imaginer que chaque « branche » peut se traiter indépendamment des autres, jusqu'à arriver sur une clique jouant le rôle de point d'intersection entre les branches. Les messages arrivant à cette clique doivent alors se situer dans une mémoire partagée. Il serait intéressant d'étudier la flexibilité des algorithmes proposées dans une optique de parallélisation des calculs, et d'élaborer des algorithmes spécifiquement adaptés à la répartition des calculs.

Bibliographie

G. Anandalingam et C. C. White (1993) "A penalty function approach to alternative pairwise comparisons in ISMAUT", *IEEE transactions on systems, man, and cybernetics*, vol. 23, pp. 330–333.

S. Arnborg, D. Corneil et A. Proskurowski (1987) "Complexity of finding embeddings in a k-tree", *SIAM J. Algebraic Discrete Methods*, vol. 8, pp. 277–284.

F. Bacchus et A. Grove (1995) "Graphical models for preference and utility", dans *Proc. of UAI*, pp. 3 – 10.

A. Berry, J.-P. Bordat, P. Heggernes, G. Simonet et Y. Villanger (2006) "A wide-range algorithm for minimal triangulation from an arbitrary ordering", *J. Algorithms*, pp. 33–66.

U. Bertele et F. Brioschi (1972) *Nonserial dynamic programming*, Academic Press, New York.

A. Berthoz (2003) *La décision*, Odile Jacob, Paris.

J. Blythe (2002) "Visual exploration and incremental utility elicitation", dans *Proc. of AAAI*, pp. 526 – 532.

H. Bodlaender, J. Gilbert et T. Hafsteinsson, H.and Kloks (1995) "Approximating treewidth, pathwidth, frontsize, and shortest elimination tree", *J. Algorithms*, vol. 18, pp. 238–255.

C. Boutilier (2002) "A POMDP formulation of preference elicitation problems", dans *Proc. of AAAI*, pp. 239 – 246.

C. Boutilier, F. Bacchus et R. Brafman (2001) "UCP-networks ; a directed graphical representation of conditional utilities", dans *Proc. of UAI*, pp. 56 – 64.

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

C. Boutilier, R. Brafman, C. Domshlak, H. Hoos et D. Poole (2004b) "Preference-based constraint optimization with CP-nets", *Computational Intelligence*, vol. 20, pp. 137–157.

D. Bouyssou, T. Marchant, P. Perny, M. Pirlot, A. Tsoukiàs et P. Vincke (2000) *Evaluation and decision models : a critical perspective*, Kluwer Academic Publishers.

R. Brafman et C. Domshlak (2002) "Introducing variable importance tradeoffs into CP-nets", dans *Proc. of UAI*, pp. 69–76.

D. Braziunas et C. Boutilier (2005) "Local utility elicitation in GAI models", dans *Proc. of UAI*, pp. 42–49.

D. Braziunas et C. Boutilier (2007) "Minimax regret-based elicitation of generalized additive utilities", dans *Proc. of UAI*.

D. Braziunas et C. Boutilier (2008) "Elicitation of factored utilities", *AI Magazine*, vol. 29, pp. 79–92.

U. Chajewska et D. Koller (2000) "Utilities as random variables : Density estimation and structure discovery", dans *Proc. of UAI*, pp. 63–71.

G. Choquet (1953) "Theory of capacities", *Annales de l'institut Fourrier*, vol. 5, pp. 131–295.

E. Dahlhaus et M. Karpinski (1989) "An efficient parallel algorithm for the minimal elimination ordering (MEO) of an arbitrary graph", *Foundations of Computer Science, Annual IEEE Symposium on*, vol. 0, pp. 454–459.

G. Debreu (1964) "Continuity properties of paretian utility", *International Economic Review*, vol. 5, pp. 285–293.

R. Dechter (2003) *Constraint processing*, Morgan Kaufmann.

T. Degris (2007) *Apprentissage par Renforcement dans les Processus de Décision Markoviens Factorisés*, Thèse de doctorat, LIP6/AnimatLab, Université Pierre et Marie Curie, Paris, France.

J.-P. Dubus, C. Gonzales et P. Perny (2009a) "Choix multiattribut à l'aide de réseaux GAI de forte densité", dans *Actes du 10ème Congrès de la Société Française de Recherche Opérationnelle et d'Aide à la Décision*, pp. 25–39.

J.-P. Dubus, C. Gonzales et P. Perny (2009b) "Choquet optimization using GAI networks for multiagent/multicriteria decision-making", dans *Algorithmic Decision Theory*, Lectures Notes in Artificial Intelligence, pp. 377–389.

J.-P. Dubus, C. Gonzales et P. Perny (2009c) "Fast recommendations using GAI models", dans *IJCAI'09, International Joint Conference on Artificial Intelligence*, pp. 1896–1901.

J.-P. Dubus, C. Gonzales et P. Perny (2009d) "Multiobjective optimization using GAI models", dans *IJCAI'09 International Joint Conference on Artificial Intelligence*, pp. 1902–1907.

Y. Engel et M. P. Wellman (2006) "CUI networks : A graphical representation for conditional utility independence", dans *Proc. of AAAI*, pp. 83–112.

A. Eyal (2001) "Efficient approximation for triangulation of minimum treewidth", dans *Proc. of UAI*, pp. 7–15.

P. C. Fishburn (1970) *Utility Theory for Decision Making*, Wiley.

J. Flores, J. Gamèz et K. Olesen (2003) "Incremental compilation of Bayesian Networks", dans *Proc. of UAI*, pp. 233–240.

J. Fodor et M. Roubens (1994) *Fuzzy Preference Modelling and Multi-Criteria Decision Aid*, Kluwer Academic Publisher.

L. Galand (2008) *Méthodes exactes pour l'optimisation multicritère dans les graphes : recherche de solutions de compromis*, Thèse de doctorat, LIP6 - université Paris 6.

M. R. Garey et D. S. Johnson (1979) *Computers and Intractability : A Guide to the Theory of NP-Completeness*, W. H. Freeman.

F. Gavril (1974) "The intersection graphs of subtrees in trees are exactly the chordal graphs", *J. Combinatorial Theory, Series B*, vol. 16, pp. 47–56.

B. Golden et P. Perny (2010) "Infinite order lorenz dominance for fair multiagent optimization", dans *Proc. of AAMAS*, pp. 383–390.

M. C. Golumbic (1980) "Triangulated graphs", *Algorithmic Graph Theory and Perfect Graphs*, pp. 98–100.

C. Gonzales et P. Perny (2004) "GAI networks for utility elicitation", dans *Proc. of KR*, pp. 224–234.

C. Gonzales et P. Perny (2005) "GAI networks for decision making under certainty", dans R. Brafman et U. Junker, réds., *Proceedings of the 19th International Joint Conference on Artificial Intelligence – workshop on advances in preference handling*, pp. 100–105.

C. Gonzales, P. Perny et S. Queiroz (2007) "Réseaux GAI pour la prise de décision", *Revue d'Intelligence Artificielle*, vol. 21, no. 4, pp. 555–587.

C. Gonzales, P. Perny et S. Queiroz (2008) "Preference aggregation with graphical utility models", dans *Proceedings of the 23rd AAAI conference on Artificial Intelligence*, pp. 1037–1042.

M. Grabisch (1996) "The application of fuzzy integrals in multicriteria decision making", *European Journal of Operational Research*, vol. 89, pp. 445–456.

M. Grabisch et P. Perny (2003) *Agrégation multicritère*, Hermès.

S. Greco, B. Matarazzo et R. Slowiński (2000) *Encyclopedia of Management*, chap. Decision rules, Farmington Hills.

P. Hansen (1980) "Bicriterion path problems", dans G. Fandel et T. Gal, réds., *Multiple Criteria Decision Making : Theory and Applications*, Springer.

G. H. Hardy, J. E. Littlewood et G. Pólya (1934) *Inequalities*, Cambridge University Press.

E. Jacquet-Lagrèze (1975) *La modélisation des préférences : préordres, quasi-ordres et relations floues*, Thèse de doctorat, Université de Paris V.

J.-Y. Jaffray (1995) "On the maximum probability which is consistent with a convex capacity", *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 3, pp. 27–34.

K. Kask et R. Dechter (1999) "Branch and bound with mini-bucket heuristics", dans *IJCAI'99, International Joint Conference on Artificial Intelligence*, pp. 426–433.

R. L. Keeney et H. Raiffa (1993) *Decisions with multiple objectives - preferences and value tradeoffs*, Cambridge University Press.

U. Kjærulff (1990) "Triangulation of graphs — algorithms giving small total state space", *Rap. tech. R-90-09*, Dept. of Maths and Computer Science, Aalborg University.

M. Kostreva et W. Ogryczak (1999) "Linear optimization with multiple equitable criteria", *RAIRO Operations Research*, vol. 33, pp. 275–297.

D. Krantz, R. D. Luce, P. Suppes et A. Tversky (1971) *Foundations of Measurement (Additive and Polynomial Representations)*, vol. 1, Academic Press.

D. H. Krantz, R. D. Luce, P. Suppes et A. Tversky (1989) *Foundations of Measurement (Geometrical, Threshold, and Probabilistic Representations)*, vol. 2, Academic Press.

S. Kullback et R. Leibler (1951) "On information and sufficiency", *Annals of Mathematical Statistics*, vol. 22, pp. 79–86.

J. Larrosa et T. Schiex (2003) "In the quest of the best form of local consistency for weighted csp", dans *Proc. of IJCAI'03*.

M. Laumanns, L. Thiele, K. Deb et E. Zitzler (2002) "Combining convergence and diversity in evolutionary multiobjective optimization", *Evolutionary Computation*, vol. 10, pp. 263–282.

H. Leimer (1993) "Optimal decomposition by clique separators", *Discrete Mathematics*, vol. 113, pp. 99–123.

R. D. Luce (1956) "Semiorders and a theory of utility discrimination", *Econometrica*, vol. 24, pp. 415–426.

R. D. Luce et H. Raiffa (1957) *Games and Decisions : Introduction and Critical Survey*, Wiley, New York.

A. Madsen et K. Olesen (2002) "Maximal prime subgraph decomposition of Bayesian Networks", *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, vol. 32, pp. 21–31.

H. Montgomery (1987) "Image theory and dominance search theory : how is decision making actually done ?", *Acta Psychologica*, vol. 66, pp. 221–224.

T. Murofushi et M. Sugeno (1991) "A theory of fuzzy measures. representation, the Choquet integral and null sets", *Journal of Mathematical Analysis and Applications*, vol. 159, pp. 532–549.

D. Nilsson (1998) "An efficient algorithm for finding the M most probable configurations in probabilistic expert systems", *Statistics and Computing*, vol. 8, no. 2, pp. 159–173.

W. Ogryczak (2000) "Inequality measures and equitable approaches to location problems", *European Journal of Operational Research*, vol. 122, pp. 374–391.

T. Ohtsuki (1976) "A fast algorithm for finding an optimal ordering in the vertex elimination on a graph", *SIAM J. Computing*, vol. 5, pp. 133–145.

C. H. Papadimitriou et Y. Yannakakis (2000) "On the approximability of trade-offs and optimal access of web sources", dans *Proc. of FOCS*.

A. Para et P. Scheffler (1997) "Characterizations and algorithmic applications of chordal graph embeddings", *Discrete Applied Mathematics*, vol. 79, pp. 171–188.

J. Pearl (1988) *Probabilistic Reasoning in Intelligent Systems*, Morgan-Kaufmann.

P. Perny et O. Spanjaard (2008) "Near admissible algorithms for multiobjective search", dans *Proc. of ECAI*, pp. 490–494.

P. Perny, O. Spanjaard et L.-X. Storme (2006) "A decision-theoretic approach to robust optimization in multivalued graphs", *Annals of Operations Research*, vol. 147, pp. 317–341.

B. W. Peyton (2001) "Minimal orderings revisited", *SIAM J. Matrix Analysis and Applications*, pp. 271–294.

S. Queiroz (2008) *Modèles graphiques décomposables pour la décision individuelle et collective*, Thèse de doctorat, LIP6 - université Paris 6.

D. Rose (1970) "Triangulated graphs and the elimination process", *Journal of Mathematical Analysis and Applications*, vol. 32, pp. 597–609.

D. Rose, R. Tarjan et G. Lueker (1976) "Algorithmic aspects of vertex elimination on graphs", *SIAM journal on Computing*, vol. 5, pp. 266–283.

M. Roubens et P. Vincke (1985) *Preference Modelling*, Springer-Verlag, Berlin.

B. Roy (1985) *Méthodologie multicritère d'aide à la décision*, Economica.

L. Savage (1954) *The Foundations of Statistics*, Wiley, New York.

D. Schmeidler (1986) "Integral representation without additivity", *Proceedings of the American Mathematical Society*, vol. 97, pp. 255–261.

A. K. Sen et J. E. Foster (1997) *On economic inequality*, Clarendon Press.

G. Shafer et P. P. Shenoy (1990) "Probability propagation", *Annals of Mathematics and Artificial Intelligence*, vol. 2, pp. 327–352.

L. Shapley (1971) "Cores of convex games", *International Journal of Game Theory*, vol. 1, pp. 11–22.

P. P. Shenoy et G. Shafer (1990) "Axioms for probability and belief-function propagation", dans *Proc. of UAI*, pp. 169–198.

K. Shoikhet et D. Geiger (1997) "A practical algorithm for finding optimal triangulations", dans *Proc. of AAAI*, pp. 185–190.

R. Steuer et E. Choo (1983) "An interactive weighted Tchebycheff procedure for multiple objective programming", *Mathematical Programming*, vol. 26, pp. 326–344.

M. Sugeno (1974) *Theory of fuzzy integrals and its applications*, Thèse de doctorat, Tokyo Institute of Technology.

R. E. Tarjan et M. Yannakakis (1984) "Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs and selectively reduce hypergraphs", *SIAM J. Computing*, vol. 13, pp. 566–579.

F. van den Eijkhof et H. L. Bodlaender (2002) "Safe reduction rules for weighted treewidth", vol. 2573 de *Lecture Notes in Computer Science*, Springer, pp. 176–185.

V. V. Vazirani (2001) *Approximation Algorithms*, Springer.

P. Vincke (1989) *L'Aide Multicritère à la Décision*, Editions de l'Université de Bruxelles.

P. Vincke (1992) *Multicriteria Decision Aid*, Wiley.

J. von Neumann et O. Morgenstern (1947) *Theory of games and economic behaviour*, Princeton University Press, 2nd edition.

P. P. Wakker (1989) *Additive Representations of Preferences, A New Foundation of Decision Analysis*, Kluwer Academic Publishers.

M. Weber (1987) "Decision making with incomplete information", *European Journal of Operational Research*, vol. 28, pp. 44–57.

C. C. White, S. Dozono et W. T. Scherer (1983) "Interactive procedure for aiding multiattribute alternative selection", *Omega*, vol. 11, pp. 212–214.

A. Wierzbicki (1986) "On the completeness and constructiveness of parametric characterizations to vector optimization problems", *OR Spektrum*, vol. 8, pp. 73–87.

R. R. Yager (1988) "On ordered weighted averaging aggregation operators in multicriteria decision making", *IEEE Transactions on Systems, Man and Cybernetics*, vol. 18, pp. 183–190.

M. Yannakakis (1981) "Computing the minimum fill-in is NP-complete", *SIAM Journal on Discrete Mathematics*, vol. 2, pp. 77–79.

Index

(ϵ, w)-dominance, 116
$(\log, w)$-dominance, 117
ϵ-couverture, 115
ϵ-dominance, 114
log-dominance, 115

Agrégateur, 27
  Choquet, 32
  OWA, 28
  Tchebycheff, 30
    Point idéal, 29
    Point Nadir, 29
Agrégation
  Forêts de jonction, 55
  Graphes markoviens, 55
  Réseaux GAI (somme), 56
Alternative
  préférée, 58
Antécédents, 132
Application partielle, 87
Arbre de jonction, 43
Attributs, 17
  instanciation, 17
  modalité, 17

Borne supérieure
  d'OWA, 70
  de Choquet, 71
  de Tchebycheff, 71

Capacité, 32
  additive, 33
  concave, 33
  convexe, 33
  duale, 33
  noyau, 71
  sousmodulaire, 33
  supermodulaire, 33
  symétrique, 33
Classe d'équivalence, 15
Combinaison ensembliste, 107
Compatibilité avec Pareto, 25
Complexité
  Choix optimal, 64
  Collecte, 64
  Diffusion, 64
  Rangement, 69
Contrainte, 56

Décomposition
  additive, 18
    généralisée, 37
Dominance
  de Lorenz, 26
  de Pareto, 23
    faible, 23

Ensemble de découpages, 85
Espace des critères, 23

Fill-in, 47
Fonction d'utilité, 15
  étendue, 56
  additive, 18
  GAI, 37
  multicritère, 22
Fonction heuristique, 127
Forêt de jonction, 43
Frontière
  de Lorenz, 26
  de Pareto, 24

GAI
  décomposition, 37
Graphe
  triangulé, 47
  acyclique, 40
  Arbre, 40
  Chaîne, 39
  Chemin, 39

Circuit, 39
Clique, 40
complet, 40
Composante connexe, 40
connexe, 39
cordal, 41
Corde, 39
Cycle, 39
Forêt, 40
markovien, 47
d'une contrainte, 57
non orienté, 38
orienté, 38
Sous-graphe, 40
induit, 40
partiel, 40

Heuristique, 127
Hypergraphe, 42
graphe dual, 42
graphe primal, 42

Indépendance généralisée, 45
Intégrale de Choquet, 32
Intersection courante, 43

Label, 135

Norme de Tchebycheff, 30

P-monotonie, 25
Point
idéal, 29
Nadir, 29
Prédecesseurs, 63
Prédecesseurs d'une arête, 91
Principe de transfert, 25
Problème
frontière de Lorenz (décision), 142
frontière de Pareto (décision), 104
maximisation OWA (décision), 144
Partition (décision), 143, 144
Sac à dos (décision), 105
Projection, 36

Réduction, 52
Réseau GAI, 43
scalarisé, 70
somme, 56
vectoriel, 105
Relation
complète, 14
d'équivalence, 15
d'indifférence, 13
de préférence, 13
stricte, 13
préordre complet, 15
réflexive, 14
symétrique, 14
transitive, 14

Séquence d'élimination, 49
Successeur, 63
Suppression saine, 92

Transformée de Lorenz, 26
Treewidth, 53
Triangulation, 47, 48

Liste des Algorithmes

1 Prise de décision en deux phases 12
2 Triangulation 48
3 Réduction 52
4 IntégrerContraintes 57
5 Collecte 63
6 Diffusion 64
7 RankingCollecte 67
8 RankingDiffusion 67
9 Rangement 68
10 RangementSymbolique 69
11 Construction d'une capacité additive d'entropie maximale 72
12 AgregationOptimale 73
13 ChoixOptTreewidth 81
14 Approximer 89
15 RangementTreewidth 98
16 CollecteVectorielle 109
17 CollecteVectorielleApprochee 118
18 CollecteHeuristique 129
19 DiffusionHeuristique 130
20 Labeliser 135
21 UnPas 136
22 InitialiserMouvement 137
23 ParetoSansHeuristique 138
24 ParetoHeuristique 139
25 LorenzHeuristique 145
26 AgregHeuristique 147

Vu :
Le Président
M.

Vu :
Les Suffragants
MM.

Vu et permis d'imprimer :
Le Vice-Président du Conseil Scientifique chargé de la Recherche de l'université PARIS VI PIERRE ET MARIE CURIE

Résumé
Les réseaux GAI sont une représentation graphique compacte et expressive des préférences d'un décideur en Décision Multiattribut, c'est-à-dire dans des situations où les alternatives sur lesquelles portent les choix du décideur sont décrites à l'aide d'un ensemble d'attributs (de caractéristiques). L'exploitation de leur structure graphique permet de définir des procédures efficaces d'élicitation de préférences (détermination des préférences à l'aide de questionnaires) ainsi que des algorithmes assez performants de prise de décision (calcul de l'alternative préférée du décideur ou des k meilleures alternatives). Le but de cette thèse est double. Tout d'abord elle vise à étendre les algorithmes de prise de décision dans des cas où les réseaux GAI sont denses, c'est-à-dire dans des situations où leur structure ne permet pas aux algorithmes de l'état de l'art de s'exécuter en un temps raisonnable. Pour cela, une nouvelle méthode de triangulation approchée a été développée, qui produit des réseaux GAI approchés sur lesquels des mécanismes d'inférence adaptés permettent d'obtenir les alternatives optimales des réseaux GAI d'origine. Ensuite, elle propose de nouvelles méthodes d'inférence en Décision multicritère. Plus précisément, elle propose des approches pour déterminer des frontières de Pareto (exactes ou approchées avec garantie de performance) ou des frontières de Lorenz. Elle propose également des algorithmes pour déterminer des solutions optimales dans les cas où les critères peuvent être agrégés via des opérateurs tels que OWA (Ordered Weighted Average), l'intégrale de Choquet ou bien encore la norme de Tchebycheff.
Mots-clés : Aide à la décision, décision multiattribut, décision multicritère, modélisation des préférences, optimisation combinatoire, graphes, recherche heuristique.

Abstract
GAI networks are a graphical model, both compact and expressive, for representing the preferences of a Decision Maker in the context of Multiattribute Decision Making, i.e., in situations where the set of alternatives among which the Decision Maker has to make decisions are described as a set of attributes (or features). GAI network's graphical structures are exploited to develop efficient elicitation procedures (determination of the Decision Maker's preferences using questionnaires) as well as effective Decision Making algorithms (e.g., computing the preferred alternative or the k-best alternatives). The goal of this PhD thesis is twofold. First, it extends the aforementioned state-of-the-art Decision Making algorithms to be able to cope with dense GAI networks, i.e., with situations where the GAI network's treewidth is too high for these algorithms to complete in a reasonable amount of time. For this purpose, a new triangulation method has been developed which produces approximated GAI networks on which tailored inference mechanisms determine the alternatives that are actually optimal for the original GAI network. Second, we have proposed new inference algorithms for Multicriteria Decision Making. More precisely, new approaches for determining Pareto-optimal sets (exact and approximate with performance guarantee) and Lorenz-optimal sets have been developed. In addition, we have also proposed new algorithms for computing the optimal solutions in situations where criteria are aggregated using various operators like OWA (Ordered Weighted Average), Choquet integrals and Tchebycheff's norm.
Keywords : Decision Aiding, Multiattribute Decision, Multicriteria Decision, Preference Modelling, Combinatorial Optimization, Graphs, Heuristic Search.