THÈSE DE DOCTORAT DE L'UNIVERSITÉ PARIS 6

Spécialité : **Informatique**
(École Doctorale d'Informatique, Télécommunications et Électronique – EDITE)

Présentée pour obtenir le titre de
DOCTEUR DE L'UNIVERSITÉ PARIS 6

par

Sergio Queiroz

Sujet de la thèse :

Modèles graphiques décomposables pour la décision individuelle et collective

soutenue le 12 Novembre 2008
devant le jury composé de :

Mme. Amal El Fallah Seghrouchni	Examinateur
M. Pierre Marquis	Rapporteur
M. Patrice Perny	Directeur de thèse
M. Geber Ramalho	Examinateur
M. Alexis Tsoukiàs	Rapporteur
M. Jean-Daniel Zucker	Examinateur

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.

Résumé

Cette thèse porte sur l'utilisation des GAI-Nets, un modèle graphique pour la représentation compacte de préférences, pour atteindre des fonctionnalités propres à un système de recommandation dans le cadre où l'espace d'alternatives a une structure combinatoire de grande taille. Typiquement, les systèmes de recommandation sur le Web utilisent des techniques bien adaptées au conseil d'articles fortement standardisés, tels que les CDs et les DVDs, mais impraticables dans un cadre combinatoire. Par ailleurs, les systèmes de recommandation pour le cadre combinatoire sont souvent fondés sur des modèles supposant une indépendance entre attributs qui assure la modélisation des préférences par une utilité additive. Les GAI-Nets permettent des interactions entre les attributs, étant ainsi plus généraux. Nos problématiques clés sont le choix et le rangement des k-meilleures alternatives. Nous étudions également le problème de la recherche de solutions de compromis selon des critères non-linéaires dans le cadre de la décision collective/multicritère, et aussi l'élicitation des GAI-Nets. Nous proposons des algorithmes adaptés à la résolution de tels problèmes et, finalement, nous construisons une application Web pour appliquer les techniques développées dans une situation décisionnelle concrète.
Mots-clés : décompositions GAI, modèles graphiques, représentation compacte de préférences, systèmes de recommandation, optimisation combinatoire.

Title: Decomposable graphical models for individual and collective decision

Abstract

This thesis focuses on the use of GAI-Nets, a graphical model for compact preference representation, in order to achieve the usual features of a recommender system in the context where the space of alternatives has a large combinatorial size. Typically, web recommender systems use techniques well suited to highly standardized items such as CDs and DVDs, but inappropriate to combinatorial contexts. In addition, recommender systems for combinatorial contexts are often based on models that assume a kind of independence between attributes that ensures modeling preferences by an additive utility. GAI-Nets allow for interactions between attributes, thus being more general. Our key problems are choice and ranking of the k-best alternatives. We also study the problem of finding compromise solutions according to non-linear criteria in the context of multiperson/multicriteria decision making, as well as the problem of eliciting GAI-Nets. We provide adequate algorithms to solve these problems and, finally, we build a web application to apply the developed techniques in a concrete decision setting.
Keywords: GAI decompositions, graphical models, compact preference representation, recommender systems, combinatorial optimization.

Cette thèse a été préparée au sein du département *Décision, Systèmes Intelligents et Recherche Opérationnelle* (DESIR) au *Laboratoire d'Informatique de Paris 6* (LIP6) – 104 Av. du Président Kennedy 75016 Paris

Remerciements

En premier lieu, je tiens à remercier le Professeur Patrice Perny qui a dirigé cette thèse. Pour les discussions enrichissantes que nous avons eues, ainsi que pour la précision de ses vues et la rigueur de ses jugements, je le remercie vivement. Sa passion et son dévouement pour la recherche sont des exemples pour les jeunes chercheurs.

Je tiens également à remercier vivement Christophe Gonzales pour les commentaires et suggestions très pertinents qui m'ont beaucoup apporté.

Mes sincères remerciements vont au Professeur Alexis Tsoukiàs ainsi qu'au Professeur Pierre Marquis, pour avoir accepté d'assurer la lourde tâche de rapporteurs.

Je suis reconnaissant à la Professeur Amal El Fallah Seghrouchni, au Professeur Jean-Daniel Zucker et au Professeur Geber Ramalho davoir accepté de faire partie du jury.

Je remercie infiniment la CAPES (Coordination pour l'Amélioration du Personnel de l'Enseignement Supérieur, un organisme du Ministère de l'Éducation Brésilien) pour le financement partiel qu'ils mont proportionné tout au long de la thèse.

Ce mémoire compterait un nombre élevé de fautes d'orthographe et de grammaire si ce n'était la relecture minutieuse de la première version du manuscrit par Ludiwine Lehmann. Hélas, j'ai probablement introduit de nouvelles fautes depuis...

Merci à Alessandro Almeida pour mener les démarches administratives pour la soutenance de la thèse, lorsque j'étais au Brésil. Je le remercie également pour son amitié.

J'ai eu le privilège d'avoir l'amitié d'un grand nombre d'autres « passagers de Paris ». Je pense spécialement à mes anciens colocataires Rodolpho et Judson, mais aussi à Eliana, Najla, Patricia, Silvia et Charles, Aydano et Jenny, Geber, Chico, Marina, Érico et Cibele, Alzennyr, Julien, Tamara, Laurette, Emmanuelle, Jacques, Jeremy, Karyn et Kurt, Simon, André, Giordano, Aluizio, Silvio et Sophie, Rogerio, Ítalo, Mirka, Juliana, Marinho, Vaninha, Raphael, Emmanuel, Lydia, Katherin, Grâce, Ghislain, Audrey et Fred, Moïse, Daniel, Libby, Edwige, Simone, Zacharie, John, Samuel...

Je ne peux pas oublier de remercier les membres de l'équipe DESIR du Laboratoire d'Informatique de Paris VI, qui ont fait de mon séjour au LIP6 une expérience agréable et enrichissante. Merci donc à Philippe, Jean-Yves, Christophe, Safia, Francis, Pierre-Henri,

Claire, Olivier, Emmanuel, Vincent, Pierre, Paul, Louis-Xavier, Antoine, Yann, Nabil, Yasmin, Nicolas, Lucie, Anna, Nina, Gildas, Jean-Philippe, Lionel.

Finalement, je souhaite remercier ma famille pour son soutien solide et toujours présent.

Table des matières

Liste des tableaux

Table des figures

Liste des Algorithmes

Introduction

La prise de décision constitue la pierre angulaire de plusieurs applications de l'intelligence artificielle (IA), telles que les systèmes spécialistes, les systèmes d'aide à la décision, les agents autonomes, et les systèmes de recommandation. Tandis que pour certaines de ces applications l'objectif ultime est d'être capable de prendre des décisions autonomes, comme par exemple dans le cas d'un aéronef inhabité (Karim et Heinze, 2005), dans la plupart des cas, l'objectif est d'aider un ou plusieurs agents humains à prendre des décisions bien informées face à des problèmes complexes. Confrontés à de tels problèmes, les individus peuvent se trouver incapables d'évaluer de façon systématique la masse d'informations interférant dans le processus décisionnel afin de choisir de manière « raisonnable » entre les différentes alternatives possibles. On appelle **théorie de la décision** l'approche scientifique pour l'aide à la décision, un domaine pluridisciplinaire au carrefour de multiples disciplines telles que l'économie, la psychologie, la philosophie, les sciences de gestion et l'informatique.

Comme le soulignent Keeney et Raiffa (1993, preface de la première édition), la théorie de la décision est « *designed for normally intelligent people who want to think hard and systematically about some important real problems* ». À partir de cette phrase, on peut distinguer à la fois des qualités de la théorie de la décision et des handicaps qui restreignent son utilisation dans certains contextes.

D'une part, on constate que la théorie de la décision a été développée dans l'intention d'être adapté à l'intervenant dans le processus décisionnel (que l'on nomme **décideur**). Ainsi, les modèles utilisés ne supposent pas que le décideur soit un spécialiste dans la théorie de la décision pour être capable de les manipuler. On remarque également l'attitude majoritairement prescriptive de la théorie de la décision, établissant un cadre constructif qui doit opérer en collaboration avec le décideur de façon à ce que celui-ci puisse réfléchir aux multiples facettes du problème considéré tout en en restant son seul maître (Keeney et Raiffa, 1993). Finalement, on note que la théorie de la décision nous aide à raisonner sur des problèmes concrets, avec un haut niveau de complexité.

D'autre part, « *think hard and systematically* » indique que l'on suppose un certain niveau d'engagement du décideur dans le processus décisionnel. Ainsi, le décideur doit

être disposé à prendre le temps de faire connaître ses préférences et d'analyser le problème sur divers points de vue, potentiellement conflictuels. Cette prémisse peut être particulièrement contraignante pour des problèmes où les enjeux en cause ne sont pas très importants. Par exemple, un décideur peut être moins disposé à investir du temps pour choisir quel film regarder le dimanche après-midi, que pour choisir où passer ses prochaines vacances.

À partir des années 1990, on a observé le fleurissement des systèmes de recommandation sur le Web. Dans la vie quotidienne, quand on doit faire un choix sans pleine connaissance des alternatives possibles, une approche courante est de compter sur les recommandations d'individus de confiance, tels qu'un critique de films, un ami, ou un consultant. Les systèmes de recommandation automatisent ce processus de recommandation. L'approche prédominante dans ces systèmes est le filtrage collaboratif (voir par exemple Herlocker *et al.* (1999)). Comme le soulignent Nguyen et Haddawy (1999), « *the work on collaborative filtering is not cast in the framework of decision theory and no theoretical framework or justification are provided for the similarity measures used in matching users' tastes* ». Néanmoins, dans la pratique, cette approche a obtenu un grand succès, typiquement pour des problèmes où les enjeux ne sont pas très importants. Par exemple, *Netflix**, le plus grand service en ligne de location de films au monde (avec plus de 6,7 million d'abonnés), utilise un système de recommandation fondé sur le filtrage collaboratif et a récemment lancé un concours où un prix de 1 million de dollars serait offert à celui qui arriverait à battre la performance de leur système de recommandation par une marge d'au moins 10%.

Toutefois, le filtrage collaboratif n'est possible que quand les articles sont fortement standardisés (c'est-à-dire, disponibles sous la même forme), comme par exemple les livres, les CDs et les DVDs (Ricci *et al.*, 2003). Quand les objets sont composés par des attributs configurables, l'énumération de toutes les configurations possibles peut s'avérer impraticable. Il s'agit de problèmes où l'espace d'alternatives a une structure combinatoire, c'est-à-dire que c'est un produit cartésien de variables. Dans ce contexte, pour raisonner sur les préférences des décideurs, des **modèles adaptés** couplés avec des **représentations efficaces** se font nécessaires.† Ainsi, des travaux récents en IA concernant la modélisation et la représentation de préférences, cherchent à atteindre un bon compromis entre deux aspects parfois antagonistes : i) le besoin de modèles suffisamment riches et flexibles pour décrire des comportements de décision sophistiqués ;

*`http://www.netflix.com/`

†Nous précisons que quand nous nous référons à des modèles, nous nous intéressons aux modèles mathématiques utilisés. Un modèle adapté aux préférences du décideur est en accord avec ses préférences. En revanche, quand nous parlons des représentations, nous nous intéressons aux aspects computationnels. Ainsi, une représentation efficace doit être performante en temps d'exécution et espace mémoire.

et ii) la nécessité pratique de maintenir l'effort d'élicitation à un niveau admissible et aussi le besoin de représentations efficaces pour résoudre des problèmes d'optimisation fondés sur des préférences.

Cet objectif justifie l'intérêt actuel pour les modèles qualitatifs de préférences et les représentations compactes comme les CP-Nets (Boutilier *et al.*, 2004a). Ces modèles sont délibérément simples pour être intégrés efficacement dans les systèmes interactifs de recommandation ; les préférences des agents doivent être capturées en ne posant que quelques questions, afin de trouver rapidement parmi les articles disponibles ceux qui sont les meilleurs selon les préférences de l'utilisateur. Dans d'autres applications (par exemple les problèmes de configuration, les problèmes de répartition équitable de ressources, les enchères combinatoires (Engel et Wellman, 2007)), on peut consacrer davantage de temps à l'étape d'élicitation afin d'obtenir une description plus affinée des préférences. Dans ce cadre, les fonctions d'utilité peuvent être plus adaptées que les modèles qualitatifs grâce à leur plus grande puissance descriptive. Par exemple, des utilités sont en général mieux adaptées pour représenter des préférences cardinales (quand le décideur exprime différentes intensités de préférences pour les alternatives).

Les fonctions d'utilité constituent un outil puissant pour modéliser les préférences. Elles fournissent une mesure de la valeur des différentes alternatives et peuvent capturer l'attitude du décideur vis-à-vis du risque et de l'incertitude. Toutefois, pour que la quantité d'information nécessaire pour construire la fonction d'utilité soit raisonnablement limitée, il est nécessaire que cette fonction ait une certaine structure. Différentes structures décrites en termes d'indépendance ont été explorées dans la théorie de l'utilité multi-attribut. Le modèle le plus utilisé suppose une indépendance entre attributs qui assure la représentabilité des préférences par une utilité additive. Une telle décomposabilité rend le processus d'élicitation très simple et rapide. Cependant, dans la pratique, cette indépendance n'est pas nécessairement vérifiée puisqu'elle élimine toute possibilité d'interaction entre les attributs. Pour augmenter la puissance descriptive des utilités additives, les **décompositions GAI** (Indépendance Additive Généralisée, de l'anglais *Generalized Additive Independence*) ont été introduites (Fishburn, 1967; Bacchus et Grove, 1995). Ces décompositions GAI permettent des interactions plus générales entre les attributs tout en préservant une certaine décomposabilité du modèle.

Pour représenter de manière efficace des modèles GAI, Gonzales et Perny (2004) ont introduit les **réseaux GAI** (GAI-Nets), structures graphiques similaires aux graphes de jonction utilisés pour les réseaux bayésiens (voir par exemple Jensen (1996)). Dans cette thèse, nous entendons mener une étude concernant l'utilisation des GAI-Nets pour atteindre des fonctionnalités typiques des systèmes de recommandation dans le cadre où l'espace d'alternatives a une structure combinatoire. Nous pensons que de tels modèles

graphiques peuvent aider à combler le fossé entre les méthodes classiques développées en la théorie de la décision, et leur utilisation dans des systèmes de recommandation pratiques, besoin ressenti par les développeurs de ces derniers (voir par exemple Ricci et del Missier (2004)).

Nos problématiques clés sont le choix et le rangement. Un système de recommandation doit être capable de non seulement trouver la meilleur alternative, mais aussi de lister les k-meilleures alternatives pour un utilisateur donné. Dans le cadre des espaces combinatoires, cela doit être accompli sans avoir besoin d'énumérer les alternatives.

Nous étudions également les possibilités d'éliciter les GAI-Nets de manière à se focaliser sur l'obtention des k-meilleures alternatives et le problème de recherche de solutions de compromis dans le cadre de la décision collective/multi-critère dans des espaces combinatoires.

Organisation de la thèse

Dans le premier chapitre, nous introduisons les concepts de base qui seront utilisés par la suite. Nous proposons également un tour d'horizon des travaux récents en modélisation/représentation de préférences, notamment les développements récents de travaux dans le domaine des préférences qualitatives, comme les CP-Nets et leurs extensions. Finalement nous introduisons les GAI-Nets et soulignons les avantages et désavantages de l'approche quantitative avec des réseaux GAI par rapport aux approches qualitatives.

Ensuite, dans le deuxième chapitre, nous nous focalisons sur la décision individuelle avec des GAI-Nets. Nous présentons une méthode pour obtenir la configuration d'utilité maximale fondée sur un algorithme du type « élimination de variables ». Nous étendons cet algorithme pour qu'il puisse être utilisé pour calculer les k-meilleures alternatives même pour un k de grande taille. Grâce à cette efficacité, nous vérifions que cet algorithme peut être appliqué dans une méthode du type *branch-and-bound* pour résoudre des instances difficiles du problème du sac-à-dos 0-1 non-décomposable.

Le troisième chapitre concerne la décision collective/multi-critère. Nous fournissons une méthode pour la détermination exacte d'une solution de meilleur compromis selon des critères non-linéaires pour un groupe d'individus (ou un ensemble de critères) avec des préférences modélisées par des décompositions GAI.

Le chapitre 4 est dédié à l'élicitation de préférences. Nous présentons les procédures pour éliciter des modèles GAI proposées dans la littérature, et proposons une méthode adaptative pour éliciter partiellement le modèle GAI en se focalisant sur les k-meilleures alternatives avec l'aide des GAI-Nets.

Le chapitre 5, enfin, développe une application Web fournissant des recommandations

de lieux à visiter dans une ville. Cette application est utilisée comme preuve de concept pour expérimenter les méthodes présentées dans cette thèse, dans un cadre plus concret, avec des données réelles.

Dans chaque chapitre, les algorithmes proposés sont testés sur des jeux de données adaptés.

Chapitre 1

Modélisation et Représentation de Préférences

« Mieux vaut de l'herbe pour nourriture, là où règne l'amour,
Qu'un bœuf engraissé, si la haine est là. »

La Bible, livre des Proverbes, chapitre 15, verset 17
(environ X^{e} siècle av. J.-C., d'après la tradition)

Résumé

Ce premier chapitre est consacré, d'une part, à présenter des concepts de base sur lesquels se développent les méthodes en théorie de la décision (section 1.2) et, d'autre part, à introduire des modèles graphiques de représentation de préférences qui se sont développés ces dernières années, notamment les CP-Nets et leurs extensions (section 1.3). À la fin du chapitre (section 1.3.5), nous présentons les GAI-Nets, qui seront particulièrement exploités tout au long de cette thèse. Nous terminons en soulignant la complémentarité des différents modèles graphiques de représentation des préférences, ce qui nous montre qu'il n'y a pas un seul modèle qui soit le plus adapté dans toutes les situations, justifiant ainsi l'intérêt dans leur diversité (section 1.3.6).

1.1 Notation

Commençons par introduire les notations classiques en théorie de la décision, qui seront utilisées tout au long de cette thèse :

- Par abus de notation, les lettres majuscules, comme A, B, X_1, représentent aussi bien des attributs (variables) que leurs domaines.
- Les minuscules, comme a, b', x_1, représentent des valeurs d'attributs.
- Les attributs indicés X_i sont caractérisés par leur indice. Leurs valeurs sont également identifiées par cet indice. Ainsi, x_i, y_i, z_i^3 représentent des valeurs de X_i.
- Les attributs non indicés, A, B, sont caractérisés par leur lettre. Leurs valeurs sont également identifiées par cette lettre. Ainsi a, a', a^3 sont des valeurs pour l'attribut A.
- Par abus de notation, si $I = \{i_1, \ldots, i_k\}$ est un sous-ensemble de $\{1, \ldots, n\}$, a_I représentera le k-uplet $(a_{i_1}, \ldots, a_{i_k}) \in A_{i_1} \times \cdots \times A_{i_k}$. De même, si a et a' sont deux éléments de $\mathcal{A}$, alors $z = (a_{-I}, a'_I)$ représentera le n-uplet $(z_1, \ldots, z_n)$ tel que $z_i = a'_i$ pour $i \in I$ et $z_i = a_i$ pour $i \notin I$.

1.2 Concepts de base sur des préférences

Pour prendre une décision bien informée il est nécessaire que le décideur soit capable d'évaluer et de comparer les différentes alternatives (ou **actes**) disponibles afin d'identifier celles qui leur sont *préférées*.

1.2.1 Relations de préférence

Soit $\mathcal{A}$ l'ensemble d'actes disponibles. Les préférences du décideur sur les éléments de $\mathcal{A}$ peuvent être décrites par une **relation binaire** sur $\mathcal{A}$, c'est-à-dire un sous-ensemble du produit cartésien $\mathcal{A} \times \mathcal{A}$. Cette relation est nommée une **relation de préférence** et nous la notons par $\succsim$. Ainsi, pour un ensemble d'alternatives $\mathcal{A}$ et des éléments $a^1, a^2 \in \mathcal{A}$, $a^1 \succsim a^2$ signifie que le décideur *préfère* a^1 à a^2 ou bien qu'il est *indifférent* entre a^1 et a^2 (on écrit $a^1 \succsim a^2$ comme sucre syntaxique pour $(a^1, a^2) \in \succsim$). Nous définissons aussi certains opérateurs additionnels à partir de la relation $\succsim$:

- **préférence stricte :** $a^1 \succ a^2 \Leftrightarrow (a^1 \succsim a^2) \wedge \neg(a^2 \succsim a^1)$
- **indifférence :** $a^1 \sim a^2 \Leftrightarrow (a^1 \succsim a^2) \wedge (a^2 \succsim a^1)$

Rappelons quelques définitions sur les relations binaires.

Définition 1.1. Une relation binaire $\succsim$ sur un ensemble $\mathcal{A}$ est dite :

- **réflexive** $\Leftrightarrow \forall a^1 \in \mathcal{A}, \quad a^1 \succsim a^1$;
- **irréflexive** $\Leftrightarrow \forall a^1 \in \mathcal{A}, \quad \neg(a^1 \succsim a^1)$;

- **transitive** $\Leftrightarrow \forall a^1,\, a^2,\, a^3 \in \mathcal{A}, \quad (a^1 \succsim a^2) \wedge (a^2 \succsim a^3) \Rightarrow a^1 \succsim a^3$;
- **complète** $\Leftrightarrow \forall a^1,\, a^2 \in \mathcal{A}, \quad a^1 \succsim a^2 \vee a^2 \succsim a^1$;
- **symétrique** $\Leftrightarrow \forall a^1,\, a^2 \in \mathcal{A}, \quad a^1 \succsim a^2 \Rightarrow a^2 \succsim a^1$;
- **asymétrique** $\Leftrightarrow \forall a^1,\, a^2 \in \mathcal{A}, \quad a^1 \succsim a^2 \Rightarrow \neg(a^2 \succsim a^1)$.

Définition 1.2. Un **préordre** est une relation binaire réflexive et transitive.

Définition 1.3. Un **ordre** est une relation binaire irréflexive et transitive (par conséquent elle est aussi asymétrique).

Définition 1.4. Une **relation d'équivalence** est une relation binaire réflexive, symétrique et transitive.

Si la relation $\succsim$ du décideur est réflexive et transitive, et que l'on définit $\succ$ et $\sim$ comme ci-dessus, ses préférences respectent les **hypothèses fortes de rationalité du décideur**.

Définition 1.5. (Barba-Romero et Pomerol, 1997, page 33) Des préférences respectent les hypothèses fortes de rationalité du décideur, si et seulement si (ssi) :

- l'intersection entre $\sim$ et $\succ$ est vide ;
- $\sim$ est réflexive et symétrique ;
- $\succ$ est asymétrique ;
- $\succsim$ est transitive.

Sous les hypothèses fortes de rationalité du décideur, $\succsim$ est un préordre, $\sim$ est une relation d'équivalence et $\succ$ est un ordre.

Au premier regard, ces hypothèses de rationalité peuvent paraître très naturelles. Toutefois elles ne sont pas toujours respectées dans la réalité. Par exemple, l'indifférence n'est pas toujours transitive.

Exemple 1.1. (Luce, 1956) Trouvez un décideur qui préfère une tasse de café avec un morceau de sucre à une tasse de café avec cinq morceaux de sucre. Maintenant préparez 401 tasses de café avec $\left(1 + \frac{i}{100}\right) x$ grammes de sucre, $i = 0, \ldots, 400$, où x est le poids d'un morceau de sucre. Il est évident qu'il sera indifférent entre la tasse i et la tasse $i+1$, pour tout i, mais par choix il n'est pas indifférent entre $i = 0$ et $i = 400$.

De tels effets sont notamment observés en raison de l'imprécision de nos organes sensoriels ou des instruments de mesure. Dans ces cas, $\succsim$ n'est plus transitive et donc elle n'est point un préordre. La manière la plus simple d'éviter le syndrome de la tasse de café est d'utiliser l'hypothèse que l'indifférence n'est pas transitive. Ainsi, nous arrivons aux **hypothèses faibles de rationalité du décideur**.

Définition 1.6. (Barba-Romero et Pomerol, 1997, page 36) Des préférences respectent les hypothèses faibles de rationalité du décideur, si et seulement si (ssi) :

- l'intersection entre $\sim$ et $\succ$ est vide ;
- $\sim$ est réflexive et symétrique ;
- $\succ$ est asymétrique et transitive ;

Une structure de préférences qui vérifie de telles hypothèses est nommée un **quasi-ordre**. Toutefois, les hypothèses fortes de rationalité du décideur nous permettent de manipuler plus facilement ses préférences en utilisant un modèle bien adapté aux préordres : une **fonction d'utilité**.

1.2.2 Fonctions d'utilité

Pour comparer des alternatives dans un problème décisionnel, une approche classique consiste à les mesurer selon une échelle de valuation (c'est-à-dire un ensemble ordonné). Étant donnée une échelle de valuation $\mathcal{E}$, on associe à chaque alternative $a \in \mathcal{A}$, une valeur $u(A) \in \mathcal{E}$ de façon à refléter l'appréciation de a du point de vue du décideur. On appelle cette valeur l'**utilité** de a. Ainsi, on transpose le problème de comparer deux alternatives $a^1, a^2 \in \mathcal{A}$ dans un problème où on compare les valeurs $u(a^1), u(a^2) \in \mathcal{E}$. Un ensemble ordonné adapté à cette utilisation est l'ensemble de nombres réels, $\mathbb{R}$. Rappelons-nous que $\mathbb{R}$ possède des préordres bien connus, à savoir, $\leq$ et $\geq$; ainsi que la relation d'équivalence $=$, et les ordres $>$ et $<$. Une **fonction d'utilité** $u : \mathcal{A} \mapsto \mathbb{R}$ associe une valeur d'utilité réelle à chaque action.

Définition 1.7. Une fonction d'utilité $u : \mathcal{A} \mapsto \mathbb{R}$ modélise $\succsim$, ssi

$$\forall a^1, a^2 \in \mathcal{A}, \quad u(a^1) \geq u(a^2) \Leftrightarrow a^1 \succsim a^2 \tag{1.1}$$

Théorème 1.1. *(Fishburn, 1970) Soit $\mathcal{A}$ un ensemble fini ou dénombrable, et soit $\succsim$ une relation binaire sur $\mathcal{A}$. Les deux affirmations suivantes sont équivalentes :*

1. *$\succsim$ est un préordre complet ;*
2. *$\exists u$, $u : \mathcal{A} \mapsto \mathbb{R}$, telle que u modélise $\succsim$.*

En plus, si u est une fonction d'utilité qui modélise un préordre complet $\succsim$, une condition suffisante et nécessaire pour qu'une fonction $v : \mathcal{A} \mapsto \mathbb{R}$ modélise aussi $\succsim$ est qu'il existe une application t strictement croissante de $u(\cdot)$ dans $\mathbb{R}$ telle que $v = t \circ u$. De ce fait, à partir d'une fonction d'utilité nous pouvons obtenir une infinité d'autres fonctions, à une transformation strictement croissante près. Ici, les fonctions d'utilité indiquent simplement l'ordre des alternatives, il s'agit donc de **fonctions d'utilité exprimées**

sur une échelle ordinale (cf. Krantz *et al.* (1971)). Considérons maintenant l'exemple suivant :

Exemple 1.2. Un décideur reçoit les dossiers de 5 candidats à 3 postes disponibles pour la même fonction dans une entreprise. Les dossiers ont été préalablement évalués et notés dans l'intervalle $[0, 10]$. Les notes obtenues par les candidats a, b, c, d, e ont été :

- $a = 10$, $b = 9$, $c = 2$, $d = 1$, $e = 1$.

Si nous considérons que ces notes correspondent simplement à la modélisation d'un préordre par une fonction d'utilité u, toute information qui peut être déduite est que $a \succ b \succ c \succ d \sim e$. Ce préordre serait aussi bien modélisé par une fonction v, telle que : $v(a) = 1000$, $v(b) = 999$, $v(c) = 998$, $v(d) = 997$, $v(e) = 997$. Ainsi, le décideur devrait choisir les candidats mieux placés : a, b et c. Néanmoins, en formulant ces notes, le décideur pourrait avoir voulu signifier que a et b sont bien meilleurs que c, d et e, ces derniers ayant des dossiers assez médiocres. Dans ce cas, le décideur préférera laisser un poste vacant et prendre seulement a et b, que de pouvoir les 3 postes vacants.

Définition 1.8. On dit qu'un décideur exprime des préférences modélisables sur une échelle (cardinale) d'intervalle quand on peut comparer les différences de préférence. C'est-à-dire, si nous savons que $a^1 \succ a^2$ et $a^3 \succ a^4$, nous pouvons dire si la différence entre a^1 et a^2 est plus grande, plus petite, ou égale à la différence entre a^3 et a^4. Pour modéliser de telles préférences, une fonction d'utilité u doit être telle que $u(a^1) - u(a^2) > u(a^3) - u(a^4)$ ssi, a^1 est « plus préférée » à a^2 que a^3 à a^4. Ainsi, le quotient $\dfrac{u(a^1) - u(a^2)}{u(a^3) - u(a^4)}$ doit être un invariant dans la famille des fonctions d'utilité qui modélisent ces préférences. Nous pouvons donc conclure que si des préférences sont modélisées par une fonction d'utilité u, elles seront aussi bien modélisées par toute fonction v telle que $v(\cdot) = u(\cdot)\alpha + \beta,\ \alpha \in \mathbb{R}^+, \beta \in \mathbb{R}$ (soit, les transformations affines strictement positives de u). Il s'agit donc de **fonctions d'utilité exprimées sur une échelle d'intervalle**.

Exemple 1.3. La météo prévoit pour demain à Paris une température minimale de 9°C et une température maximale de 18°C. Pour après-demain, la minimale prévue est de 8°C et la maximale de 17°C. Notons que nous avons la même différence entre la température maximale et la minimale prévues dans les deux jours, 9°C. La température en degrés Celsius t_C utilise une échelle cardinale d'intervalle, ainsi toute autre échelle de température doit constater une même différence entre les températures minimales et maximales pour les deux prochains jours. Effectivement, on peut par exemple passer à des températures en degrés Fahrenheit t_F en utilisant la formule $t_F = \frac{9}{5}t_C + 32$. Ainsi, les températures prévues seront de $48,2$°F min. et $64,4$°F max. pour demain et de $46,4$°F min. et $62,6$°F max. pour après-demain. Nous constatons donc qu'en degrés Fahrenheit nous avons aussi dans les deux jours qui viennent la même différence entre

les températures maximales et minimales, $16,2°$F. Remarquons que nous ne pouvons pas dire que demain la température maximale sera deux fois plus chaude que la température minimale, le chiffre deux fois plus grand étant particulier à l'échelle Celsius utilisée. En effet, le 0°C n'indique pas « l'absence de température » et correspond à 32°F.

Au contraire, si l'on considère la masse d'un objet en kilogrammes, la valeur 0 indique vraiment l'absence de la propriété masse. Ainsi, toutes les échelles de masse ont le même 0 et nous pouvons donc dire qu'un objet de 12 kilogrammes est deux fois plus lourd qu'un objet de 6 kilogrammes.

Définition 1.9. On dit qu'un décideur exprime des préférences modélisables sur une échelle (cardinale) de ratio si les deux affirmations suivantes sont vérifiées :

1. on peut comparer les différences de préférence. Ainsi, quand nous avons $a^1 \succ a^2$ et $a^3 \succ a^4$, nous pouvons dire si la différence entre a^1 et a^2 est plus grande, plus petite, ou égale à la différence entre a^3 et a^4 ;
2. il existe un élement $a \in \mathcal{A}$ qui représente un « zéro absolu ». Ainsi, pour toute fonction u qui modélise ces préférences, $u(a) = 0$.

Pour toute fonction u qui modélise les mêmes préférences d'un décideur sur un ensemble $\mathcal{A}$:

1. $\forall a^1, a^2, a^3, a^4 \in \mathcal{A}$, le quotient $\dfrac{u(a^1) - u(a^2)}{u(a^3) - u(a^4)}$ est invariant ;
2. $\forall a^1, a^2 \in \mathcal{A}$, le quotient $\dfrac{u(a^1)}{u(a^2)}$ est invariant.

Ainsi, les fonctions d'utilité qui modélisent ces préférences sont invariantes face à des transformations linéaires positives. C'est-à-dire, si u et v modélisent les mêmes préférences, $v(\cdot) = u(\cdot)\alpha,\ \alpha \in \mathbb{R}^+$. Il s'agit donc de **fonctions d'utilité exprimées sur une échelle de ratio**.

On constate que les fonctions d'utilité possèdent une grande puissance descriptive, on peut les utiliser non seulement pour indiquer l'ordre de préférence entre les alternatives, mais aussi pour comparer les différences de préférence ou des niveaux absolus d'utilité. Les trois types d'échelle ici présentées (ordinale, d'intervalle et de ratio) permettent l'expression de ces informations à travers des valeurs d'utilité. Néanmoins, il est important de souligner que, même si ces échelles sont adaptées à un grand nombre de problèmes pratiques, dans certaines situations nous sommes menés à utiliser une échelle non-classique, souvent construite à partir d'une analyse empirique du problème en question (cf. Bouyssou *et al.* (2006, chapitre 3)). On doit être toujours attentif à la fiabilité des informations utilisées pour construire la fonction d'utilité, surtout quand celle-ci est cardinale, car les moindres modifications par le décideur peuvent mener à des changements des préférences. Le décideur doit être conscient des situations où l'on suppose qu'il exprime

des cardinalités, et non simplement des ordres. Comme le souligne Barba-Romero et Pomerol (1997, page 53) :

> « Il faut être d'autant plus vigilant, dans la pratique, *qu'il ne sera jamais difficile d'obtenir des évaluations numériques de la part du décideur*. Il semble que l'être humain évalue spontanément les choses ! Cette facilité ne doit pas nous induire en erreur : il est bien plus difficile d'obtenir des évaluations robustes, qui résistent aux changements d'échelle et de contexte. Dans tous les cas, l'analyste devra faire preuve d'un grand esprit critique et surtout, contrôler la robustesse des méthodes en fonction du type d'utilité employée. »*

Comme nous avons pu constater dans l'exemple 1.1 (page 10), il y a des cas où l'on ne peut pas compter sur la transitivité de l'indifférence. Ainsi, la relation de préférence ne sera plus un préordre. Toutefois, si les hypothèses faibles de rationalité du décideur sont vérifiées et si nous avons un quasi-ordre complet, il est toujours possible de modéliser les préférences avec des fonctions d'utilité. Pour cela, nous utilisons la notion d'un **seuil d'indifférence**.

Définition 1.10. Soit $\xi \in \mathbb{R}^+$ le seuil d'indifférence. Une fonction d'utilité $u : \mathcal{A} \mapsto \mathbb{R}$ modélise un quasi-ordre quand :

$$\forall a^1, a^2 \in \mathcal{A} \begin{cases} a^1 \succ a^2 \Leftrightarrow u(a^1) > u(a^2) + \xi \\ a^1 \sim a^2 \Leftrightarrow |u(a^1) - u(a^2)| \leq \xi \end{cases} \tag{1.2}$$

1.2.3 Attributs et critères

Nous nous intéressons à des problèmes où chaque élément dans l'ensemble d'alternatives $\mathcal{A}$ peut être décrit par un ensemble de caractéristiques, ou **attributs**. Supposez un ensemble d'attributs $A_1, A_2, \ldots, A_n$. Chaque attribut a un ensemble fini ou infini de valeurs possibles. Ainsi, l'ensemble d'alternatives $\mathcal{A} = A_1 \times A_2 \times \ldots \times A_n$ a une structure multidimensionnelle, il est le produit cartésien des valeurs des attributs. Quand ces attributs codent les informations préférentielles du décideur, ils sont appelés **critères**. Les critères sont les axes d'évaluation des alternatives, c'est-à-dire qu'ils expriment la satisfaction du décideur en fonction de différents aspects.

Exemple 1.4. † Une entreprise décide d'embaucher un nouvel employé. Neuf candidats se présentent et constituent donc l'ensemble d'alternatives $\mathcal{A}$ du problème. Le département

*dans l'original : « Es preciso ser tanto más vigilante, en la práctica, *cuanto que nunca será difícil obtener evaluaciones numéricas por parte del decisor*. Parece que el ser humano sea espontáneamente evaluador!. Esta facilidad no debe engañarnos: es mucho más difícil obtener evaluaciones robustas, que resistan a los cambios de escala y de contexto. En todos los casos el analista deberá hacer prueba de un gran espíritu crítico y, sobre todo, contrastar la robustez de los métodos en función del tipo de utilidad empleada. »

†adapté de Barba-Romero et Pomerol (1997, page 85)

de Ressources Humaines décide de les caractériser en utilisant 6 différents attributs, à savoir : nombre d'années d'étude après le bac ; nombre d'années d'expérience professionnelle dans le domaine ; âge du candidat ; note dans l'entretien personnel ; note dans le test psychotechnique ; niveau d'anglais. L'entreprise veut choisir le meilleur candidat, en partant du principe que les critères d'évaluation sont : pour le niveau d'anglais, fluent $\succ$ moyen $\succ$ mauvais ; la minimisation de l'âge ; la maximisation des autres attributs. Le tableau 1.1 liste les candidats et leur caractéristiques respectives selon les attributs considérés.

TAB. 1.1 – Les candidats, caractérisés par des différents critères : C_1, nombre d'années d'études après le bac ; C_2, nombre d'années d'expérience professionnelle ; C_3, âge ; C_4, note dans l'entretien personnel ; C_5, note dans le test psychotechnique ; C_6, niveau d'anglais.

Candidat		C_1 MAX	C_2 MAX	C_3 MIN	C_4 MAX	C_5 MAX	C_6
1	Alberto	6	5	28	5	5	moyen
2	Blanca	4	2	25	10	9	moyen
3	Carlos	7	7	38	5	10	fluent
4	Daniel	5	7	35	9	6	moyen
5	Emilia	6	1	27	6	7	moyen
6	*Félix*	*5*	*7*	*31*	*7*	*8*	*moyen*
7	Germán	6	8	30	7	9	fluent
8	Hilario	5	6	26	4	8	moyen
9	Irene	3	8	34	8	7	mauvais

Nous notons que dans cet exemple, le candidat n° 6 (Félix), listé en italique dans le tableau 1.1, est battu par le candidat n° 7 (Germán) sur tous les critères d'évaluation. On dit que l'alternative n° 7 domine l'alternative n° 6, ou que l'alternative n° 6 n'est pas **Pareto-optimale**.

Définition 1.11. Soit $\mathcal{C} = \{C_1, C_2, \ldots, C_n\}$ l'ensemble de critères d'évaluation. Une alternative a^1 Pareto-domine (ou simplement domine) une autre alternative a^2 ssi, sur chacun des critères, la première alternative est préférée à la seconde, c'est-à-dire,

$$\forall i = \{1, \ldots, n\}, \quad c_i^1 \succsim c_i^2.$$

Une alternative qui n'est pas Pareto-dominée par une autre est dite Pareto-optimale.

Dans l'exemple 1.4, seulement l'alternative n° 6 n'est pas Pareto-optimale. Ainsi, dans le problème de choix du meilleur candidat, Félix serait le seul qui pourrait être tout de suite éliminé, sous prétexte qu'un autre candidat est supérieur à lui. Pour le choix parmi les autres candidats, un décideur rationnel peut trouver une bonne raison pour

choisir n'importe lequel d'entre eux, en fonction de son point de vue. Si au contraire, l'entreprise, au lieu de vouloir minimiser l'âge de l'employé, considérait plutôt que le candidat idéal doit avoir 30 ans, l'attribut « âge du candidat en années » ne serait plus un critère à minimiser. Dans ce cas, on pourrait adopter à la place de C_3 un nouveau critère à minimiser $C_3{}'$, tel que $c_3{}' = |c_3 - 30|$. Le tableau 1.2 liste les candidats selon les nouveaux critères. Notons que maintenant les candidats n° 1 (Alberto), n° 5 (Emilia), n° 6 (Félix) et n° 8 (Hilario) sont dominés.

TAB. 1.2 – Les candidats de exemple 1.4, en remplaçant C_3 par $C_3{}'$

Candidat		C_1 MAX	C_2 MAX	$C_3{}'$ MIN	C_4 MAX	C_5 MAX	C_6
1	*Alberto*	*6*	*5*	*2*	*5*	*5*	*moyen*
2	Blanca	4	2	5	10	9	moyen
3	Carlos	7	7	8	5	10	fluent
4	Daniel	5	7	5	9	6	moyen
5	*Emilia*	*6*	*1*	*3*	*6*	*7*	*moyen*
6	*Félix*	*5*	*7*	*1*	*7*	*8*	*moyen*
7	Germán	6	8	0	7	9	fluent
8	*Hilario*	*5*	*6*	*4*	*4*	*8*	*moyen*
9	Irene	3	8	4	8	7	mauvais

Dans le cas le plus général, nous n'avons pas forcément une bijection entre l'ensemble d'attributs et l'ensemble de critères. En plus, les critères peuvent être interdépendants et conflictuels, c'est-à-dire que l'optimisation d'un critère peut causer la dégradation d'autres critères.

Exemple 1.5. Considérons le problème de choix de la configuration d'une voiture, caractérisée par les attributs $\mathcal{A}$ = {couleur, nombre de portes, décapotable, direction assistée, freinage ABS, type de carburant, embrayage automatique}, selon 3 critères : *esthétique* (C_1), *confort* (C_2), et *sécurité* (C_3). Chacun des critères est modélisé à travers une fonction d'utilité sur un sous-ensemble d'attributs, comme suit : u_{C_1}(couleur, nombre de portes, décapotable), u_{C_2}(nombre de portes, direction assistée, type de carburant, embrayage automatique), et u_{C_3}(direction assistée, freinage ABS, décapotable). Notons que l'attribut « nombre de portes » influe à la fois dans l'esthétique et le confort, l'attribut booléen « décapotable » intervient à la fois dans l'esthétique et la sécurité, et l'attribut « direction assistée » apparaît à la fois dans le confort et la sécurité. Or il est possible que pour un certain modèle de voiture le décideur trouve, par exemple, que la version avec trois portes soit plus belle que la version avec cinq portes, mais que la dernière apporte plus de confort. Ainsi, dans cette situation, passer de trois portes à cinq portes créé une amélioration dans le critère confort et en même temps une dégradation dans le critère esthétique.

Ces exemples nous indiquent la difficulté d'agréger les performances en une relation de préférence globale sur les alternatives, car les performances, portées par des critères multiples, sont souvent difficiles à comparer et conflictuelles. Grabisch et Perny (2003) distinguent deux types d'approches concurrentes utilisées dans des méthodes classiques d'aide à la décision pour mener à bien l'agrégation des performances sur des critères multiples : l'approche **agréger puis comparer** (AC) et l'approche **comparer puis agréger** (CA).

Soit $g_j : \mathcal{A} \mapsto \mathbb{R}, j \in \{1, \ldots, n\}$ les n fonctions critères du problème. Pour toute alternative $a \in \mathcal{A}$, pour toute fonction critère g_j, a_j note la performance $g_j(a)$. Supposons que pour tout $a^1, a^2 \in \mathcal{A}$ la relation de préférence $\succsim$ est définie comme :

$$\succsim (a^1, a^2) = f(a_1^1, \ldots, a_n^1, a_1^2, \ldots, a_n^2)$$

Si nous adoptons la logique classique, f est une fonction $f : \mathbb{R}^{2n} \mapsto \{0, 1\}$, où $f(a^1, a^2) = 1$ signifie que a^1 est préférée ou indifférente à a^2.

En général, pour comparer les deux alternatives, la fonction f réalise deux étapes bien distinctes : une étape d'**agrégation** en utilisant une fonction d'agrégation multi-critère $\psi : \mathbb{R}^n \mapsto \mathbb{R}$ qui synthétise n valeurs en une seule ; et une étape de **comparaison** en utilisant des fonctions de comparaison de performances $\phi : \mathbb{R}^2 \mapsto \mathbb{R}$, ce qui permet de comparer deux performances données.

Dans l'approche « agréger puis comparer » nous décomposons $f(a_1^1, \ldots, a_n^1, a_1^2, \ldots, a_n^2)$ en $\phi(\psi(a_1^1, \ldots, a_n^1), \psi(a_1^2, \ldots, a_n^2))$, tandis que dans l'approche « comparer puis agréger » nous décomposons $f(a_1^1, \ldots, a_n^1, a_1^2, \ldots, a_n^2)$ en $\psi(\phi_1(a_1^1, a_1^2), \ldots, \phi_n(a_n^1, a_n^2))$. La figure 1.1 schématise les deux approches.

$$\begin{array}{ccc}
(a_1^1, \ldots, a_n^1),\ (a_1^2, \ldots, a_n^2) & \xrightarrow{\ \psi\ } & u(a^1),\ u(a^2) \\
\Big\downarrow \phi_j & & \Big\downarrow \phi \\
\phi_1(a_1^1, a_1^2), \ldots, \phi_n(a_n^1, a_n^2) & \xrightarrow[\ \psi\ ]{} & \succsim (a^1, a^2)
\end{array}$$

FIG. 1.1 – Les approches AC et CA (Grabisch et Perny, 2003)

L'approche AC est normalement fondée sur la construction d'une fonction d'utilité globale pour ψ. Comme nous l'avons mentionné précédemment, cela suppose que les préférences soient des préordres (ou quasi-ordres dans le cas d'utilisation d'un seuil d'indifférence) complets. Dans le cas multi-critère nous avons des difficultés additionnelles apportées par la nécessité de construire une évaluation globale à partir d'évaluations

partielles qui peuvent être exprimées sur des échelles différentes (échelle monétaire, temps, vitesse, notes etc.) et difficiles à comparer. Toutefois, si nous réussissons à avoir une fonction ψ adaptée, la comparaison d'utilités (ϕ) est très simple. L'étude des conditions sous lesquelles il est possible de concevoir une évaluation globale numérique pour une relation de préférences sur des espaces multi-attributs, est l'objectif de la théorie du mesurage conjoint (voir par exemple Bouyssou et Pirlot (2005) pour une introduction).

L'approche CA compare la performance de deux alternatives sur le même critère à chaque fois. Ainsi, dans ce cas il n'est pas nécessaire de faire des comparaisons intercritères, comme dans l'approche AC. Toutefois, une fois que nous avons un vecteur de résultats de comparaisons critère-à-critère, nous débouchons sur un problème difficile d'agrégation ordinale qui cherche à trouver la relation de préférence globale $\succsim$ qui synthétise au mieux le profil de relations $(\succsim_1, \ldots, \succsim_n)$.

Ainsi, le choix du type d'approche d'agrégation à utiliser est encore dépendant du type de problème et des informations disponibles.

1.2.4 Préférences décomposables

Lorsque la taille de $\mathcal{A}$ devient trop grande, il est impossible d'exprimer la relation de préférence en extension, le nombre d'alternatives possibles étant très élevé. Par exemple, avec 20 attributs d'une taille de domaine 5 chacun, nous avons 5^{20} configurations possibles (un nombre supérieur à 10^{13}). Ainsi, si les préférences dans cet ensemble sont modélisées par une fonction d'utilité, on ne peut pas représenter cette dernière par un tableau avec une valeur d'utilité pour chaque configuration possible.

Si les préférences sur les éléments de $\mathcal{A}$ ont une certaine structure (c'est-à-dire, que l'on peut observer des indépendances parmi les attributs), la relation de préférence sur $\mathcal{A}$ peut être définie sur des sous-ensembles d'attributs et représentée de manière plus compacte.

Soit $\mathcal{A} = \times_{i=1}^{n} A_i$ l'ensemble d'alternatives et $W = \{w_1, \ldots, w_k\}$ un sous-ensemble de $\{1, \ldots, n\}$. On dit que le sous-ensemble d'attributs A_W est **préférentiellement indépendant** de son complément A_{-W} (on écrit $PI(A_W, A_{-W})$) ssi, pour tout a_W, a_{-W}, a'_W, a'_{-W}, on a

$$(a_W, a_{-W}) \succeq (a'_W, a_{-W}) \Leftrightarrow (a_W, a'_{-W}) \succeq (a'_W, a'_{-W}) \tag{1.3}$$

Cela signifie que les préférences sur les valeurs des attributs indicés par W restent les mêmes une fois que l'on fixe les valeurs des autres attributs, peu importe sur lesquelles valeurs ces dernières ont été fixées. Dans ce cas on dit que a_W est préféré à a'_W ***ceteris paribus*** (toutes choses égales par ailleurs). Il convient de souligner que l'indépendance

préférentielle n'est pas toujours bidirectionnelle, c'est-à-dire que nous pouvons avoir un sous-ensemble d'attributs A_W qui est préférentiellement indépendant de son complément A_{-W}, au même temps que A_{-W} n'est pas préférentiellement indépendant de A_W.

Exemple 1.6. Supposons qu'un décideur évalue des voitures en fonction de deux attributs : le type de transmission (A_1) et la marque (A_2). Soit $A_1 = \{\text{automatique, manuelle}\}$ et $A_2 = \{\text{Citrone, Audible}\}$. Le décideur a exprimé les préférences suivantes :

$$(\text{Audible}, \text{automatique}) \succeq (\text{Audible}, \text{manuelle})$$
$$\succeq (\text{Citrone}, \text{manuelle}) \succeq (\text{Citrone}, \text{automatique})$$

Dans ce cas, $A_W = \{A_2\}$ est préférentiellement indépendant de $A_{-W} = \{A_1\}$, une fois que nous avons (Audible, automatique) $\succeq$ (Citrone, automatique) et aussi (Audible, manuelle) $\succeq$ (Citrone, manuelle). Toutefois, $\{A_1\}$ n'est pas préférentiellement indépendant de $\{A_2\}$, puisque nous avons (Audible, automatique) $\succeq$ (Audible, manuelle) mais (Citrone, manuelle) $\succeq$ (Citrone, automatique).

De manière analogue à l'indépendance préférentielle, on définit l'**indépendance conditionnelle**. Soit $\mathcal{A} = \times_{i=1}^{n} A_i$ l'ensemble d'alternatives et W, Y et Z trois ensembles non-nuls qui partitionnent $\{1, \ldots, n\}$ (c'est-à-dire que $W \cup Y \cup Z = \{1, \ldots, n\}, W \cap Y = W \cap Z = Y \cap Z = \emptyset$). Le sous-ensemble d'attributs A_W est conditionnellement indépendant du sous-ensemble d'attributs A_Y étant donné A_Z (on écrit $CPI(A_W, A_Z, A_Y)$) ssi, pour tout $a_W, a'_W, a_Y, a'_Y, a_Z$, on a

$$(a_W, a_Z, a_Y) \succeq (a'_W, a_Z, a_Y) \Leftrightarrow (a_W, a_Z, a'_Y) \succeq (a'_W, a_Z, a'_Y) \tag{1.4}$$

Cela signifie que les préférences sur les valeurs des attributs indicés par W sont préférentiellement indépendants des valeurs des attributs indicés par Y une fois que l'on a affecté une valeur pour les attributs indicés par Z.

Dans la littérature, quand on utilise des fonctions d'utilité pour modéliser des préférences dans des espaces multi-attributs, le type d'indépendance le plus utilisé suppose une indépendance entre attributs qui assure la modélisation des préférences par une **utilité additive**.

Définition 1.12. Une relation de préférence sur $\mathcal{A} = \times_{i=1}^{n} A_i$ est décomposable additivement si elle peut être modélisée par une fonction d'utilité $u(x_1, ..., x_n)$ telle que $u(x_1, ..., x_n) = \sum_{i=1}^{n} u_i(x_i)$. Autrement dit, $\forall x, y \in \mathcal{A}$

$$(x_1, ..., x_n) \succsim (y_1, ..., y_n) \Leftrightarrow \sum_{i=1}^{n} u_i(x_i) \geq \sum_{i=1}^{n} u_i(y_i) \tag{1.5}$$

Cela nous permettre de réduire l'espace de représentation de $\times_{i=1}^{n}|A_i|$ à $\sum_{i=1}^{n}|A_i|$.

Exemple 1.7. Revenons à l'épigraphe de ce chapitre (en page 7). Il exprime des préférences sur des alternatives composées par deux attributs, que nous appellerons attitude (A_1) et nourriture (A_2). Nous avons que $A_1 = \{\text{amour}\,(a_1^1),\ \text{haine}\,(a_1^2)\}$ et $A_2 = \{\text{bœuf engraissé}\,(a_2^1), \text{herbe}\,(a_2^2)\}$. Nous pouvons interpréter que l'épigraphe correspond aux préférences suivantes* :

- Dans tous les cas, l'amour, en tant qu'attitude, est préféré à la haine ;
- Un bœuf engraissé pour nourriture est préférable à de l'herbe, *ceteris paribus*.

Ainsi, son auteur utiliserait l'ordre de préférences :

$$(a_1^1, a_2^1) \succ (a_1^1, a_2^2) \succ (a_1^2, a_2^1) \succ (a_1^2, a_2^2)$$

Ces préférences peuvent être modélisées par une fonction d'utilité additive $u(a_1, a_2) = u_1(a_1) + u_2(a_2)$. Par exemple, nous pouvons adopter : $u_1(a_1^1) = 100,\ u_1(a_1^2) = 10$; $u_2(a_2^1) = 2,\ u_2(a_2^2) = 1$.

Exemple 1.8. Un voyageur cherche à acheter un billet de train. Supposons qu'un billet de train se caractérise par les trois attributs suivants : heure du départ, catégorie, vitesse du train. Ainsi, nous pouvons caractériser le problème comme suit :

- $\mathcal{A}$ de billets de train $a = (a_1, a_2, a_3)$,
- heure du départ $a_1 \in A_1 = \{\text{nuit } (a_1^1), \text{ jour } (a_1^2)\}$,
- catégorie $a_2 \in A_2 = \{\text{cabine } (a_2^1), \text{ siège } (a_2^2)\}$,
- vitesse du train $a_3 \in A_3 = \{\text{grande vitesse } (a_3^1), \text{ vitesse normale } (a_3^2)\}$.

Le voyageur a les préférences (lexicographiques) suivantes :

- Avec la plus grande priorité, je préfère voyager de nuit que pendant la journée : $a_1^1 \succ_1 a_1^2$;
- après, le plus important c'est d'avoir une cabine : $a_2^1 \succ_2 a_2^2$;
- enfin, je préfère le Train à Grande Vitesse (TGV) : $a_3^1 \succ_3 a_3^2$.

Ces préférences induisent l'ordre :

$$(a_1^1, a_2^1, a_3^1) \succ (a_1^1, a_2^1, a_3^2) \succ (a_1^1, a_2^2, a_3^1) \succ (a_1^1, a_2^2, a_3^2) \succ (a_1^2, a_2^1, a_3^1) \succ (a_1^2, a_2^1, a_3^2) \succ (a_1^2, a_2^2, a_3^1) \succ (a_1^2, a_2^2, a_3^2)$$

Cet ordre peut être modélisé par une fonction d'utilité additive $u(a_1, a_2, a_3) = u_1(a_1) + u_2(a_2) + u_3(a_3)$. Par exemple, nous pouvons utiliser : $u_1(a_1^1) = 100,\ u_1(a_1^2) = 90$; $u_2(a_2^1) = 10,\ u_2(a_2^2) = 5$; $u_3(a_3^1) = 1,\ u_3(a_3^2) = 0$.

*Notons que ce proverbe date d'une époque où garder la ligne n'était certainement pas parmi les principales préoccupations des personnes.

Dans une relation de préférence additive tous les attributs sont mutuellement indépendants, c'est-à-dire que la préférence sur chaque attribut ne dépend pas des valeurs des autres attributs. Comme toute interaction entre attributs est interdite, cette décomposition peut s'avérer inadéquate dans certains cas. D'autres formes de décomposition plus flexibles sont donc nécessaires. Une structure moins restrictive que la structure additive est la **décomposition multi-linéaire** :

Définition 1.13. Une fonction d'utilité multi-linéaire peut être écrite sous la forme :

$$u(x_1, ..., x_n) = \sum_{\emptyset \neq Y \subseteq \{1,...,n\}} k_Y \prod_{i \in Y} u_i(x_i) \tag{1.6}$$

où k_Y sont constantes et les u_i sont normalisées dans l'intervalle $[0, 1]$.

Nonobstant que plus générales par rapport aux décompositions additives, les décompositions multi-linéaires sont toujours assez limitées par rapport aux interactions possibles entre attributs. Considérez l'exemple suivant de Gonzales et Perny (2004) :

Exemple 1.9. Soit $\mathcal{A} = A_1 \times A_2$, où $A_1 = \{\text{agneau, légumes, steak}\}$ et $A_2 = \{\text{vin rouge, vin blanc}\}$. Supposez un décideur avec les préférences suivantes sur des repas :

$$\begin{aligned}(\text{agneau, vin rouge}) \succ (\text{légumes, vin rouge}) \sim (\text{agneau, vin blanc}) \\ \sim (\text{légumes, vin blanc}) \succ (\text{steak, vin rouge}) \succ (\text{steak, vin blanc})\end{aligned} \tag{1.7}$$

Une fonction d'utilité multi-linéaire pour ces préférences devrait avoir la structure suivante :

$$u(\text{plat}, \text{boisson}) = k_1 u_1(\text{plat}) + k_2 u_2(\text{boisson}) + k_3 u_1(\text{plat}) u_2(\text{boisson}) \tag{1.8}$$

Comme chaque u_i est normalisée dans l'intervalle $[0, 1]$, les préférences ci-dessus impliquent que $u_1(\text{agneau}) = 1 \geq u_1(\text{légumes}) = x \geq u_1(\text{steak}) = 0$ et que $u_2(\text{vin rouge}) = 1$ et $u_2(\text{vin blanc}) = 0$. Les différentes configurations de repas auront alors les valeurs d'utilité suivantes :

- (agneau, vin rouge) : $k_1 u_1(\text{agneau}) + k_2 u_2(\text{vin rouge}) + k_3 u_1(\text{agneau}) u_2(\text{vin rouge}) = k_1 + k_2 + k_3$;
- (légumes, vin rouge) : $k_1 u_1(\text{légumes}) + k_2 u_2(\text{vin rouge}) + k_3 u_1(\text{légumes}) u_2(\text{vin rouge}) = k_1 x + k_2 + k_3 x$;
- (agneau, vin blanc) : $k_1 u_1(\text{agneau}) + k_2 u_2(\text{vin blanc}) + k_3 u_1(\text{agneau}) u_2(\text{vin blanc}) = k_1$;
- (légumes, vin blanc) : $k_1 u_1(\text{légumes}) + k_2 u_2(\text{vin blanc}) + k_3 u_1(\text{légumes}) u_2(\text{vin blanc}) = k_1 x$;

- (steak, vin rouge) : $k_1u_1(\text{steak}) + k_2u_2(\text{vin rouge}) + k_3u_1(\text{steak})u_2(\text{vin rouge}) = k_2$;
- (steak, vin blanc) : $k_1u_1(\text{steak}) + k_2u_2(\text{vin blanc}) + k_3u_1(\text{steak})u_2(\text{vin blanc}) = 0$.

Les préférences du décideur (1.7) impliquent alors le système d'inégalités suivant :

$$k_1 + k_2 + k_3 > k_1x + k_2 + k_3x = k_1 = k_1x > k_2 > 0 \tag{1.9}$$

Ce système n'a pas de solution (remarquez que $k_1 = k_1x$ implique que $x = 1$ et $k_1 + k_2 + k_3 > k_1x + k_2 + k_3x$ implique que $x < 1$, c'est une contradiction). Ainsi, cet ensemble de préférences n'est pas représentable par une fonction d'utilité multi-linéaire.

Les modèles d'**Indépendance Additive Généralisée** (**GAI** – de l'anglais ***G**eneralized **A**dditive **I**ndependence*) permettent la modélisation de préférences où les attributs ne sont pas indépendants individuellement*, mais où l'on observe des indépendances entre des sous-ensembles d'attributs (pas forcément disjoints) (Fishburn, 1967; Bacchus et Grove, 1995).

Définition 1.14. Soit $\mathcal{A} = A_1 \times A_2 \times ... \times A_n$; $C_1, ..., C_k$ des sous-ensembles de $N = \{1, ...n\}$ tels que $N = \bigcup_{i=1}^{k} C_i$ et $X_{C_i} = \{A_j : j \in C_i\}$. Une fonction d'utilité GAI sur $\mathcal{A}$ peut être écrite sous la forme :

$$u(x_1, ..., x_n) = \sum_{i=1}^{k} u_i(x_{C_i}) \tag{1.10}$$

où chaque u_i est une fonction $u_i : X_{C_i} \mapsto \mathbb{R}$

Exemple 1.10. Considérons un nouveau voyageur pour le problème de l'exemple 1.8. Cette fois-ci, ses préférences sont comme suit :

- Le plus important c'est d'avoir un billet adapté au type du voyage. Pour des voyages diurnes je préfère un siège, mais pour les voyages nocturnes je préfère une cabine (préférence conditionnelle) : $a_1^1 : a_2^1 \succ_2 a_2^2$, $\quad a_1^2 : a_2^2 \succ_2 a_2^1$;
- ensuite, je préfère voyager la nuit que pendant la journée : $a_1^1 \succ_1 a_1^2$;
- enfin, quand je voyage pendant la journée je préfère le TGV, mais pour la nuit je préfère le train à vitesse normale : $a_1^2 : a_3^1 \succ_3 a_3^2$, $\quad a_1^1 : a_3^2 \succ_3 a_3^1$.

*Comme le souligne Bouyssou et Pirlot (2005), il est parfois argumenté dans la littérature que l'absence d'indépendance additive entre les attributs indique que l'ensemble d'attributs a été mal choisi. Or, nous considérons que même si en théorie on pouvait avoir un ensemble d'attributs indépendants pour un certain problème, en pratique il arrive souvent que l'ensemble d'attributs considéré le plus approprié par le décideur ne soit pas indépendant. Ce qui justifie l'intérêt d'avoir des modèles capables de manipuler de telles préférences.

Ces préférences induisent l'ordre :

$$(a_1^1, a_2^1, a_3^2) \succ (a_1^1, a_2^1, a_3^1) \succ (a_1^2, a_2^2, a_3^1) \succ (a_1^2, a_2^2, a_3^2) \succ$$
$$(a_1^1, a_2^2, a_3^2) \succ (a_1^1, a_2^2, a_3^1) \succ (a_1^2, a_2^1, a_3^1) \succ (a_1^2, a_2^1, a_3^2)$$

Cet ordre peut être modélisé par une fonction d'utilité GAI-décomposable $u(a_1, a_2, a_3) = u_1(a_1, a_2) + u_2(a_1, a_3)$. La figure 1.2 montre des valeurs d'utilité pour u.

$u_1(a_1, a_2)$	a_2^1	a_2^2
a_1^1	110	10
a_1^2	0	100

$u_2(a_1, a_3)$	a_3^1	a_3^2
a_1^1	0	5
a_1^2	5	0

FIG. 1.2 – Valeurs d'utilité pour $u(a_1, a_2, a_3) = u_1(a_1, a_2) + u_2(a_1, a_3)$ de l'exemple 1.10.

Exemple 1.11. Revenons à l'exemple 1.5 (page 16). Si nous décidons d'adopter une approche du type « agréger puis comparer » pour classer les voitures, nous pouvons procéder en modélisant les préférences du décideur par une fonction d'utilité u GAI-décomposable. Soient les attributs nommés comme suit : couleur (a_1), nombre de portes (a_2), décapotable (a_3), direction assistée (a_4), freinage ABS (a_5), type de carburant (a_6), embrayage automatique (a_7). Telle fonction d'utilité peut avoir la forme :

$$u(a_1, a_2, a_3, a_4, a_5, a_6, a_7) = u_1(a_1, a_2, a_3) + u_2(a_2, a_4, a_6, a_7) + u_3(a_3, a_4, a_5)$$

Comme les modèles GAI ne font pas de suppositions sur le type d'interaction entre les attributs de chaque sous-utilité u_i, ils permettent une grande flexibilité de modélisation et par conséquent un grand nombre de relations de préférence peut être modélisé par des fonctions GAI. Toutefois, les interactions entre attributs apportent des difficultés pour l'élicitation du modèle. Dans le cadre d'indépendance additive, chaque attribut est préférentiellement indépendant de l'ensemble des autres attributs. Nous avons donc que (1.3) :

$$\forall X_i \quad (x_{\{i\}}, x_{-\{i\}}) \succeq (x'_{\{i\}}, x_{-\{i\}}) \Leftrightarrow (x_{\{i\}}, x'_{-\{i\}}) \succeq (x'_{\{i\}}, x'_{-\{i\}}).$$

Par conséquent, on peut éliciter localement l'utilité pour chaque attribut. Par ailleurs, dans un modèle GAI, d'une manière générale, les attributs ne sont pas préférentiellement indépendant des autres. En plus, même les sous-utilités u_i ne représentent pas des indépendances preferentielles entre l'ensemble d'attributs dans u_i et son complèment. Considerons un modèle GAI sur un domaine de trois attributs binaires, et tel que $u(x_1, x_2, x_3) = u_1(x_1, x_2) + u_2(x_2, x_3)$. Supposons que $u_1(x_1^0, x_2^0) = u_1(x_1^1, x_2^1) = 10$; $u_1(x_1^0, x_2^1) = u_1(x_1^1, x_2^0) = 1$; $u_2(x_2^0, x_3^1) = u_2(x_2^1, x_3^0) = 100$; $u_2(x_2^1, x_3^1) = u_2(x_2^0, x_3^0) = 9$. Il est aisé de noter que $\{x_1, x_2\}$ n'est pas préférentiellement indépendant de $\{x_3\}$, une

fois que :

$$(x_1^0, x_2^0, x_3^1) \succeq (x_1^0, x_2^1, x_3^1) \text{ mais } (x_1^0, x_2^1, x_3^0) \succeq (x_1^0, x_2^0, x_3^0).$$

Cet exemple illustre que les valeurs des fonctions de sous-utilité ne correspondent pas à des préférences *ceteris paribus* pour les attributs de son domaine. Un autre point à remarquer est que la quantité d'information nécessaire pour éliciter la fonction dépend de la structure de la décomposition GAI. Pour que le nombre de questions soit raisonnable, la taille des sous-ensembles X_{C_i} ne doit pas être trop importante. En pratique, il est fréquent que cette taille ne dépasse pas 3 ou 4 attributs pour chaque X_{C_i}. Nous reviendrons sur les problèmes posés par l'élicitation dans le chapitre 4.

1.2.5 Modèles qualitatifs

Les techniques développées dans le domaine de la théorie de la décision sont en plein essor, comme le constatent Doyle et Thomason (1999) : « *As developed by philosophers, economists, and mathematicians over some 300 years, these disciplines have developed many powerful ideas and techniques, which exert major influences over virtually all the biological, cognitive, and social sciences. Their uses range from providing mathematical foundations for microeconomics to daily application in a range of fields of practice, including finance, public policy, medicine, and now even automated device diagnosis.* ». Les attentes pour la pénétration de la théorie de la décision dans la vie quotidienne sont très élevées, Edwards et Fasolo (2001) reflètent cet optimisme en affirmant que : « *decision tools will be as important in the 21st century as spreadsheets were in the 20th* ». Toutefois, des tentatives récentes dans l'Intelligence Artificielle d'appliquer des techniques traditionnelles pour automatiser le processus de décision, ont révélé des difficultés pour leur adoption.

D'un coté, le décideur peut avoir des difficultés en exprimer des informations numériques, où simplement être hésitant, voir inconsistant, comme le soulignent Doyle et Thomason (1999) : « *Traditional decision theory provides an account of the information that, in principle, suffices for making a rational decision. In practice, however, the decision maker might have never considered the type of choice in question and so might not happen to possess this information.* ».

D'autre coté, même si le décideur possède toute l'information nécessaire pour construire un modèle détaillé de ses préférences, il peut tout simplement ne pas disposer de tout le temps nécessaire pour fournir toute l'information, où encore juger que les enjeux mis en cause ne sont pas suffisamment importants pour justifier l'investissement dans la construction d'un tel modèle.

Ce scénario a motivé le développement du domaine de la décision qualitative qui

utilise des modèles et représentations s'écartant des méthodes quantitatives classiques* de la théorie de la décision, dans l'espoir d'être plus adapté aux besoins du décideur et du problème auquel il doit faire face. Ces modèles sont délibérément simples pour être intégrés efficacement dans des applications, comme les systèmes de recommandation où les préférences des individus doivent être capturées uniquement à partir de quelques questions posées. Ce type d'application justifie l'intérêt actuel pour des modèles qualitatifs graphiques pour la représentation compacte de préférences, telles que CP-Nets (Boutilier *et al.*, 2004a) et TCP-Nets (Brafman *et al.*, 2006). Nous aurons l'occasion de revenir sur ces modèles dans la section 1.3.

Une autre raison de vouloir construire des modèles qualitatifs à la place de modèles quantitatifs réside dans le fait que, dans certaines circonstances, les premiers peuvent être plus faciles à appréhender par le décideur. Ainsi, après la construction du modèle, le décideur peut l'utiliser pour expliquer ou mieux comprendre ses préférences. En se basant sur l'idée que les personnes prennent des décisions en cherchant des règles qui justifient leurs choix (Slovic, 1975), Greco *et al.* (1999) ont développé une méthode pour construire un ensemble de règles de décision du type « si..., alors... » à partir de comparaisons binaires entre les alternatives. Par ailleurs, Coste-Marquis *et al.* (2004) analisent les modèles fondés sur la logique propositionnelle. Ces modèles peuvent être à la fois compacts, expressifs et d'une bonne lisibilité, grâce aux qualités de la logique propositionnelle.

Toutefois, les modèles qualitatifs peuvent être plus difficiles à exploiter une fois construits. Par exemple, dans Brafman *et al.* (2004), après que l'on ait construit un modèle qualitatif de préférences (dans ce cas-là, un TCP-Net), on le transforme en un modèle quantitatif (une fonction d'utilité GAI) afin de pouvoir générer efficacement un rangement d'alternatives. Par ailleurs, la compilation de connaissances est un domaine de recherche très actif qui propose la compilation de modèles fondés sur la logique dans des langages plus traitables du point de vue computationnel (voir, par exemple, Marquis (1995); Selman et Kautz (1996); Cadoli et Donini (1997); Darwiche (1999); Darwiche et Marquis (2002); Fargier et Marquis (2008)). La complexité pour explorer un modèle est directement liée à sa structure et aux méthodes d'aide à la décision envisagées, c'est-à-dire qu'il faut considérer la problématique du processus décisionnel (*cf.* section 1.2.6) et donc les questions auxquelles le modèle, dans la problématique concernée, doit répondre.

1.2.6 Problématique d'aide à la décision

Les méthodes d'aide à la décision envisagées dans un processus décisionnel dépendent principalement des termes dans lesquels le problème est formulé, de ceux desquels on désire

*tels que ceux fondés sur des fonctions d'utilité

obtenir une éventuelle recommandation, où encore sur lesquels on désire prendre appui dans le processus décisionnel. Roy et Bouyssou (1993) identifient quatre problématiques d'aide à la décision :

1. **Choix** (P.α) : on pose le problème en terme de choix d'une seule « meilleure » action. Ainsi, l'aide à la décision est orientée vers la mise en évidence d'un sous-ensemble $\mathcal{A}'$ de $\mathcal{A}$ tel que :
   - On peut justifier l'élimination de toute alternative a tel que $a \notin \mathcal{A}'$;
   - $\mathcal{A}'$ soit aussi restreint que possible.
2. **Tri** (P.β) : on cherche à affecter chaque alternative à une catégorie, dans un ensemble de catégories définies au préalable. Dans cette problématique, l'affectation d'une alternative à une catégorie donnée s'effectue indépendamment de l'affectation des autres alternatives à cette catégorie ou à une autre catégorie. Ainsi, cette affectation est faite uniquement en fonction de normes portant sur la valeur intrinsèque des alternatives.
3. **Rangement** (P.γ) : on pose le problème en terme de rangement des alternatives de $\mathcal{A}$ (ou de certaines d'entre elles). Ainsi, nous nous intéressons ici à classer les alternatives, c'est-à-dire de les regrouper en classes d'indifférence et d'ordonner ces classes.
4. **Description** (P.δ) : on cherche à éclairer la décision par une description, dans un langage approprié, des actions de $\mathcal{A}$ et/ou de leurs conséquences.

Une procédure d'exploitation ayant pour objet d'aider à tirer parti d'un modèle de préférences doit être conçue en relation directe avec ces problématiques. En effet, pour P.δ le modèle de préférences est en soit l'objet d'aide à la décision. Ainsi, seulement les trois premières problématiques suscitent des procédures d'exploitation associées. Une procédure d'exploitation sera nommée une **procédure de sélection, d'affectation ou de classement**, selon que l'objectif recherché relève respectivement du choix, du tri ou du rangement.

Comme nous l'avons souligné dans l'introduction, dans ce travail nous nous intéressons à des procédures concernant le rangement, puisque l'une des caractéristiques essentielles d'un système de recommandation est que celui-ci soit capable de lister les alternatives les plus adaptées aux préférences de l'utilisateur. Nous nous intéressons aussi aux procédures de choix de la meilleure alternative, dans le cadre de la décision collective ou multi-critère.

1.2.7 Décision dans le certain et décision dans l'incertain

Quand chaque acte mène invariablement à une conséquence précise (comme dans l'exemple 1.4), nous sommes dans le domaine de la **décision dans le certain**. Toutefois,

dans certaines occasions, des facteurs d'incertitude ou stochastiques (intrinsèques au problème ou en raison de données incomplètes) font qu'à chaque acte corresponde un ensemble de conséquences possibles. Dans ces cas, nous sommes dans le domaine de la **décision dans l'incertain***. Regardons maintenant un exemple qui illustre, un problème de décision dans l'incertain.

Exemple 1.12. (Savage, 1972, pages 13 à 15) Un décideur qui prépare une omelette a déjà cassé 5 œufs dans un bol. Il veut ajouter un 6^{e} œuf, mais il n'est pas sûr que cet œuf soit frais. Il peut réaliser l'un des trois actes possibles :

- casser l'œuf dans le bol (COB) ;
- casser l'œuf dans une tasse et l'ajouter à l'omelette seulement s'il est frais (COT) ;
- jeter l'œuf suspect (JO).

Le tableau 1.3 liste les actions et les conséquences possibles. Nous constatons que chaque acte peut amener à l'une des deux conséquences possibles. Étant donnée l'incertitude sur l'état du 6^{e} œuf et son désir de préparer une omelette avec 6 œufs, le décideur pourrait avoir comme préférences : COT $\succ$ JO ; JO $\succ$ COB. Dans ce cas, si nous supposons la transitivité des préférences, nous obtenons une relation de préférences complète sur le produit cartésien $\mathcal{A} \times \mathcal{A}$.

Tab. 1.3 – Décision dans l'incertain, l'exemple de l'omelette.

	Conséquence	
Acte	*Si l'œuf était frais*	*Si l'œuf était pourri*
Casser l'œuf dans le bol (COB)	une omelette avec 6 œufs	rien à manger
Casser l'œuf dans une tasse (COT)	une omelette avec 6 œufs, une tasse à laver	une omelette avec 5 œufs, une tasse à laver
Jeter l'œuf (JO)	une omelette avec 5 œufs, un œuf gâché	une omelette avec 5 œufs

L'approche le plus classique d'aide à la décision dans l'incertain suppose que nous avons une fonction d'utilité u sur l'espace de conséquences $\mathcal{O}$ et modélise $\mathcal{A}$ comme un ensemble de distributions de probabilités (aussi nommées des lotteries) sur l'espace de conséquences $\mathcal{O}$ (Fishburn, 1999). Pour comparer les actes, le critère utilisé est l'utilité espérée (EU). Ainsi, si $a^1, a^2 \in \mathcal{A}$:

$$a^1 \succ a^2 \Leftrightarrow \sum_{o \in \mathcal{O}} a^1(o)u(o) > \sum_{o \in \mathcal{O}} a^2(o)u(o)$$

*Parfois, dans la littérature, on utilise le terme « décision dans le risque » quand la probabilité des conséquences possibles est connue, et on réserve le terme « décision dans l'incertain » pour les cas où les probabilités sont inconnues ou même sans signification.

Même si l'utilité espérée reste probablement le critère le plus utilisé dans le domaine de la décision dans l'incertain, il ne correspond pas toujours au comportement des décideurs. Plusieurs exemples peuvent être trouvés dans la littérature où les décideurs exhibent un comportement qui n'est pas en accord avec l'utilité espérée (voir p. ex. Allais (1953); Ellsberg (1961); Kahneman et Tversky (1979)).

En plus, l'approche fondée sur l'utilité espérée requiert une grande quantité d'informations disponibles pour être mise en œuvre. Notons que nous devons à la fois être capable de mesurer l'utilité des conséquences et la probabilité d'obtenir les conséquences à partir de chaque acte. Ainsi, dans des contextes où nous avons moins d'informations disponibles, tels que ceux où l'on utilise des modèles qualitatifs, on peut vouloir utiliser d'autres modèles décisionnels que EU, plus adaptés à ces situations. Ceci configure un domaine de recherche très actif dans les dernières années (voir par exemple Boutilier (1994); Dubois et Prade (1995); Lehmann (1996); Sabbadin (1998); Dubois *et al.* (2001); Giang et Shenoy (2001); Weng (2005)).

Dans ce travail nous sommes majoritairement intéressés par les situations de décision dans le certain, comme les systèmes de recommandation. Dans ce cas, l'attitude du décideur vis-à-vis du risque et de l'incertitude peut être ignorée, puisque les actes sont complètement déterministes.

1.3 Modèles graphiques de représentation de préférences

Les modèles graphiques, tels que les réseaux bayésiens, ont connu un grand succès en optimisant l'élicitation et le raisonnement avec probabilités (voir par exemple Cowell *et al.* (1999)). L'efficacité de ces modèles est fondée sur l'exploitation des indépendances existantes dans le modèle probabiliste. Ce succès a suscité l'idée d'utiliser les concepts d'indépendance des utilités pour arriver à des modèles graphiques efficaces pour l'élicitation et le raisonnement avec des utilités (Bacchus et Grove, 1995), ou bien avec des préférences qualitatives. Ainsi, au cours des dernières années, différents modèles graphiques pour représenter des préférences ont été développés.

1.3.1 CP-Nets

Les CP-Nets (acronyme pour *Conditional Preference Networks* – réseaux de préférences conditionnelles) ont été introduits par Boutilier *et al.* (1999). Un CP-Net est un modèle graphique développé pour représenter de manière compacte des préférences qualitatives du type *ceteris paribus* dans des espaces multi-attributs. Pour cela, un CP-Net utilise un graphe orienté (pas forcément acyclique) où chaque nœud correspond à un attribut. Chaque nœud est préférentiellement indépendant des autres nœuds conditionnellement à

ses *parents* dans le graphe. Formellement, un CP-Net C est un n-uplet $\langle \mathcal{A}, cp, cpt \rangle$, tel que :

1. $\mathcal{A}$ est l'ensemble de nœuds qui correspond aux attributs du problème $\{A_1, ..., A_n\}$.
2. cp est un ensemble d'arêtes orientées $\langle \overrightarrow{A_i, A_j} \rangle$ (nommées *cp-arcs*) signifiant que les préférences du décideur sur les valeurs de A_j sont dépendantes de la valeur de A_i. Soit $P_{A_j} = \{A_i | \langle \overrightarrow{A_i, A_j} \rangle \in cp\}$ l'ensemble de *parents* de A_j. Les indépendances conditionnelles du réseau impliquent que pour tout A_i, nous avons $CPI(A_i, P_{A_i}, Y_i)$ où $Y_i = \mathcal{A} - \{P_{A_i} \cup \{A_i\}\}$.
3. cpt associe pour chaque $A_i \in \mathcal{A}$, un tableau de préférences conditionnelles (CPT – de l'anglais ***C**onditional **P**reference **T**able*), c'est-à-dire une application de P_{A_i} à une relation de préférence sur A_i.

Selon la sémantique standard des CP-Nets, les préférences exprimées dans le CPT d'un CP-net sont des préférences strictes ($\succ$), mais des extensions où les préférences dans les CPT sont des préférences non-strictes ($\succsim$) ont aussi été proposées (Brafman et Dimopoulos, 2004). La sémantique standard des CP-Nets est définie en fonction des rangements des alternatives qui sont consistants avec les contraintes imposées par ses CPT. Formellement, on définit les rangements des alternatives qui sont consistants avec un CP-Net de la manière suivante :

Définition 1.15. Soit C un CP-Net sur des attributs $\mathcal{A}$, $A_i \in \mathcal{A}$ un attribut, et $P_{A_i} \subset \mathcal{A}$ les *parents* de A_i dans C. Soit $Y_i = \mathcal{A} - \{P_{A_i} \cup \{A_i\}\}$. Soit $\succ_{p_{A_i}}$ l'ordre de préférences sur A_i induit par le $CPT(A_i)$ en fonction de l'instantiation $a_{P_{A_i}} \in P_{A_i}$. Enfin, soit $\succ$ un rangement des alternatives dans $\mathcal{A}$.

1. Un rangement des alternatives $\succ$ satisfait $\succ_{p_{A_i}}$ ssi $(a_{Y_i}, a_{P_{A_i}}, a_i) \succ (a_{Y_i}, a_{P_{A_i}}, a'_i)$, pour tout $a_{Y_i} \in Y_i$, quand $a_i \succ_{p_{A_i}} a'_i$.
2. Un rangement des alternatives $\succ$ satisfait $CPT(A_i)$ ssi il satisfait $\succ_{p_{A_i}}$ pour tout $a_{P_{A_i}} \in P_{A_i}$.
3. Un rangement des alternatives $\succ$ satisfait le CP-Net C ssi il satisfait $CPT(A_i)$ pour tout attribut A_i. Dans ce cas, on dit que $\succ$ est consistant avec le CP-Net C.

Un CP-Net C est satisfiable ssi il existe un rangement des alternatives $\succ$ qui satisfait C.

Exemple 1.13. La Figure 1.3 illustre l'utilisation d'un CP-Net pour un problème de choisir un vol pour aller à une conférence à Recife (Brésil) à partir de Paris (France). Ce réseau possède quatre attributs correspondants aux paramètres du vol :

- D – Date du départ. Le décideur préfère partir un jour avant la conférence (d^{1j}) plutôt que deux jours avant (d^{2j}).

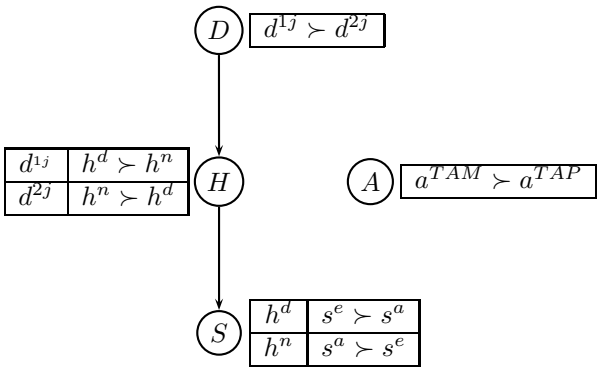


FIG. 1.3 – Un CP-Net pour l'exemple du choix de vols

- A – Compagnie aérienne. Le décideur préfère TAM (a^{TAM}) à TAP (a^{TAP}).
- H – Horaire du vol. Pour partir deux jours avant la conférence, le décideur préfère un vol nocturne, h^n (comme ça il peut travailler pendant la journée). Pour partir un jour avant la conférence, le décideur préfère un vol diurne, h^d (sinon il ne va pas arriver à temps).
- S – Classe de service. Pour un vol nocturne, le décideur préfère la classe d'affaires (s^a) puisqu'il n'arrive pas à dormir dans les sièges inconfortables de la classe économique (s^e). Pour un vol pendant la journée les préférences s'inversent puisque le décideur ne pense pas à dormir.

Comme on peut le constater dans l'exemple, les CP-Nets permettent l'expression de deux types de préférences *ceteris paribus* : *des préférences inconditionnelles*, comme « je préfère TAM à TAP » ; et des *préférences conditionnelles*, comme « si je pars un jour avant la conférence je préfère un vol diurne ; dans le cas contraire je préfère un vol nocturne ». La figure 1.4 montre les préférences induites par le CP-Net. Les flèches sont orientées d'une alternative moins préférée vers une alternative plus préférée selon une préférence *ceteris paribus*. Par exemple, une flèche lie $(d^{2j}, h^n, s^e, a^{TAM})$ et $(d^{1j}, h^n, s^e, a^{TAM})$ parce que ces deux alternatives diffèrent seulement dans la date du départ et, toutes choses égales par ailleurs, $d^{1j} \succ d^{2j}$ (préférence inconditionnelle). Analogiquement, une flèche lie $(d^{1j}, h^n, s^e, a^{TAM})$ et $(d^{1j}, h^n, s^a, a^{TAM})$ parce que ces deux alternatives diffèrent seulement dans la classe de service et, étant donné qu'il s'agit d'un vol nocturne (h^n), $s^a \succ s^e$ (préférence conditionnelle).

Notons que le CP-net induit une relation de préférences partielle (quand le graphe de préférences induit est acyclique, nous avons un ordre partiel sur des alternatives). Par exemple, nous ne pouvons pas décider si $(d^{1j}, h^n, s^a, a^{TAP}) \succ (d^{1j}, h^n, s^e, a^{TAM})$ ou vice-versa. Toutefois, il y a des préférences qui ne sont pas dérivées directement des CPT mais qui sont révélées par transitivité, par exemple, nous pouvons affirmer que

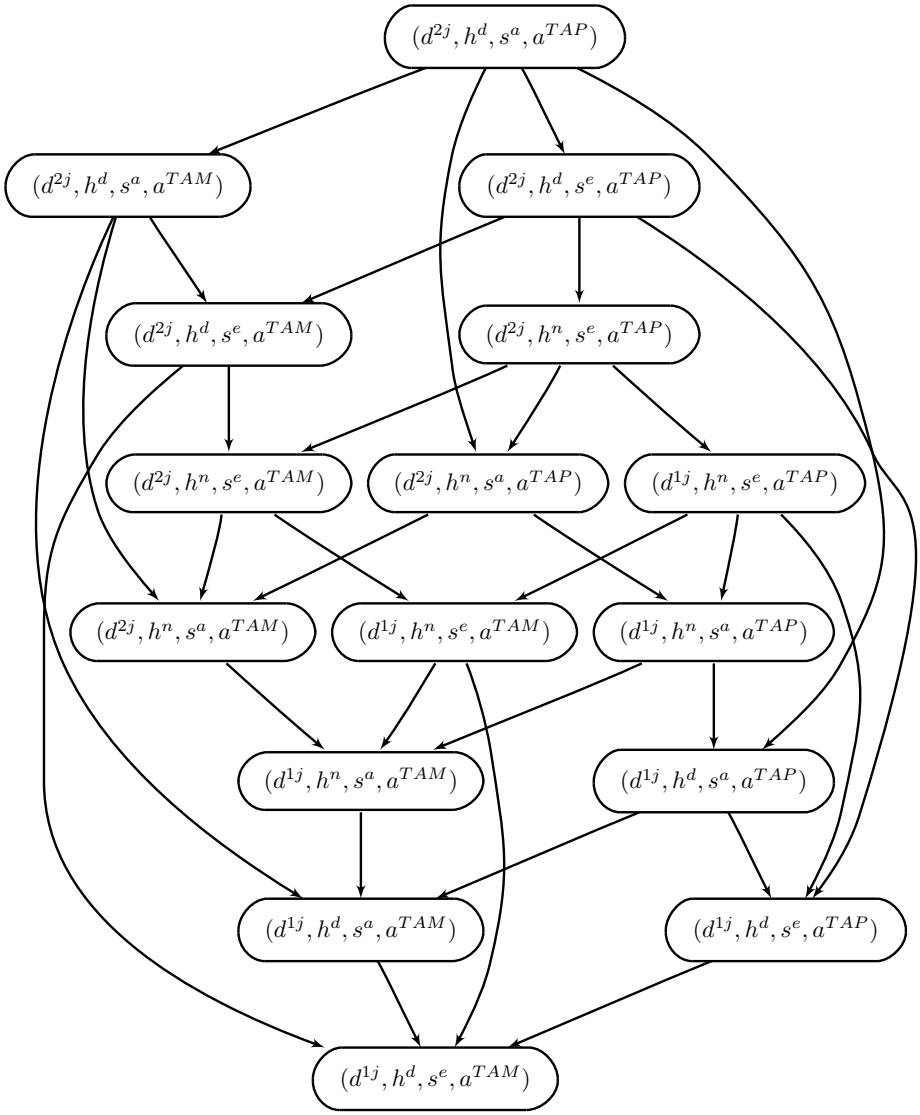


FIG. 1.4 – Préférences sur des configurations pour l'exemple de choix de vols

$(d^{2j}, h^n, s^a, a^{TAM}) \succ (d^{2j}, h^d, s^e, a^{TAP})$. En effet, déterminer si on peut conclure que $a \succ a'$ à partir d'un CP-Net C (nous notons cela par $C \vDash a \succ a'$), équivaut à trouver un chemin de a' à a dans le graphe de préférences induites. Dans la littérature de CP-Nets, on le nomme une **séquence d'échanges améliorants** (en anglais, on l'appelle une *improving flipping sequence*).

Définition 1.16. (Boutilier *et al.*, 1999) Soit C un CP-Net sur les attributs $\mathcal{A}$; P_{A_i} l'ensemble de *parents* de $A_i \in \mathcal{A}$ et $Y_i = \mathcal{A} - \{P_{A_i} \cup \{A_i\}\}$. Soit $(a_{P_{A_i}}, a'_i, a_{Y_i})$ une alternative dans $\mathcal{A}$. Une alternative $(a_{P_{A_i}}, a_i, a_{Y_i})$ peut être obtenue par un **échange améliorant** à partir de l'alternative $(a_{P_{A_i}}, a'_i, a_{Y_i})$ par rapport à l'attribut A_i si $a_i \succ a'_i$

étant donné $a_{P_{A_i}}$ dans C.* Une **séquence d'échanges améliorants** dans C est toute séquence d'alternatives $a^1, \ldots, a^k$ telle que a^{j+1} est obtenue par un échange améliorant à partir de a^j par rapport à un attribut quelconque (pout tout $j < k$).† Toute séquence d'échanges améliorants telle que $a^1 = a'$ et $a^k = a$ est une séquence d'échanges améliorants de l'alternative a' à l'alternative a.

Théorème 1.2. *(Boutilier* et al.*, 1999) Soit C un CP-Net sur les attributs $\mathcal{A}$:*

$$\forall a, a' \in \mathcal{A}, \quad C \vDash a \succ a' \Leftrightarrow \textit{il existe une séquence d'échanges améliorants de } a' \textit{ à } a.$$

Il s'avère que la recherche d'une séquence d'échanges améliorants est une tâche difficile (même quand les attributs ont un domaine binaire). Sa compléxité varie en fonction de la structure du CP-Net. Nous savons que (Boutilier *et al.*, 2004a; Goldsmith *et al.*, 2005) :

1. Quand le CP-Net binaire est un arbre orienté, la complexité est quadratique dans le nombre d'attributs.
2. Quand le CP-Net binaire est un multi-arbre (c'est-à-dire que le graphe non-orienté induit n'a pas de cycles), la complexité est polynomiale dans la taille de la description du réseau.
3. Quand le CP-Net binaire est un graphe orienté connecté simple (c'est-à-dire qu'il y a au maximum un chemin orienté entre deux nœuds), le problème est NP-complet.
4. Le problème reste NP-complet si le nombre de chemins orientés entre deux nœuds est limité par une constante.
5. Le problème est PSPACE-complet‡ si le CP-Net possède des cycles.

Ainsi, étant donné deux alternatives $a', a \in \mathcal{A}$, et un CP-Net C sur $\mathcal{A}$, déterminer si nous pouvons affirmer $C \vDash a \succ a'$ ou $C \vDash a' \succ a$ (dans la littérature des CP-Nets, on l'appelle le **problème de dominance**) est loin d'être une tâche facile pour une bonne partie des CP-Nets. Toutefois, si l'objectif est de trouver la meilleure alternative dans $\mathcal{A}$ (**problème d'optimisation**), la tâche peut être plus simple (Domshlak *et al.*, 2003; Boutilier *et al.*, 2004a,b) :

*De manière analogue, une alternative $(a_{P_{A_i}}, a_i, y_i)$ peut être obtenue par un **échange empirant** à partir de l'alternative $(a_{P_{A_i}}, a'_i, a_{Y_i})$ par rapport à l'attribut A_i si $a'_i \succ a_i$, étant donné $a_{P_{A_i}}$ dans C.

†De manière analogue, une **séquence d'échanges empirants** dans C est toute séquence d'alternatives $a^1, \ldots, a^k$ telle que a^{j+1} est obtenue par un échange empirant à partir de a^j par rapport à un attribut quelconque (pout tout $j < k$).

‡PSPACE est la classe de tous les langages reconnaissables en espace polynomial par un programme qui utilise une machine de Turing déterministe et qui termine pour toute entrée. Le fait qu'un problème soit PSPACE-complet est une évidence encore plus forte qu'il soit intractable que s'il était NP-complet, en effet nous savons que P $\subseteq$ NP $\subseteq$ PSPACE (Garey et Johnson (1979, pages 170–178), Papadimitriou (1994, pages 455–490.)).

1. Si le CP-Net est un graphe acyclique, la complexité est polynomiale dans le nombre d'attributs. Intuitivement, on parcourt le réseau de haut en bas (c'est-à-dire des *parents* aux *descendants*), en fixant la valeur de chaque attribut à sa valeur préférée étant donnée l'instantiation de ses *parents*. Toutefois, s'il y a des contraintes sur des configurations possibles, le problème devient NP-complet.
2. Si le CP-Net possède des cycles, déterminer s'il y a une alternative non-dominée est un problème NP-complet.

L'ordre partiel induit par un CP-Net suppose implicitement que les attributs-*parents* sont plus importants que les attributs-*fils*, c'est-à-dire que les préférences sur les attributs-*parents* ont plus de priorité que les préférences sur les attributs-*fils*. Toutefois, les CP-Nets ne permettent pas que le décideur exprime l'importance entre deux attributs qui ne sont pas un *descendant* de l'autre dans le graphe.

Exemple 1.14. Dans l'exemple 1.7 (page 20), nous avons vu que pour l'auteur de l'épigraphe de ce chapitre, les préférences sur l'attribut « attitude » ont une priorité plus élevée que les préférences sur l'attribut « nourriture ». Toutefois, ces deux attributs sont conditionnellement indépendants. Un CP-Net ne permet pas d'exprimer une telle relation de préférences, malgré le fait qu'elle soit suffisamment simple pour être modélisable par une fonction d'utilité additive.

Exemple 1.15. Le CP-Net de la figure 1.3 ne peut pas déterminer si l'option $(d^{1j}, h^d, s^e, a^{TAP})$ (la préférence sur A est violée) est meilleure que l'option $(d^{1j}, h^n, s^e, a^{TAM})$ (la préférence sur H est violée). Or il est fort probable que pour le décideur il soit plus important d'arriver dans les temps que de voyager avec sa compagnie aérienne favorite, donc la première option serait préférable à la deuxième.

Les TCP-Nets (*Trade-off* CP-Nets) ont été introduits in Brafman et Domshlak (2002) pour permettre l'expression de l'importance relative entre attributs*.

1.3.2 TCP-Nets

Les TCP-Nets étendent les CP-Nets en permettant l'expression de deux types d'importance relative entre attributs : *l'importance relative inconditionnelle*, par exemple, « l'horaire du vol est plus important que la compagnie aérienne » ; et *l'importance relative conditionnelle*, par exemple, « l'horaire du vol est plus important que la compagnie aérienne, si je présente dans le premier jour de la conférence. Sinon, la compagnie aérienne est plus importante ». Formellement, on peut définir l'importance relative entre attributs comme suit (Brafman *et al.*, 2006) :

*Un formalisme logique qui généralise les CP-Nets (c'est-à-dire que les CP-Nets peuvent être représentés dans ce langage) et qui est capable de représenter des préférences lexicographiques a aussi été proposé (Wilson, 2004). Toutefois, dans ce cas nous n'avons plus de modèle graphique.

Définition 1.17. Soient A_i, A_j deux attributs conditionnellement indépendants étant donné $W = \mathcal{A} - \{A_i, A_j\}$ ($CPI(A_i, W, A_j)$). On dit que A_i est inconditionnellement plus important que A_j (on écrit $A_i \rhd A_j$), si pour tout $a_W, a_i, a'_i, a_j, a'_j$ tels que $a_i \succ a'_i$ étant donné a_W, alors :

$$(a_i, a_j, a_W) \succ (a'_i, a'_j, a_W)$$

Définition 1.18. Soient A_i, A_j deux attributs conditionnellement indépendants étant donné $W = \mathcal{A} - \{A_i, A_j\}$ ($CPI(A_i, W, A_j)$), et soit $Z \subseteq W$. On dit que A_i est plus important que A_j étant donné a_Z (on écrit $A_i \rhd_z A_j$), ssi pour tout $a_V \in \mathcal{A} - (\{A_i, A_j, \} \cup Z), a_j, a'_j,\ a_i, a'_i$ tels que $a_i \succ a'_i$ étant donnés a_Z, a_V :

$$(a_i, a_j, a_Z, a_V) \succ (a'_i, a'_j, a_Z, a_V)$$

Si pour tout $a_Z \in Z$ nous avons, soit $A_i \rhd_z A_j$, soit $A_j \rhd_z A_i$, nous disons que l'importance relative de A_i et A_j est conditionnelle à Z (on écrit $RI(A_i, A_j|Z)$).

Pour exprimer des importances relatives entre attributs, les TCP-Nets introduisent deux nouveaux types d'arêtes dans le graphe de préférences conditionnelles : les *i-arcs* $\longrightarrow$, qui sont des arêtes orientées pour les importances relatives inconditionnelles ; et les *ci-arcs* $\multimap$, qui sont des arêtes non-orientées pour les importances relatives conditionnelles.

Formellement, un TCP-Net C est un n-uplet $\langle \mathcal{A}, cp, cpt, i, ci, cit \rangle$, tel que :

1. $\mathcal{A}$, *cp* et *cpt* sont comme dans les CP-Nets.
2. i est un ensemble d'arêtes orientées $\langle \overrightarrow{A_i, A_j} \rangle$ (nommées *i-arcs*). Un *i-arc* de A_i à A_j signifie que $A_i \rhd A_j$.
3. *ci* est un ensemble d'arêtes non-orientées $\langle A_i, A_j \rangle$ (nommées *ci-arcs*) signifiant que l'importance relative entre les deux attributs A_i et A_j est conditionnée à la valeur des variables d'un ensemble de sélection (*selector set*, en anglais) $Z \subseteq \mathcal{A} - \{A_i, A_j\}$ ($RI(A_i, A_j|Z)$).
4. *cit* associe pour chaque *ci-arc* entre $\{A_i, A_j\}$ une fonction du domaine de Z à un ordre $\triangleright$ entre $\{A_i, A_j\}$.

Exemple 1.16. Les préférences induites par l'épigraphe de ce chapitre (voir exemple 1.7, page 20) peuvent être facilement réprésentées par un TCP-Net, comme le montre la figure 1.5.

Exemple 1.17. Revenons au problème de choix d'un vol pour aller à une conférence à Recife (Brésil) au départ de Paris (France). Supposons que nous avons maintenant un nouvel attribut :

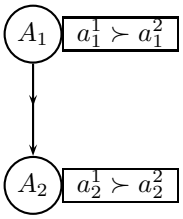


FIG. 1.5 – Un TCP-net pour l'épigraphe de ce chapitre. Un *ci-arc* indique que $A_1 \rhd A_2$

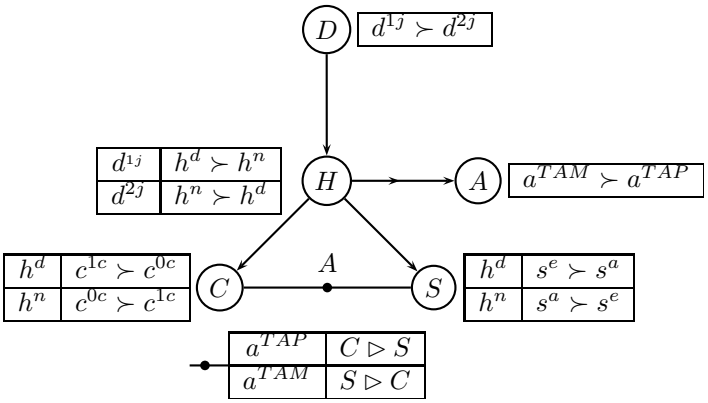


FIG. 1.6 – Un TCP-net pour l'exemple du choix de vols réitéré

- C – correspondance. Pour un vol diurne, le décideur aimerait faire une pause, donc il préfère les vols avec une escale. (c^{1c}). Pour un vol nocturne, le décideur préfère dormir pendant le voyage, il préférera donc un vol direct (c^{0c}).

Le voyageur exprime encore les préférences additionnelles suivantes :

- l'horaire du vol est plus important que la compagnie aérienne ;
- pour les vols de TAP, la correspondance est plus importante que la classe (Lisbonne est sur le trajet, donc faire une escale là-bas ne prendrait pas beaucoup de temps additionnel), mais pour les vols de TAM, la classe est plus importante que la correspondance (l'escale à São Paulo représente un grand détour).

La figure 1.6 illustre l'utilisation d'un TCP-Net pour exprimer les nouvelles préférences du voyageur.

La complexité pour résoudre des problèmes de dominance et optimisation avec des TCP-Nets a été moins étudiée que pour les CP-Nets. Toutefois, comme un CP-Net est un cas particulier d'un TCP-Net, les résultats de complexité pour le second seront, dans tous les cas, au moins aussi difficiles à resoudre que pour le premier. Les algorithmes pour raisonner avec des TCP-Nets sont aussi plus difficiles à construire. Ainsi, l'une des stratégies possible est de transformer le TCP-Net en un modèle plus tractable avant de l'utiliser. C'est l'approche adoptée dans Brafman *et al.* (2004), où le TCP-Net est

d'abord transformé dans une fonction d'utilité GAI-décomposable pour que l'on puisse raisonner d'une manière efficace avec le modèle. Pourtant, tous les TCP-Nets ne sont pas modélisables par une fonction GAI, les conditions suffisantes et nécessaires pour qu'un TCP-Net soit modélisable par une fonction GAI sont introduites dans Brafman *et al.* (2004). Réciproquement, notons que toutes les fonctions GAI ne sont pas représentables par un TCP-Net. Supposons que concernant la compagnie aérienne on avait une troisième option, a^{MEN}, avec des préférences $a^{TAM} \succ a^{TAP} \succ a^{MEN}$ et que comme le voyageur ne fait pas confiance à la compagnie a^{MEN}, l'horaire du vol est plus important que la compagnie aérienne mais uniquement pour a^{TAM} et a^{TAP}, car il préfère arriver un peu en retard en prenant une autre compagnie (a^{TAM} ou a^{TAP}) plutôt que d'empreinter a^{MEN}. On voit ici une limite descriptive intrinsèque des TCP-Nets en raison de l'impossibilité de prendre en compte les intensités de préférences entre les attributs en fonction de leur valeurs respectives.

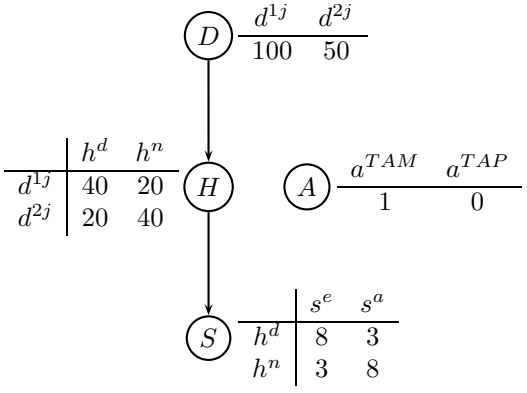
1.3.3 UCP-Nets

Une autre extension des CP-Nets combine la structure du graphe de préférences conditionnelles avec des fonctions d'utilité. Le modèle résultant est un CP-Net où *cpt* est remplacé pour des fonctions d'utilité. Formellement, un UCP-Net C est un n-uplet $\langle \mathcal{A}, cp, u \rangle$, tel que :

1. $\mathcal{A}$ et cp sont comme dans les CP-Nets.
2. u est une fonction $u : \mathcal{A} \mapsto \mathbb{R}$, telle que $u(a_1, ..., a_n) = \sum_i u_i(a_i, P_{a_i})$.
3. Le graphe de l'UCP-Net est orienté et acyclique (DAG – de l'anglais ***D**irected **A**cyclic **G**raph*) et les utilités respectent les propriétés d'indépendance conditionnelle pour $\succsim$, c'est-à-dire que pour tout $A_i \in \mathcal{A}$, $CPI(A_i, P_{A_i}, Y_i)$, où $Y_i = \mathcal{A} - \{P_{A_i} \cup \{A_i\}\}$.

Exemple 1.18. La Figure 1.7 illustre un UCP-net pour l'exemple de choix de vols. Ce réseau correspond à la fonction GAI $u(d, h, s, a) = u_1(d) + u_2(h, d) + u_3(s, h) + u_4(a)$. Chacun des facteurs u_i possède un tableau de préférences dans le réseau. Les tableaux de préférences et l'interprétation GAI du réseau établissent la fonction d'utilité du décideur. Dans l'exemple, à l'alternative $(d^{1j}, h^d, s^a, a^{TAM})$ correspond l'utilité $u(d^{1j}, h^d, s^a, a^{TAM}) = u_1(d^{1j}) + u_2(h^d, d^{1j}) + u_3(s^a, h^d) + u_4(a^{TAM}) = 100 + 40 + 3 + 1 = 144$. Avec les utilités cardinales nous avons pu exprimer le fait que le choix de la compagnie aérienne est moins important que les autres attributs. Notons que, du point de vue de la décomposition GAI, le terme $u_1(d)$ est redondant (il pourrait être intégré à $u_2(h, d)$), mais en gardant la structure graphique des CP-Nets, on préserve la représentation explicite des dépendances

conditionnelles, qui d'une part, facilite l'élicitation, et d'autre part, permet l'utilisation d'algorithmes de résolution plus efficaces.



$(d^{1j}, h^d, s^e, a^{TAM}) = 149$ $(d^{2j}, h^n, s^a, a^{TAM}) = 99$
$(d^{1j}, h^d, s^e, a^{TAP}) = 148$ $(d^{2j}, h^n, s^a, a^{TAP}) = 98$
$(d^{1j}, h^d, s^a, a^{TAM}) = 144$ $(d^{2j}, h^n, s^e, a^{TAM}) = 94$
$(d^{1j}, h^d, s^a, a^{TAP}) = 143$ $(d^{2j}, h^n, s^e, a^{TAP}) = 93$
$(d^{1j}, h^n, s^a, a^{TAM}) = 129$ $(d^{2j}, h^d, s^e, a^{TAM}) = 79$
$(d^{1j}, h^n, s^a, a^{TAP}) = 128$ $(d^{2j}, h^d, s^e, a^{TAP}) = 78$
$(d^{1j}, h^n, s^e, a^{TAM}) = 124$ $(d^{2j}, h^d, s^a, a^{TAM}) = 74$
$(d^{1j}, h^n, s^e, a^{TAP}) = 123$ $(d^{2j}, h^d, s^a, a^{TAP}) = 73$

FIG. 1.7 – À gauche, un UCP-Net pour l'exemple du choix de vols. À droite, les utilités des différentes alternatives

Les UCP-Nets sont très efficaces pour résoudre les problèmes de dominance et d'optimisation (cf. page 32). D'une part, pour les problèmes de dominance, étant donné deux alternatives $a', a \in \mathcal{A}$, et un UCP-Net C sur $\mathcal{A}$, déterminer si nous pouvons affirmer $C \vDash a \succ a'$ ou $C \vDash a' \succ a$ est trivial, il suffit de calculer $u(a)$, $u(a')$ et de les comparer. D'autre part, on peut trouver la configuration optimale (problème d'optimisation) en temps linéaire au nombre d'attributs ; pour cela il suffit de parcourir à partir d'un ordre topologique, les attributs du réseau $A_1, \ldots, A_n$, en affectant le A_i actuel à sa valeur maximale en tenant compte de l'affectation de ses *parents* (Boutilier *et al.*, 2001).

Soulignons qu'un UCP-Net (étant un cas particulier d'une fonction d'utilité), induit une relation de préférences complète, ce qui normalement n'est pas le cas pour les CP-Nets. Toutefois, les exigences que le graphe soit un DAG et que les conditions d'indépendances conditionnelles soient respectées, limitent non seulement la topologie des décompositions GAI pouvant être représentées par un UCP-Net, mais aussi les valeurs d'utilité acceptables dans les tableaux d'utilité (souvenons-nous que la propriété CPI implique que les préférences sur les attributs-*parents* seront toujours plus intenses que les préférences sur des attributs-*fils*). Un nombre exponentiel de tests est necéssaire, dans le cas le plus général, pour vérifier si une décomposition GAI est représentable par un UCP-Net. Des conditions plus simples à tester, suffisantes, mais non nécessaires, sont présentées dans Boutilier *et al.* (2001).

1.3.4 mCP-Nets

Les mCP-nets ont été introduits par Rossi *et al.* (2004) comme une extension aux CP-Nets destinée au cadre de la décision collective où chaque individu exprime ses

(propres) préférences à travers un modèle graphique. Un mCP-net peut contenir des attributs sur lesquels l'individu n'a pas exprimé de préférences, c'est ce que l'on appelle un CP-Net partiel.

Pour accommoder cette extension, la sémantique du réseau a été légèrement modifiée. Une alternative $(a_{P_{A_i}}, a_i, a_{Y_i})$ peut être obtenue à partir d'un échange empirant de l'alternative $(a_{P_{A_i}}, a'_i, a_{Y_i})$ par rapport à l'attribut A_i si :

- $a'_i \succ a_i$ étant donné $a_{P_{A_i}}$ dans C, *quand l'individu a exprimé ses préférences sur* A_i ;
- $(a_{P_{A_i}}, a_i, a_{Y_i})$ peut être obtenue par un échange empirant pour tous les CPT des attributs dépendants de A_i dans C, *quand l'individu n'a pas exprimé ses préférences sur* A_i.

Les échanges dits **indifférents** sont ceux qui concernent un attribut A_i sur lequel l'individu n'a pas émis de préférences, et pour lesquels tous les CPT des attributs dépendant de A_i dans C ne sont ni améliorants, ni empirants. Les échanges dits **incomparables**, sont ceux qui ne sont ni améliorants, ni empirants, ni indifférents. Ainsi :

- $a' \succ a$ ssi, il y a une séquence d'échanges de a' à a, telle que chaque échange est empirant ou indifférent, et s'il y a au moins un échange empirant.
- $a' \sim a$ ssi, il y a une séquence d'échanges entre les deux alternatives, composée uniquement d'échanges indifférents.
- a' et a sont incomparables si, ni $a' \succ a$, ni $a \succ a'$, ni $a' \sim a$.

Les mCP-nets ont été étudiés uniquement pour des graphes acycliques. Dans de tels cas, les coûts pour les problèmes de dominance et d'optimisation sont du même ordre que pour les CP-Nets.

Pour la décision collective, une sémantique de vote est adoptée, dans laquelle chaque individu fait des tests de dominance selon son CP-Net partiel pour déterminer si une alternative est préférée à une autre. Rossi *et al.* (2004) considèrent ensuite différentes stratégies de vote, comme la pluralité, la majorité, le consensus. Comme les tests de dominance possèdent un coût élevé, la complexité du processus de vote est presque toujours exponentielle (voir Rossi *et al.* (2004) pour les détails).

1.3.5 GAI-Nets

Les réseaux GAI ou GAI-Nets sont des modèles graphiques qui représentent directement des décompositions GAI (Gonzales et Perny, 2004). Ces modèles graphiques sont similaires aux graphes de jonction utilisés pour les réseaux bayésiens (Jensen, 1996).

Définition 1.19. (Gonzales *et al.*, 2007) Soit $\mathcal{A} = \times_{i=1}^{n} A_i$. Soit $C_1, \ldots, C_k$ des sous-ensembles de $N = \{1, \ldots, n\}$, tels que $N = \bigcup_{i=1}^{k} C_i$. Supposons que $\succsim$ est modélisable

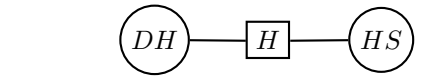
par une utilité GAI $u(a) = \sum_{i=1}^{k} u_i(a_{C_i})\ \forall a \in \mathcal{A}$. Alors un **réseau GAI** représentant $u(\cdot)$ est un graphe non-orienté $\mathcal{G} = (\mathcal{C}, \mathcal{E})$ qui satisfait les propriétés suivantes :

- Propriété 1 : $\mathcal{C} = \{A_{C_1}, \ldots, A_{C_k}\}$;
- Propriété 2 : $(A_{C_i}, A_{C_j}) \in \mathcal{E} \Rightarrow C_i \cap C_j \neq \emptyset$. $\forall A_{C_i}, A_{C_j}$ tels que $C_i \cap C_j = T_{ij} \neq \emptyset$, il existe un chemin dans $\mathcal{G}$ qui connecte X_{C_i} et X_{C_j} de façon à ce que tous ses nœuds contiennent tous les indices de T_{ij} (propriété d'intersection courante).

Les nœuds de $\mathcal{C}$ sont appelés *cliques*. Chaque arête $\langle \overline{A_{C_i}, A_{C_j}} \rangle \in \mathcal{E}$ est étiquetée par $A_{T_{ij}} = A_{C_i \cap C_j}$ et est appelée *séparateur*. Les cliques sont représentées sous forme d'ellipses, et les séparateurs sous forme de rectangles.

Pour toute décomposition GAI, selon la définition 1.19, les cliques du réseau GAI contiennent des sous-ensembles de variables. Les arêtes reliant les cliques, indiquent la présence de certains attributs dans plusieurs facteurs. Autrement dit, les arêtes représentent des intersections entre les cliques. Or, l'intersection étant une opération commutative, il convient de représenter le réseau GAI par un graphe non-orienté. Notons que cela contraste avec les UCP-Nets où les relations de dépendances entre les facteurs sont conditionnelles et justifient l'utilisation de graphes orientés.

Exemple 1.19. La figure 1.8 illustre un réseau GAI pour l'exemple du choix de vols. Ce réseau correspond à la fonction GAI $u(d, h, s, a) = u_1(d, h) + u_2(h, s) + u_3(a)$. Dans l'exemple, à l'alternative $(d^{1j}, h^d, s^a, a^{TAM})$ correspond l'utilité $u(d^{1j}, h^d, s^a, a^{TAM}) = u_1(d^{1j}, h^d) + u_2(h^d, s^a) + u_3(a^{TAM}) = 140 + 3 + 1 = 144$. Notez qu'à ce réseau GAI correspondent les mêmes valeurs d'utilité que dans l'exemple avec un UCP-Net. Toutefois le terme redondant sur D a été éliminé (en intégrant ses valeurs d'utilité au terme sur (D, H)).



	h^d	h^n
d^{1j}	140	120
d^{2j}	70	90

	s^e	s^a
h^d	8	3
h^n	3	8

a^{TAM}	a^{TAP}
1	0

FIG. 1.8 – Un GAI-Net pour l'exemple du choix de vols

Exemple 1.20. Si $u(a, b, c, d, e, f, g) = u_1(a, b) + u_2(c, e) + u_3(b, c, d) + u_4(b, d, f) + u_5(b, g)$ alors, comme le montre la figure 1.9, les cliques sont : AB, CE, BCD, BDF et BG. Par la propriété 2 de la définition 1.19, l'ensemble d'arêtes d'un réseau GAI peut être déterminé par des algorithmes qui préservent la propriété d'intersection courante (voir la littérature sur les réseaux bayésiens Cowell *et al.* (1999)).

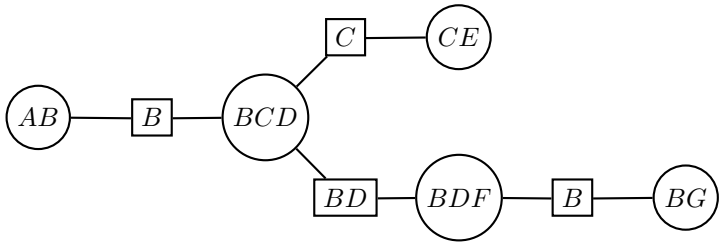


FIG. 1.9 – Un GAI-Net

Quand le réseau GAI a la structure d'un arbre, il est possible d'utiliser des algorithmes de résolution efficaces, fondés sur des passages de messages locaux entre des cliques voisines (voir le chapitre 2). Des arbres GAI ont aussi été utilisés pour guider l'élicitation de préférences dans l'incertain, à travers une séquence de questions concernant des loteries simples qui capturent efficacement les caractéristiques principales de l'attitude du décideur vis-à-vis du risque (Gonzales et Perny, 2004).

Ainsi, nous nous intéresserons davantage à des arbres GAI. Ceci n'est pas très restrictif puisque des réseaux GAI généraux peuvent toujours être recompilés dans des arbres GAI (avec possibilité d'avoir de plus grandes cliques).

Pour transformer un réseau GAI en arbre GAI, on construit d'abord son *réseau de Markov*. Soit $\mathcal{G} = (\mathcal{C}, \mathcal{E})$ un réseau GAI. Le **réseau de Markov** de $\mathcal{G}$ est un graphe non-orienté $G = (V, E)$ où $V = \{A_1, \ldots, A_n\}$ et E est tel qu'il y a une arête (A_i, A_j) entre toutes paires d'attributs A_i, A_j qui paraissent dans une même clique dans $\mathcal{G}$. Ensuite, on transforme G dans un graphe triangulé. Un graphe est triangulé si tout cycle de taille 4 ou plus possède une corde. Une triangulation d'un graphe est complètement définie à partir d'un ordre d'élimination des nœuds σ. Par un ordre d'élimination des nœuds d'un graphe $G = (V, E)$, on entend une bijection $\sigma : V \rightleftharpoons \{1, \ldots, |V|\}$. Un algorithme de triangulation procède de la manière suivante :

- Soit σ un ordre d'élimination des nœuds d'un graphe $G(V, E)$. Nous traitons chaque nœud $v \in V$ dans l'ordre σ de la façon suivante :
  - Soit $Adj(v)$ l'ensemble de nœuds adjacents à v dans G.
  - Ajouter des arêtes entre les nœuds $v' \in Adj(v)$ jusqu'à ce que le sous-graphe induit par les nœuds $v \cup Adj(v)$ soit complet.
  - Soit $T_{\sigma(v)}$ l'ensemble d'arêtes ajoutées.
  - Retirer v ainsi que toutes les arêtes du graphe qui contiennent ce v.
- Créer le graphe triangulé en ajoutant l'ensemble d'arêtes $T = \{T_1 \cup \ldots \cup T_{|V|}\}$ au graphe G initial.

Un arbre GAI peut être obtenu à partir du graphe triangulé de façon à ce que ses nœuds soient les cliques maximales du graphe triangulé, et les arêtes forment un arbre

entre les cliques, respectant la *propriété d'intersection courante*. Pour obtenir un arbre qui respecte la *propriété d'intersection courante*, on utilise le théorème suivant :

Théorème 1.3. *(Shibata, 1988; Jensen et Jensen, 1994) Soit G un graphe triangulé avec l'ensemble de cliques maximales $\mathcal{C} = \{A_{C'_1}, \ldots, A_{C'_m}\}$. Un graphe de jonction $G'(V, E)$ pour G est tel que : $V = \mathcal{C}$, $E = \{(A_{C'_i}, A_{C'_j}) | A_{C'_i} \cap A_{C'_j} \neq \emptyset\}$. Le coût d'une arête $(A_{C'_i}, A_{C'_j}) \in E$ est défini comme $|A_{C'_i} \cap A_{C'_j}|$. Dans ces conditions, les deux affirmations qui suivent sont équivalentes :*

1. *T est un arbre de jonction sur $\mathcal{C}$;*
2. *T est un arbre couvrant de coût maximal sur G'.*

Ainsi, nous pouvons simplement utiliser un algorithme d'arbre couvrant de coût maximal, pour trouver la structure de l'arbre GAI. Pour finir, chaque sous-fonction d'utilité est placée dans une clique qui contient tous les attributs de son domaine.

Exemple 1.21. * Soit $u(a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p) = u_1(a, b, c) + u_2(b, c, d) + u_3(c, d, e) + u_4(b, d, m) + u_5(f, d) + u_6(f, h) + u_7(h, j, k) + u_8(h, k, o) + u_9(e, g) + u_{10}(g, i, n) + u_{11}(i, l, p)$. Le réseau de Markov correspondant à cette fonction d'utilité est représenté dans la figure 1.10.

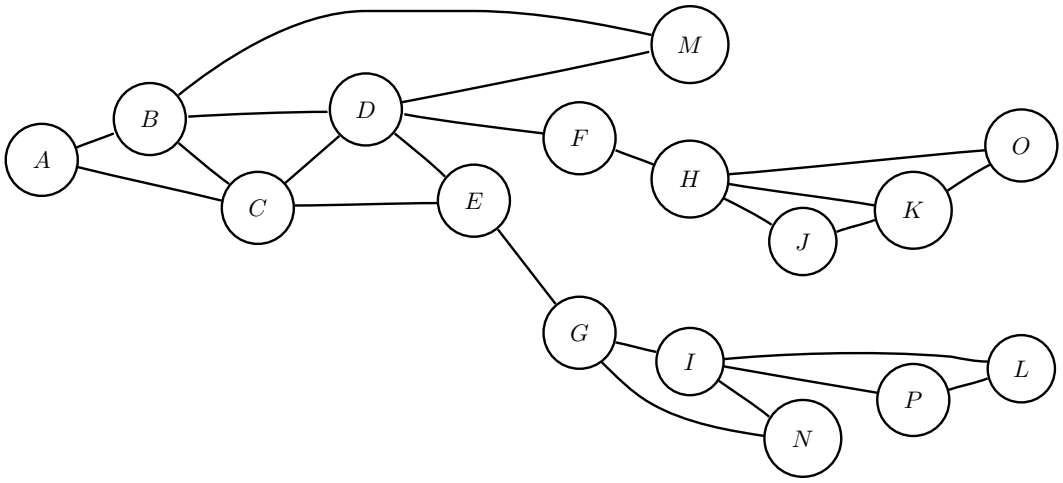


FIG. 1.10 – Réseau de Markov de $u(\cdot)$

La figure 1.11 montre le graphe triangulé selon l'ordre d'élimination des nœuds : L, J, K, I (les arêtes $\langle \overline{N, P} \rangle$ et $\langle \overline{G, P} \rangle$ sont ajoutées), H (l'arête $\langle \overline{F, O} \rangle$ est ajoutée), A, C (l'arête $\langle \overline{B, E} \rangle$ est ajoutée), D (les arêtes $\langle \overline{M, E} \rangle$, $\langle \overline{M, F} \rangle$, $\langle \overline{B, F} \rangle$ et $\langle \overline{E, F} \rangle$ sont ajoutées), P, G (l'arête $\langle \overline{E, N} \rangle$ est ajoutée), O, N, F, E, M, B. Les cliques maximales de ce graphe sont ABC, $BCDE$, $BDEFM$, EGN, FHO, $GINP$, HKO, ILP, HJK. La

*adapté de Jensen *et al.* (1994)

figure 1.12a montre le graphe de jonction pour le graphe triangulé. Notons que ce graphe possède deux arbres de jonction (des arbres couvrants avec un coût maximale de 15) comme le signalent les figures 1.12b et 1.12c. Puisque les arbres de jonction préservent la propriété de l'intersection courante, nous pouvons construire un arbre GAI directement à partir d'eux. Notons que la transformation du modèle en arbre GAI a résulté dans une clique maximale de taille 5, alors qu'initialement la sous-fonction d'utilité maximale de u impliquait 3 termes.

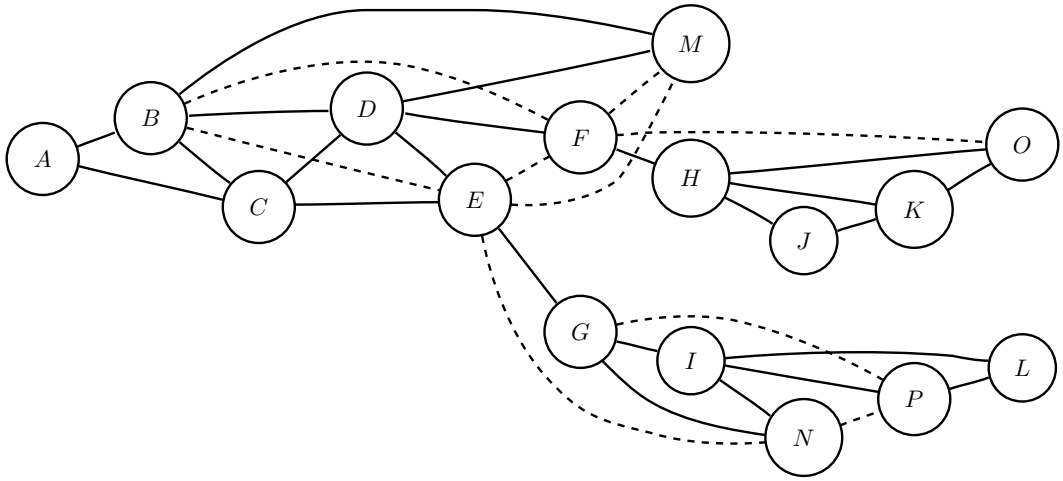
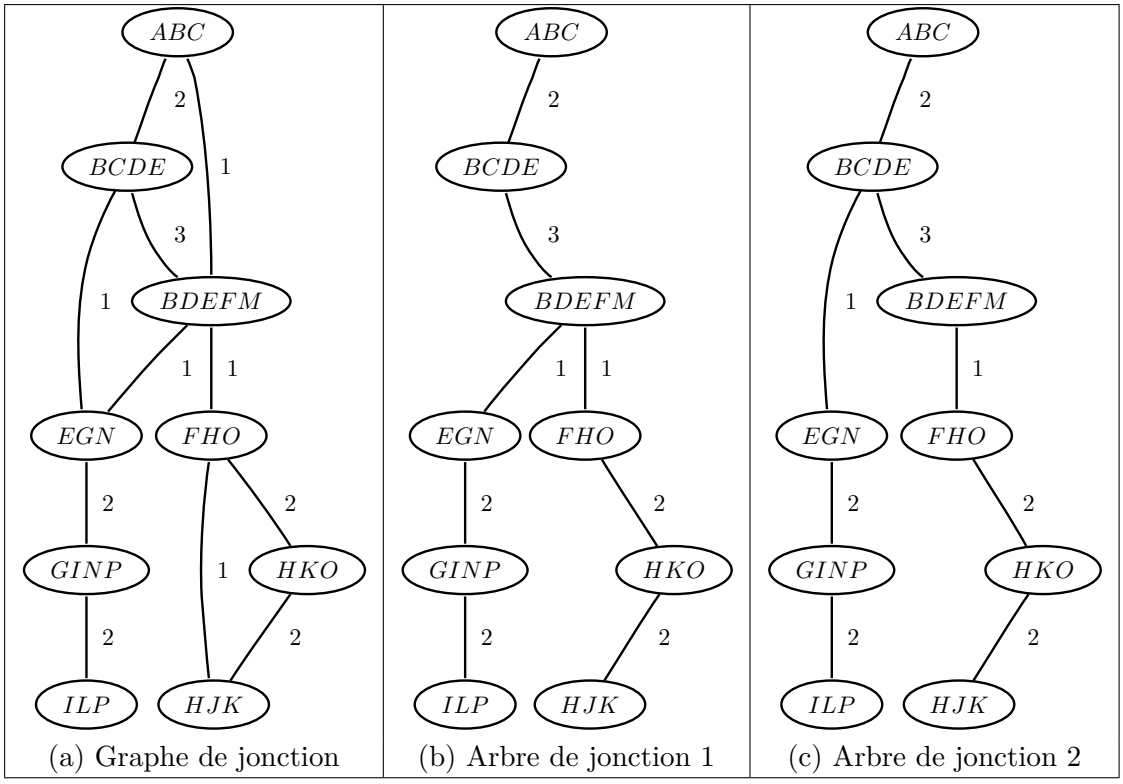


FIG. 1.11 – Graphe triangulé. Les arêtes ajoutées lors de la triangulation sont affichées en pointillés

Puisque la complexité de représenter et de manipuler une clique est proportionelle au produit cartésien des domaines de ses variables, nous avons intérêt à utiliser un ordre d'élimination qui résulte en une triangulation ayant une clique maximale de taille minimale. Il s'avère que le problème de trouver un ordre d'élimination qui induit un graphe triangulé ayant une clique maximale de taille minimale est un problème NP-complet (Arnborg *et al.*, 1987). Ainsi, les algorithmes tractables de triangulation font appel à des heuristiques pour déterminer l'ordre d'élimination à utiliser. Par exemple, Kjærulff et A/s (1990) décrivent une méthode heuristique pour l'obtention d'un ordre d'élimination, fondée sur un critère glouton de minimisation du nombre d'arêtes à ajouter lors de l'élimination d'un nœud v. À chaque étape, v est choisi de telle sorte que le nombre d'arêtes qui doit être ajouté pour lier les nœuds $v' \in Adj(v)$ soit minimal. Cette méthode a obtenu de bons résultats lors de tests pratiques, bien qu'il soit possible de trouver des cas pathologiques où l'ordre d'élimination obtenu est loin d'être optimal.



(a) Graphe de jonction (b) Arbre de jonction 1 (c) Arbre de jonction 2

FIG. 1.12 – Graphe de jonction et arbres de jonction pour le graphe triangulé.

1.3.6 L'intérêt de différentes représentations graphiques

Comme nous l'avons vu au cours de ce chapitre, chaque représentation graphique apporte ses avantages et ses inconvénients :

- les GAI-Nets ne se limitent pas à l'expression de préférences *ceteris paribus* et peuvent décrire n'importe quel préordre total sur un espace combinatoire, ce qui n'est pas possible avec des CP-Nets, des TCP-Nets ou des UCP-Nets ;
- les tables d'utilités locales exprimées dans les GAI-Nets (et de manière plus restrictive dans les UCP-Nets aussi, en raison du besoin de respect de l'indépendance conditionnelle) permettent de quantifier la valeur d'un n-uplet et donc de traduire des intensités de préférences, ce qui n'est pas possible dans un CP-net ou TCP-net ;
- les GAI-Nets et UCP-Nets ne permettent pas de décrire des structures de préférences partielles*, alors que les CP-Nets et TCP-Nets permettent de décrire certaines d'entre elles ;
- les CP-Nets traduisent des structures de préférences plus simples, donc plus faciles à éliciter (surtout quand les attributs ont des domaines binaires). L'élicitation des

*Sauf à valuer la fonction d'utilité dans un ensemble partiellement ordonné, ce que nous ne traitons pas ici.

TCP-Nets et des UCP-Nets restent plus simples que celle des GAI-Nets ;

- les questions de dominance du type $C \models a \succ a'$ (où C est le modèle graphique, a et a' sont deux alternatives) peuvent être résolues en temps $O(k)$ pour les UCP-Nets et les GAI-Nets (où k est le nombre de nœuds du graphe), tandis que pour les CP-Nets et les TCP-Nets le problème peut être NP-complet ou même PSPACE-complet.
- la complexité pour répondre aux questions d'optimisation dépend de la structure des préférences. Pour des graphes acycliques, les CP-Nets et leurs extensions ont des algorithmes polynomiaux, s'il n'y a pas de contraintes sur les configurations possibles. Pour les autres cas, le problème est NP-complet. Pour les GAI-Nets, le problème est d'ordre exponentiel au nombre d'attributs de la plus grande clique.
- dans le cadre de l'agrégation multicritère, l'emploi d'utilités peut nous permettre d'échapper à un problème d'agrégation ordinale où des conséquences impliquées par des résultats comme le théorème d'impossibilité d'Arrow, rendent plus difficile le processus d'agrégation (Pini *et al.*, 2005).

Ces systèmes de représentation sont donc complémentaires. On préfèrera l'un ou l'autre suivant le niveau d'information dont on dispose sur les préférences de l'individu, la problématique désirée et le degré de sophistication que l'on souhaite atteindre dans les résultats, et aussi le temps dont on dispose pour éliciter les préférences. Il est encore possible que le décideur veuille représenter certaines préférences de manière qualitative, et d'autres de manière quantitative. Considérant les cas où la coexistence entre les préférences des deux types dans le même système est nécessaire, ainsi que le transfert d'information d'une représentation à l'autre, des méthodes pour compiler des préférences qualitatives en fonctions d'utilité ont aussi été proposées (McGeachie et Doyle, 2004).

1.4 Positionnement du travail

Dans ce chapitre nous avons introduit les concepts de base utilisés en théorie de la décision. Nous avons également réalisé un survol des modèles graphiques récemment développés dans le domaine de modélisation et représentation de préférences, notamment les CP-Nets (ainsi que leurs extensions) et les GAI-nets. Nous avons remarqué que l'adéquation de chacun de ces modèles à un problème varie en fonction de plusieurs facteurs tels que le niveau d'information dont on dispose sur les préférences du décideur, la problématique désirée, le degré de sophistication que l'on souhaite atteindre dans les résultats, ou encore le temps dont on dispose pour éliciter les préférences.

Dans ce travail, nous explorons l'utilisation de GAI-Nets pour le raisonnement avec des préférences modélisables par des fonctions GAI-décomposables dans des systèmes interactifs de recommandation dont les objets à recommander sont composés par des

attributs configurables, ce qui résulte en un espace d'alternatives avec une structure combinatoire. Nous supposons que cet espace soit suffisamment large pour que l'énumération des configurations possibles soit impraticable. Nous enfrentons ainsi deux difficultés simultanées :

- la structure non-classique (c'est-à-dire, non-additive) des préférences ;
- l'espace combinatoire des solutions potentielles.

Pour aborder ces deux problèmes simultanément, l'utilisation de GAI-Nets nous paraît spécialement appropriée, une fois que d'un coté, la modélisation des préférences par des fonctions GAI nous permet de traiter des préférences non-classiques et, d'autre coté, l'utilisation d'arbres de jonction nous possibilite le développement d'algorithmes efficaces pour des espaces combinatoires. Ainsi, dans le chapitre suivant, nous investiguons l'utilisation de GAI-Nets pour traiter deux problèmes clés pour un système de recommandation : *le choix* (détermination de la meilleure alternative) et *le rangement* (génération d'une liste des k-meilleures alternatives).

Toutefois, il n'est pas toujours le cas que l'on cherche une recommandation pour une seule personne, ou selon un seul critère d'évaluation. Il se peut que nous cherchons une bonne solution pour un groupe d'individus, ou selon plusieurs critères, possiblement conflictuels. Ainsi, dans le chapitre 3 nous nous penchons sur le problème d'agrégation de préférences représentées par des réseaux GAI. Nous proposons une méthode opérationnelle pour trouver des solutions Pareto-optimales (qui peuvent refléter des critères de compromis non-linéaires) quand l'ensemble de préférences à agréger est modélisé par un ensemble de fonctions d'utilité GAI.

Le cadre de préférences non-classiques sur des espaces combinatoires nous pose aussi des nouveaux défis pour l'élicitation de ces préférences. Dans le chapitre 4 nous soulignons ces difficultés, ainsi que des possibles approches pour les traiter.

Enfin, dans le chapitre 5 nous appliquons les diverses concepts développés tout au long de la thèse dans la création d'une application Web pratique pour fournir des recommandations de lieux à visiter dans une grande ville touristique.

Chapitre 2

Décision Individuelle avec des Réseaux GAI

« Une décision rapide est une décision dangereuse. »

Sophocle, auteur grec
(V^{e} siècle av. J.-C.)

Ce chapitre s'appuie en partie sur des résultats présentés dans Gonzales *et al.* (2007, 2008)

Résumé

Lorsque les préférences du décideur ont été traduites dans un modèle de représentation de préférences, celui-ci doit pouvoir être employé pour des tâches de recommandation. Pour les modèles GAI, les questions clés sont :

1. **questions globales de choix :** trouver la configuration préférée dans l'ensemble d'alternatives.
2. **questions de rangement :** déterminer et ordonner les k configurations préférées dans l'ensemble des alternatives.

Ce chapitre se focalise sur ces deux types de question. Nous présentons comment la structure des réseaux GAI permet d'organiser les calculs efficacement en se ramenant à des séquences d'optimisations locales qui sont exploitées par un algorithme d'élimination de variables. Cet algorithme est développé pour des questions de choix (section 2.3) et des questions de rangement (section 2.4). Ensuite nous traitons le cas où il y a des contraintes qui rendent invalides certaines parties du produit cartésien. Nous constatons que le cas où les contraintes n'ajoutent pas énormément de dépendances entre les attributs, peut être trivialement intégré dans la méthode précédemment utilisée (section 2.4.1). Enfin, nous développons un algorithme de branch-and-bound qui peut être utilisé quand les dépendances entre les variables créeraient des cliques de taille trop grande. L'algorithme est utilisé pour le problème du sac-à-dos, dans sa version non-décomposable (section 2.5). Les algorithmes sont testés sur des jeux de données réalistes.

2.1 Notation

Dans la suite nous utiliserons les notations additionelles suivantes :

- Si $\mathcal{G} = (\mathcal{C}, \mathcal{E})$ est un arbre GAI, $\mathcal{C} = \{X_{C_1}, \ldots, X_{C_k}\}$ représente l'ensemble des cliques de $\mathcal{G}$ ordonnées de l'extérieur vers l'intérieur, cela signifie que, pour tout i, si $\mathcal{G}_{\mathcal{C}_i}$ représente le sous-graphe de $\mathcal{G}$ induit par $\mathcal{C}_i = \mathcal{C} \backslash \{X_{C_j} : j < i\}$, alors $\mathcal{G}_{\mathcal{C}_i}$ est connexe. De plus, pour tout $i \in \{1, \ldots, |\mathcal{C}| - 1\}$, la clique X_{C_i} a exactement une clique voisine dans $\mathcal{G}_{\mathcal{C}_i}$, que l'on désignera par $X_{C_{n(i)}}$.
- Nous noterons $S_i = C_i \cap C_{n(i)}$ et $D_i = C_i \backslash S_i$. Ils correspondront donc au séparateur entre X_{C_i} et $X_{C_{n(i)}}$, et aux attributs de X_{C_i} n'appartenant pas à ce séparateur, respectivement.

2.2 Recommandation avec des réseaux GAI

Une fois les préférences du décideur traduites dans un modèle de représentation de préférences, il doit pouvoir être employé pour des tâches de recommandation. Plusieurs questions intéressantes peuvent être distinguées :

1. **questions globales de choix :** trouver le n-uplet a^* préféré dans l'ensemble $\mathcal{A}$.
2. **questions de choix contraints :** trouver le n-uplet a^* préféré dans l'ensemble $\mathcal{A}$ sous la contrainte que certains attributs prennent des valeurs spécifiques.
3. **questions de comparaison :** trouver quelle est l'alternative préférée par le décideur parmi deux alternatives données $(a, a') \in \mathcal{A} \times \mathcal{A}$.
4. **questions de rangement :** déterminer et ordonner les k n-uplets préférés dans l'ensemble $\mathcal{A}$.
5. **questions de valeur d'utilité :** attribuer une valeur d'utilité à une alternative a arbitrairement choisie dans $\mathcal{A}$.

Si on utilise des représentations fondées sur des fonctions d'utilité, comme les GAI-Nets, nous remarquons que les *questions de choix contraints* ne sont qu'un cas simplifié des *questions globables de choix* ; nous avons simplement fixé les valeurs affectées à un sous-ensemble des variables. Pour les *questions de valeur d'utilité*, les GAI-nets permettent que le mécanisme pour y répondre soit trivialement implémenté. Pour connaître la valeur d'utilité pour une alternative a il suffit de calculer $u(a) = \sum_{i=1}^{k} u_i(a_{C_i})$. Ceci nous permet également de répondre trivialement aux *questions de comparaison* : pour déterminer si une alternative a est préférée à une alternative a' il suffit de comparer $u(a)$ et $u(a')$.

Il nous reste alors les questions globales de choix et les questions de rangement. Si la taille du domaine de l'ensemble des alternatives est petite nous pouvons simplement calculer l'utilité de toutes les configurations possibles et les ordonner par ordre décroissant

d'utilité (c'est l'approche utilisée dans Brafman *et al.* (2004), par exemple). Toutefois l'énumération de tous les éléments du domaine est infaisable pour les domaines de grande taille à structure combinatoire.

Étant données les similitudes entre les réseaux GAI et les réseaux bayésiens, nous pouvons adapter les méthodes utilisées dans ces derniers pour les GAI-Nets. Ainsi, pour trouver la configuration maximale d'un modèle GAI, les méthodes d'élimination de variables (*variable elimination*, en anglais) seront des candidats naturels (Zhang et Poole, 1994; Dechter, 1999; Kask *et al.*, 2005). Dans les prochaines sections nous entendons présenter la manière dont la structure des réseaux GAI permet d'organiser les calculs efficacement, en se ramenant à des séquences d'optimisation locales qui sont exploitées par un algorithme d'élimination de variables. Dans un premier temps, nous nous concentrons sur des questions globales de choix et nous présenterons ensuite une procédure de rangement. Pour la clarté de la présentation, la procédure est d'abord illustrée sur un exemple « jouet » et, dans un deuxième temps, un algorithme général est présenté.

2.3 Choix optimal

Considérons le problème consistant à déterminer la configuration maximale pour l'ensemble $\mathcal{X} = A \times B \times C \times D \times E \times F \times G$ où $A = \{a^0, a^1, a^2\}$, $B = \{b^0, b^1\}$, $C = \{c^0, c^1\}$, $D = \{d^0, d^1\}$, $E = \{e^0, e^1, e^2\}$, $F = \{f^0, f^1\}$, $G = \{g^0, g^1\}$, et où les préférences du décideur sont représentées par la fonction d'utilité $u(a, b, c, d, e, f, g) = u_1(a, b) + u_2(c, e) + u_3(b, c, d) + u_4(b, d, f) + u_5(b, g)$ avec les valeurs d'utilité de la figure 2.1. Remarquons que l'utilité u est complètement caractérisée par seulement 32 valeurs tandis que le stockage de u en extension requiert $|\mathcal{X}| = 288$ entiers. Si les attributs étaient continus, les tables de la figure 2.1 seraient remplacées par des fonctions fournissant une représentation analytique de u. Cette décomposition correspond au réseau GAI illustré dans la figure 1.9 du chapitre 1.

$u_1(a,b)$	b^0	b^1
a^0	8	2
a^1	4	3
a^2	1	7

$u_2(c,e)$	e^0	e^1	e^2
c^0	6	3	5
c^1	3	4	0

$u_3(b,c,d)$	b^0		b^1	
	d^0	d^1	d^0	d^1
c^0	0	2	7	1
c^1	5	1	2	4

$u_4(b,d,f)$	b^0		b^1	
	f^0	f^1	f^0	f^1
d^0	4	2	5	8
d^1	3	8	9	0

$u_5(b,g)$	g^0	g^1
b^0	0	9
b^1	6	4

FIG. 2.1 – Valeurs d'utilité pour $u(\cdot)$

Trouver la configuration optimale correspond à résoudre le problème suivant : $\max_{a,b,c,d,e,f,g} u_1(a,b) + u_2(c,e) + u_3(b,c,d) + u_4(b,d,f) + u_5(b,g)$. Les propriétés ci-dessous peuvent être exploitées pour résoudre efficacement ce problème :

1. le « max » sur un ensemble de variables $X_1, \ldots, X_n$ de $u(X_1, \ldots, X_n)$, peut être décomposé comme $\max_{X_1} \max_{X_2} \ldots \max_{X_n} u(X_1, \ldots, X_n)$ où l'ordre des « max » n'a aucune importance ;
2. si $u(X_1, \ldots, X_n)$ peut être décomposé comme $f() + g()$ où $f()$ ne dépend pas de la variable X_i, alors $\max_{X_i}[f() + g()] = f() + \max_{X_i} g()$;
3. dans un réseau GAI, la propriété d'intersection courante garantit qu'une variable appartenant à une clique externe C et qui n'appartient pas à la clique voisine de C n'apparaît dans aucune autre clique du réseau GAI.

Les propriétés 2 et 3 suggèrent une stratégie dans laquelle, pour calculer l'utilité maximale, on maximise en jouant d'abord sur les variables figurant uniquement dans les cliques externes. On transmet ensuite les résultats à la clique voisine en éliminant les cliques externes. On itère ainsi ce processus de l'extérieur vers l'intérieur du réseau, jusqu'à ce que toutes les cliques soient éliminées. Dans l'exemple, on résout le problème d'optimisation :

$$\begin{aligned} \max_{b,c,d}[\ & u_3(b,c,d) + \max_f[u_4(b,d,f) + \max_g u_5(b,g)] \\ & + [\max_e u_2(c,e)] + [\max_a u_1(a,b)]] \end{aligned} \tag{2.1}$$

à travers les opérations suivantes :

1. dans la clique AB, calculer $u_1^*(b) = \max_{a \in A} u_1(a,b)$ pour tout $b \in B$;
2. dans la clique CE, calculer $u_2^*(c) = \max_{e \in E} u_2(c,e)$ pour tout $c \in C$;
3. dans la clique BG, calculer $u_5^*(b) = \max_{g \in G} u_5(b,g)$ pour tout $b \in B$;
4. dans la clique BDF, substituer $u_4(b,d,f)$ par $u_4(b,d,f) + u_5^*(b)$ pour tout n-uplet $(b,d,f) \in B \times D \times F$. Ensuite, calculer $u_4^*(b,d) = \max_{f \in F} u_4(b,d,f)$ pour tout n-uplet $(b,d) \in B \times D$;
5. dans la clique BCD, substituer $u_3(b,c,d)$ par $u_3(b,c,d) + u_1^*(b) + u_2^*(c) + u_4^*(b,d)$ pour tout n-uplet $(b,c,d) \in B \times C \times D$. Ensuite, calculer $\max_{b,c,d} u_3(b,c,d)$, l'utilité maximale du réseau GAI (34, dans l'exemple).

La figure 2.2 montre le contenu des u_i^* et u_i après les substitutions. À la fin de l'étape 5 nous avons calculé la valeur maximale de l'utilité, ici 34, définie par l'équation 2.1.

Au terme de cette phase de collecte de valeurs, on dispose donc de la valeur optimale de la fonction u sur $\mathcal{X}$. Pour déterminer à quelle configuration des attributs correspond cette valeur, il suffit de réaliser une phase d'affectation des attributs qui consiste à

	b^0	b^1
$u_1^*(b)$	8	7

	c^0	c^1
$u_2^*(c)$	6	4

	b^0	b^1
$u_5^*(b)$	9	6

$u_4(b,d,f)$	b^0		b^1	
	f^0	f^1	f^0	f^1
d^0	13	11	11	14
d^1	12	17	15	6

$u_4^*(b,d)$	b^0	b^1
d^0	13	14
d^1	17	15

$u_3(b,c,d)$	b^0		b^1	
	d^0	d^1	d^0	d^1
c^0	27	33	34	29
c^1	30	30	27	30

FIG. 2.2 – Contenu des u_i^* et u_i après les substitutions

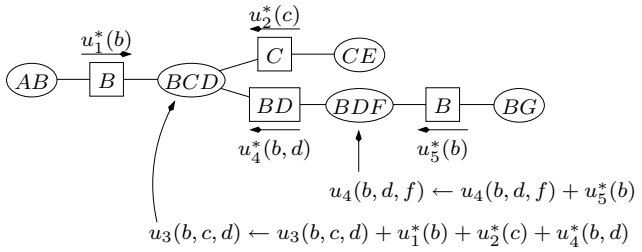


FIG. 2.3 – Étapes 1 à 5 pour calculer l'utilité à l'optimum.

propager en sens inverse du sens de la collecte, les arguments des calculs opérés. Ainsi, à la dernière étape de notre phase de collecte, on voit que l'utilité 34 correspond en u_3 au n-uplet (b^1, c^0, d^0), ce qui permet de déduire que, dans la configuration optimale, on a $B = b^1, C = c^0, D = d^0$. À l'étape 4, $u_4^*(b^1, d^0)$ correspond à $u_4(b^1, d^0, f^1) = 14$ ce qui implique que $F = f^1$. Ensuite, à l'étape 3, on constate que $u_5^*(b^1) = 6$ correspond à $u_5(b^1, g^0)$ et par conséquent $G = g^0$, ce qui achève de caractériser le n-uplet optimal. Pour procéder de manière efficace il faut, lors du calcul de chaque u_i^*, sauvegarder une fonction $M_i^* : X_{S_i} \mapsto X_{D_i}$, où X_{S_i} et X_{D_i} sont définis comme dans la section 2.1, M_i^* mémorisant les arguments pour lesquels u_i^* est atteint. Dans l'exemple :

1. $M_1^*(b) = \text{Argmax}_{a \in A} u_1(a, b)$ pour tout $b \in B$;
2. $M_2^*(c) = \text{Argmax}_{e \in E} u_2(c, e)$ pour tout $c \in C$;
3. $M_5^*(b) = \text{Argmax}_{g \in G} u_5(b, g)$ pour tout $b \in B$;
4. $M_4^*(b, d) = \text{Argmax}_{f \in F} u_4(b, d, f)$ pour tout $(b, d) \in B \times D$;
5. $\text{Argmax}_{b,c,d} u_3(b, c, d)$.

Ainsi, afin de trouver les valeurs des attributs pour la configuration optimale, il suffit de consulter les fonctions M_i^*. La phase d'instanciation reprend en sens inverse les étapes de la phase de collecte. Ainsi, dans l'exemple :

- *Étape 5 :* $\text{Argmax}_{b,c,d} u_3(b, c, d) = (b^1, c^0, d^0)$;
- *Étape 4 :* $M_4^*(b^1, d^0) = \text{Argmax}_{f \in F} u_4(b^1, d^0, f) = f^1$;
- *Étape 3 :* $M_5^*(b^1) = \text{Argmax}_{g \in G} u_5(b^1, g) = g^0$;
- *Étape 2 :* $M_2^*(c^0) = \text{Argmax}_{e \in E} u_2(c^0, e) = e^0$;

- *Étape 1 :* $M_1^*(b^1) = \text{Argmax}_{a \in A} u_1(a, b^1) = a^2$.

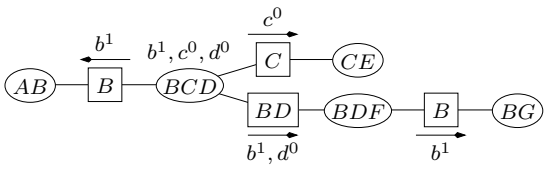


FIG. 2.4 – Étapes 5 à 1 pour calculer l'instanciation à l'optimum

La configuration optimale est donc $(a^2, b^1, c^0, d^0, e^0, f^1, g^0)$.

La fonction `Optimal_choice` ci-dessous implante la procédure utilisée pour trouver la configuration maximale. Son principe est similaire à celui utilisé pour trouver l'explication la plus probable dans un réseau bayésien (Jensen, 1996; Cowell *et al.*, 1999). L'algorithme possède deux phases : d'abord la fonction `Collect` qui calcule les utilités maximales, et ensuite la fonction `Instantiate` qui calcule la configuration optimale correspondante. Dans les fonctions ci-dessous, nous supposerons que l'ensemble des cliques $\mathcal{C} = \{X_{C_1}, \ldots, X_{C_k}\}$ est ordonné dans l'ordre inverse des appels de la fonction `Collect` (ce qui garantit que les cliques sont effectivement ordonnées de l'extérieur vers l'intérieur). X_{C_k} est donc la dernière clique de $\mathcal{C}$. Enfin, dans la fonction `Optimal_choice`, la valeur de x_{C_k} passée en paramètre lors de l'appel à `Instantiate` est une instanciation quelconque des attributs de X_{C_k}.

Entrées : Un arbre GAI $\mathcal{G}$ avec un ensemble de cliques ordonnées $\mathcal{C} = \{X_{C_1}, \ldots, X_{C_k}\}$
Sorties : La configuration optimale x^*

```
1 Collect(X_{C_k}, X_{C_k})
2 x* ⟵ Instantiate(X_{C_k},X_{C_k},x_{C_k},∅)
```

Fonction `Optimal_choice`

`Collect` effectue la collecte en X_{C_i} tout en évitant la clique X_{C_r}. Cela permet de ne pas boucler indéfiniment sur la boucle 1-6 de `Collect`.

La fonction `Instantiate` réalise l'instanciation des attributs. À l'instar de la fonction `Collect`, son second argument, X_{C_r}, évite que la boucle 8-9 soit infinie. L'argument $x^*_{C_r}$ contient les valeurs optimales des attributs de X_{C_r}. Enfin, *Forbidden* désigne un ensemble de configurations interdites de la clique X_{C_i}. Ce champ ne nous servira que pour le rangement, c'est-à-dire dans la section suivante.

Notons qu'à la place d'utiliser une méthode fondée sur l'élimination de variables, nous pourrions avoir utilisé une méthode de recherche, comme il est plus habituel dans le domaine de contraintes souples (voir par exemple Larrosa et Schiex (2003); de Givry *et al.* (2005)). Pourtant, en utilisant une méthode fondée sur l'élimination de variables,

Entrées : X_{C_i}, la clique à partir de laquelle effectuer la collecte ; X_{C_r}, la clique à éviter

pour tous les *cliques* $X_{C_j} \in \{$*cliques adjacentes à* $X_{C_i}\} \backslash \{X_{C_r}\}$ **faire**
  `Collect(`X_{C_j}, X_{C_i}`)`
  pour chaque $x_{C_i} \in X_{C_i}$ **faire**
    $u_i(x_{C_i}) \leftarrow u_i(x_{C_i}) + u_j^*(x_{S_j})$, où x_{S_j} est la projection de x_{C_i} sur $X_{C_i \cap C_j}$
  fin
fin
si $X_{C_i} \neq X_{C_r}$ **alors**
  pour chaque $x_{S_i} \in X_{S_i}$ **faire** $u_i^*(x_{S_i}) \leftarrow \max_{x_{D_i}} u_i(x_{C_i})$
fin

Procédure `Collect`

Entrées : X_{C_i}, la clique à calculer la configuration optimale ; X_{C_r}, une clique voisine ; $x^*_{C_r}$, les instantiations des variables de la clique voisine ; *Forbidden*, instantiations interdites
Sorties : Une instantiation des variables, *solution*

si $X_{C_i} = X_{C_r}$ **alors**
  $x^*_{C_i} \leftarrow \text{Argmax}\{u_i(x_{C_i}) : x_{C_i} \in X_{C_i} \setminus \textit{Forbidden}\}$
sinon
  $x^*_{S_i} \leftarrow$ la projection de $x^*_{C_r}$ sur $X_{C_i \cap C_r}$
  $x^*_{C_i} \leftarrow \text{Argmax}\{u_i(x^*_{S_i}, x_{D_i}) : x_{D_i} \in X_{D_i}$ et $(x^*_{S_i}, x_{D_i}) \notin \textit{Forbidden}\}$
fin
solution $\leftarrow x^*_{C_i}$
pour tous les *cliques* X_{C_j} *dans* $\{$*cliques adjacentes à* $X_{C_i}\} \backslash \{X_{C_r}\}$ **faire**
  $x^*_{C_j} \leftarrow$ `Instantiate(`X_{C_j}, X_{C_i}, $x^*_{C_i}$, $\emptyset$`)`
  solution $\leftarrow$ (*solution*, $x^*_{C_j}$)
fin
retourner *solution*

Fonction `Instantiate`

nous avons à la fin du processus de collecte **compilé*** une masse d'informations qui nous permettra de trouver rapidement les solutions suivantes dans le cadre de la problématique du rangement, comme nous le verrons dans la prochaine section.

2.4 Rangement

Pour réaliser un rangement des meilleures solutions il faut, après avoir déterminé le n-uplet optimal comme indiqué dans la section précédente, être capable de trouver le

*Nous utilisons ce terme en analogie à « knowledge compilation », obtenue grâce à l'utilisation d'une méthode d'élimination de variables dans le cadre de la résolution de problèmes binaires de satisfiabilité propositionnelle (SAT). Cette *knowledge compilation* permet de trouver des modèles satisfiables de manière très efficace, une fois que le premier modèle a été trouvé. (Rish et Dechter, 2000)

deuxième meilleur élément, puis le suivant, etc.

Nilsson (1998) a proposé un algorithme *diviser pour régner* pour trouver les k solutions les plus probables d'un réseau bayésien à partir de la construction itérative de partitions de l'ensemble des alternatives. Supposons que nous avons trouvé les $j-1$ meilleures alternatives $a^1, a^2, \ldots, a^{j-1}$ et l'ensemble $\mathcal{A}\backslash\{a^1, a^2, \ldots, a^{j-1}\}$ a été découpé en m sous-ensembles non-nuls $\mathrm{E}1, \ldots, \mathrm{E}m$, c'est-à-dire $\cup_{i=1}^{m} \mathrm{E}i = \mathcal{A}\backslash\{a^1, a^2, \ldots, a^{j-1}\}$ et $\forall k, l \in \{1, \ldots, m\}\ \mathrm{E}k \cap \mathrm{E}l = \emptyset$. Dans ces conditions, la j-ième meilleure alternative sera celle qui est la meilleure parmi les meilleures alternatives des m sous-ensembles. Ainsi, nous cherchons deux qualités (conflictuelles) dans les partitions :

1. Les sous-ensembles de la partition doivent être faciles à évaluer, c'est-à-dire la configuration $a^* \in \mathrm{E}i$ telle que $u(a^*) = \mathrm{Argmax}_{a \in \mathrm{E}i} u(a)$ doit être facile à obtenir.
2. Le nombre d'ensembles m doit être le plus petit possible.

Retournons à l'exemple antérieur, où la fonction `Optimal_choice` nous a renvoyé la meilleure solution $x^* = (a^2, b^1, c^0, d^0, e^0, f^1, g^0)$. La deuxième meilleure solution, x^2, diffère de x^* en au moins un attribut, autrement dit il y a une clique X_{C_i} telle que la projection de x^2 sur X_{C_i} diffère de celle de x^*. Comme nous ne savons pas dans quelle clique X_{C_i} se produit cette différence, on considère toutes les possibilités en divisant l'espace d'alternatives restantes selon la partition suivante :

- E1 : $(B, C, D) \neq (b^1, c^0, d^0)$;
- E2 : $(B, C, D) = (b^1, c^0, d^0)$ et $(B, D, F) \neq (b^1, d^0, f^1)$;
- E3 : $(B, C, D, F) = (b^1, c^0, d^0, f^1)$ et $(B, G) \neq (b^1, g^0)$;
- E4 : $(B, C, D, F, G) = (b^1, c^0, d^0, f^1, g^0)$ et $(C, E) \neq (c^0, e^0)$;
- E5 : $(B, C, D, E, F, G) = (b^1, c^0, d^0, e^0, f^1, g^0)$ et $(A, B) \neq (a^2, b^1)$.

Trouver le choix optimal dans l'ensemble E1 revient à résoudre :

$$\max_{(b,c,d) \neq (b^1, c^0, d^0)} [\ u_3(b, c, d) + \max_f (u_4(b, d, f) + \max_g u_5(b, g)) + (\max_e u_2(c, e)) + (\max_a u_1(a, b))\ ].$$

Cela suggère d'utiliser la fonction `Collect` comme dans la section précédente et ensuite d'appeler `Instantiate` en interdisant le choix de (b^1, c^0, d^0) dans l'étape 5 : c'est-à-dire qu'on appelle `Instantiate`$(BCD, BCD, (b^1, c^0, d^0), \{(b^1, c^0, d^0)\})$. De la même manière, pour trouver le choix optimal dans l'ensemble E2 on doit résoudre :

$$u_3(b^1, c^0, d^0) + \max_{f \neq f^1} [u_4(b^1, d^0, f) + \max_g u_5(b^0, g)] + [\max_e u_2(c^0, e)] + [\max_a u_1(a, b^0)].$$

Cela consiste à appeler la fonction `Collect` comme précédemment, et ensuite `Instantiate`$(BDF, BCD, (b^1, c^0, d^0), \{(b^1, d^0, f^1)\})$. Il est clair que cet appel n'affectera que les cliques BDF et BG, les autres cliques n'étant pas appelées par la fonction puisque, BCD étant passé en 2ème argument, la ligne 8 de `Instantiate` interdira tout appel récursif passant par la clique BCD. Pour former le n-uplet recherché, on affectera aux attributs de ces cliques les valeurs qu'ils avaient à l'optimum. Ainsi

`Instantiate`$(BDF, BCD, (b^1, c^0, d^0), \{(b^1, d^0, f^1)\})$ est précisément ce qui est nécessaire pour déterminer le choix optimal dans E2. Naturellement, cela se généralise aux autres ensembles et, dans notre exemple, nous obtenons :

- E1 : `Instantiate`$(BCD, BCD, (b^1, c^0, d^0), \{(b^1, c^0, d^0)\})$, $u(x) = 33$, $x = (a^0, b^0, c^0, d^1, e^0, f^1, g^1)$;
- E2 : `Instantiate`$(BDF, BCD, (b^1, c^0, d^0), \{(b^1, d^0, f^1)\})$, $u(x) = 31$, $x = (a^2, b^1, c^0, d^0, e^0, f^0, g^0)$;
- E3 : `Instantiate`$(BG, BDF, (b^1, d^0, f^1), \{(b^1, g^0)\})$, $u(x) = 32$, $x = (a^2, b^1, c^0, d^0, e^0, f^1, g^1)$;
- E4 : `Instantiate`$(CE, BCD, (b^1, c^0, d^0), \{(c^0, e^0)\})$, $u(x) = 33$, $x = (a^2, b^1, c^0, d^0, e^2, f^1, g^0)$;
- E5 : `Instantiate`$(AB, BCD, (b^1, c^0, d^0), \{(a^2, b^1)\})$, $u(x) = 30$, $x = (a^1, b^1, c^0, d^0, e^0, f^1, g^0)$.

Le deuxième n-uplet préféré est ainsi le choix optimal dans E1 ou E4. Arbitrairement, nous choisissons le n-uplet dans E1. Le prochain n-uplet, x^2, est le n-uplet préféré qui est donc différent de $(a^2, b^1, c^0, d^0, e^0, f^1, g^0)$ et de $(a^0, b^0, c^0, d^1, e^0, f^1, g^1)$. Il peut être recherché en utilisant le même processus. Comme x^1 est dans E1, nous devons découper E1 de la manière suivante pour exclure x^1 :

- E1.1 : $(B, C, D) \notin \{(b^1, c^0, d^0), (b^0, c^0, d^1)\}$;
- E1.2 : $(B, C, D) = (b^0, c^0, d^1)$ et $(B, D, F) \neq (b^0, d^1, f^1)$;
- E1.3 : $(B, C, D, F) = (b^0, c^0, d^1, f^1)$ et $(B, G) \neq (b^0, g^1)$;
- E1.4 : $(B, C, D, F, G) = (b^0, c^0, d^1, f^1, g^1)$ et $(C, E) \neq (c^0, e^0)$;
- E1.5 : $(B, C, D, E, F, G) = (b^0, c^0, d^1, e^0, f^1, g^1)$ et $(A, B) \neq (a^0, b^0)$;

et ensuite réitérer le même processus. Ce découpage peut être implanté grâce à la fonction `Split` ci-dessous : chaque ensemble de la partition courante est décrit par un quintuplet de la forme $(X_{C_i}, X_{C_r}, \mathit{Forbidden}_{C_i}, u^*, y^*)$, où X_{C_i} correspond à la clique sur laquelle on impose les contraintes d'exclusion, X_{C_r} est la clique intérieure voisine à X_{C_i}, $\mathit{Forbidden}_{C_i}$ désigne l'ensemble des $|C_i|$-uplets interdits pour la clique X_{C_i}, u^* est la valeur d'utilité de la configuration maximale dans cet ensemble, y^* est un n-uplet optimal lors de la création de cet ensemble. Par exemple, les ensembles E1 à E5 ci-dessus sont représentés par :

- E1 : $(BCD, \quad BCD, \quad \{(b^1, c^0, d^0)\}, \quad 33, \quad (a^2, b^1, c^0, d^0, e^0, f^1, g^0))$;
- E2 : $(BDF, \quad BCD, \quad \{(b^1, d^0, f^1)\}, \quad 31, \quad (a^2, b^1, c^0, d^0, e^0, f^1, g^0))$;
- E3 : $(BG, \quad BDF, \quad \{(b^1, g^0)\}, \quad 32, \quad (a^2, b^1, c^0, d^0, e^0, f^1, g^0))$;
- E4 : $(CE, \quad BCD, \quad \{(c^0, e^0)\}, \quad 33, \quad (a^2, b^1, c^0, d^0, e^0, f^1, g^0))$;
- E5 : $(AB, \quad BCD, \quad \{(a^2, b^1)\}, \quad 30, \quad (a^2, b^1, c^0, d^0, e^0, f^1, g^0))$.

Lorsque l'on veut découper un ensemble, comme par exemple E1, la fonction `Split` doit être appelée en passant en paramètre la clique X_{C_i} de l'ensemble, la clique intérieure voisine à X_{C_i}, le n-uplet optimal qui sera enlevé, l'utilité de ce n-uplet, et les $|C_i|$-uplets interdits.

Entrées : X_{C_i}, clique sur laquelle on impose les contraintes d'exclusion ; X_{C_r}, une clique voisine ; y^*, un n-uplet optimal dans l'ensemble considéré ; $u(y^*)$, l'utilité de y^* ; $\mathit{Forbidden}_{C_i}$, $|C_i|$-uplets interdits pour la clique X_{C_i}
Sorties : $\mathcal{S}$, une partition des alternatives restantes

$\mathit{Forbidden}_{C_i} \leftarrow \mathit{Forbidden}_{C_i} \cup \{y^*_{C_i}\}$
si $X_{C_i} = X_{C_r}$ **alors**
  $u^* = \max\{u_i(x_{C_i}) : x_{C_i} \in X_{C_i} \setminus \mathit{Forbidden}_{C_i}\}$
sinon
  $u^*_{x_{C_i}} \leftarrow \max\{u_i(y^*_{S_i}, x_{D_i}) : x_{D_i} \in X_{D_i}$ et $(y^*_{S_i}, x_{D_i}) \notin \mathit{Forbidden}_{C_i}\}$
  $u^* = u(y^*) - (u_i(y^*_{C_i}) - u^*_{x_{C_i}})$
fin
$\mathcal{S} \leftarrow \{(X_{C_i}, X_{C_r}, \mathit{Forbidden}_{C_i}, u^*, y^*)\}$
pour tous les *cliques* X_{C_j} *dans* {*cliques adjacentes à* X_{C_i}}\{X_{C_r}} **faire**
  $\mathcal{S} \leftarrow \mathcal{S} \cup$ `Split(`X_{C_j}, X_{C_i}, y^*, $u(y^*)$, $\emptyset$`)`
fin
retourner $\mathcal{S}$

Fonction `Split`

En utilisant la fonction `Split`, on peut définir la fonction de rangement `k-best`.

Entrées : $\mathcal{G}$, un arbre GAI ; K, le nombre de solutions désirées
Sorties : les k meilleures solutions $(x^*, x^2, \ldots, x^K)$

$x^* \leftarrow$ `Optimal_choice(`$\mathcal{G}$`)`
$\mathcal{S} \leftarrow$ `Split(`X_{C_k}, X_{C_k}, x^*, $u(x^*)$, $\emptyset$`)`
$i \leftarrow 2$
tant que $i \leq K$ *et* $\mathcal{S} \neq \emptyset$ **faire**
  $S \leftarrow$ l'élément $(X_{C_l}, X_{C_j}, \mathit{Forbidden}_{C_l}, u^*, y^*)$ de $\mathcal{S}$ ayant la plus grande valeur de u^*
  $x^i \leftarrow$ `Instantiate(`X_{C_l}, X_{C_j}, $y^*_{C_j}$, Forbidden$_{C_l}$`)`
  compléter les attributs manquants dans x^i avec leurs valeurs correspondantes dans y^*
  $\mathcal{S} \leftarrow \mathcal{S} \cup$ `Split(`X_{C_l}, X_{C_j}, x^i, u^*, Forbidden$_{C_l}$`)` $\setminus \{S\}$
  $i \leftarrow i + 1$
fin
retourner $(x^*, x^2, \ldots, x^K)$

Fonction `k-best`

La complexité de cet algorithme se déduit directement de celle donnée par Nilsson (1998). Soit :

- k = le nombre de cliques de l'arbre GAI ;
- K = la taille de la tête de classement recherchée (top-K éléments) ;
- $\bar{c} = \frac{\sum_{j=1}^{k} |X_{C_j}|}{k}$, et $|X_{C_j}| = |\{x_{C_j} \in X_{C_j}\}|$ représente le nombre de configurations d'une clique ;
- $\bar{d} = \frac{|X_{C_k}| + \sum_{j=1}^{k-1} |X_{D_j}|}{k}$ et $|X_{D_j}| = |\{x_{D_j} \in X_{D_j}\}|$.

La complexité de l'algorithme `k-best` peut être répartie dans les parties suivantes :

1. Trouver la meilleure solution (ligne 1 de `k-best`) ;
2. Trouver l'utilité maximale des sous-ensembles (lignes 2-7 de `Split`) ;
3. Trouver les sous-ensembles d'utilité maximale et mettre à jour la structure de données qui les stocke (lignes 5, 8 de `k-best`) ;
4. Trouver la configuration maximale à laquelle corresponde l'utilité maximale d'un sous-ensemble (lignes 6, 7 de `k-best`).

Ces étapes ont des coûts comme suivent :

1. Pour passer les valeurs d'utilité d'une clique C_i à sa clique voisine $C_{n(i)}$ (dans l'appel à `Optimal_choice`) on utilise $|C_i| + |C_{n(i)}|$ opérations. Par conséquent, le nombre d'opérations nécessaires pour trouver la meilleure solution x^* est moins de $2k\bar{c}$ opérations.
2. À chaque solution x^i, `k-best` crée au maximum k sous-ensembles (ligne 8). Ainsi, pour trouver les utilités maximales des sous-ensembles (lignes 2-7 de `Split`) créés pour K solutions demande au maximum $kK\bar{d}$ opérations.
3. À chaque étape i, on a au maximum ik sous-ensembles. Ainsi, le nombre d'opérations pour garder les sous-ensembles générés ordonnés par utilité pour $i = 1, ..., K$ est de $\mathrm{O}(kK \log kK)$ opérations.
4. Finalement, trouver la configuration maximale à laquelle corresponde l'utilité maximale d'un sous-ensemble, demande au maximum $k\bar{d}$ opérations. Ainsi, pour K solutions le nombre d'opérations nécessaires est au maximum $kK\bar{d}$.

En ajoutant les termes, on obtient que le nombre d'opérations pour générer les K meilleures solutions est d'ordre de :

$$2k\bar{c} + 2kK\bar{d} + kK \log kK$$

Notons que l'un des termes dominants de la complexité, $2kK\bar{d}$, est dépendant du nombre de solutions désirées, ce qui occasionne une dégradation de la performance par rapport à la croissance de K. Nous pouvons modifier `k-best` pour atteindre de meilleures performances pour des grandes valeurs de K. Dans les étapes 2 et 4, le coût

d'exécution est dû aux calculs des max étant données les affectations des séparateurs. Or, les expérimentations montrent que l'on calcule souvent avec les mêmes valeurs de séparateurs x_{S_i}. Cela suggère donc de préparer ces calculs en triant (sur demande) les portions d'utilités sur lesquelles portent ces calculs. Autrement dit, on trie les tables d'utilité d'une clique x_{C_i} par affectation de son séparateur x_{S_i}. Ainsi, les lignes 3 et 5 dans `Split` et les lignes 2 et 5 dans `Instantiate` peuvent être calculées en O(1) en utilisant des curseurs dans les tables triées. Avec cette modification, les coûts des étapes 2 et 4 passent de $kK\bar{d}$ à kK. Toutefois, dans le pire des cas (si l'on trie toutes les valeurs dans les cliques), on ajoute le terme de complexité $k\bar{c}\log\bar{c}$. Ainsi, le coût maximal de l'algorithme avec des tables d'utilité triées est d'ordre de :

$$2k\bar{c} + 2kK + kK\log kK + k\bar{c}\log\bar{c},$$

ce qui sera toujours avantageux dès que $K > \dfrac{\bar{c}\log\bar{c}}{2(\bar{d}-1)}$.

2.4.1 Intégration de contraintes

Jusqu'à présent, nous avons considéré que toutes les configurations du produit cartésien $\mathcal{X}$ étaient réalisables. Toutefois, dans de nombreuses situations, certaines de ces configurations sont impossibles ou indisponibles. Ces contraintes peuvent être intégrées de façon directe dans le modèle GAI en ajoutant des nouveaux termes d'utilité dans lesquels la valeur est 0 pour les configurations possibles et $-\infty$ dans le cas contraire. La fonction d'utilité joue alors le rôle des fonctions d'appartenance dans les CSP* flexibles (Schiex *et al.*, 1997), les préférences étant vues comme des contraintes flexibles et les contraintes de faisabilité comme des contraintes dures. Supposons que les paires (a^0, e^1), (a^1, e^2), (a^2, e^1) et (a^2, e^2) soient impossibles, on doit alors ajouter le terme d'utilité $u_6(a, e)$ avec les valeurs :

$$\begin{array}{lll} u_6(a^0,e^0) = -\infty; & u_6(a^0,e^1) = 0; & u_6(a^0,e^2) = 0; \\ u_6(a^1,e^0) = 0; & u_6(a^1,e^1) = 0; & u_6(a^1,e^2) = -\infty; \\ u_6(a^2,e^0) = 0; & u_6(a^2,e^1) = -\infty; & u_6(a^2,e^2) = -\infty; \end{array}$$

De même, si les paires (e^0, f^0) et (e^1, f^1) sont aussi interdites, on ajoute un terme d'utilité $u_7(e, f)$ avec les valeurs :

$$\begin{array}{lll} u_7(e^0,f^0) = -\infty; & u_7(e^1,f^0) = 0; & u_7(e^2,f^0) = 0; \\ u_7(e^0,f^1) = 0; & u_7(e^1,f^1) = -\infty; & u_7(e^2,f^1) = 0; \end{array}$$

*De l'anglais *Constraint Satisfaction Problem*

Après l'addition des termes d'utilité permettant d'exprimer les contraintes, on a de nouvelles cliques à gérer et il est nécessaire de construire un nouvel arbre de jonction. Pour cela, on retriangule le réseau résultant après l'addition de nouvelles cliques. La figure 2.5 montre l'arbre GAI de l'exemple après l'introduction des termes u_6 et u_7.

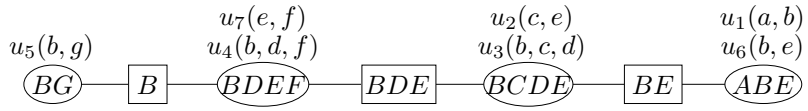


FIG. 2.5 – L'arbre GAI après l'introduction des contraintes

On remarque que les nouvelles dépendances entre variables introduites par u_6 et u_7 ont créé un arbre avec des cliques plus grandes qu'avant. Ceci demande plus d'espace pour stocker les fonctions relatives aux cliques. Toutefois il est possible d'utiliser une implantation « *lazy* », dans laquelle les utilités intervenant dans chaque clique ne sont pas fusionnées mais gardées dans leur forme originelle et stockées dans des listes chaînées (dans notre exemple, les termes d'utilité composant chaque clique sont indiqués sur la figure 2.5).

2.4.2 Expérimentations

Pour tester l'algorithme d'énumération des k-meilleures solutions, nous avons exécuté ce dernier sur différents jeux de données générés automatiquement. Les temps de calculs pour trouver le meilleur élément ainsi que les 50 et 100 meilleurs ont été enregistrés. On a créé 5 classes de problèmes $K_1, \ldots, K_5$, caractérisées par le nombre d'attributs, la taille du domaine de chaque attribut, le nombre de contraintes, la taille de la décomposition (c'est-à-dire le nombre de termes d'utilité), et l'arité des termes d'utilité (ainsi que des contraintes) :

	Nb. Attr.	Taille Dom.	Nb. Contr.	Taille decomp.	Arité des termes
K_1	40	2	5	5	4
K_2	35	5	30	30	2
K_3	15	10	20	20	2
K_4	30	5	5	5	4
K_5	10	10	5	5	3

Pour chaque classe on a généré aléatoirement 50 instances avec des nombres différents de n-uplets interdits par contrainte. Les temps d'exécution ont d'abord été pris sans tenir compte des contraintes (c'est-à-dire, toutes les configurations dans le produit cartésien étant valables). Ensuite les contraintes ont été introduites dans le réseau à travers de nouveaux termes d'utilité, le nouveau réseau retriangulé, et l'algorithme a utilisé le

Time	sans contraintes			avec contraintes		
(in ms)	Best	Top 50	Top 100	Best	Top 50	Top 100
K_1	7	18	29	40	55	67
K_2	1	3	6	121	132	142
K_3	1	3	4	2067	2070	2073
K_4	40	45	50	4399	4410	4419
K_5	663	664	665	12662	12663	12665

TAB. 2.1 – Temps d'exécution (en ms) pour les différentes classes de problème. Implémentation non-paresseuse

Time	sans contraintes				avec contraintes			
(in ms)	TBBest	Best	Top 50	Top 100	TBBest	Best	Top 50	Top 100
K_1	6153	8	44	82	2047	28	64	102
K_2	< 10	2	8	17	< 10	65	100	133
K_3	< 10	4	9	14	< 10	1272	1334	1365
K_4	35510	40	65	82	11257	2553	2647	2702
K_5	60	625	719	780	72	7858	7971	8062

TAB. 2.2 – Temps d'exécution (en ms) pour les différentes classes de problème. Implémentation paresseuse. TBBest est le temps pour le programme Toolbar

nouvel arbre de jonction induit par ce nouveau graphe. Les expérimentations ont été faites avec un programme en Java en utilisant un PC 2.4GHz. Les temps moyens sur les 50 instances sont résumés dans le tableau 2.1. Dans ce tableau, l'implémentation n'utilise pas une stratégie paresseuse.

Le tableau 2.2 montre les temps d'exécution pour l'implémentation paresseuse. On affiche aussi avec le temps que Toolbar* prend pour trouver la meilleure solution pour les mêmes problèmes (colonne TBBest). Toolbar est un logiciel de référence (librement disponible, implémenté en langage C) dans le domaine de contraintes souples et utilise des algorithmes basés sur recherche. On n'a pas pu utiliser Toolbar pour énumérer les k-meilleures solutions puisque il n'implémente pas cette fonctionnalité (ajouter des contraintes qui interdisent la meilleure solution déjà générée n'est pas possible non plus, une contrainte sur tous les attributs cause l'explosion de l'utilisation de mémoire).

Dans le cas de produits cartésiens complets, on peut remarquer que les algorithmes fondés sur les réseaux GAI sont très efficaces pour résoudre les problèmes de choix et de rangement. Nous notons que dans certains cas, l'approche fondée sur des algorithmes de recherche de Toolbar a été moins performante pour trouver la meilleure solution que notre approche fondée sur l'élimination de variables pour trouver un rangement des 100 meilleures solutions. Les méthodes fondées sur l'élimination de variables ont traditionellement été peu utilisées pour la résolution de problèmes avec contraintes, où

*`http://carlit.toulouse.inra.fr/cgi-bin/awki.cgi/ToolBarIntro`

des algorithmes de recherche sont plus amplement appliqués. Pourtant, Larrosa *et al.* (2005) ont récemment constaté, avec une étude de cas, que les premières méthodes (celles fondées sur l'élimination de variables) peuvent être supérieures à celles traditionnellement utilisées et même, dans certains cas, permettre la résolution de problèmes intraitables par les meilleurs algorithmes de recherche.

Quand des contraintes sont ajoutées, les temps de calcul demeurent satisfaisants en dépit de la complexité additionnelle. On peut remarquer, en particulier, que le rangement des éléments de la deuxième à la centième position ne prend pas plus de temps que l'obtention du premier élément. L'augmentation du temps de calcul observée après avoir ajouté des contraintes est causée par les dépendances inter-attributs additionnelles induites par ces mêmes contraintes. Ces nouvelles dépendances provoquent la réunion de cliques initialement disjointes, augmentant ainsi la taille moyenne des cliques, et dégradant du même coup les performances de l'algorithme de choix ou de rangement. Cependant, dans des applications pratiques, l'arité des facteurs de sous-utilité est généralement petite (≤ 3 dans la plupart des cas) ; le temps de calcul dépend donc essentiellement de l'arité des contraintes de faisabilité. Pour certains problèmes, les contraintes de faisabilité peuvent impliquer un grand nombre de variables, ce qui empêche leur intégration dans le GAI-net comme nous l'avons fait jusqu'ici. Dans de tels cas, nous pouvons garder les contraintes séparées du réseau et utiliser un algorithme mixte qui combine recherche et élimination de variables. Dans la prochaine section nous allons présenter une instance de cette méthode, en utilisant le problème du sac-à-dos comme cas d'étude.

2.5 Le sac-à-dos non-linéaire : une approche mixte avec des GAI-Nets

L'intégration de contraintes comme des sous-fonctions d'utilité dans le réseau GAI n'est possible que si les contraintes n'impliquent pas un grand nombre de variables. Dans le cas contraire, le nouvel arbre GAI aurait des cliques de taille trop importante après la triangulation du nouveau réseau, en raison des interdépendances entre un grand nombre de variables. Ainsi, dans de tels cas, nous pouvons envisager la combinaison des méthodes d'élimination de variables avec des méthodes de recherche, puisque l'espace nécessaire pour ces dernières est indépendant de la structure des dépendances du réseau. Dans le domaine des CSP valués, des algorithmes qu'utilisent des arbres de jonction combinés à des méthodes de recherche ont été proposés, notamment pour des contraintes binaires* (Terrioux et Jégou, 2003; de Givry *et al.*, 2006). Typiquement, le point de départ

*Il s'avère que tout ensemble de contraintes non-binaires avec des domaines finites peut être transformé dans un ensemble de contraintes binaires. Pourtant, dans la pratique, ceci peut produire des problèmes

de ces algorithmes est la construction de l'arbre de jonction du réseau de contraintes. La propriété qui ressort de l'arbre de jonction est, qu'une fois qu'un séparateur a été instantié, le problème initial est séparé en deux sous-problèmes qui peuvent être résolus séparément. Toutefois, nous ne pouvons pas compter sur cette propriété pour le problème du sac-à-dos ; comme nous allons le voir par la suite, ces problèmes possèdent des contraintes qui portent sur tout l'ensemble de variables, ce qui engendrerait un arbre avec une seule clique. Nous proposons donc une nouvelle approche qui adapte l'algorithme de rangement présenté dans la section précédente en le couplant avec une méthode de « branch and bound ».

2.5.1 Le problème du sac-à-dos non-linéaire GAI-décomposable

Les problèmes de sac-à-dos consistent à choisir un sous-ensemble parmi n objets, de telle sorte que l'utilité totale du sous-ensemble choisi soit maximisée sans dépasser la capacité du sac-à-dos (ou la capacité de chacun des m sacs-à-dos). Dans sa forme la plus classique, le *sac-à-dos 0–1*, chaque objet $j = 1, \ldots, n$ a une utilité associée u_j et un poids w_j, et on doit choisir un sous-ensemble des n objets de telle sorte que la somme des utilités des objets choisis soit maximisée et la somme de leur poids ne dépasse pas la capacité c d'un sac à dos. Cela suppose que les préférences sur les objets sont représentables par une fonction d'utilité additive $u : X_1 \times \ldots \times X_n \mapsto \mathbb{R}$ telle que $u(x_1, \ldots, x_n) = \sum_{j=1}^{n} u_j x_j$, où $x_j \in \{0, 1\}$ sont des variables binaires qui assument la valeur 1 si l'objet j doit être inclus dans le sac-à-dos, et 0 dans le cas contraire. Formellement, cela correspond au problème de maximisation suivant :

$$\begin{aligned} \text{maximiser} \quad & \sum_{j=1}^{n} u_j x_j \\ \text{sous la contrainte que} \quad & \sum_{j=1}^{n} w_j x_j \leq c\,, \\ & x_j \in \{0, 1\}, \quad j = 1, \ldots, n. \end{aligned} \tag{2.2}$$

Considérons maintenant un problème de sac-à-dos où l'on doit choisir les objets à mettre dans une mallette. Parmi les objets nous avons un *téléphone portable* et son *chargeur*. Sans le téléphone portable le chargeur est inutile, alors si l'on prend le chargeur sans le téléphone portable nous avons une utilité zéro. Mais, si l'on prend le téléphone portable sans le chargeur, cela a une certaine utilité, puisque nous pouvons effectuer des appels tant que la batterie du téléphone portable n'est pas déchargée. Enfin, en choisissant en même temps le téléphone portable et son chargeur, nous obtenons une

d'efficacité. (Bessière *et al.*, 2002; Bacchus *et al.*, 2002)

utilité supérieure à l'addition des utilités individuelles qui correspondent à choisir l'un d'entre eux et pas l'autre. Dans ce cas, l'utilité de l'ensemble d'objets ne se décompose plus additivement, seulement le poids reste additivement décomposable. Comme autre exemple, nous pouvons penser au problème qui se pose lorqu'ayant un budget limité, il nous faut choisir des projets à financer parmi un panel de projets. L'utilité totale des projets sélectionnés n'est pas nécessairement additivement décomposable, puisque nous pouvons avoir des projets complémentaires (de sorte qu'il peut être bon d'exploiter leur effet gestaltique en les choisissant ensemble), ainsi que des projets partiellement redondants (ce n'est donc pas une bonne idée de les choisir ensemble). Toujours dans ce cas, les poids peuvent être additifs (par exemple, si chaque projet a une proposition de budget fixe et indépendant).

Ce type de problème de sac-à-dos peut être formulé comme le problème de maximisation suivant :

$$\begin{aligned} \text{maximiser} \quad & u(x) \\ \text{sous la contrainte que} \quad & \sum_{j=1}^{n} w_j x_j \leq c\,, \\ & x = (x_1, x_2, \ldots, x_n) \in \{0,1\}^n. \end{aligned} \tag{2.3}$$

Nous supposerons que tous les coefficients w_j, ainsi que c sont des nombres réels non-négatifs et que $u(x)$ est une fonction telle que $u : \{0,1\}^n \mapsto \mathbb{R}$.

Lorsque la fonction d'utilité $u(x)$ n'est point additivement décomposable, le problème de sac-à-dos résultant est beaucoup plus difficile à résoudre que pour le cas additif, étant donné que nous ne pouvons pas exploiter les indépendances structurelles pour développer des algorithmes efficaces (Bretthauer et Shetty, 2002). L'existence d'une structure sous-jacente pour la fonction d'utilité est indispensable pour traiter les préférences dans des domaines complexes (Keeney et Raiffa, 1993).

Dans cette section nous traitons le problème du *sac-à-dos 0–1* comme nous l'avons précédemment formulé (équation 2.3), pour le cas où la fonction $u(x)$ est GAI-décomposable. Nous le nommerons « problème du sac-à-dos non-linéaire GAI-décomposable » (**NGKP**, de l'anglais *Nonlinear GAI-decomposable Knapsack Problem*). Nous développons un algorithme exact pour résoudre le NGKP, en couplant la méthode de rangement précédemment présentée avec une recherche heuristique du type branch-and-bound.

2.5.2 Un algorithme exact pour le NGKP

À partir de l'algorithme de rangement présenté dans la section 2.4, nous pouvons construire un algorithme de branch-and-bound pour des instances du NGKP.

Supposons que nous avons un sac-à-dos de capacité $c = 95$. Les préférences du décideur sur un ensemble de 7 objets sont données par la fonction d'utilité $u(a, b, c, d, e, f, g) = u_1(a, b) + u_2(c, e) + u_3(b, c, d) + u_4(b, d, f) + u_5(b, g)$. La figure 2.6 liste les valeurs d'utilité pour u, ainsi que le poids des objets. La structure de l'arbre GAI de l'exemple est la même que celle montrée dans la figure 1.9 (page 40).

$u_1(a,b)$	b^0	b^1
a^0	8	2
a^1	4	3

$u_2(c,e)$	e^0	e^1
c^0	6	3
c^1	3	4

$u_3(b,c,d)$	b^0		b^1	
	d^0	d^1	d^0	d^1
c^0	0	2	7	1
c^1	5	1	2	4

$u_4(b,d,f)$	b^0		b^1	
	f^0	f^1	f^0	f^1
d^0	4	2	5	8
d^1	3	8	9	0

$u_5(b,g)$	g^0	g^1
b^0	0	9
b^1	6	4

Objet	a	b	c	d	e	f	g
Poids	40	10	20	35	55	60	15

FIG. 2.6 – Valeurs d'utilité pour l'exemple. Pour un objet x, x^1 signifie que x a été inclus dans le sac-à-dos ($x = 1$), et x^0 qu'il n'a pas été inclus dans le sac-à-dos ($x = 0$)

Avant de commencer le processus de branch-and-bound, nous devons trouver une solution qui sera initialement la borne inférieure d'utilité. Afin de la trouver rapidement, nous pouvons trier les objets par ordre croissant de poids, comme nous le montre la figure 2.7. Nous remarquons que si nous n'ajoutons pas les objets a, e et f au sac-à-dos, toute combinaison des autres objets ne dépasse pas la capacité du sac-à-dos. Ainsi, la borne inférieure peut être la configuration d'utilité maximale qui exclut les objets a, e et f. Notons que, puisque l'utilité du sac-à-dos est non-décomposable (c'est-à-dire que nous n'avons pas une utilité associée à chaque objet), nous ne pouvons pas utiliser la relaxation continue du problème pour établir la borne initiale (Dantzig, 1957), l'approche classique pour les problèmes de sac-à-dos décomposables.*

Pour trouver cette borne inférieure, nous affectons les valeurs d'utilité de $u_1(1, b)$, $u_2(c, 1)$ et $u_4(b, d, 1)$ à $-\infty$ et, en utilisant cette fonction u modifiée, nous appelons la procédure `Collect`, qui résoudra l'expression :

$$\max_{b,c,d}\{\ \max_{a} u_1(a,b) + \max_{e} u_2(c,e) + \max_{f}[u_4(b,d,f) + \max_{g} u_5(b,g)] + u_3(b,c,d)\ \} \tag{2.4}$$

à travers les opérations suivantes :

- **Étape 1 :** dans la clique AB, calculer $u_1^*(b) = \max_{a \in A} u_1(a, b)$ pour tout $b \in B$;
- **Étape 2 :** dans la clique CE, calculer $u_2^*(c) = \max_{e \in E} u_2(c, e)$ pour tout $c \in C$;

*Toutefois, il serait possible d'utiliser la relaxation continue si nous approchons la fonction d'utilité $u(x)$ par une fonction d'utilité additive $g(x)$ telle que $\forall x g(x) < u(x)$. Nous reviendrons à ce point.

Objet	b	g	c	d	a	e	f
Poids	10	15	20	35	40	55	60
Poids cumulé	10	25	45	80	120	175	235

FIG. 2.7 – Objets triés par ordre croissant de poids. Dans la dernière ligne, nous montrons le poids cumulé si l'on inclut tous les objets du plus léger à l'objet courant dans le sac-à-dos

- **Étape 3 :** dans la clique BG, calculer $u_5^*(b) = \max_{g \in G} u_5(b, g)$ pour tout $b \in B$;
- **Étape 4 :** dans la clique BDF, remplacer $u_4(b, d, f)$ par $u_4(b, d, f) + u_5^*(b)$ pour tout $(b, d, f) \in B \times D \times F$. Ensuite calculer $u_4^*(b, d) = \max_{f \in F} u_4(b, d, f)$ pour tout $(b, d) \in B \times D$;
- **Étape 5 :** dans la clique BCD, remplacer $u_3(b, c, d)$ par $u_3(b, c, d) + u_1^*(b) + u_2^*(c) + u_4^*(b, d)$ pour tout $(b, c, d) \in B \times C \times D$. Calculer $\max_{b,c,d} u_3(b, c, d)$.

Les contenus des u_i^* et des u_i après les substitutions sont donnés dans la figure 2.8. À la fin de l'Étape 5, nous avons calculé la valeur de l'équation 2.4, et donc la valeur d'utilité maximale pour la fonction u modifiée (ici, 29).

	b^0	b^1
$u_1^*(b)$	8	2

	c^0	c^1
$u_2^*(c)$	6	3

$u_4(b,d,f)$	b^0		b^1	
	f^0	f^1	f^0	f^1
d^0	13	$-\infty$	11	$-\infty$
d^1	12	$-\infty$	15	$-\infty$

	b^0	b^1
$u_5^*(b)$	9	6

$u_4^*(b,d)$	b^0	b^1
d^0	13	11
d^1	12	15

$u_3(b,c,d)$	b^0		b^1	
	d^0	d^1	d^0	d^1
c^0	27	28	26	24
c^1	29	24	18	24

FIG. 2.8 – Contenus des u_i^* et des u_i après les substitutions (pour la fonction u modifiée)

À la valeur d'utilité maximale calculée dans l'Étape 5, soit 29, correspond la configuration $u_3(b, c, d) = (b^0, c^1, d^0)$. Maintenant $u_4^*(b^0, d^0)$, calculé à l'Étape 4, correspond à $u_4(b^0, d^0, f^0) = 13$, alors à la configuration optimale $F = f^0$. À l'Étape 3, $u_5^*(b^0) = 9$ correspond à $u_5(b^0, g^1)$. Ainsi, dans la configuration optimale $G = 1$. Par conséquent, la configuration correspondant à l'utilité maximale qui est obtenue quand nous itérons en arrière de l'Étape 5 à l'Étape 1 (avec la fonction `Instantiate`) est $(a^0, b^0, c^1, d^0, e^0, f^0, g^1)$. Cette configuration sera notre borne inférieure initiale, avec utilité 29 et poids 35. Cette méthode pour trouver la borne inférieure initiale est implantée par la fonction `Initial_lower_bound`.

Maintenant nous pouvons commencer à explorer l'espace des configurations possibles en utilisant une méthode de branch-and-bound. Nous énumérons u, en explorant ses parties qui semblent être les plus intéressantes selon une heuristique adaptée et en rejetant les parties qui sont au-dessous de la borne inférieure actuelle.

Entrées : $\mathcal{G}$, un arbre GAI avec un ensemble de cliques ordonnées $\mathcal{C} = \{X_{C_1}, \ldots, X_{C_k}\}$; $w = \{w_1, \ldots, w_n\}$, une liste de poids ; c, la capacité du sac-à-dos
Sorties : La configuration qui sera utilisée pour la borne inférieure x_*

```
1  Soit (X_{σ1}, ..., X_{σn}) la liste des objets triés par ordre croissant de poids
2  G' ⟵ G
3  pour j = (premier σ_j tel que Σ_{t=1}^{j} w_{σt} > c) à n faire
4      pour chaque X'_{Ci} faire
5          si X_{σj} ∈ X'_{Ci} alors
6              affecter u'_i à −∞ pour les élements du domaine où x_{σj} = 1
7          fin
8      fin
9  fin
10 Collect(X'_{Ck}, X'_{Ck})
11 x* ⟵ Instantiate(X'_{Ck},X'_{Ck},x'_{Ck},∅)
```

Fonction `Initial_lower_bound`

Pour commencer, nous appelons `Collect` pour la fonction u, ce qui calcule l'expression de l'équation 2.4 à travers les opérations 1–5 comme précédemment, mais cette fois-ci avec la fonction u originale. La figure 2.9 montre les contenus des u_i^* et des u_i à la fin de ce processus. Notons que les tableaux d'utilité de $u_1(a, b)$, $u_2(c, e)$ et $u_5(b, g)$ restent inchangés (avec les mêmes valeurs que dans la figure 2.6), puisqu'il s'agit de cliques extérieures.

	b^0	b^1
$u_1^*(b)$	8	3

	c^0	c^1
$u_2^*(c)$	6	4

$u_4(b,d,f)$	b^0		b^1	
	f^0	f^1	f^0	f^1
d^0	13	11	11	14
d^1	12	17	15	6

	b^0	b^1
$u_5^*(b)$	9	6

$u_4^*(b,d)$	b^0	b^1
d^0	13	14
d^1	17	15

$u_3(b,c,d)$	b^0		b^1	
	d^0	d^1	d^0	d^1
c^0	27	33	30	25
c^1	30	30	23	26

FIG. 2.9 – Contenus des u_i^* et des u_i après les substitutions pour u

Ensuite nous instancions la meilleure configuration de u. La configuration avec la plus grande utilité (ici, 33) est alors $(a^0, b^0, c^0, d^1, e^0, f^1, g^1)$. Malheureusement, cette configuration pèse 110 et donc dépasse la capacité du sac-à-dos. Nous découpons alors l'ensemble d'alternatives restantes comme dans la procédure de rangement précédemment présentée, ce qui nous donne les ensembles :

- E1 : $(B, C, D) \neq (b^0, c^0, d^1)$. Nous pouvons calculer l'utilité de la meilleure configuration dans cet ensemble sans l'instancier, en regardant la deuxième meilleure

valeur de $u_3(b, c, d)$ dans la figure 2.9. Elle est égale à 30.

- E2 : $(B, C, D) = (b^0, c^0, d^1)$ et $(B, D, F) \neq (b^0, c^0, f^1)$. Pour calculer l'utilité de la meilleure configuration dans cet ensemble, nous observons $u_4^*(b, d)$ et notons qu'avant nous avions une utilité 17 pour (b^0, d^1), et maintenant nous n'avons que 12 ($u_4(b^0, d^1, f^0)$). Ainsi, l'utilité de la meilleure configuration dans cet ensemble sera $33 - (17 - 12) = 28$. Cette utilité est inférieure à notre borne inférieure, alors cet ensemble peut être rejeté sans aucune exploration additionnelle. Notons que les configurations dans cet ensemble pèsent au moins 35 (puisque $D = 1$).
- E3 : $(B, C, D, F) = (b^0, c^0, d^1, f^1), (B, G) \neq (b^0, g^1)$. Les configurations dans cet ensemble pèsent au moins 95, de sorte qu'elles pourraient toujours être adaptées au sac-à-dos. Toutefois, l'utilité maximale pour l'ensemble est $33 - (9 - 0) = 24$ (voir $u_5(b, g)$) ce qui est inférieur à notre borne inférieure actuelle. Alors cet ensemble de configurations peut également être écarté.
- E4 : $(B, C, D, F, G) = (b^0, c^0, d^1, f^1, g^1), (C, E) \neq (c^0, e^0)$. Les configurations dans cet ensemble pèsent au moins 110, alors il peut aussi être écarté.
- E5 : $(B, C, D, E, F, G) = (b^0, c^0, d^1, e^0, f^1, g^1), (A, B) \neq (a^0, b^0)$. Cet ensemble sera aussi écarté puisque les configurations qui lui appartiennent pèsent au moins 110. Notons que nous n'avons même pas besoin de considérer cet ensemble parce que l'ensemble 4 a été déjà écarté et étant donnée la méthode utilisée pour découper les ensembles, les configurations dans les ensembles d'indices plus grands sont au moins aussi lourdes que la configuration plus légère de l'ensemble d'indice inférieur généré dans la même phase.

La fonction `NGKPSplit` est une modification de la fonction `Split` utilisée dans la procedure de rangement. Elle prend en compte la capacité du sac-à-dos et l'utilité de la borne inférieure courante pour ne pas créer d'ensembles dans lesquels la solution optimale n'est pas forcément contenue, comme ci-dessus.

Comme seul E1 est resté, nous passerons directement à son exploration. Si, par contre, plusieurs ensembles étaient restés après le découpage, nous les trierions selon une heuristique et commencerions à explorer le mieux classé selon l'ordre obtenu. Notons qu'à ce moment nous ne pouvons connaître que l'utilité de la configuration d'utilité maximale dans l'ensemble et le poids dû aux variables qui ont leurs valeurs fixées pour l'ensemble et à qui a été affectée la valeur 1. Ainsi, l'heuristique que nous adoptons pour ordonner les ensembles à considérer utilise la valeur d'utilité maximale dans l'ensemble, pondérée par le « poids disponible pour variable à instancier ». La fonction `Heuristic_Choose` implante cette heuristique.

Ensuite, nous instancions E1 et nous obtenons sa configuration avec la plus grande

Entrées : X_{C_i}, clique sur laquelle l'on impose les contraintes d'exclusion ; X_{C_r}, une clique voisine ; y^*, un n-uplet optimal dans l'ensemble considéré ; $u(y^*)$, l'utilité de y^* ; $Forbidden_{C_i}$, $|C_i|$-uplets interdits pour la clique X_{C_i} ; c, la capacité du sac-à-dos ; u_*, la borne inférieure d'utilité ; $w = \{w_1, \ldots, w_n\}$, une liste de poids

Sorties : $\mathcal{S}$, une partition des alternatives restantes

$\mathcal{S} \leftarrow \emptyset$
si *poids total des vars = 1 pour cet ensemble* $\leq c$ **alors**
  $Forbidden_{C_i} \leftarrow Forbidden_{C_i} \cup \{y^*_{C_i}\}$
  si $X_{C_i} = X_{C_r}$ **alors**
    $u^* = \max\{u_i(x_{C_i}) : x_{C_i} \in X_{C_i} \setminus Forbidden_{C_i}\}$
  sinon
    $u^*_{x_{C_i}} \leftarrow \max\{u_i(y^*_{S_i}, x_{D_i}) : x_{D_i} \in X_{D_i}$ et $(y^*_{S_i}, x_{D_i}) \notin Forbidden_{C_i}\}$
    $u^* = u(y^*) - (u_i(y^*_{C_i}) - u^*_{x_{C_i}})$
  fin
  si $u^* > u_*$ **alors**
    $\mathcal{S} \leftarrow \{(X_{C_i}, X_{C_r}, Forbidden_{C_i}, u^*, y^*)\}$
  fin
  pour tous les *cliques* X_{C_j} *dans* $\{$*cliques adjacentes à* $X_{C_i}\}\backslash\{X_{C_r}\}$ **faire**
    $\mathcal{S} \leftarrow \mathcal{S} \cup$ `NGKPSplit(`X_{C_j}, X_{C_i}, y^*, $u(y^*)$, $\emptyset$, c, u_*`)`
  fin
fin
retourner $\mathcal{S}$

Fonction `NGKPSplit`

utilité*, $(a^1, b^1, c^0, d^0, e^0, f^1, g^0)$. Cette configuration a une utilité 30 (comme nous le savions déjà depuis la création de l'ensemble) et pèse 110. Ainsi, son poids dépasse la capacité du sac-à-dos et elle n'est donc pas une solution pour le problème. La solution est dans l'un des ensembles qu'on obtient en découpant E1 :

- E1.1 : $(B, C, D) \notin \{(b^0, c^0, d^1), (b^1, c^0, d^0)\}$. L'utilité de la meilleure configuration dans cet ensemble est aussi 30 (la 3^e meilleure valeur de $u_3(b, c, d)$).
- E1.2 : $(B, C, D) = (b^1, c^0, d^0)$ et $(B, D, F) \neq (b^1, d^0, f^1)$. L'utilité de la meilleure configuration dans cet ensemble est $30 - (14 - 11) = 27$, inférieure à notre borne inférieure. Ainsi, cet ensemble est écarté.
- E1.3 : $(B, C, D, F) = (b^1, c^0, d^0, f^1), (B, G) \neq (b^1, g^0)$. L'utilité de la meilleure configuration dans cet ensemble est $30 - (6 - 4) = 28$, inférieure à notre borne inférieure. Cet ensemble est donc également écarté.
- E1.4 : $(B, C, D, F, G) = (b^1, c^0, d^0, f^1, g^0), (C, E) \neq (c^0, e^0)$. L'utilité de la meilleure configuration dans cet ensemble est $30 - (6 - 3) = 27$, inférieure à notre borne inférieure. Cet ensemble peut aussi être écarté.

*En réalité, il y a 3 configurations d'utilité maximale, nous avons choisi l'une d'entre elles

Entrées : Une liste d'ensembles, $\mathcal{S}$ (chaque ensemble dans $\mathcal{S}$ a le format : $\{(X_{C_i}, X_{C_r}, Forbidden_{C_i}, u^*, y^*)\}$) ; c, la capacité du sac-à-dos ; u_*, la borne inférieure d'utilité
Sorties : Un élément $S \in \mathcal{S}$ choisi heuristiquement

pour tous les $S^j \in \mathcal{S}$ **faire**
  si u^* *de* $S^j \leq u_*$ **alors** Écarter S^j **sinon**
    $$h^{S^j} = u^* \times \frac{(c - \text{poids total des vars} = 1 \text{ pour cet ensemble})}{\text{nombre de vars non instantiées}}$$
  fin
fin
$S \longleftarrow S^j \in \mathcal{S}$ avec h^{S^j} maximale

Procédure `Heuristic_Choose`

- E1.5 : $(B, C, D, E, F, G) = (b^1, c^0, d^0, e^0, f^1, g^0), (A, B) \neq (a^1, b^1)$. L'utilité de la meilleure configuration dans cet ensemble est $30 - (3 - 2) = 29$, elle n'est donc pas supérieure à l'utilité de la configuration qui est notre borne inférieure. Cet ensemble peut être écarté.

Maintenant nous instancions E1.1, et nous obtenons sa configuration avec l'utilité maximale*, $(a^0, b^0, c^1, d^1, e^1, f^1, g^1)$. Cette configuration a une utilité 30 et pèse 185. Alors, nous devons continuer à chercher la solution en découpant E1.1 :

- E1.1.1 : $(B, C, D) \notin \{(b^0, c^0, d^1), (b^1, c^0, d^0), (b^0, c^1, d^1)\}$. L'utilité de la meilleure configuration dans cet ensemble est toujours 30 (4^{e} meilleure valeur de $u_3(b, c, d)$).
- E1.1.2 : $(B, C, D) = (b^0, c^1, d^1), (B, D, F) \neq (b^0, d^1, f^1)$. L'utilité de la meilleure configuration dans cet ensemble est $30 - (17 - 12) = 25$, donc plus petite que notre borne inférieure. Ainsi, cet ensemble est écarté.
- E1.1.3 : $(B, C, D, F) = (b^0, c^1, d^1, f^1), (B, G) \neq (b^0, g^1)$. Les configurations dans cet ensemble pèsent au moins 115, donc on peut l'écarter. Nous n'avons pas besoin de considérer ni E1.1.4 ni E1.1.5 pusqu'ils seront nécessairement trop lourds.

Nous instancions E1.1.1, et nous obtenons la configuration $(a^0, b^0, c^1, d^0, e^1, f^0, g^1)$. Elle a une utilité 30 et pèse 90. Ainsi, elle est une solution au problème. Nous mettons à jour notre borne inférieure en lui affectant cette solution. Nous n'avons pas besoin de découper E1.1.1, parce que la configuration que nous venons d'obtenir est déjà sa meilleure configuration. Comme nous n'avons aucun autre ensemble restant, il n'y a pas d'autre configuration avec une utilité plus grande qui soit compatible avec la contrainte du sac-à-dos. Nous pouvons donc terminer la recherche. Ainsi, en choisissant les objets C, E et G nous avons l'utilité maximale pour cet instance du NGKP.

La fonction `NGKPSearch` implante l'algorithme décrit dans cette section. On commence par l'obtention de la configuration qui sera notre borne inférieure en appelant

*Cette fois-ci nous avons 2 options de configuration avec la même utilité maximale

`Initial_lower_bound` (ligne 1). Après nous calculons la configuration d'utilité maximale du problème (ligne 2). Si cette configuration est compatible avec le sac-à-dos, elle est la solution au problème. Sinon nous découpons l'ensemble d'alternatives en lui extrayant cette configuration maximale (ligne 7). Ensuite nous passons à l'exploration des sous-ensembles (lignes 8–17). Le prochain ensemble à explorer est choisi par la fonction `Heuristic_Choose` (qui, de sa part, élimine les sous-ensembles dominés). Si la configuration avec la plus grande utilité de l'ensemble choisi est compatible avec le sac-à-dos, nous mettons à jour la borne inférieure, sinon nous redécoupons l'ensemble choisi pour extraire cette solution. Ce processus progresse jusqu'à ce que l'utilité maximale de tous les ensembles restants soit plus petite que l'utilité de la borne inférieure courante.

Entrées : $\mathcal{G}$, un arbre GAI avec un ensemble de cliques ordonnées $\mathcal{C} = \{X_{C_1}, \ldots, X_{C_k}\}$; $w = \{w_1, \ldots, w_n\}$, une liste de poids ; c, la capacité du sac-à-dos
Sorties : x^*, une configuration d'utilité maximale compatible avec le sac à dos

1 $x^* \leftarrow$ `Initial_lower_bound(`$\mathcal{G}$, w, c`)`
2 $x^c \leftarrow$ `Optimal_choice(`$\mathcal{G}$`)`
3 **si** $weight(x^c) \leq c$ **alors**
4 $\quad x^* \leftarrow x^c$
5 $\quad$ **retourner**
6 **fin**
7 $\mathcal{S} \leftarrow$ `NGKPSplit(`X_{C_k}, X_{C_k}, x^c, $u(x^c)$, $\emptyset$, c, $u(x^*)$, w`)`
8 **tant que** *l'utilité de l'ensemble d'utilité maximale dans* $\mathcal{S} > u(x^*)$ **faire**
9 $\quad \{(X_{C_i}, X_{C_r}, Forbidden_{C_i}, u^*, y^*)\} \leftarrow$ `Heuristic_Choose(`$\mathcal{S}$, c, u_*`)`
10 $\quad x^c \leftarrow$ `Instantiate(`X_{C_i}, X_{C_r}, $y^*_{C_r}$, Forbidden$_{C_i}$`)`
11 $\quad$ **si** $weight(x^c) \leq c$ **alors**
12 $\quad\quad x^* \leftarrow x^c$
13 $\quad$ **fin**
14 $\quad$ **sinon**
15 $\quad\quad \mathcal{S} \leftarrow \mathcal{S} \cup$ `NGKPSplit(`X_{C_i}, X_{C_r}, x^c, u^*, Forbidden$_{C_i}$, c, $u(x^*)$, w`)`
16 $\quad$ **fin**
17 **fin**

Fonction `NGKPSearch`

2.5.3 Expérimentations

Nous avons implanté la procédure proposée en Java et nous l'avons exécutée en utilisant un PC 2.13GHz. Pour les instances de test, nous avons généré 8 classes de problème différentes décrites par le nombre d'objets, la taille des termes de sous-utilité dans la décomposition GAI ($|x_{C_i}|$ pour chaque $u_i(x_{Z_i})$), et le nombre de termes de sous-utilité dans la décomposition GAI (k for $u(x_1, ..., x_n) = \sum_{i=1}^{k} u_i(x_{C_i})$). Les classes de problème sont listées dans le tableau 2.3. Par exemple, pour K_6, nous avons $u(x_1, \ldots, x_{40}) = \sum_{i=1}^{15} u_i(x_{C_i})$,

chaque x_{C_i} est une combination de 3 variables choisies aléatoirement parmi les 40 variables (avec répétition, c'est-à-dire que les intersections entre les x_{C_i} sont permises). Les valeurs d'utilité sont des nombres réels choisis aléatoirement dans l'intervalle $[0, 1]$.

	Nombre d'objets	Taille des termes	Nombre de termes
K_1	10	2	6
K_2	10	3	4
K_3	20	2	12
K_4	20	3	8
K_5	40	2	22
K_6	40	3	15
K_7	80	2	45
K_8	80	3	30
K_9	160	2	90
K_{10}	160	3	60
K_{11}	320	2	180
K_{12}	320	3	120
K_{13}	640	2	360

TAB. 2.3 – Classes de problème

Des instances du problème du sac-à-dos peuvent avoir des poids indépendants ou des poids corrélés à l'utilité des objets. Le premier cas a lieu dans des situations où il n'y a pas de raison pour supposer que les objets plus lourds ont plus de valeur. Par exemple, quand on met des objets dans une mallette, des objets très légers peuvent avoir beaucoup de valeur, tandis que le contraire peut être vrai pour des objets lourds. L'autre cas a lieu quand l'utilité d'un objet est corrélé à son poids. Par exemple, dans les finances, le retour d'un investissement est en général corrélé à la somme qui a été investie. Les problèmes avec des poids aléatoires sont normalement plus faciles à résoudre que des instances où les poids des objets sont corrélés à leur utilité (Pisinger, 1995). Nous avons voulu tester les deux types d'instance de problème. Notons que dans notre cas, parler de l'utilité d'un objet ne convient pas, puisque la fonction d'utilité n'est pas additivement décomposable. Ainsi, pour simuler des instances où le poids des objets est corrélé à leur utilité dans le sac-à-dos, nous avons procédé de la manière suivante :

- Après la définition des fonctions d'utilité, nous avons généré les 50 000 meilleures configurations.
- Nous avons calculé combien de fois chaque objet est paru dans ces meilleures configurations.
- Le poids d'un objet sera plus élevé s'il paraît plus de fois dans les meilleures configurations et vice-versa. Supposez que nous avons compté m parutions pour l'objet qui est paru le plus souvent dans ces configurations. Nous avons utilisé comme poids pour un objet qui est paru dans i meilleures configurations $(i/m)^2 *$

$maxWeight$.

Pour les instances où les objets ont des poids aléatoires, le poids utilisé est un entier qui a été choisi aléatoirement dans l'intervalle $[0, 1000]$. Soit n le nombre d'objets avec des poids w_j, $j = 1, \ldots, n$. Nous avons utilisé un sac-à-dos de capacité $c = \frac{1}{2}\sum_{j=1}^{n} w_j$.

Les tableaux 2.4 et 2.5 montrent les temps d'exécution pour chacune des classes de problème dans les deux scénarios : des poids aléatoires ou corrélés. Les résultats affichés portent sur 20 exécutions. Nos résultats reflètent le fait que les instances avec des poids corrélés sont plus difficiles que les instances avec des poids aléatoires. Pour ce dernier type d'instance, nous avons pu résoudre des problèmes dans toutes les classes K_1–K_{13}, tandis que pour les instances avec des poids corrélés, seulement les instances de K_1–K_6 ont pu être résolues en moins de 20 minutes. Nous remarquons que l'approche a été très efficace pour une grande partie des instances résolues. Pour certaines classes de problème, par exemple K_8 avec des poids aléatoires, l'approche a été très efficace pour une grande majorité des instances, mais certaines instances ont eu une performance notamment moins bonne. Remarquons que pour K_8 avec des poids aléatoires, 75% des cas ont été résolus en au plus 0,08 seconds (voir Q3, le troisième quartile), tandis que l'instance qui a pris le maximum de temps a été résolue en 340,22 seconds. Les instances avec 10 et 20 variables ont été rapidement résolues, dans tous les cas, tandis que les instances avec 40 objets ont déjà pris plus de temps quand elles avaient des poids corrélés. Le temps de calcul est naturellement affecté par la difficulté de l'instance, il peut être très élevé pour un grand nombre d'objets et surtout avec des poids corrélés. Toutefois, dans des applications pratiques avec des modèles de représentation de préférences plus élaborés comme des décompositions GAI, le nombre de variables est rarement énorme, puisque nous devons exprimer des préférences sur ces différentes interactions. Ainsi, il est fort probable que la limite cognitive soit atteinte plus rapidement que la limite d'efficacité de l'algorithme de résolution.

Par ailleurs, notons que la qualité d'une méthode de branch and bound est fortement influencée par la qualité de la borne initiale utilisée. D'autres approches peuvent être utilisées pour essayer de trouver une meilleure borne initiale (ici, implantée par l'approche `Initial_lower_bound`). Par exemple, si nous approchons la fonction d'utilité $u(x)$ par une fonction d'utilité additive $g(x)$ telle que $\forall x, g(x) \leq u(x)$, nous pouvons obtenir la borne initiale à partir de cette fonction additive $g(x)$ qui, de sa part, nous pose dans le cadre d'un problème de sac-à-dos décomposable. Ainsi, des méthodes classiques pourront être utilisées pour l'obtention de la borne initiale (voir, par exemple, Pisinger (1995)). Toutefois, cela sera d'autant plus efficace que la fonction additive $g(x)$ utilisée soit proche de la fonction originale $u(x)$. Par conséquent, l'efficacité d'une borne ainsi obtenue sera aussi dépendant du jeu de données utilisé.

Classe de problème	Temps (s)						# d'instances $< 20'$
	min	Q1	médiane	Q3	max	moyenne	
K_1	0	0	0	0,01	0,01	0	20
K_2	0	0,01	0,01	0,01	0,06	0,01	20
K_3	0,01	0,01	0,02	0,03	0,08	0,03	20
K_4	0	0,01	0,01	0,03	0,07	0,02	20
K_5	0,01	0,01	0,02	0,03	0,08	0,03	20
K_6	0,01	0,02	0,03	0,05	0,25	0,05	20
K_7	0,04	0,05	0,05	0,17	8,47	0,74	20
K_8	0,03	0,04	0,07	0,08	340,22	17,09	20
K_9	0,31	0,32	0,37	2,83	59,05	7,65	19
K_{10}	0,21	0,22	0,23	0,52	444,31	23,48	20
K_{11}	2,51	2,56	2,67	5,49	540,43	41,81	19
K_{12}	1,64	1,68	1,77	4,29	620,48	38,25	18
K_{13}	4,29	26,27	27,79	56,29	597,16	76,18	14

TAB. 2.4 – Poids aléatoires. Temps d'exécution en secondes (sur 20 exécutions)

Classe de problème	Temps (s)						# d'instances $< 20'$
	min	Q1	médiane	Q3	max	moyenne	
K_1	0	0	0,01	0,01	0,05	0,01	20
K_2	0	0	0,01	0,02	0,06	0,01	20
K_3	0	0,01	0,01	0,02	0,05	0,01	20
K_4	0	0,01	0,03	0,05	0,21	0,04	20
K_5	0,19	1,73	5,00	16,37	32,32	9,34	20
K_6	0,02	0,41	2,54	6,71	105,45	12,06	20

TAB. 2.5 – Poids corrélés. Temps d'exécution en secondes (sur 20 exécutions)

2.6 Conclusion

Dans ce chapitre, nous avons montré comment les réseaux GAI peuvent être utilisés pour réaliser, de manière efficace, des recommandations (choix et rangement) pour un individu dans le cadre des espaces combinatoires de grande taille. Nous avons aussi vu que des contraintes interdisant certaines affectations de valeurs aux variables peuvent être intégrées directement dans le modèle GAI, permettant ainsi le traitement des cas où quelques configurations sont interdites. Toutefois, quand nous avons des contraintes de grande arité, leur intégration dans le modèle n'est pas faisable, en raison des cliques de grande taille générées. Dans de tels cas, il est nécessaire de traiter les contraintes séparément, en utilisant des outils développés dans les communautés de CSP et de recherche opérationnelle. Nous avons analysé le cas des problèmes de sac-à-dos 0–1 non-linéaire, GAI-décomposable. Ici, si nous intégrons les contraintes de sac-à-dos dans le modèle GAI, elles impliqueront l'existence d'une clique avec toutes les variables, ce qui rend cette approche infaisable. Nous avons donc proposé une approche qui adapte l'algorithme de rangement présenté dans ce chapitre en le couplant avec une méthode de « branch and bound ». Des bons résultats expérimentaux ont été obtenus.

Par ailleurs, des recommandations ne sont pas toujours construites à partir d'un seul point de vue. Il se peut que nous devons considérer l'opinion de plusieurs individus, ou analyser le problème selon un ensemble de critères, potentiellement conflictuels. Dans le chapitre suivant, nous traitons le problème d'agrégation de préférences, quand l'utilité des alternatives est décrite par un ensemble de fonctions d'utilité GAI sur les attributs du domaine. Chaque fonction d'utilité peut correspondre aux préférences d'un agent, ou à l'utilité des alternatives selon un critère d'évaluation. Dans ce cadre, nous cherchons une bonne solution de compromis, qui ait une bonne valeur d'utilité selon l'ensemble de fonctions. Nous développerons ainsi une méthode efficace pour la recherche de solutions non-dominées selon des critères de compromis adaptés.

Chapitre 3

Agrégation de Réseaux GAI

« Deux hommes marchent-ils ensemble,
Sans en être convenus ? »

La Bible, livre d'Amos, chapitre 3, verset 3
(VII^{e} siècle av. J.-C.)

Ce chapitre s'appuie en partie sur des résultats présentés dans Gonzales *et al.* (2006); Queiroz *et al.* (2007); Queiroz (2007b); Gonzales *et al.* (2008)

Résumé

Dans ce chapitre, nous considérons le cas où l'on dispose d'un ensemble de modèles GAI qui représentent des points de vue (différents critères ou groupe d'individus) sur l'ensemble d'alternatives. Nous cherchons la meilleure solution de compromis selon des critères notamment non-linéaires (max-min, min-max regret, normes de Tchebycheff), ce qui engendre un problème NP-difficile. Pour chacun de ces critères d'agrégation, nous proposons une procédure efficace pour la détermination exacte de la solution optimale sur le produit cartésien. Ensuite nous considérons une catégorie d'intégrales de Choquet qui peut être traitée par la même approche, comme un sous-ensemble des majorités floues fondées sur des critères linguistiques. Dans chacun des cas, des résultats expérimentaux qui montrent l'efficacité pratique de la procédure proposée sont présentés.

3.1 La prise de décision collective ou multi-critère

Supposons que l'on dispose d'un ensemble de fonctions d'utilité préalablement élicitées $\{u^1, u^2, \ldots, u^m\}$, où chaque $u^j : \mathcal{X} \mapsto \mathbb{R}, j \in \{1, \ldots, m\}$ est une fonction GAI-décomposable qui code l'utilité des éléments du produit cartésien $\mathcal{X} = \{X_1, X_2, \ldots, X_n\}$ selon un point de vue différent. Un point de vue peut provenir d'une personne ou d'un critère. Dans le premier cas, nous sommes dans le cadre de la **décision collective**. Chaque fonction d'utilité modélise les préférences d'un agent. Dans l'autre cas, nous sommes dans le cadre de la **décision multi-critère**. Chaque fonction exprime l'utilité des éléments de $\mathcal{X}$ selon l'évaluation d'un aspect différent.

Dans ces conditions, l'une des manières classiques de trouver la meilleure solution de compromis pour un ensemble de valeurs d'utilité est de définir une fonction d'utilité globale $u(x)$, telle que pour tout $x \in \mathcal{X}$, $u(x)$ donne la valeur d'utilité de x pour l'ensemble d'agents/critères*. On considère $u(x) = h(u^1(x), \ldots, u^m(x))$, où h est une fonction d'agrégation qui établit implicitement le type de compromis désiré pour la solution. La meilleure solution de compromis est celle optimisant u sur $\mathcal{X}$. Quand la fonction h est non-décroissante en chaque composante, les solutions u-optimales sont des solutions faiblement Pareto-optimales (c'est-à-dire qu'il n'y a pas d'autre solution qui améliore l'utilité pour tous les agents). De plus, si h est strictement croissante en chaque composante alors les solutions u-optimales sont Pareto-optimales (c'est-à-dire qu'il n'y a pas d'autre solution qui améliore l'utilité pour un agent, sans parallèlement diminuer l'utilité pour un autre).

Notons que les solutions qui nous intéressent sont dans l'ensemble de solutions non-dominées au sens de Pareto, mais cet ensemble peut être excessivement nombreux et inclure des solutions avec des profils d'utilité très différents. Regardons la figure 3.1. Ici, les alternatives de $\mathcal{X}$ sont évaluées selon deux fonctions d'utilité $u^1, u^2 : \mathcal{X} \mapsto \mathbb{R}$. Ainsi, chaque alternative dans $\mathcal{X}$ est renvoyée dans l'espace $\mathbb{R}^2$.† Les alternatives Pareto-optimales sont représentées par des cercles remplis (en rouge), tandis que les alternatives faiblement Pareto-optimales sont représentées par des cercles hachurés. Les autres alternatives dominées sont représentées par des cercles sans remplissage. Considerons maintenant les alternatives **a**, **b** et **c** dans la figure. Elles s'agissent, toutes les trois, d'alternatives Pareto-optimales. Toutefois, elles ont des profils d'utilité bien différents. Tandis que **a** possède une bonne utilité selon u^2 et une mauvaise utilité selon u^1, la situation pour **c** est l'inverse. Pourtant, l'alternative **b** peut être une bonne solution de compromis.

*Puisque le lien entre le cadre collectif et le cadre multi-critère est assez direct, nous nous limitons, sans perte de généralité, à utiliser la nomenclature du cadre collectif dans la suite de la présentation.

†En décision multicritère, chaque $u^j(x)$ correspond à l'évaluation de l'alternative x selon un critère j ($j \in \{1, \ldots, m\}$, où m est le nombre de critères). Ainsi, nous nommons cet espace ($\mathbb{R}^m$, si chaque $u^j(x)$ est une fonction $\mathcal{X} \mapsto \mathbb{R}$) l'**espace des critères**.

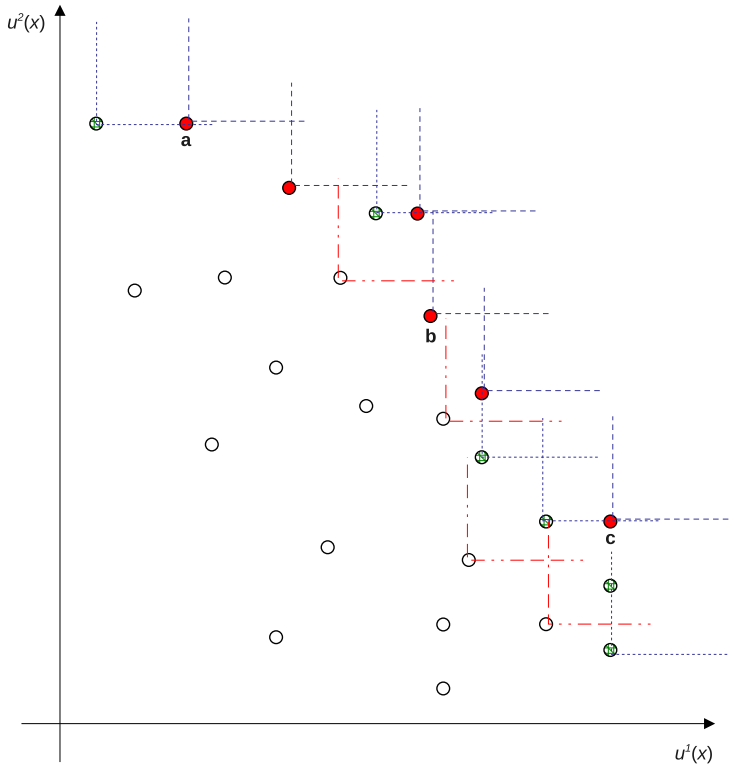


FIG. 3.1 – Évaluation des alternatives selon deux fonctions d'utilité $u^1(x), u^2(x)$.

Quand la fonction d'agrégation h est linéaire, alors u est une somme d'utilités GAI, ce qui, par conséquent, fait également de u une fonction GAI. Ainsi, le problème se réduit à un problème monoagent, avec une fonction d'utilité GAI u (comme dans le chapitre 2). Toutefois, les fonctions d'agrégation linéaires ne sont pas une bonne option, car elles peuvent choisir une solution x avec un profil d'utilité $(u^1(x), \ldots, u^m(x))$ très déséquilibré, concernant l'utilité selon chacun des axes d'évaluation.

Exemple 3.1. Considérons un problème avec 3 agents, tel que $\mathcal{X} = \{x, y, z, w\}$ avec $u^1(x) = 0, u^2(x) = u^3(x) = 100$, $u^2(y) = 0, u^1(y) = u^3(y) = 100$, $u^3(z) = 0, u^1(z) = u^2(z) = 100$, $u^1(w) = u^2(w) = u^3(w) = 65$. Seule la solution w est acceptable par tous les agents. Elle constitue donc la meilleure solution de compromis. Toutefois, malgré le fait que w ne soit pas Pareto-dominé, cette alternative ne peut pas être trouvée par la maximisation d'une combinaison linéaire (avec des coefficients positifs) des utilités des agents.

En effet, l'optimisation d'une fonction d'agrégation linéaire ne nous permet que l'exploration des solutions Pareto-optimales dans l'enveloppe convexe de points dans l'espace des profils d'utilité. Comme nous pouvons remarqué dans la figure 3.2, des solutions Pareto-optimales représentant des compromis intéressants peuvent se situer

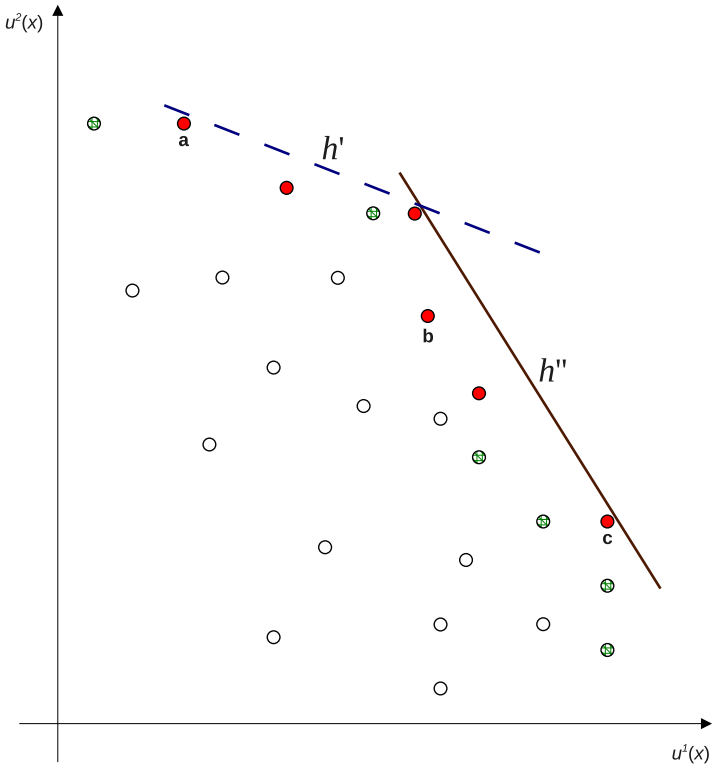


FIG. 3.2 – Optimisation selon deux fonctions d'agrégation linéaires h' et h''.

en dehors de la frontière de cette enveloppe convexe. Dans de tels cas, des fonctions d'agrégation non-linéaires se montrent plus adaptées.

3.2 L'agrégation non-linéaire

Considérons à nouveau l'exemple 3.1. Si maintenant nous maximisons $u(\cdot) = \min_{j\in\{1,2,3\}} u^j(\cdot)$, w sera notre solution maximale, puisque $u(x) = 0$, $u(y) = 0$, $u(z) = 0$ et $u(w) = 65$. Ce critère amenant à choisir l'alternative qui maximise la satisfaction de l'agent le moins satisfait, est nommé **max-min** :

- **Le critère max-min :** Soit $\mathcal{D} = \{1, \ldots, m\}$ l'ensemble d'agents. On maximise sur $\mathcal{X}$ le critère $u(x) = \min_{j\in\mathcal{D}} u^j(x)$. Ceci correspond à maximiser la satisfaction de l'agent le moins satisfait dans l'ensemble d'agents.

Le max-min met en exergue le fait qu'aucun agent n'est très insatisfait avec la solution choisie. Dans la décision collective, cela peut être une caractéristique désirable, par exemple, des études psychologiques indiquent que l'existence d'individus très insatisfaits dans un groupe compromet sa longévité (Mohammed et Ringseis, 2001). Toutefois, notons que ce critère rend le choix facilement manipulable puisqu'il pratiquement donne le droit

de veto à chaque agent.

Un critère plus flexible, utilisé dans la littérature d'optimisation mutiobjectif pour générer des solutions de compromis est le critère de Tchebycheff (Steuer et Choo, 1983; Wierzbicki, 1986). Pour générer des solutions de compromis nous minimisons la fonction :

$$f_w(x) = \|w(\overline{u} - u(x))\|_\infty = \max_{j\in\{1,\ldots,m\}} \{w_j|\overline{u}^j - u^j(x)|\}, \quad \text{où}$$

- $\overline{u} = (\overline{u}^1, \ldots, \overline{u}^1)$ représente un point idéal qui correspond à la situation, normalement fictive, où tous les fonctions d'utilité obtiennent simultanément l'alternative d'utilité maximale. Ce point idéal est une borne supérieure de l'ensemble des solutions non-dominées au sens de Pareto ;
- w est un vecteur de poids (positifs).

Notons que, en minimisant la distance selon la norme-infinie (le max), nous nous focalisons sur le composant où l'écart entre l'utilité obtenue par l'alternative et l'utilité idéale est le plus grand. Ainsi, nous favorisons les solutions proches au point de référence $\overline{u}$ en toutes dimensions de l'espace d'utilité. Cela nous permet de trouver des bonnes solutions de compromis. Les fonctions $f_w(x)$ possèdent deux propriétés importantes (Wierzbicki, 1986) :

- **Propriété 1 :** Si $\forall j \in \{1, \ldots, m\}, w_j > 0$, alors toute solution qui minimise $f_w(x)$ sur l'ensemble d'alternatives $\mathcal{X}$ est faiblement Pareto-optimale. En plus, au moins une de ces solutions est Pareto-optimale.
- **Propriété 2 :** Si $\forall j \in \{1, \ldots, m\}, \overline{u}^j > \sup_{x\in\mathcal{X}} u^i(x)$, alors pour toute solution Pareto-optimale x, il existe un vecteur de poids w tel que x est la seule solution qui minimise $f_w(x)$ sur $\mathcal{X}$.

La propriété 1 nous montre que la minimisation de $f_w(x)$ obtient au moins une solution Pareto-optimale. La propriété 2 nous montre que, en contraste avec les fonctions d'agrégation linéaires, toute solution Pareto-optimale peut être atteinte par la minimisation de $f_w(x)$ en utilisant un vecteur de poids approprié. Cela explique pourquoi la minimisation de $f_w(x)$ est préférable aux sommes pondérées dans le cadre de l'optimisation multiobjectif sur des ensembles non-convexes (Steuer et Choo, 1983; Wierzbicki, 1986). Notons que la recherche d'une solution de compromis est souvent un processus interactif. Des procédures couramment employées dans la littérature normalement commencent par utiliser un vecteur w visant le centre de l'ensemble des alternatives Pareto-optimales. Ensuite, w est interactivement ajusté pour explorer des régions plus spécifiques selon l'intérêt de l'agent (ou des agents). La figure 3.3 montre les distances à un point idéal selon la norme infinie pour l'exemple des figures 3.1 et 3.2, en utilisant le vecteur identité pour w. Remarquons que, en modifiant le vecteur de poids, nous

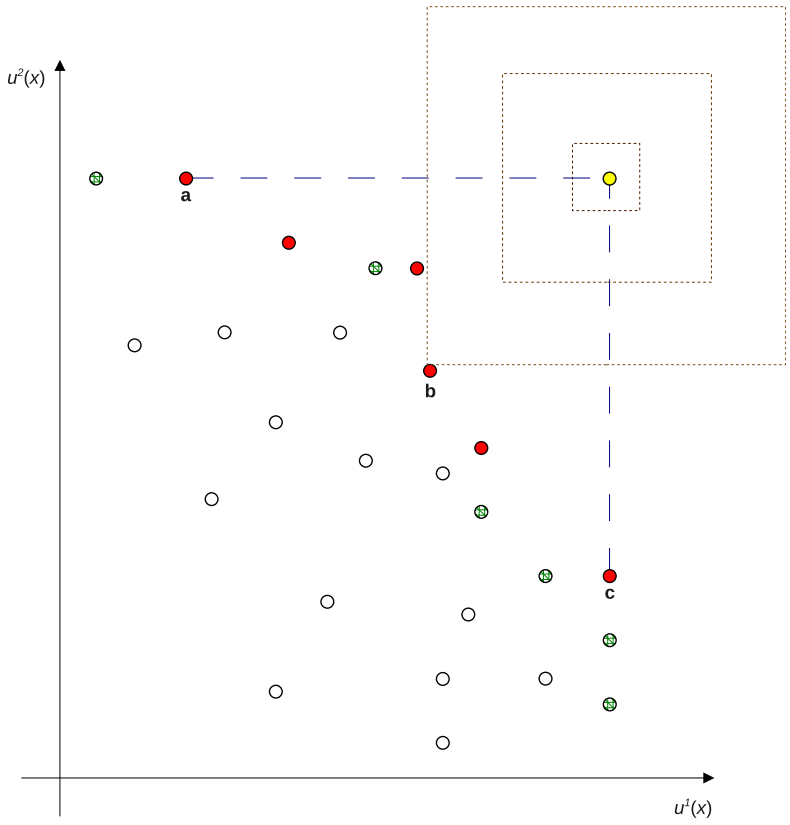


FIG. 3.3 – Distances (selon la norme infinie) à un point idéal (fictif).

déformons l'hypercube de la norme infinie de façon à atteindre d'autres régions d'intérêt.

Dans le cadre de la décision collective, nous pouvons interpréter le terme $|\overline{u}^j - u^j(x)|$, que nous nommerons r^j, comme représentant le **regret** de l'agent j une fois que l'on a choisi la solution x au lieu de son alternative préférée (d'utilité $\overline{u}^j$). Ainsi, le critère de Tchebycheff définit naturellement le critère **min-max regret** (qui minimise le regret r^j maximal pour $j \in \{1, \ldots, m\}$). Toutefois, le min-max regret suppose que les utilités des différents agents sont commensurables et exprimées sur une même échelle. Le critère de Tchebycheff généralise le min-max regret par l'introduction du vecteur de poids w. Nous pouvons, par exemple, utiliser des poids tels que $w_j = \alpha_j/(\overline{u}^j - \underline{u}^j)$ (où $\underline{u}^j$ est l'utilité de l'alternative moins bonne selon l'agent j) pour à la fois contrôler l'importance de l'agent j par moyens de α_j, et normaliser l'échelle d'utilité (si les agents ont attribué leurs valeurs d'utilité dans des intervalles différents) à travers le facteur $1/(\overline{u}^j - \underline{u}^j)$.

3.3 Recherche de la solution de compromis non-linéaire avec l'aide d'une fonction GAI

La détermination de la solution qui minimise une fonction telle que $f_w(x)$, n'est pas une tâche facile en raison de deux éléments principaux : la nature combinatoire de $\mathcal{X}$, et la non-decomposabilité de la fonction $f_w(x)$ (qui n'est pas une fonction GAI). En effet, la détermination de la meilleure solution de compromis pour $f_w(x)$ est un problème NP-difficile dès qu'il y a $n \geq 3$ attributs, $m \geq 2$ agents, chacun possédant une fonction d'utilité GAI avec au moins un terme de taille égale ou supérieure à 3. Ceci peut être prouvé en utilisant une réduction de 3-SAT. Considérons en effet une instance de 3-SAT avec n variables et m clauses. À chaque variable nous associons un attribut booléen X_i, et à chaque clause C_j sur des variables, nous associons un agent avec une fonction d'utilité booléenne u^j. Par exemple $C_j = x \vee y \vee \neg z$ sera représentée par la fonction $u^j(x, y, z) = 1 - (1 - x)(1 - y)z$. Supposons que $\overline{u}^j = 1$ pour tout $j = \{1, \ldots, m\}$ et nous utilisons le vecteur identité pour w. Ainsi, la valeur optimale de $f_w(x)$ sur $\mathcal{X} = X_1 \times \cdots \times X_n$ et les fonctions $u^1, \ldots, u^m$ est 0 si et seulement si le problème 3-SAT initial est satisfaisable. Ceci montre la difficulté à trouver efficacement la solution de meilleur compromis.

Pour faire face au problème d'optimisation d'un critère non-linéaire h sur $\mathcal{X}$, nous proposons de recourir à une technique de rangement. Ainsi nous proposons une procédure en 3 étapes :

Étape 1 : reformulation. Nous reformulons le problème comme un problème monoagent, en utilisant un critère global $g(x)$ défini comme une combinaison linéaire des utilités individuelles. Une telle fonction est plus facile à optimiser que h puisque la somme de fonctions GAI est elle-même une fonction GAI.

Étape 2 : rangement. Nous énumérons les solutions de $\mathcal{X}$ dans l'ordre décroissant de l'utilité selon $g(x)$. Pour cela, on utilise la méthode présenté dans le chapitre 2 (section 2.4), ce qui permet une énumération rapide de g.

Étape 3 : critère d'arrêt. Nous devons arrêter l'énumération le plus vite possible en raison de la taille exponentielle de $\mathcal{X}$. Pour cela nous utilisons la proposition suivante :

Proposition 3.1. *Considérons une fonction h à maximiser et une fonction g, telles que $h, g : \mathcal{X} \mapsto \mathbb{R}$. h est une fonction quelconque et g est une fonction GAI-décomposable, telle que $h(x) \leq g(x)$, $\forall x \in \mathcal{X}$. Soit $x^1, ..., x^k$ les k-meilleures solutions selon la fonction*

g (c'est-à-dire les solutions ayant les k valeurs maximales pour g). Si $g(x^k) \leq h(x^*)$ avec $x^* = Argmax_{i=1,\ldots,k} h(x^i)$, alors x^* est maximal pour h, c'est-à-dire $h(x^*) = \max_{x \in \mathcal{X}} h(x)$.

Démonstration : Pour tout $i > k$, nous avons par construction, $h(x^i) \leq g(x^i) \leq g(x^k)$. Comme $g(x^k) \leq h(x^*)$ par hypothèse, nous avons $h(x^i) \leq h(x^*)$, ce qui montre qu'aucune solution trouvée après x^k dans le rang ne peut améliorer la solution optimale actuelle x^*. ■

La figure 3.4 illustre la procédure d'optimisation d'une fonction $h(x)$ quelconque à travers le rangement d'une fonction GAI-décomposable $g(x)$, telles que $h(x) \leq g(x)$, $\forall x \in \mathcal{X}$. Dans la figure, les points en croix représentent les utilités pour les k-meilleures solutions de $\mathcal{X}$ pour la fonction g. Les points en **x** représentent les utilités obtenues par ces solutions selon h. En $k = 5$ nous pouvons arrêter l'énumeration des alternatives, une fois que $g(x^5) < h(x^*)$. La solution optimale x^* a été obtenue à l'étape $k = 4$.

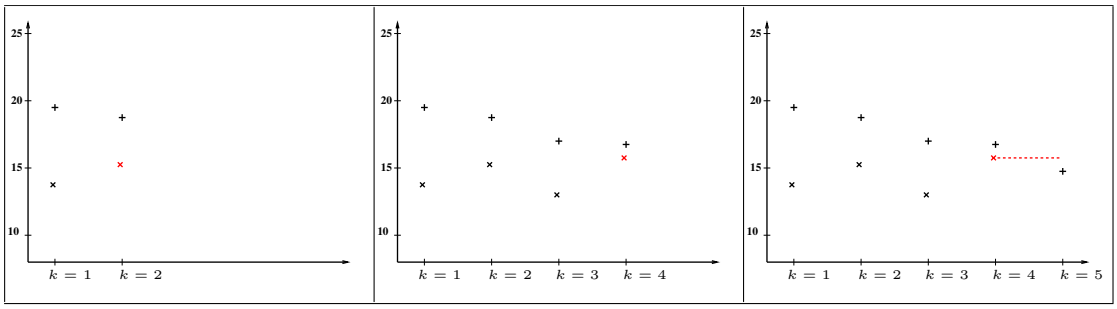


FIG. 3.4 – Optimisation par le rangement d'une fonction GAI

Ce résultat simple peut être utilisé pour optimiser des critères d'agrégation non-linéaires. Le tableau 3.1 suivant définit g, l'approximation GAI à utiliser, pour trois cas : le max-min ; le critère de Tchebycheff avec le vecteur identité pour w, appelé min-max regret ; et le critère de Tchebycheff utilisant un vecteur de poids w, que nous appelons simplement « Tchebycheff ».

TAB. 3.1 – Critères non-linéaires et approximations GAI à utiliser

Critères	$h(x)$	$g(x)$
max-min	$\min_j(u^j(x))$	$\frac{1}{m}\sum_j u^j(x)$
min-max regret	$-\max_j(r^j(x))$	$-\frac{1}{m}\sum_j r_j(x)$
Tchebycheff	$-\max_j(w_j r^j(x))$	$-\frac{1}{m}\sum_j w_j r_j(x)$

Comme $-1/m \sum_j r_j(x)$ diffère de $1/m \sum_j u^j(x)$ seulement par un terme constant, la même méthode de rangement peut être utilisée pour max-min et min-max regret.

Tchebycheff est simplement une extension de min-max regret avec des poids, ce qui lui permet aussi d'être traité de manière analogue.

Nous pouvons ainsi énoncer l'algorithme de recherche de la solution de compromis selon un critère d'agrégation non-linéaire, par l'énumération d'une fonction GAI-décomposable (algorithme 3.1). Notons que cet algorithme possède une caractéristique *anytime**. À tout moment nous pouvons calculer la perte maximale d'utilité que nous pouvons avoir si nous arrêtons l'algorithme et gardons la solution x^* au moment de l'interruption. Cette perte u^p sera :

$$u^p \leq (g(x^k) - h(x^*)), \tag{3.1}$$

puisque avant la fin de l'algorithme, $g(x^k) > h(x^*)$.

La figure 3.5 nous montre la perte maximale d'utilité u^{p^k} à chaque étape k de l'algorithme pour l'exemple de la figure 3.4. Dans la figure, nous pouvons aussi voir la perte réelle à chaque étape, représentée par la distance entre $h(x^k)$ et $h(x^*)$ (ligne horizontale en pointillé).

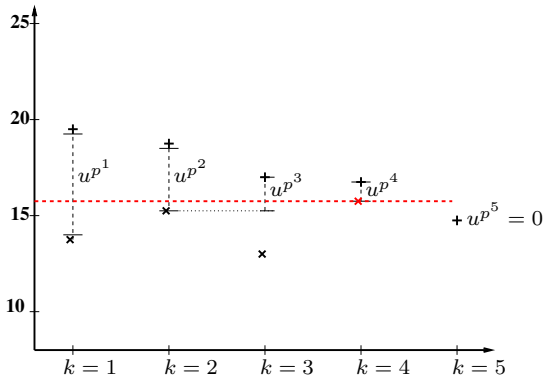


FIG. 3.5 – Perte maximale d'utilité u^p à chaque étape de l'algorithme

La caractéristique *anytime* de l'algorithme est d'autant plus importante une fois que dans des cas pathologiques, il est possible que la condition d'arrêt ne soit pas atteinte, et nous devons énumérer toutes les solutions de l'espace pour être sûr d'avoir la solution optimale. Toutefois, comme nous allons voir dans la section suivante, nous avons obtenu des bonnes résultats dans la pratique.

La fonction `recherche-compromis` modifie `k-best` (vue page 57) pour implémenter la procédure de recherche de la solution optimale de compromis. Dans la ligne 3, la

*Un algorithme *anytime* est un algorithme dont la qualité de la sortie s'améliore progressivement avec le temps, de sorte que nous avons une sortie raisonnable prête si nous interrompons son exécution.(Russell et Norvig, 2003, page 971)

Entrées : $[u^1, u^2, \ldots, u^m]$, un vecteur de fonctions d'utilité GAI ; h, un critère non-linéaire ; g, l'approximation GAI à utiliser
Sorties : x^*, une solution qui maximise h

```
1  k ← 0
2  sol ← ∅
3  soit u* = −∞
4  répéter
5      k ← k + 1
6      soit x^k la k-meilleure solution obtenue par l'optimisation de l'arbre GAI de g
7      si h(x^k) > u* alors
8          u* ← h(x^k)
9          x* ← x^k
10     fin
11 jusqu'à g(x^k) ≤ u*
```

Algorithme 3.1 : Recherche de la solution de compromis

fonction `MakeGAITree` combine les m fonctions GAI dans un seul arbre GAI, selon l'approximation g. Notons que dans le cas où les m fonctions ont la même structure, l'arbre GAI des m fonctions combinées selon une approximation g aura exactement la même structure que chacune des m fonctions. Toutefois, le cas échéant, le graphe résultant peut ne pas être un arbre GAI, et `MakeGAITree` doit alors retrianguler le réseau GAI résultant.

3.3.1 Expérimentations

Pour évaluer l'efficacité pratique de notre méthode, nous avons fait des expérimentations avec différents jeux d'essai. Nous avons observé le temps de calcul ainsi que le nombre de solutions énumérées pour renvoyer la solution de meilleur compromis, selon les critères considérés. Les expérimentations ont été faites avec un PC 2.1GHz en utilisant une implantation en Java.

Jeux d'essai. Pour les expérimentations, nous avons généré des données artificielles pour des préférences GAI-décomposables. Toutes les décompositions GAI concernaient 20 variables, avec 10 termes d'utilité $u_i(x_{C_i})$ de taille $|x_{C_i}|$ choisie aléatoirement entre 2 et 4 (il semble irréaliste de considérer des interactions encore plus complexes, car l'élicitation serait trop difficile à réaliser). Le domaine de chaque terme u_i a été aléatoirement choisi. Pour les variables qui n'ont été sélectionnées dans aucun domaine, nous avons créé des termes d'utilité unaires. Ensuite, nous avons créé m fonctions d'utilité différentes avec la structure précédemment générée. Ces fonctions représentent les préférences de m agents. Nous avons réalisé des expérimentations pour $m \in \{3, 5, 7\}$. Pour chaque terme

Entrées : $[u^1, u^2, \ldots, u^m]$, un vecteur de fonctions d'utilité GAI ; h, un critère non-linéaire ; g, l'approximation GAI à utiliser
Sorties : x^*, une solution qui maximise h

```
1  x* ← ∅
2  soit u* = −∞
3  G ← MakeGAITree([u^1, u^2, ..., u^m], g)
4  x ← Optimal_choice(G)
5  S ← Split(X_{C_k}, X_{C_k}, x, u(x), ∅)
6  répéter
7      si h(x) > u* alors
8          u* ← h(x)
9          x* ← x
10     fin
11     S ← l'élément (X_{C_l}, X_{C_j}, Forbidden_{C_l}, u'*, y*) de S ayant la plus grande valeur de u'*
12     x ← Instantiate(X_{C_l}, X_{C_j}, y*_{C_j}, Forbidden_{C_l})
13     compléter les attributs manquants dans x avec leurs valeurs correspondantes dans y*
14     S ← S ∪ Split(X_{C_l}, X_{C_j}, x, u'*, Forbidden_{C_l}) \ {S}
15 jusqu'à g(x) ≤ u* ou S = ∅
16 retourner x*
```

Fonction `recherche-compromis`

d'utilité u_i^j d'un agent j, nous générons d'abord sa valeur maximale d'utilité $\max(u_i^j)$ dans l'intervalle $[0, 1]$. Ensuite, nous générons uniformément les valeurs d'utilité pour toutes les configurations de u_i^j dans l'intervalle $[0, \max(u_i^j)]$. Ceci nous donne m fonctions d'utilité GAI différentes avec la même structure. Nous avons généré des données pour les tailles de domaine 2, 5 et 10, ce qui induit respectivement 2^{20}, 5^{20} et 10^{20} configurations possibles. Chaque terme d'utilité u_i^j peut avoir une table d'utilité avec jusqu'à 10 000 éléments (4 variables avec taille de domaine 10). Pour le critère « Tchebycheff », nous avons utilisé des poids $w_j = 1/(\overline{u}^j - \underline{u}^j)$, qui normalisent l'échelle d'utilité et donnent la même importance à chacun des agents.

Résultats. Les résultats moyens (t : temps en ms, et $\#gen$: nombre de solutions générées) sur 100 exécutions sont résumés dans le tableau 3.2.

Nous avons obtenu des temps d'exécution moyens entre 0,2 et 100 secondes. Pour les groupes de 3 et 5 agents, les temps moyens ont été petits, ce que suggère que l'algorithme proposé peut être utilisé pour donner des réponses « en temps réel » à des petits groupes, même quand l'espace de solutions est grand (notons que, en passant d'un problème à 20 attributs avec un domaine de taille 5 à un problème à 20 attributs avec un domaine de taille 10, nous multiplions le nombre de configurations par plus de 10^6, tandis que

TAB. 3.2 – Temps d'exécution et nombre de configurations énumérées pour les différents critères non-linéaires

Nombre de agents	Taille du domaine	max-min		min-max Regret		Tchebycheff	
		t(ms)	$\#gen$	t(ms)	$\#gen$	t(ms)	$\#gen$
3	2	20	601	18	184	16	177
	5	33	1146	47	649	44	484
	10	72	1846	151	745	155	917
5	2	96	4491	59	2318	45	1622
	5	2074	80889	809	30136	766	29869
	10	2642	106437	1816	67446	1911	70770
7	2	952	40560	271	11134	234	9454
	5	43452	1465662	14347	494899	18550	645142
	10	99484	2908370	64430	1885490	67193	1960612

le nombre de solutions énumérées a normalement augmenté d'un facteur inférieur à 3). Pour $m = 7$, nous notons que l'efficacité a été déterminé par la taille du problème. Quand m est grand, nous nécessitons d'énumérer un plus grand nombre d'alternatives pour arriver au critère d'arrêt, une fois que l'utilité des meilleures solutions de l'approximation GAI utilisée diminue plus lentement. Toutefois, dans des problèmes où des réponses immédiates ne sont pas nécessaires, un temps de réponse dans l'ordre d'une minute peut être toujours acceptable.

Pour mieux visualiser la dispersion des données obtenues, la figure 3.6 montre des Box Plots (ou boîtes à moustaches) pour les différentes configurations testées. Son interprétation est celle usuelle pour des Box Plots (Chambers *et al.*, 1983) :

- la base de la boîte indique le premier quartile (c'est-à-dire que 25% des instances ont eu un temps d'éxécution inférieur au temps marqué) ;
- chaque boîte est coupée par la médiane (c'est-à-dire que 50% des instances ont eu un temps d'éxécution inférieur au temps marqué) ;
- le plafond de la boîte indique le troisième quartile (c'est-à-dire que 75% des instances ont eu um temps d'éxécution inférieur au temps marqué) ;
- si les « entailles » au tour de la médiane de deux boîtes ne s'interposent pas, il y a une forte évidence que les deux médianes différent avec une degré de confidence de 95% ;
- la longueur des « moustaches » vaut 1,5 fois l'écart interquartile.* Dans le cas où les données peuvent être modélisées en utilisant une loi normale, la théorie montre

*L'écart interquartile est la différence entre le troisième et le premier quartile. Il correspond à l'étendue de la série statistique après l'élimination de 25% des valeurs les plus faibles et de 25% des valeurs les plus fortes.

que les extremités des « moustaches » sont voisines du premier et 99^{e} centile. Cela signifie que 1% des instances ont eu un temps inférieur à la moustache inférieure, et 1% des instances ont eu un temps supérieur à la moustache supérieure. Telles instances sont considérées comme des données exceptionnelles (*outliers*).

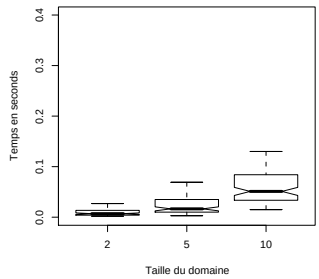
Nous observons que la croissance de la taille du problème non seulement génère des temps d'exécution moyens plus grands, mais aussi les temps d'exécution sont plus dispersés. Toutefois, pour les groupes de 3 et 5 agents, les temps d'exécution ont été toujours petits. Pour les groupes de 7 agents, le temps d'exécution peut s'élever à plusieurs minutes pour certaines instances de grande taille (variables de domaine de taille 10). Grâce à la caractéristique *anytime* de la méthode, dans des instances plus exceptionnelles, nous pouvons opter pour interrompre l'algorithme et utiliser la meilleure solution au moment de l'interruption (potentiellement sous-optimale, avec une perte d'utilité maximale donnée par l'équation 3.1).

Nous avons aussi réalisé des expérimentations où chaque agent avait une fonction GAI de structure différente. Dans ce cas, pour générer la fonction GAI agrégée, on doit d'abord trianguler le graphe induit par les fonctions d'utilité des agents. Quand les structures des fonctions GAI des agents sont très différentes, le nouvel arbre GAI agrégé peut avoir des cliques très grandes, ce qui empêche l'utilisation de l'algorithme. Toutefois, dans bon nombre de situations pratiques, la différence de préférences entre les agents est due principalement aux valeurs d'utilité, tandis que la structure d'indépendance reste presque la même d'un agent à l'autre. Ceci devrait contribuer à garantir une certaine efficacité de la procédure proposée dans une situation réelle.

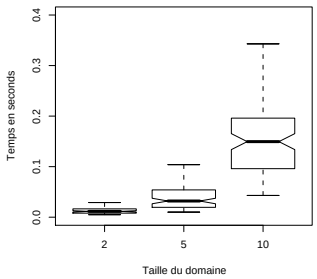
3.4 La recherche non-interactive de compromis

Le critère de Tchebycheff est un outil très adapté pour l'exploration interactive de l'ensemble des solutions non-dominées au sens de Pareto (voir par exemple Steuer et Choo (1983); Steuer *et al.* (1993); Kim et Kim (2006)). Toutefois, dans certaines situations, au lieu d'explorer de façon interactive les solutions non-dominées, nous pouvons être intéressés par la solution optimale selon des critères non-linéaires fixes, par exemple, des critères linguistiques flous, qui cherchent à représenter le discours humain ; par exemple, la solution où *autant plus que possible* des agents soient satisfaits. Comme il a été suggéré par Yager (1988), ces fonctions de décision floues peuvent être implantées en utilisant des opérateurs de moyenne pondérée ordonnée (OWA, de l'anglais *Ordered Weight Averaging*). De leur côté, les opérateurs OWA sont un cas particulier de l'intégrale de Choquet, une fonction d'agrégation floue fondée sur une capacité (Grabisch, 1995). Dans cette section nous analysons l'applicabilité de l'algorithme de recherche de la solution de compromis

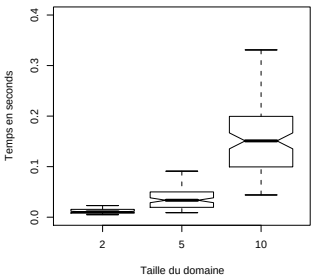
3 agents



(a) max-min

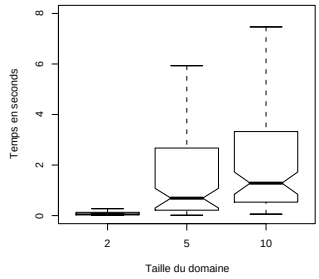


(b) min-max Regret

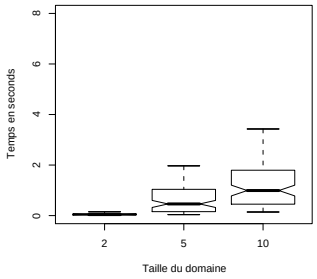


(c) Tchebycheff

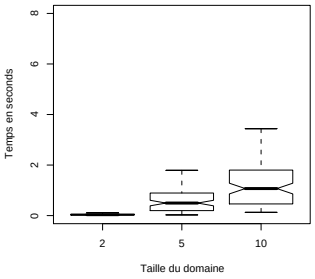
5 agents



(d) max-min

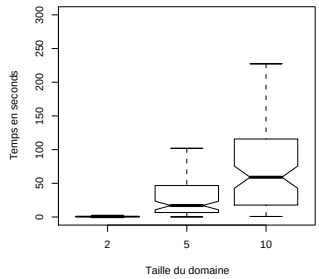


(e) min-max Regret

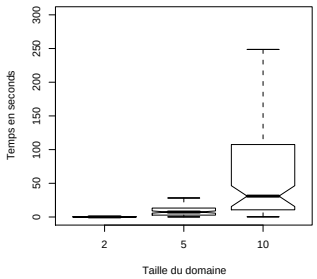


(f) Tchebycheff

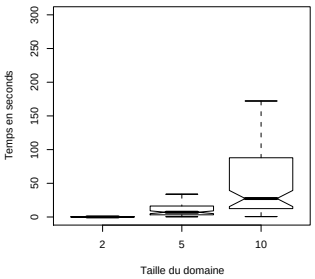
7 agents



(g) max-min



(h) min-max Regret



(i) Tchebycheff

FIG. 3.6 – Box Plots pour les différentes configurations

avec l'aide d'une fonction GAI, quand la fonction d'agrégation non-linéaire utilisée est une intégrale de Choquet.

3.4.1 Opérateurs OWA, quantifieurs linguistiques et intégrale de Choquet

Définition 3.1. Soit $w = (w_1, w_2, \ldots, w_m)$ un vecteur de poids, tel que $w_j \in [0,1], \forall j \in \{1, \ldots, m\}$ et $\sum_{j=1}^{m} w_j = 1$. Alors l'opérateur OWA est défini comme :

$$\mathrm{OWA}_w(x) = \sum_{j=1}^{m} w_j u^{(j)} \tag{3.2}$$

où (.) représente une permutation de $\{1, \ldots, m\}$, telle que $u^{(1)} \leq u^{(2)} \leq \ldots \leq u^{(m)}$. Notons que le type d'agrégation réalisé est très dépendant du vecteur de poids utilisé. Par exemple, avec un vecteur de poids W, tel que $w_1 = 1$ et $w_j = 0, \forall j \neq 1$, nous obtenons le Min, tandis que si W est tel que $w_m = 1$ et $w_j = 0, \forall j \neq m$ nous obtenons le Max.

Les opérateurs OWA peuvent représenter des critères linguistiques, tels que *tous*, *la plupart*, *au moins la moitié*, en utilisant des quantifieurs flous non-décroissants qui représentent le discours humain (nommés *quantifieurs linguistiques* par Zadeh (1983)) pour choisir leur vecteur de poids associé.

Définition 3.2. Une fonction d'appartenance $Q : [0,1] \mapsto [0,1]$, telle que $Q(0) = 0$, $Q(1) = 1$ et si $x > y$, alors $Q(x) \geq Q(y)$ est un *quantifieur flou non-décroissant.*

Un **quantifieur flou non-décroissant** peut être représenté comme :

$$Q(r) = \begin{cases} 0 & \text{si } r < a, \\ \frac{r-a}{b-a} & \text{si } a \leq r \leq b, \\ 1 & \text{si } r > b, \end{cases} \tag{3.3}$$

avec $a, b, r \in [0,1]$.

Des quantifieurs flous sont utilisés pour représenter le concept de majorité floue. Traditionnellement, la majorité est définie comme un seuil en fonction du nombre d'individus. Par exemple, pour dix individus donnés nous pouvons définir « six » comme étant le seuil. La majorité floue, en revanche, est une notion plus souple, comprise dans la logique floue, qui utilise des quantifieurs linguistiques flous pour exprimer le discours humain. Des *quantifieurs flous non-décroissants* peuvent être utilisés pour représenter des termes tels que *la plupart*, *au moins la moitié*, *autant que possible*. Dans ce contexte,

pour tout $r \in [0,1]$, $Q(r)$ indique la mesure dans laquelle la quantité r est compatible avec le terme linguistique représenté par Q.

La procédure utilisée pour la génération de poids OWA à partir d'un quantifieur Q est (Yager, 1988) :

$$w_j = Q\left(\frac{m-j+1}{m}\right) - Q\left(\frac{m-j}{m}\right) \quad \text{pour } j = 1, \ldots, m$$

Les opérateurs OWA sont un cas particulier de l'intégrale de Choquet, une fonction d'agrégation floue fondée sur une capacité. La caractéristique distinctive de l'intégrale de Choquet est sa capacité de représenter un certain type d'interaction entre les éléments qui seront agrégés. Cette interaction peut aller de la redondance (interaction négative) à la synergie (interaction positive) (Grabisch, 1996). Dans le contexte de la décision collective, cela ouvre la possibilité de coalitions entre les individus.

Définition 3.3. Soit $\mathcal{D} = \{1, \ldots, m\}$ un ensemble d'indices. Une capacité* sur $\mathcal{D}$ est une fonction d'ensemble $\mu : 2^{\mathcal{D}} \mapsto [0,1]$, telle que $\forall S, T \in 2^{\mathcal{D}}$, $S \subseteq T \Rightarrow \mu(S) \leq \mu(T)$; $\mu(\emptyset) = 0$ et $\mu(\mathcal{D}) = 1$.

Définition 3.4. L'intégrale de Choquet d'un vecteur $(u^1, u^2, \ldots, u^m) \in \mathbb{R}_+^m$ par rapport à une capacité μ est définie comme :

$$C_\mu([u^1, \ldots, u^m]) = \sum_{j=1}^{m} \left[\mu(\mathcal{D}_{(j)}) - \mu(\mathcal{D}_{(j+1)})\right] u^{(j)} \tag{3.4}$$

$$= \sum_{j=1}^{m} \left[u^{(j)} - u^{(j-1)}\right] \mu(\mathcal{D}_{(j)}) \tag{3.5}$$

où (.) représente une permutation de $\{1, \ldots, m\}$, telle que $(0 = u^{(0)}) \leq u^{(1)} \leq \ldots \leq u^{(m)}$; $\mathcal{D}_{(j)} = \{k \in \{1, \ldots m\}, u^k \geq u^{(j)}\} = \{(j), (j+1), \ldots, (m)\}$ pour $j \leq m$ et $\mathcal{D}_{(m+1)} = 0$. Notons que $\mathcal{D}_{(j+1)} \subset \mathcal{D}_{(j)}$, alors $\mu(\mathcal{D}_{(j)}) \geq \mu(\mathcal{D}_{(j+1)})$ pour tout j.

Proposition 3.2 (Torra (1998))**.** *Pour tout quantifieur flou non-décroissant Q, nous avons* $\text{OWA}_Q = C_{\mu_Q}$*, où μ_Q est définie comme :*

$$\mu_Q(B) = Q(|B|/|\mathcal{D}|) \quad \forall B \in 2^{\mathcal{D}},$$

nous nommerons μ_Q une capacité fondée sur Q.

Définition 3.5. Une capacité μ est nommée *convexe* (ou supermodulaire) quand $\mu(A \cup B) + \mu(A \cap B) \geq \mu(A) + \mu(B)$, et elle est nommée *concave* (ou submodulaire) quand $\mu(A \cup B) + \mu(A \cap B) \leq \mu(A) + \mu(B)$, $\forall A, B \in 2^{\mathcal{D}}$. Une capacité est nommée *additive*

*Aussi appelé une mesure floue.

quand $\mu(A \cup B) + \mu(A \cap B) = \mu(A) + \mu(B)$, $\forall A, B \in 2^{\mathcal{D}}$, et dans ce cas elle est appelée une *mesure de probabilité*. Étant donnée une capacité μ, nous pouvons obtenir sa *duale* $\bar{\mu}(A) = 1 - \mu(\mathcal{D} \backslash A), \forall A \in 2^{\mathcal{D}}$. On sait que μ est convexe ssi $\bar{\mu}$ est concave, et vice-versa.

Une intégrale de Choquet définie à partir d'une capacité μ donnera plus de poids dans l'agrégation aux valeurs plus petites à agréger, si μ est convexe. Inversement, si μ est concave, l'intégrale de Choquet donnera plus de poids dans l'agrégation aux valeurs plus grandes (Wakker, 2001). Dans notre cadre, si l'on considère que nous agrégeons les utilités de différents agents dans la recherche d'une solution de compromis, des intégrales de Choquet favorisant les minorités semblent être plus adaptées. Dans ce contexte, les intégrales fondées sur une capacité convexe peuvent être des bonnes candidates.

3.4.2 Recherche de la solution de compromis pour une intégrale de Choquet

Quand le compromis recherché par la fonction h correspond à une intégrale de Choquet par rapport à une capacité convexe, la proposition suivante nous sera utile pour trouver une fonction g qui satisfait les conditions de la proposition 3.1, et ainsi nous permettra d'utiliser l'algorithme présenté dans la section 3.1.

Proposition 3.3 (Shapley (1971))**.** *Quand μ est convexe, nous avons* $1 \geq \mu(A) + \mu(\mathcal{D} \backslash A) \Leftrightarrow 1 \geq \mu(A) + 1 - \bar{\mu}(A) \Leftrightarrow \bar{\mu}(A) \geq \mu(A)$*. Dans ce cas, nous avons :*

$$\text{core}(\mu) = \{P \in \mathcal{L}, \mu(A) \leq P(A) \leq \bar{\mu}(A)\} \neq \emptyset, \forall A \in 2^{\mathcal{D}}$$

où $\mathcal{L}$ est l'ensemble de mesures de probabilité dans $\mathcal{D}$.

Cette proposition permet d'arriver au résultat suivant :

Proposition 3.4 (Galand et Perny (2007))**.** *Si μ est convexe, alors $\forall P \in \text{core}(\mu)$ la propriété suivante est vérifiée : $C_\mu([u^1, \ldots, u^m]) \leq \sum_{j=1}^m p_j u^j$, où $p_j = P(\{j\})$.*

Démonstration : $C_\mu = \sum_{j=1}^m \left[u^{(j)} - u^{(j-1)}\right] \mu(\mathcal{D}_{(j)}) \leq \sum_{j=1}^m \left[u^{(j)} - u^{(j-1)}\right] P(\mathcal{D}_{(j)})$, puisque $\mu(\mathcal{D}_{(j)}) \leq P(\mathcal{D}_{(j)}), \forall P \in \text{core}(\mu)$ (proposition 3.3). Aussi, $\sum_{j=1}^m \left[u^{(j)} - u^{(j-1)}\right] P(\mathcal{D}_{(j)}) = \sum_{j=1}^m \left[P(\mathcal{D}_{(j)}) - P(\mathcal{D}_{(j+1)})\right] u^{(j)} = \sum_{j=1}^m p_{(j)} u^{(j)} = \sum_{j=1}^m p_j u^j$. ■

Ainsi, toute mesure de probabilité $P \in \text{core}(\mu)$ peut être utilisée pour obtenir une fonction g compatible avec la proposition 3.1. Notons qu'étant une combinaison linéaire de fonctions GAI, g sera aussi une fonction GAI. Pour trouver P, nous pouvons utiliser des méthodes disponibles dans la littérature pour obtenir une probabilité dans le core d'une capacité convexe. Nous avons testé l'algorithme de l'entropie maximale de Jaffray (Jaffray,

1995; Baroni et Vicig, 2006), implanté par la fonction `maximum-entropy`, et des valeurs de Shapley (Shapley, 1971), implanté par la fonction `shapley`.

Entrées : $\mu : 2^{\mathcal{D}} \mapsto [0,1]$, une capacité convexe
Sorties : $P \in \text{core}(\mu)$, une mesure de probabilité

soit $\bar{\mu}$ la duale de μ
$A^0 \leftarrow \emptyset, B^0 \leftarrow \emptyset, k \leftarrow 1$
répéter
  $A^k = \text{Argmin}_{\emptyset \neq E \subseteq (\mathcal{D} \setminus B^{k-1})} \frac{\bar{\mu}(B^{k-1} \cup E) - \bar{\mu}(B^{k-1})}{|E|}$
  $B^k \leftarrow B^{k-1} \cup A^k$
  pour tous les $\omega \in A^k$ **faire**
    $P(\omega) \leftarrow \frac{\bar{\mu}(B^{k-1} \cup A^k) - \bar{\mu}(B^{k-1})}{|A^k|}$
  fin
  $k \leftarrow k + 1$
jusqu'à $B^{k-1} = \mathcal{D}$
retourner P

Fonction `maximum-entropy`

Entrées : $\mu : 2^{\mathcal{D}} \mapsto [0,1]$, une capacité convexe
Sorties : $P \in \text{core}(\mu)$, une mesure de probabilité

pour tous les $\omega \in \mathcal{D}$ **faire**
  $P(\omega) \leftarrow \sum_{E \subseteq D \setminus \omega} \frac{|E|!(|\mathcal{D}| - |E| - 1)!}{|D|!} (\mu(E \cup \omega) - \mu(E))$
fin
retourner P

Fonction `shapley`

Comme nous l'avons vu, la méthode proposée n'est adaptée que pour des cas où le compromis recherché par la fonction h correspond à une intégrale de Choquet par rapport à une capacité convexe. Nous voyons maintenant ce que cela signifie dans le cadre de majorités floues fondées sur des quantifieurs flous non-décroissants.

Proposition 3.5. *Une capacité μ_Q fondée sur un quantifieur flou non-décroissant Q avec $b = 1$ est convexe.*

Démonstration : Tout d'abord, notons qu'à partir de la proposition 3.2 et de l'équation 3.3, nous pouvons conclure que $\mu_Q(A \cup B) \geq \mu_Q(A \cap B)$, que $\mu_Q(A \cup B) \geq \mu_Q(A)$ et que $\mu_Q(A \cup B) \geq \mu_Q(B)$, pour tout A, $B \subseteq X$. Nous devons considérer trois cas :

- $|A \cup B|/|X| < a$: ici $\mu_Q(A \cup B) = \mu_Q(A \cap B) = \mu_Q(A) = \mu_Q(B) = 0$, satisfaisant ainsi la propriété de convexité.

- $|A \cup B|/|X| \geq a$ et $|A \cap B|/|X| \geq a$: ici,

$$\mu_Q(A \cup B) + \mu_Q(A \cap B) = \frac{\frac{|A \cup B|}{|X|} - a}{1-a} + \frac{\frac{|A \cap B|}{|X|} - a}{1-a} = \frac{\frac{|A \cup B| + |A \cap B|}{|X|} - 2a}{1-a}$$
$$= \frac{\frac{|A|+|B|-|A \cap B|+|A \cap B|}{|X|} - 2a}{1-a} = \frac{\frac{|A|}{|X|} - a}{1-a} + \frac{\frac{|B|}{|X|} - a}{1-a} = \mu_Q(A) + \mu_Q(B),$$

  satisfaisant ainsi la propriété de convexité.

- $|A \cup B|/|X| \geq a$ et $|A \cap B|/|X| < a$: ici,

$$\mu_Q(A \cup B) + \mu_Q(A \cap B) = \frac{\frac{|A \cup B|}{|X|} - a}{1-a} + 0 = \frac{\frac{|A|+|B|}{|X|} - \frac{|A \cap B|}{|X|} - a}{1-a}$$
$$> \frac{\frac{|A|+|B|}{|X|} - 2a}{1-a}, \text{ puisque } \frac{|A \cap B|}{|X|} < a, \text{ par définition.}$$

  Puisque $\frac{\frac{|A|+|B|}{|X|} - 2a}{1-a} \geq \mu_Q(A) + \mu_Q(B)$, nous avons $\mu_Q(A \cup B) + \mu_Q(A \cap B) > \mu_Q(A) + \mu_Q(B)$, satisfaisant ainsi la propriété de convexité. ■

Proposition 3.6. *Une capacité μ_Q fondée sur un quantifieur flou non-décroissant Q avec $a = 0$ est concave.*

Démonstration : Nous devons considérer trois cas :

- $|A \cup B|/|X| > b$ et $|A \cap B|/|X| > b$: ici $\mu_Q(A \cup B) = \mu_Q(A \cap B) = \mu_Q(A) = \mu_Q(B) = 1$, satisfaisant ainsi la propriété de concavité.
- $|A \cup B|/|X| \leq b$ et $|A \cap B|/|X| \leq b$: ici,

$$\mu_Q(A \cup B) + \mu_Q(A \cap B) = \frac{\frac{|A \cup B|}{|X|}}{b} + \frac{\frac{|A \cap B|}{|X|}}{b} = \frac{\frac{|A|+|B|-|A \cap B|+|A \cap B|}{|X|}}{b}$$
$$= \frac{\frac{|A|}{|X|}}{b} + \frac{\frac{|B|}{|X|}}{b} = \mu_Q(A) + \mu_Q(B),$$

  satisfaisant ainsi la propriété de concavité.

- $|A \cup B|/|X| > b$ et $|A \cap B|/|X| \leq b$: ici,

$$\mu_Q(A \cup B) + \mu_Q(A \cap B) = 1 + \frac{\frac{|A \cap B|}{|X|}}{b} \leq \mu_Q(A) + \mu_Q(B),$$
$$\text{puisque } \frac{\frac{|A \cup B|}{|X|}}{b} > 1, \text{ par définition ;}$$
$$\mu_Q(A \cap B) \leq \mu_Q(A) \text{ et } \mu_Q(A \cap B) \leq \mu_Q(B) \text{ pour tout } A, B \subseteq X,$$

  satisfaisant ainsi la propriété de concavité. ■

Proposition 3.7. *Une capacité μ_Q fondée sur un quantifieur flou non-décroissant Q avec $a > 0$ et $b < 1$, peut être ni concave, ni convexe.*

Exemple 3.2. Considérons Q avec $a = 0{,}3$ et $b = 0{,}8$, $X = \{1, 2, 3, 4, 5\}$.

- μ_Q n'est pas convexe. Par exemple, avec $A = \{1, 2\}$ et $B = \{1, 3, 4, 5\}$, nous avons :

$$\mu_Q(A \cup B) + \mu_Q(A \cap B) = Q(5/5) + Q(1/5) = 1$$
$$\mu_Q(A) + \mu_Q(B) = Q(2/5) + Q(4/5) = 1{,}2$$

- μ_Q n'est pas concave. Par exemple, avec $A = \{1\}$ et $B = \{2\}$, nous avons :

$$\mu_Q(A \cup B) + \mu_Q(A \cap B) = Q(2/5) + Q(0/5) = 0{,}2$$
$$\mu_Q(A) + \mu_Q(B) = Q(1/5) + Q(1/5) = 0$$

De cette manière, lorsque l'on utilise des majorités floues, la méthode proposée est appropriée quand nous avons $b = 1$ pour le quantifieur flou. C'est le cas par exemple pour le quantifieur *autant que possible*, qui normalement a $a = 0{,}5$ et $b = 1$ comme ses paramètres associés.

3.4.3 Recherche de la solution de compromis pour un opérateur OWA

La méthode d'optimisation présentée dans la section 3.1 est également applicable quand la fonction d'agrégation s'agit d'un opérateur OWA avec un jeu de poids décroissants, grâce au résultat suivant :

Proposition 3.8 (Galand et Spanjaard (2007))**.** * *Pour tout vecteur* $u \in \mathbb{R}_+^m$ *et tout jeu de poids décroissants* $w = (w_1, \ldots, w_m)$*, tel que* $w_j \in [0, 1], \forall j \in \{1, \ldots, m\}$ *et* $\sum_{j=1}^m w_j = 1$*, on vérifie* $\mathrm{OWA}_w(x) \leq \mathrm{AVG}(x)$*, où* $\mathrm{AVG}(x) = \frac{1}{m}\sum_{j=1}^m u^j$ *.*

Démonstration : Considérons la différence $D = \mathrm{AVG}(x) - \mathrm{OWA}_w(x)$. On a $D = \sum_{j=1}^m \frac{1}{m}u^j - \sum_{j=1}^m w_j u^{(j)} = \sum_{j=1}^m \left(\frac{1}{m} - w_j\right) u^{(j)}$. Soit $k \in \{1, \ldots, m\}$ tel que $\forall j \in \{1, \ldots, k\}$, $w_j > \frac{1}{m}$ et $\forall j \in \{k+1, \ldots, m\}$, $\frac{1}{m} \geq w_j$. On peut décomposer D en : $\sum_{j=1}^k \left(\frac{1}{m} - w_j\right) u^{(j)} + \sum_{j=k+1}^m \left(\frac{1}{m} - w_j\right) u^{(j)}$. Comme $u^{(1)} \leq u^{(2)} \leq \ldots \leq u^{(m)}$, alors :

- $\sum_{j=1}^k \left(\frac{1}{m} - w_j\right) u^{(j)} \geq \sum_{j=1}^k \left(\frac{1}{m} - w_j\right) u^{(k)}$, puisque $\forall j \in \{1, \ldots, k\}, \left(\frac{1}{m} - w_j\right) < 0$, par construction ;
- $\sum_{j=k+1}^m \left(\frac{1}{m} - w_j\right) u^{(j)} \geq \sum_{j=k+1}^m \left(\frac{1}{m} - w_j\right) u^{(k+1)}$, puisque $\forall j \in \{k+1, \ldots, m\}$, $\left(\frac{1}{m} - w_j\right) \geq 0$, par construction.

Ainsi,

$$D \geq \sum_{j=k+1}^{m} \left(\frac{1}{m} - w_j\right) u^{(k+1)} - \sum_{j=1}^{k} \left(w_j - \frac{1}{m}\right) u^{(k)} \tag{3.6}$$

*Le résultat présenté ici diffère légèrement de celui obtenu par Galand et Spanjaard (2007) une foi que nous cherchons à maximiser l'utilité tandis qu'ils cherchent la minimisation de coûts. Aussi, nous utilisons une permutation en ordre croissant dans la définition de l'opérateur OWA (équation 3.2), tandis que Galand et Spanjaard (2007) utilisent la définition analogue avec une permutation en ordre décroissant.

Notons que $\sum_{j=1}^{m}\left(w_j - \frac{1}{m}\right) = 0$ puisque $\sum_{j=1}^{m} w_j = 1 = \sum_{j=1}^{m} \frac{1}{m}$. Ainsi,

$$\sum_{j=1}^{k}\left(w_j - \frac{1}{m}\right) + \sum_{j=k+1}^{m}\left(w_j - \frac{1}{m}\right) = 0 \Rightarrow \sum_{j=1}^{k}\left(w_j - \frac{1}{m}\right) = \sum_{j=k+1}^{m}\left(\frac{1}{m} - w_j\right).$$

Nommons $W = \sum_{j=1}^{k}\left(w_j - \frac{1}{m}\right) = \sum_{j=k+1}^{m}\left(\frac{1}{m} - w_j\right)$. Nous pouvons donc récrire l'équation (3.6) comme :

$$D \geq W(u^{(k+1)} - u^{(k)}) \tag{3.7}$$

Il est aisé de vérifier que $W(u^{(k+1)} - u^{(k)})$ est positif puisque nous avons $u^{(1)} \leq u^{(2)} \leq \ldots \leq u^{(m)}$ et W est positif par construction. On en conclut alors que $\mathrm{AVG}(x) \geq \mathrm{OWA}_w(x)$. ■

Grâce à la proposition 3.8, la méthode d'optimisation présentée dans la section 3.1 s'applique aussi quand $h(x)$ s'agit d'un opérateur OWA avec un vecteur de poids décroissants, une fois que la moyenne satisfait les conditions de la proposition 3.1 pour la fonction $g(x)$.

L'utilisation d'un opérateur OWA_w avec un vecteur w de poids décroissants favorise des solutions de compromis dans le sens que l'on vérifie l'axiome suivant :

- **Principe de transfert :** Soit $x \in R_+^m$ tel que $x_i > x_j$ pour quelque i, j. Alors, pour tout ϵ tel que $0 \leq \epsilon \leq x_i - x_j$, $x - \epsilon e_i + \epsilon e_j \succsim x$, où e_i (respectivement e_j) est le vecteur avec le i^{e} (respectivement j^{e}) composant égal à 1, tous les autres composants étant nuls.

Cet axiome exprime l'idée d'équité dans la mesure où pour un profil d'utilité $x \in R_+^m$, avec $x_i > x_j$, une légère amélioration de x_j au détriment de x_i produit un profil d'utilité plus équilibré et donc un meilleur compromis. Tels transferts, connus comme des **transferts de Pigou-Dalton** dans la littérature du choix social, sont utilisés pour réduire l'inégalité de la distribution du revenu dans une population (voir, par exemple, Sen (1997)). Ils ont aussi été utilisés dans le cadre de l'optimisation robuste de graphes finis avec des coûts variant en fonction du scénario (Perny *et al.*, 2006).

3.4.4 Expérimentations

Pour évaluer empiriquement la méthode de recherche de la solution de compromis dans le cadre d'intégrales de Choquet convexes, nous avons réalisé des expérimentations avec le même jeux d'essai utilisé dans la section 3.3.1. Nous avons testé trois capacités convexes différentes :

- μ_1 : la capacité qui correspond à la majorité floue *autant que possible*, comme nous l'avons précédemment décrite ;
- $\mu_2(A) = \sum_{j \in A} p_j, \forall A \in 2^{\mathcal{D}}$: pour une distribution de probabilité tirée aléatoirement

$(p_1, \ldots, p_m)$;

- $\mu_3(A) = \sum_{B \subseteq A} \varphi(B), \forall A \in 2^{\mathcal{D}}$: où $\varphi(C), C \subseteq \mathcal{D}$ sont des masses positives de Möbius tirées aléatoirement et qui somment 1 (voir par exemple Chateauneuf et Jaffray (1995) pour les détails).

Résultats. Les résultats moyens (t : temps en ms et $\#gen$: nombre de configurations générées) sur 30 exécutions sont résumés dans le tableau 3.3.

TAB. 3.3 – Temps d'exécution et nombre de configurations énumérées

Taille du	μ_1		μ_2		μ_3	
domaine	t(ms)	$\#gen$	t(ms)	$\#gen$	t(ms)	$\#gen$
2	60	1060	20	96	45	1035
5	298	8634	81	469	230	6250
10	911	26436	242	1288	777	22515

Comme nous pouvons le constater, tous les temps moyens obtenus sont en-dessous d'1 seconde. Si nous comparons ces résultats avec ceux du tableau 3.2, nous constatons que la méthode a été plus efficace avec les intégrales de Choquet qu'avec les critères utilisés dans la section 3.3. De même, avec ce qui a été vu précédemment, la taille du domaine des attributs a été la variable la plus importante dans le temps d'exécution (et le nombre de configurations énumérées). Aussi, le nombre d'éléments qui ont dû être énumérés avant l'arrêt de l'algorithme augmente beaucoup plus lentement que la taille du problème.

Ces résultats ont été obtenus en utilisant `maximum-entropy` pour trouver $P \in \text{core}(\mu)$. Nous avons observé que si nous utilisons `shapley`, dans certaines instances, l'énumération perdure à plusieurs milliers de configurations, sans que la condition d'arrêt soit atteinte. Le tableau 3.4 montre les temps moyens obtenus en utilisant les valeurs de Shapley. Pour μ_3 et les variables avec un domaine de taille 10, la condition d'arrêt n'a pas été atteinte dans un temps limite de 2 minutes pour quelques instances.

TAB. 3.4 – Temps d'exécution et nombre de configurations énumérées : valeurs de Shapley

Taille du	μ_1		μ_2		μ_3	
domaine	t(ms)	$\#gen$	t(ms)	$\#gen$	t(ms)	$\#gen$
2	54	1060	40	724	53	1270
5	299	8634	4456	135323	453	14131
10	896	26436	2400	80113	-	-

3.5 Conclusion

Dans ce chapitre, nous avons considéré le cas où nous ne cherchons pas les meilleures solutions selon un seul point de vue, mais chaque alternative est évaluée selon un ensemble de point de vues. Plus spécifiquement, nous nous posons dans le cadre où chaque alternative est évaluée selon un ensemble de fonctions d'utilité, et chacune de ces fonctions est une fonction GAI-décomposable sur des attributs du domaine.

Après avoir constaté que des opérateurs linéaires tels que la somme pondérée ne constituent pas des bonnes fonctions d'agrégation, nous nous tournons vers des opérateurs non-linéaires, tels que les normes de Tchebytcheff, l'opérateur OWA et l'intégrale de Choquet. Malgré le fait que l'optimisation de ces critères non-linéaires soit un problème NP-difficile, nous avons pu développer une méthode efficace dans la pratique pour la détermination exacte de la solution de meilleur compromis selon de tels critères non-linéaires d'agrégation.

Dans le prochain chapitre, nous analysons une autre difficulté pour l'utilisation de fonctions GAI dans la modélisation de préférences, la construction du modèle d'utilité, c'est-à-dire, son élicitation.

Chapitre 4

Élicitation de modèles GAI

« Le cœur est tortueux par-dessus tout, et il est méchant : Qui peut le connaître ? »

La Bible, livre de Jérémie chapitre 17, verset 9
(siècles VI-VII av. J.-C.)

Ce chapitre s'appuie en partie sur des résultats présentés dans Queiroz (2007a); Gonzales *et al.* (2007)

Résumé

La construction d'un modèle de préférences qui soit à la fois capable de refléter avec fidélité les préférences d'un décideur tout en restant traitable d'un point de vue computationnel est un défi majeur de la problématique décisionnelle.

Pour le modèle le plus utilisé, l'additif, la littérature nous présente des techniques classiques qui exploitent les indépendances du modèle afin de réduire considérablement l'effort d'élicitation. Des techniques focalisées sur la résolution d'un problème spécifique et/ou approximatives peuvent aussi collaborer dans la réduction de cet effort.

Dans la mesure où les modèles GAI-décomposables généralisent les utilités additives, nous pouvons nous inspirer des techniques utilisées dans le cas additif pour développer de techniques d'élicitation adaptées aux modèles GAI-décomposables.

Dans ce chapitre, nous rappelons les techniques classiques d'élicitation de préférences additivement décomposables. Nous présentons ensuite les techniques d'élicitation de préférences GAI-décomposables récemment développées dans la littérature des préférences, notamment l'élicitation fondée sur les GAI-Nets. Une méthode d'élicitation dérivée de la technique d'élicitation fondée sur les GAI-Nets ici presentée sera utilisée dans une application Web que nous développerons dans le chapitre suivant. Enfin, nous introduisons une nouvelle méthode d'élicitation partielle heuristique spécialement adaptée au problème de rangement des k-meilleures solutions selon un modèle GAI.

4.1 La construction de modèles d'utilité

Construire un modèle capable de refléter les préférences d'un décideur s'avère être une tâche très difficile. Comme le souligne Doyle (2004), la structure psychologique d'un individu est tellement riche que l'existence d'un modèle formel, à la fois fidèle et traitable d'un point de vue computationnel, peut être remise en question. En particulier, des modèles utilisées dans le domaine de la philosophie, telles que celle du libre arbitre de Frankfurt (1971), suggèrent que les personnes possèdent des préférences circulaires et potentiellement conflictuelles. Par ailleurs, les modèles les plus utilisés se sont concentrés traditionnellement sur ce que Doyle (2004) appelle des « *nice preferences* », c'est-à-dire des préférences avec des caractéristiques listées dans la figure 4.1.

1. **Coherent preferences:** the agents preferences fit together transitively, in that the agent prefers a to c whenever it prefers a to b and b to c.
2. **Consistent preferences:** the agent does not prefer a to b at the same time as it prefers b to a.
3. **Complete preferences:** for each a and b, the agent either prefers a to b, prefers b to a, or holds them equally desirable.
4. **Comparatively constant preferences:** the agents preferences do not change from one instant to the next, or else change very slowly with respect to decisions and actions.
5. **Concrete preferences:** the agents preferences (as opposed to multiattribute representations) relate concrete alternatives, not abstract qualities or properties of alternatives.
6. **Conveniently cleaved preferences:** the agents preferences divide along convenient conceptual lines, admitting additive representations over independent dimensions.
7. **Completely comparative preferences:** preferences serve only to indicate comparisons between alternatives in making choices.

FIG. 4.1 – Caratéristiques des « nice preferences », selon Doyle (2004)

.

Les hypothèses ci-dessus simplifient considérablement la construction du modèle de préférences. Dans le cadre des décompositions GAI, quoique certaines de ces suppositions soient toujours appliquées (comme, par exemple, l'hypothèse 4), nous nous écartons du cadre classique, une fois que la décomposition additive n'est plus observée. Pourtant, la décomposabilité joue un rôle très important pour limiter la complexité de la construction du modèle. La décomposabilité du modèle aura un impact non seulement sur le nombre de

questions nécessaires pour construire une fonction d'utilité qui modélisera les préférences du décideur à partir de questions posées à ce dernier, mais aussi sur la complexité de chaque question. Du fait de la relative complexité du modèle GAI, on peut s'attendre à ce que les procédures d'élicitation soient un peu lourdes à mettre en place. Heureusement, nous pouvons observer que l'élicitation d'utilités GAI-décomposables est étroitement liée à celle des utilités additives. À titre d'exemple, considérons l'arbre GAI de la figure 4.2. Cet arbre peut être utilisé pour représenter la décomposition GAI suivante :

$$u(a,b,c) = u_1(a,b) + u_2(b,c)\ \forall (a,b,c) \in A \times B \times C, A = \{a^0, a^1\}, B = \{b^0, b^1\}, C = \{c^0, c^1\}.$$

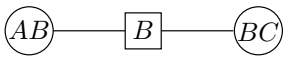


FIG. 4.2 – Arbre GAI pour $u(a,b,c) = u_1(a,b) + u_2(b,c)$.

Fixons maintenant la valeur de l'attribut B, par exemple à $B = b^0$. Alors, $u(a,b^0,c) = u_1(a,b^0) + u_2(b^0,c)$ devient une utilité additive sur $A \times C$. Ainsi, pour chaque valeur $b \in B$, les techniques classiques d'élicitation d'utilités additives nous permettent de construire des fonctions additives $u_1(A,b) + u_2(b,C)$. Pour achever la construction de $u(a,b,c)$ sur $A \times B \times C$, il est seulement nécessaire de « raccorder » les morceaux obtenus par ces collections de fonctions additives de manière à former une fonction d'utilité globale. Ainsi, en tenant compte de cette liaison entre utilités additives et utilités GAI-décomposables, nous rappelons dans les sections suivantes les techniques classiques d'élicitation des fonctions additives qui seront utilisées dans la suite dans le processus de construction d'utilités GAI-décomposables.

4.2 L'élicitation d'utilités additives

Dans le cas le plus décomposable, nous avons des utilités additives. Les procédures d'élicitation pour de telles utilités tirent parti des indépendances entre les attributs de manière à construire la fonction d'utilité multi-attribut en deux étapes bien distinctes : tout d'abord, chaque facteur d'utilité est construit indépendamment des autres facteurs ; ensuite, ces facteurs sont agrégés de manière à former une utilité globale. Procéder ainsi en deux temps nous permet d'avoir une méthode d'élicitation à la fois rapide et fiable. Elle est rapide car l'élicitation d'une utilité sur chaque X_i évite l'explosion combinatoire engendrée par le produit cartésien des X_i. Et elle est fiable car les questions posées au décideur sont cognitivement simples, dans la mesure où elles ne font intervenir que peu d'attributs ayant des valeurs différentes. Pour agir de cette manière, les procédures

d'élicitation d'utilités additives s'appuient principalement sur une propriété particulière de ces utilités, à savoir leur unicité à une transformation affine strictement positive près. Celle-ci dérive des ensembles d'axiomes garantissant l'existence même de ces utilités (Fishburn, 1970; Krantz *et al.*, 1971; Wakker, 1989; Nakamura, 2002).

Supposons qu'une relation de préférences $\succsim$ soit modélisable par une utilité additive $u(x_1, \ldots, x_n) = \sum_{i=1}^{n} u_i(x_i)$. La première étape de l'élicitation consiste en la construction, pour chaque attribut X_i et indépendamment des autres facteurs, d'un facteur d'utilité $v_i : X_i \to \mathbb{R}$. À la fin de cette étape, nous ne savons pas encore si les fonctions $u_i(\cdot)$ que nous recherchons sont égales aux $v_i(\cdot)$ élicitées. Nous savons seulement que, puisque les $u_i(\cdot)$ et les $v_i(\cdot)$ sont des utilités sur la projection de $\succsim$ sur X_i, il existe une fonction strictement croissante $\varphi_i : \mathbb{R} \to \mathbb{R}$ telle que $u_i(\cdot) = \varphi_i(v_i(\cdot))$. En général, l'élicitation de φ_i peut être aussi coûteuse que l'élicitation de v_i. Heureusement, sous l'hypothèse d'unicité des fonctions d'utilité à une transformation affine strictement positive près, nous avons que les fonctions $\varphi_i(\cdot)$ sont des transformations affines strictement positives, autrement dit, il existe des nombres réels $k_i \in \mathbb{R}_+^*$ et $r_i \in \mathbb{R}$, $i \in \{1, \ldots, n\}$, tels que $u_i(\cdot) = k_i v_i(\cdot) + r_i$. Par conséquent, pour achever la construction de la fonction d'utilité $u(\cdot)$ sur l'ensemble de $\mathcal{X}$, il ne reste plus qu'à déterminer les valeurs de k_i et de r_i. Une procédure générale d'élicitation peut donc être décrite comme dans l'algorithme 4.1.

```
1 pour chaque i dans {1, ..., n} faire
2     Construire un facteur d'utilité v_i(·) sur X_i
3     Déterminer k_i et r_i de telle sorte que v_i s'accorde avec les v_j(·), j < i
4 fin
```

Algorithme 4.1 : Procédure générale pour l'élicitation d'utilités additives

Puisque toute transformée affine strictement positive d'une utilité produit une autre utilité, le processus d'élicitation peut être simplifié en affectant la valeur 0 à tous les r_i. En effet, cela revient juste à rajouter une valeur constante à $u(\cdot)$. Pour la même raison, on peut affecter la valeur 1 à k_1. Voyons maintenant comment l'on peut déterminer les valeurs des k_i pour $i \geq 2$. Supposons que, pour des valeurs $a_i, b_i \in X_i$, et $a_j, b_j \in X_j$, $j < i$, le décideur soit capable d'émettre des jugements du type « je suis indifférent à (a_i, a_j) et (b_i, b_j), tous les autres attributs étant fixés à une certaine valeur ». Soit x_p la valeur des attributs X_p pour $p \neq i, j$. Si nous supposons une parfaite cohérence des réponses avec le modèle, on peut conclure que :

$$k_i v_i(a_i) + k_j v_j(a_j) + \sum_{p \neq i,j} k_p v_p(x_p) = k_i v_i(b_i) + k_j v_j(b_j) + \sum_{p \neq i,j} k_p v_p(x_p),$$

ou, en d'autres termes,

$$k_i = k_j \frac{v_j(b_j) - v_j(a_j)}{v_i(a_i) - v_i(b_i)} \quad \forall i \geq 2. \tag{4.1}$$

Dans l'algorithme 4.1, les valeurs des k_i sont déterminées après celles de $v_i(\cdot)$, $v_j(\cdot)$ et k_j. L'équation (4.1) permet donc d'identifier de manière unique les k_i. Par conséquent, l'algorithme 4.1 permet de construire itérativement une utilité additive représentant $\succsim$. Notons que, dans les jugements mentionnés ci-dessus, les x_p peuvent être choisis librement puisqu'ils n'apparaissent pas dans l'équation (4.1).

4.3 L'élicitation d'utilités additives dans le certain

Dans le cas le plus général, contrairement à la décision dans l'incertain ou dans le risque, les utilités additives dans le certain ne sont pas nécessairement uniques à des transformations affines strictement positives près (Adams, 1965; Gonzales, 2003). Par conséquent, la procédure précédemment décrite ne peut pas être directement appliquée. Fort heureusement, des ensembles d'axiomes peu contraignants dans la pratique, et qui garantissent l'unicité à des transformations affines strictement positives près, ont été développés. L'axiomatisation de Krantz *et al.* (1971) fournit des axiomes assez peu restrictifs (et donc favorables) à l'utilisation de l'hypothèse d'unicité à des transformations affines strictement positives près (voir par exemple Wakker (1988)).

Le premier des axiomes, que nous allons supposer être vrai dans la suite, est une hypothèse structurelle, la **solvabilité restreinte**. Pour certaines applications, cette hypothèse peut être trop contraignante (voir par exemple Gonzales et Jaffray (1998)), mais elle peut aussi être allégée (Gonzales, 2003).

Axiome 4.1 (solvabilité restreinte)**.** Soit $\mathcal{X} = X_1 \times \ldots \times X_n$. $\forall i \in \{1, \ldots, n\}$, si $(x_1^0, \ldots, x_{i-1}^0, x_i^0, x_{i+1}^0, \ldots, x_n^0) \precsim (x_1, \ldots, x_n) \precsim (x_1^0, \ldots, x_{i-1}^0, x_i^1, x_{i+1}^0, \ldots, x_n^0)$, alors $\exists x_i^2 \in X_i$ tel que $(x_1^0, \ldots, x_{i-1}^0, x_i^2, x_{i+1}^0, \ldots, x_n^0) \sim (x_1, \ldots, x_n)$.

En décision dans le certain, l'idée force pour éliciter les $v_i(\cdot)$ réside dans la construction de ce que l'on appelle des séquences standards : soit $\succsim$ une relation de préférence sur $X_1 \times X_2$ représentable par une utilité additive $k_1 v_1(\cdot) + k_2 v_2(\cdot)$. Soit $x_1^0 \in X_1$, $x_2^0, x_2^1 \in X_2$ des éléments quelconques tels que $(x_1^0, x_2^1) \succ (x_1^0, x_2^0)$. Puisque toute transformée affine strictement positive préserve la représentabilité des fonctions d'utilité, on peut supposer sans perte de généralité que $k_1 v_1(x_1^0) = k_2 v_2(x_2^0) = 0$ et $k_2 v_2(x_2^1) = 1$. Soit $x_1^1 \in X_1$ tel que $(x_1^1, x_2^0) \sim (x_1^0, x_2^1)$ (voir la figure 4.3 qui représente des courbes d'indifférence dans l'espace $X_1 \times X_2$), alors :

$$k_1 v_1(x_1^1) + k_2 v_2(x_2^0) = k_1 v_1(x_1^0) + k_2 v_2(x_2^1).$$

En d'autres termes, $k_1 v_1(x_1^1) = 1$. Plus généralement, pour tout $i \in \{1, \ldots, n\}$, soit x_1^i, x_1^{i+1} tels que $(x_1^{i+1}, x_2^0) \sim (x_1^i, x_2^1)$. Alors, si $\succsim$ est représentable par une utilité additive, $k_1 v_1(x_1^i) = i$. La suite (x_1^i) est appelée une séquence standard :

Définition 4.1 (séquence standard)**.** Soit N un ensemble d'entiers consécutifs. L'ensemble $\{x_1^k, k \in N\}$ est une séquence standard pour X_1 ssi $\exists x^0 = (x_2^0, \ldots, x_n^0)$ et $\exists x^1 = (x_2^1, \ldots, x_n^1)$ tels que $(x_1^0, x_{-1}^0) \not\sim (x_1^0, x_{-1}^1)$ et, $\forall k, k+1 \in N$, $(x_1^{k+1}, x_{-1}^0) \sim (x_1^k, x_{-1}^1)$. L'ensemble $\{x^0, x^1\}$ est appelé la **maille** de la séquence standard *. Des définitions analogues s'appliquent aux autres X_i, $i > 1$.

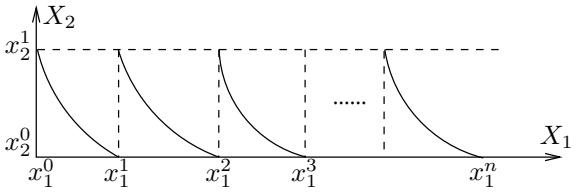


FIG. 4.3 – Une séquence standard pour X_1

Naturellement, pour pouvoir éliciter une fonction d'utilité en utilisant des séquences standards, il faut que tout l'espace puisse être parcouru par celles-ci, ou du moins que le maillage de ces séquences soit suffisamment fin pour pouvoir inférer les valeurs de la fonction d'utilité sur tout l'espace (par exemple par densité). Cela nécessite l'axiome suivant.

Axiome 4.2 (Axiome archimédien)**.** Soit $\{x_1^k, k \in N\}$ une séquence standard par rapport à X_1 de maille $\{x^0, x^1\}$. Si $\exists y, z \in \mathcal{X}$ tels que $y \succ (x_1^k, x_{-1}^0) \succ z \ \ \forall x_1^k \in X_1$, alors N est fini. Des définitions analogues s'appliquent aux autres X_i, $i > 1$.

Le dernier axiome nécessaire à l'existence d'utilités additives est une condition d'indépendance entre les attributs qui capture essentiellement le fait que l'utilité se décompose comme une somme de fonctions définies sur des espaces complètement distincts (les X_i).

Axiome 4.3 (Indépendance en coordonnées)**.** $\forall i \in \{1, \ldots, n\}$, $\forall z_i, t_i \in X_i$ et $\forall x_j, y_j \in X_j$, $j \neq i$, $(z_i, x_{-i}) \succsim (z_i, y_{-i}) \Leftrightarrow (t_i, x_{-i}) \succsim (t_i, y_{-i})$.

Cet axiome entraîne par récurrence l'existence d'une relation de préférence $\succsim_i$ sur chaque X_i définie par : $x_i \succsim_i y_i$ si et seulement s'il existe un $(n-1)$-uplet $(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n) \in \times_{j \neq i} X_j$ tel que $(x_i, x_{-i}) \succsim (y_i, x_{-i})$. Il a deux conséquences importantes en ce qui concerne la procédure d'élicitation :

*Rappelons que si I est un ensemble d'indices d'attributs, alors $z = (x_{-I}, y_I)$ représente le n-uplet $(z_1, \ldots, z_n)$ tel que $z_i = y_i$ pour $i \in I$ et $z_i = x_i$ pour $i \notin I$. Par abus de notation, (x_{-1}, y_1) représente le n-uplet $(y_1, x_2 \ldots, x_n)$.

1. le choix des mailles des séquences standards n'a aucune influence sur la valeur des $v_i(\cdot)$ construits ;
2. on n'a pas besoin d'éliciter la fonction d'utilité « globale » sur l'ensemble de $\mathcal{X}$, mais seulement sur des sous-espaces beaucoup plus petits (les X_i), et ces constructions « locales » s'étendent naturellement pour former une utilité sur l'ensemble de $\mathcal{X}$.

Bien entendu, seuls les attributs impliqués dans les prises de décision devraient être pris en compte lors de la phase d'élicitation et on devrait donc restreindre $\mathcal{X}$ au produit cartésien des seuls attributs non ainsi superflus. C'est ce qu'indique le dernier axiome utilisé dans l'axiomatique de (Krantz *et al.*, 1971) :

Axiome 4.4 (essentialité)**.** $\forall i \in \{1, \ldots, n\}$, $\exists\ x_i, y_i \in X_i$ et $\exists\ z_j \in X_j\ \forall j \neq i$, tels que $(z_{-i}, x_i) \succ (z_{-i}, y_i)$.

La combinaison des axiomes ci-dessus avec une condition dite de Thomsen (que nous ne détaillerons pas ici car nous ne l'utiliserons pas par la suite) garantit que $\succsim$ est représentable par une utilité additive et qu'en outre l'hypothèse d'unicité est vérifiée (Krantz *et al.*, 1971) :

Théorème 4.1 (Existence et unicité des utilités additives)**.** *Supposons que la solvabilité restreinte et l'essentialité soient vérifiées pour chaque attribut. Alors les deux affirmations suivantes sont équivalentes :*

1. *$\succsim$ est un préordre large total vérifiant, pour chaque attribut, l'indépendance en coordonnées, la condition de Thomsen (si $\mathcal{X}$ a seulement deux attributs) et l'axiome archimédien ;*
2. *$\succsim$ est représentable par une fonction d'utilité additive et, de plus, celle-ci est unique à une transformation affine strictement positive près.*

Maintenant que l'on a la garantie que l'hypothèse d'unicité est vérifiée, voyons comment nous pouvons construire chaque facteur d'utilité u_i indépendamment des autres facteurs. Choisissons arbitrairement deux valeurs x_n^0 et x_n^1, de l'attribut X_n, telles que $x_n^1 \succ_n x_n^0$, autrement dit, telles que $x_n^1 \succsim_n x_n^0$ et $x_n^1 \precsim_n x_n^0$. Choisissons de plus une valeur x_j^0 arbitraire pour les autres attributs. Si $\succsim$ est représentable par une utilité additive $\sum_{i=1}^n u_i(\cdot)$, alors il existe $\alpha > 0$ et $\beta_i \in \mathbb{R}$, tels que $\alpha u_1(x_1^0) + \beta_1 = \ldots = \alpha u_n(x_n^0) + \beta_n = 0$, et $\alpha u_n(x_n^1) + \beta_n = 1$. Les $v_i(\cdot) = \alpha u_i(\cdot) + \beta_i$ forment une utilité additive représentant $\succsim$ puisque $v(\cdot) = \sum_{i=1}^n v_i(\cdot)$ est une transformée affine de $u(\cdot)$. La construction d'une séquence standard $(x_1^k)_{k \in N}$ de maille $\{(x_n^1, x_{-\{1n\}}^0), (x_n^0, x_{-\{1n\}}^0)\}$ sur l'attribut X_1 a pour conséquence immédiate que $v_1(x_1^k) = k$ pour tout $k \in N$. Donc l'utilité $v_1(\cdot)$ peut être élicitée sur une grille $(x_1^k)_{k \in N}$ simplement en construisant une séquence standard. Cette fonction peut alors être étendue sur l'ensemble de l'espace X_1 en affinant itérativement

la grille par utilisation de mailles de plus en plus fines, ou bien encore en utilisant des fonctions comme des *splines* interpolant ou approximant $v_1(x_1^k)$.

4.3.1 Le cas discret

La propriété de solvabilité restreinte (Axiome 4.1) implique normalement que le produit cartésien $\mathcal{X}$ contienne de nombreux éléments, voire même un nombre infini d'éléments. Deux différentes approches sont habituellement utilisées pour étendre l'applicabilité de l'axiomatisation présentée au cas discret, en le transformant en continu (Barba-Romero et Pomerol, 1997, page 185) :

1. La première approche consiste à introduire des probabilités et à passer ainsi au cadre de l'incertitude. Dans ce cas, nous supposons implicitement que toutes les lotteries sur les alternatives sont aussi des élections possibles, ce qui nous place dans le cadre d'utilités linéaires continues du type von Neumann-Morgenstern (von Neumann et Morgenstern, 1944; Savage, 1972). Nous prenons le risque aussi que la fonction élicitée soit biaisée par une aversion pour le risque de la part du décideur (Friedman et Savage, 1948; Markowitz, 1952).
2. La deuxième approche consiste à étendre le domaine des attributs non-solvables de sorte que la solvabilité restreinte soit vérifiée. Par exemple, supposons que pour chaque attribut X_i, $x_i^\top$ et $x_i^\perp$ soient les alternatives qui fournissent la plus grande et la plus petite valeur d'utilité $u_i(x_i^\top)$ et $u_i(x_i^\perp)$, respectivement. Nous transposons alors le problème dans l'intervalle $[u_i(x_i^\perp), u_i(x_i^\top)]$. Ainsi, nous supposons que lorsque l'on prend les décisions, une fois qu'une valeur d'utilité ne correspond à aucune alternative, cette valeur à quand même du sens pour le décideur (par exemple, nous pouvons le projeter sur l'espace d'origine). Quant à l'applicabilité de l'élicitation, l'aspect infini de $\mathcal{X}$ n'est pas très problématique dans le sens où les courbes d'indifférence —les ensembles d'éléments appartenant à la même classe d'indifférence selon $\sim$ (la partie symétrique de $\succsim$) — forment des courbes en général très lisses et peuvent donc être approchées en connaissant seulement un nombre relativement limité de points (par exemple en utilisant des splines ou des fonctions affines par morceaux).

4.4 L'élicitation d'utilités GAI

Gonzales et Perny (2004) ont proposé une procédure d'élicitation dans l'incertain pour des modèles GAI fondée sur des questions standard de jeu (*standard gamble query* – SGQ), où on demande au décideur s'il préfère la configuration x à une loterie dans laquelle la meilleure configuration arrive avec probabilité l et la plus mauvaise configuration

arrive avec probabilité $1 - l$. Leur procédure exploite la décomposabilité du modèle GAI pour réduire le nombre de questions nécessaires pour l'élicitation complète de la fonction d'utilité. Les SGQ posées concernent tous les attributs, toutefois le nombre d'attributs avec des valeurs différentes pour les deux alternatives présentées à chaque question est petit, réduisant ainsi la complexité cognitive des questions.

Par ailleurs, Braziunas et Boutilier (2005) (voir aussi Braziunas et Boutilier (2006)) ont montré que le processus d'élicitation peut être conduit en posant des questions locales sur les attributs des facteurs d'utilité de la décomposition GAI et ses **ensembles conditionnants** (*conditioning set*, en anglais).

Définition 4.2. Soient $\mathcal{X} = X_1 \times X_2 \times \ldots \times X_n$; $C_1, \ldots, C_k$ des sous-ensembles de $N = \{1, \ldots n\}$ tels que $N = \bigcup_{i=1}^{k} C_i$, $X_{C_i} = \{X_j : j \in C_i\}$ qui définissent une décomposition GAI d'une fonction d'utilité u sur $\mathcal{X}$. L'ensemble conditionnant d'un facteur d'utilité X_{C_i} de u est défini comme :

$$EC_i = \{X_j : X_j \notin X_{C_i} \wedge \exists l \in \{1, \ldots, k\} \text{ tel que } j \in C_l \text{ et } C_l \cap C_i \neq \emptyset\}$$

Une fois que les attributs de EC_i sont affectés à un point de référence x^0, les valeurs des attributs qui n'appartiennent ni à X_{C_i}, ni à EC_i n'influencent pas les préférences sur X_{C_i}. Ainsi, nous pouvons omettre ces attributs dans les SGQ pour éliciter les utilités locales. Plus précisement, si $x_i^\top$ et $x_i^\perp$ sont les configurations qui fournissent respectivement la plus grande et la plus petite valeur d'utilité locale, avec $v_i(x_i^\top) = 1$ et $v_i(x_i^\perp) = 0$, alors $v_i(x_i) = p$ ssi :

$$(x_i, x^0_{EC_i}) \sim \langle p, (x_i^\top, x^0_{EC_i}); 1 - p, (x_i^\perp, x^0_{EC_i})\rangle$$

Par exemple, considérons la fonction GAI $u(a, b, c, d, e, f, g) = u_1(a, b) + u_2(c, e) + u_3(b, c, d) + u_4(b, d, f) + u_5(b, g)$ représentée par le GAI-Net de la figure 1.9 (page 40). Ainsi, $X_{C_1} = \{A, B\}$, $X_{C_2} = \{C, E\}$, $X_{C_3} = \{B, C, D\}$, $X_{C_4} = \{B, D, F\}$ et $X_{C_5} = \{C, E\}$. Les ensembles conditionnants et les attributs concernés dans les SGQ locales sont alors :

$$\begin{array}{lll}
EC_1 = \{C, D, F, G\} & \therefore & X_{C_1} \cup EC_1^0 = \{a, b, c^0, d^0, f^0, g^0\} \\
EC_2 = \{B, D\} & \therefore & X_{C_2} \cup EC_2^0 = \{b^0, c, d^0, e\} \\
EC_3 = \{A, E, F, G\} & \therefore & X_{C_3} \cup EC_3^0 = \{a^0, b, c, d, e^0, f^0, g^0\} \\
EC_4 = \{A, C, G\} & \therefore & X_{C_4} \cup EC_4^0 = \{a^0, b, c^0, d, f, g^0\} \\
EC_5 = \{A, C, D, F\} & \therefore & X_{C_5} \cup EC_5^0 = \{a^0, b, c^0, d^0, f^0, g\}
\end{array}$$

Notons que le nombre d'attributs concernés dans les SGQ locales dépend de la structure du GAI-Net. Par exemple, ci-dessus les questions pour éliciter l'utilité locale

du terme X_{C_3} concernent tous les attributs. Dans certains cas, les questions pour éliciter les termes locaux peuvent systématiquement concerner tous les attributs, même si la fonction d'utilité se décompose dans des termes d'utilité avec peu d'attributs chacun ; il suffit qu'un attribut soit présent dans tous les séparateurs du GAI-Net (comme dans l'arbre GAI de la figure 2.5, page 60). En effet, pour les arbres GAI, nous avons la propriété suivante :

Proposition 4.1. *Soit $\mathcal{G}$ un arbre GAI sur les variables $\mathcal{X} = \{X_1, \ldots, X_n\}$ avec un ensemble de cliques $\mathcal{C} = \{X_{C_1}, \ldots, X_{C_k}\}$. Si*

1. *$\exists j \in \{1, \ldots, n\}$ tel que $X_j \in S_k$ pour tout séparateur S_k de $\mathcal{G}$, alors*
2. *$\forall i \in \{1, \ldots, k\}, X_{C_i} \cup EC_i = \mathcal{X}$.*

Démonstration : Puisque toute variable $\in \mathcal{X} \backslash X_{C_i}$ appartient à une clique X_{C_l} tel que $C_l \cap C_i \neq \emptyset$, alors $EC_i = \mathcal{X} \backslash X_{C_i}$ et donc $X_{C_i} \cup EC_i = \mathcal{X}$. ■

Remarquons que la condition (1) ci-dessus est suffisante pour que la conclusion (2) soit vraie mais elle n'est même pas nécessaire. Nous pouvons aussi avoir des arbres GAI où (2) est vraie mais qui ne vérifient pas (1), comme exemplifie la figure 4.4.

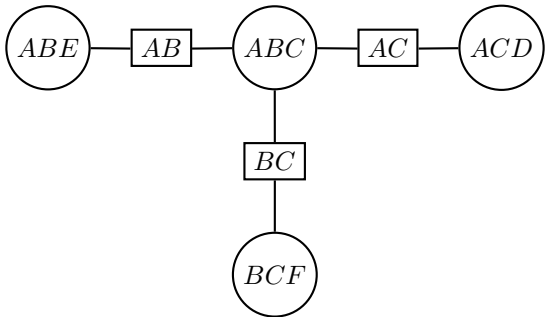


FIG. 4.4 – Example de arbre GAI où $\forall i, X_{C_i} \cup EC_i = \mathcal{X}$

Les utilités locales v_i élicitées par les SGQ locales ne sont pas les u_i de la fonction d'utilité finale. Elles doivent être calibrées à travers une une expression de Fishburn (1967) qui emploie les valeurs d'utilité, précédemment connues, de trois configurations complètes, nommées des « configurations-ancre » : la meilleure configuration $x^\top$, la configuration moins bonne $x^\perp$, e une configuration de référence x^0. Cette méthode est présentée dans Braziunas et Boutilier (2005).

Ces approches pour l'élicitation dans l'incertain peuvent être appliquées dans le certain si nous adoptons l'alternative 1 décrite dans la section 4.3.1 (page 109). Toutefois, comme nous l'avons souligné, cela présente des inconvénients. C'est pourquoi l'alternative 2, qui étend le domaine des attributs non-solvables et utilise des séquences standard sans introduire des probabilités mérite aussi d'être envisagée.

4.4.1 L'élicitation fondée sur un réseau GAI

Gonzales et Perny (2006) proposent une nouvelle voie, fondée sur les réseaux GAI. Comme nous l'avons mentionné dans la section 4.2, l'unicité à une transformation affine strictement positive près est une propriété fondamentale pour l'applicabilité des procédures d'élicitation des fonctions d'utilité additives. Cette propriété peut être utilisée dans le certain sous l'hypothèse structurelle de solvabilité restreinte, comme nous l'avons vu dans la section 4.3. Toutefois, lorsqu'il s'agit d'utilités GAI-décomposables, cette unicité ne peut pas être garantie, même sous l'hypothèse de solvabilité restreinte. En effet, une relation de préférence $\succsim$ sur $X_1 \times X_2 \times X_3$ modélisée par $u_1(x_1, x_2) + u_2(x_2, x_3)$ est tout aussi modélisable par $u'_1(x_1, x_2) + u'_2(x_2, x_3)$, avec $u'_1(x_1, x_2) = u_1(x_1, x_2) + g(x_2)$, $u'_2(x_2, x_3) = u_2(x_2, x_3) - g(x_2)$ et $g(\cdot)$ une fonction quelconque de X_2 dans $\mathbb{R}$. À moins que $g(\cdot)$ ne soit une constante, il y a peu de chances pour que la nouvelle fonction soit une transformée affine de u.

Le problème venant de la possibilité d'opérer des transferts d'utilité d'un facteur à l'autre, nous proposons (Gonzales *et al.*, 2007), à la suite de Gonzales et Perny (2006), une nouvelle condition structurelle qui restreint les fonctions GAI possibles à celles où ces transferts d'utilités sont interdits. Cette proposition n'est pas significativement restrictive dans le sens où, si une relation de préférence $\succsim$ est modélisable par une fonction d'utilité GAI, il existera une fonction GAI vérifiant les propriétés de la proposition et qui modélise elle aussi la relation $\succsim$:

Proposition 4.2. *Soit $\mathcal{X} = \times_i X_i$ et soit $u(\cdot)$ une utilité décomposable selon un arbre GAI $\mathcal{G} = (\mathcal{C}, \mathcal{E})$, où $\mathcal{C} = \{X_{C_1}, \ldots, X_{C_k}\}$ est l'ensemble des cliques de $\mathcal{G}$ ordonnées de l'extérieur vers l'intérieur (cf. la section 2.1 en page 49 pour la définition de la notation ici utilisée), autrement dit $u(x) = \sum_{i=1}^{|\mathcal{C}|} u_i(x_{C_i})$. Soit x^0 un élément quelconque de $\mathcal{X}$, alors :*

- *Propriété 1 : $\forall i \in \{1, \ldots, |\mathcal{C}| - 1\}$, la clique X_{C_i} a exactement une clique voisine dans $\mathcal{G}_{\mathcal{C}_i}$, que l'on notera $X_{C_{n(i)}}$ ($n(i)$ étant le numéro de la clique voisine).*
- *Propriété 2 : Soit $S_i = C_i \cap C_{n(i)}$ et $D_i = C_i \backslash S_i$. Il existe une utilité v GAI décomposable selon $\mathcal{G}$ telle que $\forall i < |\mathcal{C}|$ et $\forall x_{S_i}$, $v_i(x^0_{D_i}, x_{S_i}) = 0$, et telle que $v_{|\mathcal{C}|}(x^0_{C_{|\mathcal{C}|}}) = 0$. Une telle utilité est dite* non transférable *par rapport au couple $(x^0, \mathcal{C})$.*
- *Propriété 3 : Si u et v sont des utilités GAI décomposables vérifiant la propriété 2 ci-dessus, alors il n'existe aucune fonction de transfert non-nulle transformant u en v. Autrement dit, il n'existe pas un ensemble de fonctions $f_i : S_i \mapsto \mathbb{R}$, $i \in \{1, \ldots, k-1\}$, telles que :*
  - *$v_1(x_{C_1}) = u_1(x_{C_1}) - f_1(x_{S_1})$ et $v_k(x_{C_k}) = u_k(x_{C_k}) + \sum_{j \in n^{-1}(k)} f_j(x_{S_j})$,*

– $v_i(x_{C_i}) = u_i(x_{C_i}) - f_i(x_{S_i}) + \sum_{j \in n^{-1}(i)} f_j(x_{S_j})\ \forall i \geq 2$,
– $\exists i \in \{2, \ldots, k-1\}$ *et* $\exists x_{S_i} \in \times_{j \in C_i} X_j$ *tels que* $f_i(x_{S_i}) \neq 0$.

Exemple 4.1. (Gonzales *et al.*, 2007) Considérons le réseau GAI de la figure 1.9 (page 40) : Nous pouvons ordonner $\mathcal{C}$ de l'extérieur vers l'intérieur comme $\{AB, CE, BCD, BDF, BG\}$. En effet, puisque AB et CE sont des cliques externes, les supprimer conserve la connexité du graphe. De même, après avoir supprimé AB et CE, la clique BCD devient externe puisque la connexité du graphe induit n'est pas remise en cause par la suppression additionnelle de BCD, et ainsi de suite. La propriété 1 stipule simplement que la clique AB est connectée à seulement une clique adjacente, ici BCD ; après avoir supprimé AB et CE, BCD reste connectée à seulement une clique, en l'occurrence BDF, etc. La propriété 2 stipule que si $(a^0, b^0, c^0, d^0, e^0, f^0, g^0)$ est un élément quelconque de $\mathcal{X}$, alors il existe une utilité GAI décomposable

$$v(a, b, c, d, e, f, g) = v_1(a, b) + v_2(c, e) + v_3(b, c, d) + v_4(b, d, f) + v_5(b, g)$$

telle que

$$v_1(a^0, b) = v_2(c, e^0) = v_3(b, c^0, d) = v_4(b, d^0, f^0) = v_5(b^0, g^0) = 0.$$

L'idée sous-jacente à cette propriété est la suivante : puisque les X_{S_i} sont des attributs appartenant aux séparateurs, la propriété peut être établie en transférant par l'intermédiaire du séparateur X_{S_i} une certaine quantité qui dépend seulement du X_{S_i} d'une clique à sa clique voisine. Par exemple, supposons que $v_1(a^0, b) \neq 0$ pour un b donné, alors en remplaçant $v_1(a, b)$ par $v_1(a, b) - v_1(a^0, b)$ pour tout $(a, b) \in A \times B$ et $v_3(b, c, d)$ par $v_3(b, c, d) + v_1(a^0, b)$ pour tout $(b, c, d) \in B \times C \times D$, la propriété 2 se trouve trivialement vérifiée pour v_1 et la fonction GAI décomposable ainsi obtenue est encore une utilité représentant $\succsim$. Le même procédé peut être appliqué sur la deuxième clique de $\mathcal{C}$ et, par récurrence, à toutes les cliques de $\mathcal{C}$. Quant à la dernière, en soustrayant la constante $v_5(b^0, g^0)$ de $v_5(b, g)$ on obtient la propriété 2. Ce processus est illustré dans les figures 4.5 à 4.8. Ici nous partons des valeurs d'utilité de la figure 2.1 (page 50) et nous obtenons à la fin une utilité non-transférable par rapport à l'ensemble des cliques ordonnées de l'extérieur vers l'intérieur $\mathcal{C} = \{AB, CE, BCD, BDF, BG\}$ et de l'élément $(a^0, b^0, c^0, d^0, e^1, f^0, g^0)$.

Il s'avère que la non-transférabilité n'est pas suffisante pour assurer l'unicité à une transformation affine strictement positive près. Gonzales et Perny (2006) ont donc proposé une condition structurelle additionelle, nommée la **connexité des séparateurs**. Cette condition nous garantit que les séparateurs ne jouent pas un rôle de « dictateur » dans

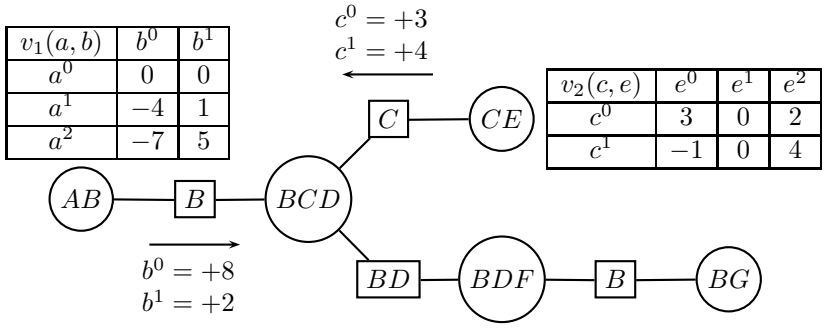


FIG. 4.5 – Transfert d'utilité par les séparateurs : AB et CE

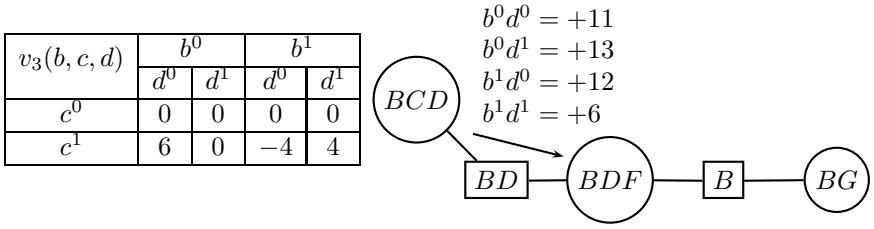


FIG. 4.6 – Transfert d'utilité par les séparateurs : BCD

les préférences du décideur, c'est-à-dire que leur poids n'est pas suffisamment grand pour que, dès qu'un élément $x \in \mathcal{X}$ a x_{S_i} pour valeur d'un séparateur X_{S_i} donné, x soit préféré à tout autre élément $y \in \mathcal{X}$ tel que $x_{S_i} \neq y_{S_i}$, et ce quelles que soient les valeurs des autres attributs. Formellement, l'axiome de connexité des séparateurs est défini comme suit :

Définition 4.3 (connexité des séparateurs)**.** Soit $\mathcal{X} = \times_{i=1}^{n} X_i$ et soit $\mathcal{G} = (\mathcal{C}, \mathcal{E})$ un arbre GAI, avec $\mathcal{C} = \{X_{C_1}, \ldots, X_{C_k}\}$. Soit x un élément quelconque de $\mathcal{X}$. Soit X_{C_i} et X_{C_j} deux cliques, pas obligatoirement voisines dans l'arbre GAI, et soit $X_{S_{ij}} = X_{C_i \cap C_j}$ leur intersection/séparateur non-vide. Notons $D_i = C_i \backslash S_{ij}$, $D_j = C_j \backslash S_{ij}$ et $E = \{1, \ldots, n\} \backslash (C_i \cup C_j)$.

Alors, $\forall x_{S_{ij}}, y_{S_{ij}} \in X_{S_{ij}}$, il existe une suite finie $(z^k_{S_{ij}})_{k=0}^{p}$ d'éléments de $X_{S_{ij}}$ telle que $z^0_{S_{ij}} = x_{S_{ij}}$, $z^p_{S_{ij}} = y_{S_{ij}}$, et $\forall k \in \{2, \ldots, p\}$ $\exists\, a^k, b^k, c^k, d^k \in \mathcal{X}$ tels que :

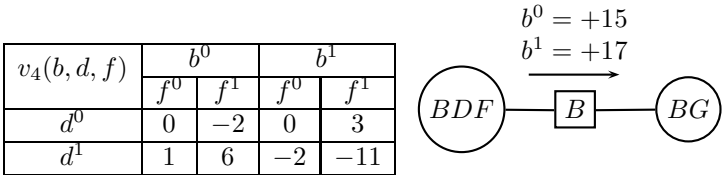
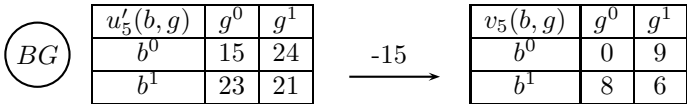


FIG. 4.7 – Transfer d'utilité par les séparateurs : BDF



$u'_5(b,g)$	g^0	g^1
b^0	15	24
b^1	23	21

$v_5(b,g)$	g^0	g^1
b^0	0	9
b^1	8	6

FIG. 4.8 – Obtention de $v_5(b,g)$

1. $(a^k_{D_i}, z^{k-1}_{S_{ij}}, a^k_{D_j}, x_E) \sim (b^k_{D_i}, z^k_{S_{ij}}, b^k_{D_j}, x_E)$;
2. $(c^k_{D_i}, z^{k-1}_{S_{ij}}, c^k_{D_j}, x_E) \sim (d^k_{D_i}, z^k_{S_{ij}}, d^k_{D_j}, x_E)$;
3. $(a^k_{D_i}, z^{k-1}_{S_{ij}}, a^k_{D_j}, x_E) \not\sim (c^k_{D_i}, z^{k-1}_{S_{ij}}, c^k_{D_j}, x_E)$.

Munis de la connexité des séparateurs, Gonzales et Perny (2006) montrent que la propriété d'unicité à une transformation affine strictement positive près peut être garantie :

Proposition 4.3. *Soit $\succsim$ une relation de préférence sur $\mathcal{X} = \times_{i=1}^n X_i$ représentable par une fonction d'utilité GAI décomposable selon un arbre GAI $\mathcal{G}$. Soit $\mathcal{C} = \{X_{C_1}, \ldots, X_{C_n}\}$ l'ensemble des cliques de $\mathcal{G}$ ordonné des cliques extérieures de $\mathcal{G}$ vers les cliques intérieures. Soit x^0 un élément quelconque de $\mathcal{X}$. Supposons que la solvabilité restreinte soit vérifiée pour tout attribut et que l'axiome de connexité des séparateurs le soit aussi pour tout couple de cliques X_{C_i}, X_{C_j} d'intersection non vide. Alors les fonctions d'utilité GAI décomposables selon $\mathcal{G}$ et non transférables selon $(x^0, \mathcal{C})$ sont uniques à une transformation linéaire strictement positive près.*

Ainsi, avec des fonctions d'utilité GAI uniques à une transformation affine strictement positive près, il est possible de procéder de façon similaire au cas des utilités additives, en élicitant les facteurs d'utilité des cliques de l'extérieur vers l'intérieur du réseau GAI (voir Gonzales *et al.* (2007)). La procédure d'élicitation se déroule en **trois étapes** :

1. L'élicitation des utilités dans les cliques en fixant les valeurs du séparateur ;
2. Dans chaque clique, le raccordement des valeurs élicités pour chacune des valeurs du séparateur (que nous nommerons **agrégation intra-clique**) ;
3. La mise en échelle des cliques (que nous nommerons **agrégation inter-cliques**).

Dans la suite, nous illustrons la procédure d'élicitation ici présentée d'abord sur un exemple simple*, associé au réseau GAI montré dans la figure 4.9, avant de passer au cas général.

FIG. 4.9 – Arbre GAI pour $u(a,b,c) = u_1(a,b) + u_2(b,c) + u_3(c,d)$.

*Tiré de Gonzales *et al.* (2007).

Étape 1 : l'élicitation des utilités dans les cliques

Pour l'étape 1, nous pouvons utiliser les techniques adaptées aux utilités additives, comme la construction de séquences standard (section 4.3). En effet, comme nous l'avons mentionné dans la section 4.2, nous obtenons des utilités additives une fois la valeur du séparateur fixée. Regardons, par exemple, l'arbre GAI de la figure 4.9. Si l'on fixe la valeur du séparateur B à une valeur b' quelconque, on obtient une utilité additive $u_1(A, b') + [u_2(b', C) + u_3(C, D)]$. Ainsi, nous pouvons éliciter les valeurs d'utilité pour $u_1(A, b')$ tel que pour une utilité additive, c'est-à-dire que nous pouvons appliquer, dans le cas général, l'algorithme 4.1.

Étape 2 : l'agrégation intra-clique

Une fois que l'on a élicité, dans l'exemple, les valeurs $u_1(A, b')$ et $u_1(A, b'')$, il faut les raccorder pour obtenir une utilité sur $\{b', b''\}$. Grâce à la non-transférabilité, $u_1(A, b')$ et $u_1(A, b'')$ sont uniques à une transformation linéaire près. Par conséquent, nous pouvons les agréger en multipliant $u_1(A, b'')$ par un nombre $k \in \mathbb{R}_+^*$ de telle sorte que $u_1(A, b')$ et $ku_1(A, b'')$ forment une utilité sur $A \times \{b', b''\}$. On peut trouver un tel k si nous sommes capables de trouver des éléments a^i, c^i, d^i tels que :

$$(a^1, b, c^1, d^1) \sim (a^2, b', c^2, d^2) \text{ et } (a^3, b, c^1, d^1) \sim (a^4, b', c^2, d^2), \tag{4.2}$$

ainsi on a :

$$\begin{aligned} u_1(a^1, b) + u_2(b, c^1) + u_3(c^1, d^1) &= ku_1(a^2, b') + u_2(b', c^2) + u_3(c^2, d^2), \\ u_1(a^3, b) + u_2(b, c^1) + u_3(c^1, d^1) &= ku_1(a^4, b') + u_2(b', c^2) + u_3(c^2, d^2), \end{aligned}$$

en soustrayant ces deux équations, on obtient :

$$k = \frac{u_1(a^1, b) - u_1(a^3, b)}{u_1(a^2, b') - u_1(a^4, b')}.$$

Le cas général : Les indifférences de l'équation (4.2) correspondent dans le cas général à celles de l'axiome de connexité des séparateurs. Si l'on suppose que l'on a élicité $u_i(x_{D_i}, z_{S_i}^{k-1}) : X_{D_i} \mapsto \mathbb{R}$ et $u_i(x_{D_i}, z_{S_i}^{k}) : X_{D_i} \mapsto \mathbb{R}$ pour deux valeurs $z_{S_i}^{k-1}$ et $z_{S_i}^{k}$ du séparateur X_{S_i}, si l'on fixe les valeurs des attributs sur D_j et E dans l'axiome de connexité des séparateurs à x^0 (l'élément de référence de la non transférabilité), les trois équations de l'axiome de connexité des séparateurs deviennent :

1. $(a_{D_i}^k, z_{S_i}^{k-1}, x_{-C_i}^0) \sim (b_{D_i}^k, z_{S_i}^k, x_{-C_i}^0)$;
2. $(c_{D_i}^k, z_{S_i}^{k-1}, x_{-C_i}^0) \sim (d_{D_i}^k, z_{S_i}^k, x_{-C_i}^0)$;

3. $(a^k_{D_i}, z^{k-1}_{S_i}, x^0_{-C_i}) \not\sim (c^k_{D_i}, z^{k-1}_{S_i}, x^0_{-C_i})$.

Lorsque l'on traduit ces équations sous forme d'utilités et que l'on soustrait les deux premières équations, l'on peut constater que tous les termes $u_p(\cdot)$, pour $p < i$, sont égaux à 0 (par non transférabilité) et que les termes $u_p(\cdot)$, pour $p > i$, s'annulent tous. On obtient donc que k, le nombre réel par lequel on doit multiplier la fonction $u_i(x_{D_i}, z^k_{S_i})$ est égal à :

$$k = \frac{u_i(a^k_{D_i}, z^{k-1}_{S_i}) - u_i(c^k_{D_i}, z^{k-1}_{S_i})}{u_i(b^k_{D_i}, z^k_{S_i}) - u_i(d^k_{D_i}, z^k_{S_i})}, \tag{4.3}$$

et l'on peut ainsi construire une utilité sur le terme X_{C_i}.

Étape 3 : l'agrégation inter-cliques

Pour l'agrégation inter-cliques, un procédé similaire peut être apliqué. Grâce à la non-transférabilité, agréger les différents facteurs u_i revient à les multiplier par des nombres réels k_i strictement positifs. Dans l'exemple de la figure 4.9, si l'on peut trouver des éléments a^i, b^i, c, c', d, d' tels que :

$$(a^1, b^1, c, d) \sim (a^2, b^2, c', d') \text{ et } (a^3, b^3, c, d) \sim (a^4, b^4, c', d'), \tag{4.4}$$

alors, en termes d'utilité, on obtient :

$$k_2 = \frac{u_1(a^2, b^2) + u_1(a^3, b^3) - u_1(a^1, b^1) - u_1(a^4, b^4)}{u_2(b^1, c) - u_2(b^2, c') + u_2(b^3, c) - u_2(b^4, c')},$$

et ainsi l'on peut agréger u_2 et u_1. Nous procédons de façon analogue pour u_3.

Le cas général : Pour généraliser les indifférences de l'équation (4.4), nous procédons de façon similaire à la généralisation de l'agrégation intra-clique. Supposons que tous les facteurs $u_p(\cdot)$, $p < i$ aient déjà été agrégés et supposons que l'on ait élicité un nouveau facteur $u_i(\cdot)$. Soit $S_1 = C_1 \cap C_{n(1)}$, autrement dit X_{S_1} est le séparateur entre X_{C_1} et son voisin dans l'arbre GAI. Soit $D_1 = C_1 \backslash S_1$, $E_1 = C_1 \backslash C_i$, $F_1 = C_1 \backslash E_1$ et $E = \{1, \ldots, n\} \backslash (D_1 \cup D_i)$. Autrement dit, X_{E_1} sont les attributs appartenant à X_{C_1} mais pas à X_{C_i}, X_{F_1} sont les attributs communs à X_{C_1} et X_{C_i}, et X_{D_1} sont les attributs n'appartenant qu'à la clique X_{C_1}. Alors, comme dans le paragraphe précédent, les trois préférences suivantes :

$$\begin{aligned}
&1) \quad (a_{D_1}, a_{D_i}, x^0_E) \sim (b_{D_1}, b_{D_i}, x^0_E),\\
&2) \quad (c_{D_1}, c_{D_i}, x^0_E) \sim (d_{D_1}, d_{D_i}, x^0_E),\\
&3) \quad (a_{D_1}, a_{D_i}, x^0_E) \not\sim (c_{D_1}, c_{D_i}, x^0_E),
\end{aligned}$$

impliquent par soustraction des utilités relatives aux deux premières préférences que :

$$k_i = \frac{u_1(b_{F_1}, x^0_{E_1}) - u_1(a_{F_1}, x^0_{E_1}) + u_1(c_{F_1}, x^0_{E_1}) - u_1(d_{F_1}, x^0_{E_1})}{u_i(a_{D_i}, x^0_{S_i}) - u_i(b_{D_i}, x^0_{S_i}) - u_i(c_{D_i}, x^0_{S_i}) + u_i(d_{D_i}, x^0_{S_i})}. \tag{4.5}$$

Notons que tant l'agrégation intra-clique que l'agrégation inter-cliques supposent l'existence d'un quadruplet de points qui vérifient l'axiome de connexité des séparateurs. Cela ne doit pas s'avérer être un problème si nous avons beaucoup de valeurs dans chaque clique et chaque séparateur. Toutefois, si chaque clique ou séparateur ne possède que peu de valeurs, nous pouvons être incapable de trouver un quadruplet de points qui vérifie cet axiome. Dans le chapitre suivant, nous modifierons la méthode d'élicitation de Gonzales et Perny (2006) pour la rendre applicable dans de tels cas.

4.5 L'élicitation partielle de fonctions d'utilité

Comme nous pouvons le remarquer, l'élicitation de modèles d'utilité peut nécessiter de poser un grand nombre de questions, aboutissant à un processus élaboré et fastidieux. C'est pourquoi, dans les dernières années, une attention grandissante a été consacrée au développement de méthodes d'élicitation rapide approchée, qui permettent à la fois l'**élicitation partielle** des modèles d'utilité, et le raisonnement avec de tels modèles. Normalement, ces méthodes recourent à des heuristiques pour arriver rapidement à des modèles approchés. Malgré l'approximation, ces modèles peuvent en pratique disposer d'une qualité suffisante pour le problème traité, et ainsi, dans certaines situations, être préférés aux méthodes exactes grâce à la difficulté cognitive allégée pour la construction des modèles approchés.

Dans cette direction, Chajewska *et al.* (1998) ont examiné le cas où l'on ne peut supposer aucune indépendance entre les attributs de la fonction d'utilité à éliciter (c'est-à-dire qu'elle est considérée comme un seul terme d'utilité qui englobe tous les attributs) et on dispose d'une base de données de telles fonctions d'utilité (précédemment élicitées), correspondant aux préférences d'autres décideurs. Dans ce cas, des questions portant sur des alternatives complètes (c'est-à-dire des configurations impliquant tous les attributs) sont posées au décideur afin d'identifier un groupe de décideurs potentiels (*cluster*) avec lequel le décideur peut être associé. Les questions à poser sont déterminées par un arbre de décision qui répartit les décideurs potentiels dans des différents groupes. Naturellement, puisque des fonctions d'utilité non-décomposables sont ici utilisées, l'applicabilité de cette approche est restreinte aux cas où le nombre de variables est réduit, en raison de la nature combinatoire du problème.

Chajewska *et al.* (2000) présentent une méthode pour éliciter des fonctions GAI de

façon incrémentale en utilisant des distributions a priori de fonctions d'utilité couvrant les différentes structures de préférences que l'on peut rencontrer. On cherche alors à situer le décideur auquel nous avons à faire par une séquence de questions. Choisir la question à poser à chaque interaction avec le décideur peut être vu comme un problème de décision séquentielle, où l'on cherche l'arbitrage optimal entre le coût cognitif des questions et l'amélioration du modèle grâce aux nouvelles informations obtenues. Toutefois, la stratégie optimale est très difficile à calculer ; ainsi, une stratégie « gloutonne » a été adoptée, fondée sur un critère « myope » de **valeur espérée d'information** (EVOI, de l'anglais *Expected Value Of Information*).

L'EVOI myope : Supposons que nous avons un ensemble fini de questions $Q = \{q^1, \ldots, q^n\}$, et que chaque question q^i possède un ensemble fini de réponses possibles $R_i = \{r_i^1, \ldots, r_i^m\}$. Les réponses aux questions dépendent de la fonction d'utilité u du décideur. Toutefois, nous ne connaissons pas cette dernière. Un modèle probabiliste adapté à plusieurs situations pratiques suppose que nous disposons d'une distribution de probabilité a priori $\pi(u)$ sur des fonctions d'utilité possibles $u \in U$. Ainsi, la probabilité d'obtenir une réponse r_i^j une fois posée la question q^i est :

$$P(r_i^j|q_i, \pi) = \int_{u \in U} P(r_i^j|q^i, u)\pi(u)du.$$

Soit $EU(x, \pi)$ l'utilité espérée de l'alternative x quand π est la distribution sur des fonctions d'utilité. Nous définissons $MEU(\pi)$ comme l'utilité espérée maximale de π (de l'anglais, *Maximum Expected Utility*) :

$$MEU(\pi) = \max_{x \in \mathcal{X}} EU(x, \pi). \tag{4.6}$$

Une réponse r à une question q nous fournit de nouvelles informations sur la fonction d'utilité u du décideur. Ainsi, π sera mise à jour selon la règle de Bayes :

$$\forall u \in U, \pi^r(u) = \pi(u|r) = \frac{P(r|u)\pi(u)}{P(r|\pi)}.$$

Après la réponse r, l'utilité espérée maximale est mise à jour selon l'équation (4.6), en calculant $MEU(\pi^r)$. Pour calculer la valeur d'une question q^i, nous calculons l'utilité a posteriori espérée pour q^i (EPU, de l'anglais, *Expected Posterior Utility*), c'est-à-dire :

$$EPU(q^i, \pi) = \sum_{r \in R_i} P(r|q^i, \pi)MEU(\pi^r).$$

La valeur espérée d'information (EVOI) pour une question q^i est son utilité a posteriori espérée diminuée de l'utilité espérée maximale pour le π actuel :

$$EVOI(q^i, \pi) = EPU(q^i, \pi) - MEU(\pi). \tag{4.7}$$

Une stratégie « gloutonne » fondée sur l'EVOI est « myope », car à chaque étape elle choisi la question avec la plus grande EVOI sans considérer que cela peut s'avérer être une mauvaise direction à l'avenir. Pour aller plus loin, Boutilier (2002) a proposé une formulation du problème d'élicitation comme un processus de décision markovien partiellement observable (POMDP) qui prend en compte la valeur des questions futures dans la décision du choix de la question à l'étape actuelle. Toutefois, le coût de calcul est très élevé, et la méthode n'est pour l'instant applicable qu'à de très petits problèmes.

Boutilier *et al.* (2005) supposent qu'au lieu d'avoir des distributions a priori des fonctions d'utilité GAI, nous avons des limites pour leurs paramètres. Formellement, soit $u(x_1, \ldots, x_n) = \sum_{i=1}^{k} u_i(x_{C_i})$ une fonction d'utilité GAI (cf. définition 1.14 en page 22). On suppose ici que chaque $u_i(x_{C_i})$ ne correspond pas à un nombre réel mais à un intervale réel $[u_{i\downarrow}(x_{C_i}), u_{i\uparrow}(x_{C_i})]$. Ils présentent une procédure d'élicitation fondée sur un critère de minimax regret. On essaie de choisir l'alternative x qui dispose du max-regret minimum, où max-regret est la plus grande quantité que nous pouvons regretter, en choisissant x (si nous permettons à la fonction d'utilité de varier dans les limites de ses paramètres). Le type de questions qu'ils utilisent sont des questions de limite (*bound queries*, en anglais) consistant à demander au décideur si l'un de ses paramètres d'utilité $u_i(x_{C_i})$ se situe au-dessus d'une certaine valeur q, $u_{i\downarrow}(x_{C_i}) < q < u_{i\uparrow}(x_{C_i})$. Une réponse positive conduit à augmenter la borne inférieure de ce paramètre (c'est-à-dire que $u_{i\downarrow}(x_{C_i}) = q$), tandis qu'une réponse négative abaisse sa borne supérieure (c'est-à-dire que $u_{i\uparrow}(x_{C_i}) = q$) : dans les deux cas, l'incertitude par rapport à la fonction d'utilité du décideur est réduite. Ce type de question est de son côté équivalent à une SGQ. Toutefois, comme nous l'avons mentionné dans la section 4.3.1, l'introduction de probabilités dans le certain n'est pas anodine, elle peut biaiser la fonction obtenue.

Le principe de base des méthodes d'élicitation partielle est la supposition que certaines régions de la fonction d'utilité sont plus importantes que d'autres pour la résolution du problème considéré. Ainsi, nous devons accorder la priorité à ces régions lors de l'élicitation. Or, si nous ne connaissons pas encore la fonction d'utilité, nous ne pouvons pas savoir a priori quelles sont ces régions. Les méthodes d'élicitation partielle dépendent alors d'heuristiques pour déterminer les régions à éliciter.

L'heuristique à utiliser dépend de la problématique en question. Nous devons prendre en considération la nature de la solution souhaitée (recherchons-nous la meilleure solution,

une liste des meilleures solutions ?). Souvent, nous nous attendons à ce qu'un système d'aide à la décision ne soit pas seulement capable de trouver la meilleure solution pour un décideur, mais qu'il puisse aussi lister les k-meilleures alternatives pour celui-ci. Par exemple, pour un système de recommandation, Herlocker (2000) identifie deux caractéristiques essentielles qu'un tel système devrait comprendre : (1) le système doit être en mesure de lister les k-meilleures recommandations, (2) le système doit être capable de donner une valeur absolue d'utilité pour toute alternative arbitrairement choisie.

Contrairement aux travaux précédents sur l'élicitation de modèles GAI, qui supposent que nous cherchions la meilleure alternative, nous supposons maintenant que l'objectif du système est de découvrir les k-meilleures solutions pour le décideur. Nous proposons donc, dans la section suivante, une méthode d'élicitation partielle qui se focalise sur la détermination des k-meilleures alternatives.

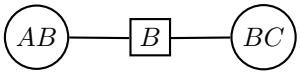
4.6 Une méthode d'élicitation partielle focalisée sur les k-meilleures alternatives

Dans cette section nous proposons une nouvelle méthode d'élicitation partielle de fonctions GAI qui se différencie des autres précédemment utilisées dans la littérature pour se focaliser sur l'obtention d'une bonne liste de k-meilleures solutions. Le scénario considéré est un processus interactif où le système questionne le décideur à propos de sa fonction d'utilité, raffinant ainsi la connaissance du système au sujet des préférences du décideur. À chaque étape, le système est capable de présenter au décideur les k-meilleures recommandations pour le modèle courant de préférences.

On suppose que le système a une base de données de préférences d'individus. Ces préférences sont GAI-décomposables, et les individus peuvent être placés dans différents *clusters* (classes d'individus). Nous devons pouvoir utiliser cette information pour focaliser le processus d'élicitation d'un nouveau décideur sur des questions qui nous permettront d'identifier rapidement ses préférences. Notez que les *clusters* peuvent différer en structure et en valeurs, comme souligné dans Chajewska et Koller (2000). Cependant, ici nous supposons que tous les *clusters* partagent la même structure de décomposition, différant seulement en valeurs. Si tel n'était pas le cas, on pourrait d'abord poser des questions au décideur en ayant pour objectif de déterminer quelle structure parmi les candidates était la plus appropriée pour lui, dans un processus d'élicitation en deux phases : *choix de la structure*, suivie d'*élicitation des utilités*.

Chaque *cluster* de décideurs est représenté par un modèle $\mathcal{M}_j$ composé par une collection de distributions normales multidimensionnelles indépendantes (matrice de variance-covariance diagonale). Nous avons utilisé une distribution normale multidimen-

sionnelle pour chaque terme d'utilité. La figure 4.10 exemplifie un modèle pour une décomposition GAI $u(a, b, c) = u_1(a, b) + u_2(b, c)$ sur des variables binaires A, B, C.



AB	μ	σ	BC	μ	σ
ab	0,62	0,11	bc	0,20	0,16
$a\bar{b}$	0,25	0,05	$b\bar{c}$	0,25	0,20
$\bar{a}b$	0,37	0,09	$\bar{b}c$	0,03	0,16
$\bar{a}\bar{b}$	0,05	0,11	$\bar{b}\bar{c}$	0,07	0,13

FIG. 4.10 – Modèle $\mathcal{M}_j$ pour un cluster de décideurs

L'objectif de la stratégie d'élicitation est d'atteindre les k-meilleures configurations pour le décideur (classées par ordre décroissant d'utilité) en employant le moins de questions possible. À chaque interaction avec le décideur, nous lui posons une SGQ, comme dans Boutilier *et al.* (2005). Dans le cadre des systèmes de recommandation, il est habituel de demander à l'utilisateur d'un tel système d'explicitement noter des articles, on peut alors considérer dans ce cadre qu'une SGQ demande si le décideur note une certaine configuration avec au moins l. Une fois qu'une réponse a été obtenue, nous restreignons les limites dans lesquelles la valeur d'utilité pour le point élicité peut varier (Elicited_bounds). Ensuite, nous mettons à jour le modèle utilisé, pour prendre en compte ses nouvelles limites. L'algorithme 4.2 résume le processus d'élicitation. Nous le détaillons dans la suite.

1 Au départ, le *modèle GAI courant* est affecté à l'utilité moyenne des modèles $\mathcal{M}$
2 **répéter**
3 calculer les k meilleures solutions selon le *modèle GAI courant*
4 choisir une SGQ à travers l'heuristique courante et la poser au décideur
5 mettre à jour Elicited_bounds selon la réponse du décideur
6 appeler `Update_GAI` pour mettre à jour le *modèle GAI courant*
7 **jusqu'à** *la variance maximale du modèle soit* $< \epsilon$

Algorithme 4.2 : Élicitation partielle adaptative de fonctions GAI

Au début de l'interaction avec le décideur, nous utilisons les valeurs moyennes des modèles $\mathcal{M}$ comme valeurs d'utilité pour le modèle GAI courant du décideur (ligne 1). Nous calculons ensuite, en employant le modèle GAI courant, les k-meilleures solutions (ligne 3). Pour cela, nous utilisons l'algorithme de rangement présenté dans le chapitre 2. Pour choisir la SGQ à poser au décideur (ligne 4), nous avons testé deux heuristiques différentes :

- **L'heuristique de variance maximale :** la première heuristique considérée vise à réduire la variance globale du modèle. Ceci suggère une stratégie simple où à chaque interaction avec le décideur il est questionné sur l'utilité du facteur ayant

Entrées : Un arbre GAI $\mathcal{G}$; $\mathcal{M}$, un ensemble de modèles normaux ;
Elicited_bounds, les limites élicitées
Sorties : Le $\mathcal{G}$ mis à jour

1 **pour chaque** *modèle-candidat* $\mathcal{M}_j \in \mathcal{M}$ **faire**
2 calculer la probabilité d'observer Elicited_bounds
3 **fin**
4 soit $\mathcal{M}_*$ le modèle avec la plus grande probabilité
5 **pour chaque** *terme d'utilité* $u_i(x_{Z_i})$ *dans* $\mathcal{G}$ **faire**
6 **si** $u_i(x_{Z_i})$ *possède Elicited_bounds* **alors**
7 fixer $u_i(x_{Z_i})$ à la valeur espérée de $\mathcal{M}_*$ conditionnée à Elicited_bounds
8 **sinon**
9 fixer $u_i(x_{Z_i})$ à la valeur espérée de $\mathcal{M}_*$
10 **fin**
11 **fin**

Fonction `Update_GAI`

la plus grande variance. On utilise alors une SGQ pour démander au décideur si la valeur d'utilité du facteur ayant la plus grande variance est plus grande que la valeur espérée (c'est-à-dire la moyenne).

- **L'heuristique focalisée sur les meilleures solutions :** l'heuristique de variance maximale essaie de réduire rapidement la variance totale du modèle sans tenir compte du fait que les questions les plus importantes à poser sont celles sur les configurations ayant une chance plus grande d'être bien classées par le décideur. Par conséquent, une approche alternative est de se concentrer sur le raffinement de la connaissance au sujet des meilleures solutions. La *stratégie de variance locale maximale (VLM)* questionne sur l'utilité du facteur des *k-meilleures solutions courantes* qui possède la plus grande variance. Les k-meilleures solutions courantes sont celles trouvéés k-optimales selon le modèle GAI courant (en ligne 3).

Ensuite, nous mettons à jour le le modèle GAI courant. Pour cela, la fonction `Update_GAI` calcule d'abord la probabilité d'observer les limites élicitées pour chacun des modèles candidats (ligne 2) et met ensuite à jour le modèle GAI courant. Puisque nous avons utilisé des modèles normaux multidimensionnels indépendants, le calcul de la probabilité d'observer les limites élicitées pour un modèle $\mathcal{M}_j$ peut être fait efficacement. Soit une configuration avec une limite supérieure l_{up} et une limite inférieure l_{down} (élicitées par des SGQ), la probabilité d'observer ces valeurs pour chaque modèle $\mathcal{M}_j$ est $F_j(l_{up}) - F_j(l_{down})$ où $F(\cdot)_j$ est la fonction de répartition de M_j. Ainsi, nous pouvons calculer la probabilité d'observer les limites élicitées pour chaque un des modèles considérés (ligne 2).

Une fois que, dans `Update_GAI`, nous avons trouvé le modèle le plus probable $\mathcal{M}_*$,

nous mettons à jour le modèle GAI courant pour le décideur, en utilisant $\mathcal{M}_*$ et les limites élicitées. Notons que, après l'élicitation des limites pour une configuration, nous n'avons plus une loi normale pour cette configuration mais une loi normale tronquée par les limites l_{up} et l_{down}. Pour estimer l'espérance μ de cette nouvelle loi tronquée, nous utilisons la moyenne d'échantillons tirés selon cette loi (en utilisant la méthode de la transformée inverse, cf. l'appendice A-1 en page 179). Afin de limiter le nombre d'échantillons nécessaire pour obtenir une estimation fiable de μ, nous tirons des échantillons jusqu'à un intervalle de confiance pour μ soit atteint. Classiquement, dans le domaine de la statistique, pour établir un intervalle de confiance à $100(1-\alpha)\%$ pour μ après n échantillons, nous utilisons la loi t (aussi connue comme Loi de Student). Dans ce cas, nous utilisons l'intervalle de confiance donné par $\hat{\mu}(n) \pm t_{n-1,1-\alpha/2}\sqrt{\frac{\hat{\sigma}^2(n)}{n}}$, où $t_{n-1,1-\alpha/2}$ est le point critique de la loi t avec $n-1$ degrés de liberté, $\hat{\mu}(n)$ est la moyenne de l'échantillon et $\hat{\sigma}^2(n)$ est sa variance. Dans les expérimentations qui suivent, pour estimer μ nous tirons des échantillons jusqu'à que la longueur de l'intervalle de confiance pour μ soit plus petite que $0,01$ avec 95% de confiance.

4.6.1 Expérimentations

Nous avons testé les stratégies d'élicitation sur des jeux de données générés synthétiquement avec différents jeux de variables et différentes structures GAI. Comme les résultats ont été similaires, nous les affichons ici pour l'une des configurations d'essai. Pour cette configuration, nous avons 30 variables, chacune avec 5 valeurs possibles (c'est-à-dire 5^{30} alternatives possibles) et 23 facteurs d'utilité, dont 1 avec 5 variables, 2 avec 4 variables, 14 avec 3 variables et 6 avec 1 variable. Chaque facteur d'utilité a ses valeurs dans l'intervalle $[0,1]$.

Critère d'évaluation : Pour évaluer la qualité de la liste ordonnée générée dans les expérimentations, nous employons une approche inspirée par celle que Breese *et al.* (1998) ont utilisé pour évaluer des algorithmes de filtrage collaboratif. L'utilité d'une liste ordonnée pour un individu a est definie comme la probabilité que l'individu a verra les articles affichés sur cette liste multipliée par les utilités des articles. Afin d'estimer combien il est probable que l'individu voit un article sur la liste, on suppose que chaque article successif a moins de chance d'être vu par l'individu avec une décroissance exponentielle. Ainsi, l'utilité prévue d'une liste ordonnée de n articles pour l'individu a (avec l'index $j \in \{1,...,n\}$ et triée dans l'ordre décroissant de u^a) est donnée par :

$$R^a = \sum_j \frac{u^{a,j}}{2^{(j-1)/(\alpha-1)}} \tag{4.8}$$

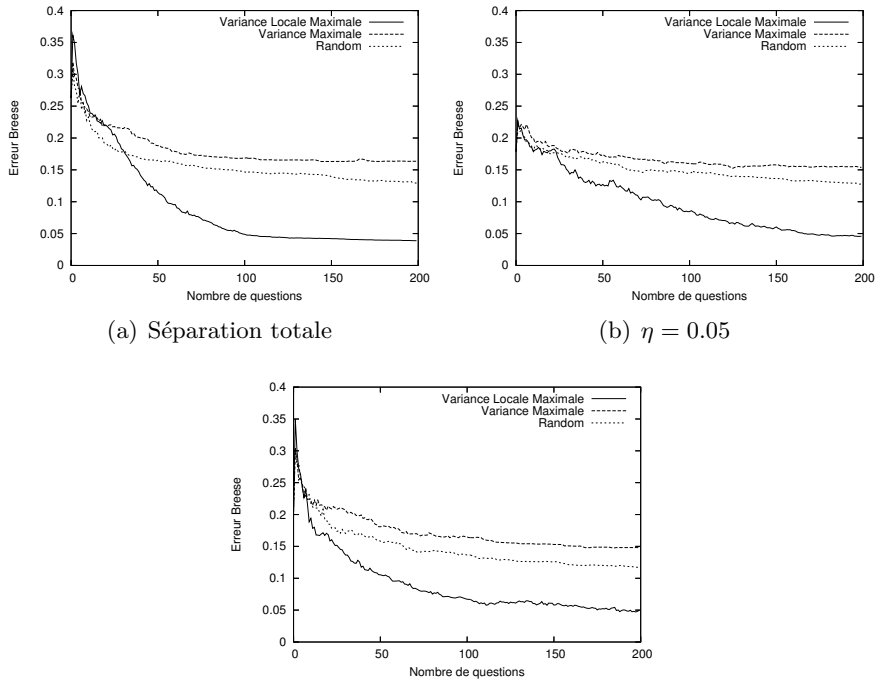
où α est la demi-vie, c'est-à-dire l'indice de l'élément sur la liste tel que l'individu a 50% de chance de voir cet élément. Le score final d'une liste est : $S^a = R^a/R^a_{max}$, où R^a est le score pour la liste et R^a_{max} est le score maximum atteignable, c'est-à-dire, le score pour la liste des véritables n meilleurs éléments pour a dans le bon ordre. On appelle $(1 - S^a)$ l'erreur Breese. Remarquez que l'erreur Breese est toujours dans l'intervalle $[0, 1]$. Dans nos expérimentations, on considère l'erreur Breese moyenne sur l'ensemble des individus. On a adopté une demi-vie de 5 et mesuré l'erreur Breese pour les 50 meilleurs élements.

Séparation des classes d'individus : Quand les classes d'individus sont bien séparées il est facile d'identifier à quel modèle l'individu appartient avec la plus grande probabilité. Dans ces cas, après quelques questions le modèle choisi (ligne 4 de `Update_GAI`) ne change plus. Pour évaluer la performance de l'élicitation avec des données plus difficiles, nous devons tester des cas où les classes d'individus s'intersectent. Afin de simuler cela, nous avons introduit des dépendances entre les modèles. Au lieu de choisir aléatoirement la valeur pour la moyenne de chaque modèle normal multidimensionnel, pour les modèles $\mathcal{M}_j$, $j \neq 0$ nous fixons $\mu_{\mathcal{M}_j} = \mu_{\mathcal{M}_1} + \text{Normal}(0, \eta)$, où $\text{Normal}(0, \eta)$ est un échantillon provenant d'une loi normale multidimensionnel avec moyenne 0 et matrice de variance-covariance ηI. À la mesure que η s'approche de 0, la séparation entre les classes diminue.

Nous avons créé 3 classes différentes, composées chacune de 30 individus. Afin de créer une classe d'individus, pour chaque terme d'utilité nous produisons d'abord un vecteur de valeurs uniformément distribuées dans l'intervalle $[0, 1]$. Ce vecteur sera la moyenne de la gaussienne multidimensionnelle pour le modèle. Ensuite nous générons une matrice diagonale de variance-covariance avec les valeurs uniformément distribuées dans un intervalle (nous avons employé l'intervalle $[0.05, 0.2]$). Les sous-fonctions d'utilité pour les utilisateurs dans une classe seront des échantillons provenant de cette gaussienne multidimensionnelle (tronquée dans l'intervalle $[0, 1]$).

Résultats expérimentaux : Nous avons évalué l'efficacité des stratégies d'élicitation en mesurant l'Erreur Breese moyenne pour les 90 individus. La figure 4.11 montre l'Erreur Breese moyenne observée sur 200 questions en utilisant l'algorithme `Elicit_Preferences`. Dans la figure 4.11(a) les classes sont totalement séparées, c'est-à-dire que les moyennes pour les modèles ont été indépendamment générées. Les figures 4.11(b) et 4.11(c) montrent les résultats pour les modèles corrélés. La figure 4.11(b) dépeint les modèles fortement corrélés, avec $\eta = 0.05$, tandis que la figure 4.11(c) montre les résultats observés pour les modèles légèrement corrélés, avec $\eta = 1.25$.

Nous voyons que la stratégie qui se concentre sur les k-meilleures solutions a une performance bien supérieure à celle des stratégies non dirigées. Dans tous les cas, la



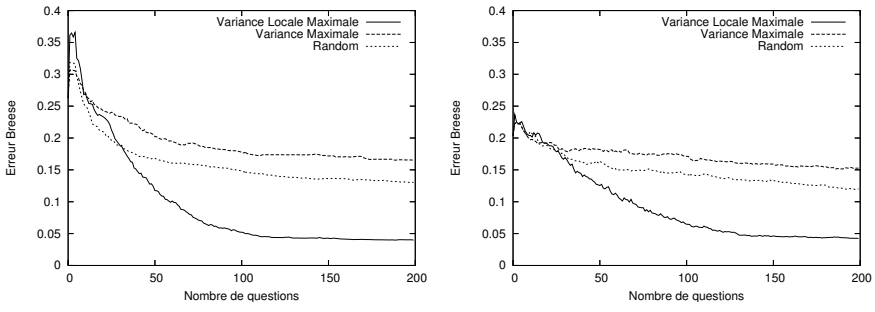
(a) Séparation totale (b) $\eta = 0.05$

(c) $\eta = 1.25$

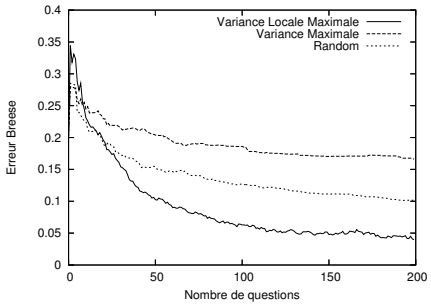
FIG. 4.11 – Erreur Breese moyenne sur 90 individus en 3 classes (30 individus par classe)

stratégie VLM a obtenu la réduction la plus rapide pour l'Erreur Breese moyenne. La stratégie de variance maximale a été particulièrement mauvaise, étant même inférieure à une stratégie aléatoire (où la prochaine question est aléatoirement choisie). Ceci montre l'importance de se focaliser sur les parties de l'espace de l'utilité qui contribueront le plus probablement aux k-meilleures solutions au lieu d'essayer d'améliorer le modèle dans l'ensemble.

La figure 4.11(b) montre que pour les modèles fortement corrélés, l'Erreur Breese a diminué beaucoup plus lentement que pour les modèles plus séparés. Ceci s'est produit parce que dans ces cas, l'algorithme en `Update_GAI` (ligne 4) commute fréquemment entre les différents modèles candidats. Par conséquent, beaucoup de questions ont été posées pour des points sous-optimaux dans l'espace d'utilité. Pour traiter ce problème, nous avons introduit un terme d'*inertie* en modifiant la ligne 4 d'`Update_GAI` pour utiliser un nouveau modèle $\mathcal{M}_*$ uniquement s'il est le meilleur depuis i interactions. La figure 4.12 montre les résultats pour le même ensemble de données en utilisant $i = 3$. Nous pouvons remarquer que l'Erreur Breese a diminué plus rapidement qu'avant pour les modèles fortement corrélés en utilisant les stratégies focalisées sur les meilleures solutions, tandis que le comportement pour les modèles faiblement corrélés n'a pas changé significativement.



(a) Séparation totale (b) $\eta = 0.05$



(c) $\eta = 1.25$

FIG. 4.12 – Erreur Breese moyenne sur 90 individus en 3 classes (30 individus par classe), inertie 3

Une critique suscitée par Doyle (2004) qui s'applique à tous les méthodes d'élicitation partielles ici mentionées, est que ces approches peuvent prétendre représenter plus d'information sur le décideur que le système n'a effectivement en raison du manque de distinction claire entre ce que le système connaît et ce que le système suppose.

Une autre préoccupation concerne la sémantique des questions. Comme nous l'avons mentionné dans la section 1.2.4, les valeurs d'utilité des termes d'une fonction GAI ne correspondent pas à des préférences *ceteris paribus* pour les attributs de ses domaines. Ainsi, pour pouvoir formuler des questions locales avec une sémantique claire, il est nécessaire d'introduire des conditions qui garantissent que les préférences élicitées sur des sous-ensembles d'attributs puissent être étendues à des préférences globales. Ceci est le sujet de la section suivante.

4.7 Discussion

La stratégie d'élicitation à adopter est largement dépendante du problème traité, du modèle utilisé et des données disponibles et aussi des attentes du constructeur du système d'aide à la décision en question et de ses utilisateurs. Ces facteurs détermineront si une élicitation partielle ou complète est plus appropriée et les types de question qui

seront posées. Le type de question à poser à l'utilisateur afin de disposer de l'information nécessaire pour le processus d'élicitation et en même temps n'être pas trop demandant est un problème clé à considérer dans les applications des méthodes décisionnelles.

Dans le cadre des modèles GAI, pour la méthode d'élicitation dans le certain fondée sur les GAI-Nets de Gonzales et Perny (2006), la construction de séquences standard se traduise naturellement dans des questions de comparaison entre alternatives, mais leur nombre peut être prohibitif. Par ailleurs, il peut être moins évident de répondre aux SGQ. Braziunas et Boutilier (2007) ont considéré l'utilisation d'autres types de questions à poser à la place ou en plus des SGQ. Dans ce cadre, seule l'utilisation de questions de comparaison s'est montrée inefficace, mais de bons résultats ont été obtenus en les combinant avec d'autres types de questions. Leurs résultats ont été obtenus empiriquement, en utilisant des données artificiellement générées.

En effet, toutes les approches d'élicitation considérées ici, que ce soit celle que nous avons présentée dans la section 4.6, ou les méthodes de Chajewska *et al.* (2000), Braziunas et Boutilier (2005), Boutilier *et al.* (2005), Braziunas et Boutilier (2007), ont été testées seulement en utilisant des données artificielles, générées pour les expérimentations. Pour évaluer les méthodes de façon plus réaliste, il est nécessaire de développer des applications grandeur nature, où ces méthodes seront aussi mises à l'épreuve de la problématique de HCI*. Toutefois, le développement d'un système de recommandation pour des produits multi-attribut où les décideurs doivent investir du temps pour exprimer ses préférences est difficilement réalisable hors du contexte d'un besoin réel de la part du décideur. Nous n'avons, pour le moment, pas eu l'opportunité de travailler dans un tel contexte.

En outre, l'application dans un cadre réel pourrait aussi nous permettre de spécialiser le processus d'élicitation, en le rendant plus simple. Dans le chapitre 5, nous introduisons *GVisite?*, une application Web dévéloppée pour expérimenter les méthodes présentées dans cette thèse dans un cadre plus concret, avec des données réeles. Nous utilisons une méthode d'élicitation fondée sur les GAI-Nets dérivée de celle presentée dans la section 4.4.1 et abordons le problème traité tel qu'une instance du NGKP (présenté dans la section 2.5, page 62).

*De l'anglais *Human-computer interaction*. Selon Hewett *et al.* (1992), « *Human-computer interaction is a discipline concerned with the design, evaluation and implementation of interactive computing systems for human use and with the study of major phenomena surrounding them.* »

Chapitre 5

GVisite? Le choix de ce qu'il faut visiter dans une ville avec l'aide de GAI-Nets

« J'ai toujours cru que le beau n'était que le bon mis en action, que l'un tenait intimement à l'autre, et qu'ils avaient tous deux une source commune dans la nature bien ordonnée. »

Jean-Jacques Rousseau, citoyen de Genève
(1712-1778)

Résumé

Dans ce chapitre, nous nous servons des GAI-Nets pour créer une application Web réelle d'aide à la décision. Nous considérons le problème où un voyageur doit choisir ce qu'il ira visiter dans une ville lors d'un court séjour.

GVisite?, l'application développée, se sert des données obtenues en temps réel dans le processus de conseil. Les informations présentées à chaque instant à l'utilisateur sont choisies de manière totalement automatique et adaptative, permettant ainsi l'utilisation à toutes destinations pour lesquelles des données peuvent être obtenues.

Le problème est ensuite construit comme une instance du problème de sac-à-dos non-linéaire GAI-décomposable (le NGKP, traité dans la section 2.5, à partir de la page 62). Nous terminons le chapitre en soulignant les possibilités des GAI-Nets dans de tels systèmes, ainsi que les carences qui ont pu être constatées à partir de la construction d'un système réel. Ces carences ouvrent des directions futures de recherche.

5.1 Le scénario

John Doe, chercheur américain, va à Paris pour présenter un article lors d'une conférence. C'est la première fois qu'il vient dans la capitale française. Il arrive un lundi et, étant un chercheur très impliqué, il compte passer toute la semaine à la conférence. Cependant, celle-ci se termine le vendredi, et Monsieur Doe ne repart que le dimanche en fin de soirée pour avoir l'opportunité de visiter la ville pendant le week-end.

Monsieur Doe sait qu'il faut absolument qu'il aille voir la Tour Eiffel, mais il n'a pas beaucoup d'idées par rapport à ce qu'il pourrait encore visiter lors de son week-end à Paris. Il pointe son navigateur sur un site de voyages, où les utilisateurs postent leurs plans de voyage*. En regardant quelques plans de voyage pour Paris, Monsieur Doe s'aperçoit qu'il y a vraiment trop de choses à faire à Paris ! Il a l'impression d'avoir remarqué au moins une centaine de choses différentes dans ces plans†. Monsieur Doe aimerait bien avoir un système qui puisse l'aider à choisir ce qu'il visitera lors de son passage à Paris.

5.2 Le problème

Nous pouvons modéliser le problème de choix des items à visiter dans une ville étant donné un temps limité comme un sac-à-dos 0-1, où un objet choisi correspond à un lieu à visiter (ou une activité à faire). Le poids de chaque objet correspond au temps que l'on investit dans sa visite, tandis que la fonction d'utilité à maximiser dépend des préférences du voyageur. Cependant il peut y avoir des interactions entre les items à visiter car ces choix ne sont pas indépendants. Par exemple, pour deux items qui intéressent le voyageur et qui se trouvent être géographiquement proches dans la ville, nous pouvons supposer que l'utilité de visiter les deux items est plus grande que la somme des utilités de visiter un item et pas l'autre. Par ailleurs, nous pouvons aussi avoir des interactions négatives, par exemple, il peut être redondant de visiter deux endroits qui permettent d'avoir une vue aérienne de la ville. De ce fait une fonction GAI peut être plus apropriée pour modéliser les préférences du décideur

Une fois que les préférences du voyageur seront modélisées par une fonction GAI et nous connaissions les poids des items, nous arrivons à une instance du NGKP, abordé dans la section 2.5 (page 62). Ainsi nous pourrons ensuite apliquer la méthode dévéloppée pour le NGKP pour résoudre ce problème.

Toutefois, lorsqu'il s'agit d'une application grandeur nature, un certain nombre de problèmes doit être abordé avant d'avoir une fonction d'utilité qui modélise les préférences

*Tel que *Yahoo ! Travel*, `http://travel.yahoo.com`

†Nous avons comptabilisé 200 choses différentes parmi 50 plans de voyage pour Paris chez Yahoo ! Travel

du voyageur :

1. **La construction du modèle d'utilité :** nous envisageons que le système soit capable de construire un modèle d'utilité après un petit nombre d'interactions avec l'utilisateur. Ainsi, pour notre problème, il est déjà inacceptable que l'utilisateur soit contraint à s'exprimer sur l'utilité de tous les items touristiques disponibles pour sa ville de destination. Comme nous avons pu le constater, il peut y en avoir des centaines. Ce problème se généralise à toute application où le nombre de variables est trop élevé. Dans de tels cas, le système doit être capable de filtrer les variables de sorte à proposer à l'utilisateur uniquement celles qui sont susceptibles de l'intéresser. Idéalement ce filtrage doit être fait de manière adaptative, en tenant compte des préférences déjà explicitées par l'utilisateur.
2. **Les données du système :** pour être capable d'aider l'utilisateur dans un problème réel, il est évidemment nécessaire que le système possède des données réelles sur ce problème. Comme nous l'avons souligné à la fin du chapitre 4 pour le cas des méthodes d'élicitation, la majorité des applications des modèles GAI se limitent jusqu'à maintenant à utiliser des données générées artificiellement (ou des jeux de données très petits). L'obtention de données réelles et leur mise à jour continuelle pour qu'un système ne devienne pas très vite obsolète constitue déjà un défi. En outre, il est fort probable que la masse de données nécessaire pour un problème réel soit beaucoup plus importante que celle utilisée dans les essais avec des données artificielles. Par exemple, si nous imaginons un système pour aider à choisir les choses à voir dans une ville, nous avons besoin d'un catalogue de villes touristiques et leurs items d'intérêt touristique. Ainsi, de nouveaux problèmes liés à l'augmentation de la masse de données traitée peuvent se poser.

Pour mettre à l'épreuve le concept d'un système capable d'aider un voyageur à sélectionner ce qu'il doit visiter lors de son passage dans une ville choisie, nous avons développé l'application Web *GVisite?*.

5.3 Le filtrage adaptatif des items

Le grand nombre d'items disponible suscite le besoin d'avoir une méthode pour filtrer les items de sorte à exposer l'utilisateur, à chaque instant, seulement à ceux qui semblent être les plus intéressants pour lui. Dans notre cadre, ce besoin est dû au grand nombre d'items touristiques existants pour une ville donnée. Par exemple, nous avons pu comptabiliser 200 items différents pour la ville de Paris. Il n'est donc pas désirable que l'utilisateur soit contraint à analyser la possibilité de visiter chacun de ces items (même en considérant ces questions comme mutuellement indépendantes).

Nous avons étudié l'utilisation de **règles d'association** afin de guider le processus de filtrage. Les règles d'association sont largement utilisées pour découvrir des relations de co-occurrence entre les activités réalisées par des individus. Un exemple intuitif et bien connu est l'analyse du comportement des consommateurs afin de découvrir des ventes croisées, du type « les consommateurs qui ont acheté le livre A ont également acheté le livre B ». Ainsi, dans le domaine du commerce de détail, des règles d'association peuvent aider à guider l'organisation des items d'un supermarché dans le but de mettre en exergue les items qui normalement apparaissent ensemble dans un panier d'achat typique.

Nous pouvons penser que notre voyageur, au fur et à mesure qu'il sélectionne les items touristiques qui lui semblent intéressants, est en train de composer son « panier de visite ». Ainsi, le problème de filtrer les items à suggérer à l'utilisateur peut être considéré de manière analogue à celui de l'organisation des items d'un supermarché dans le domaine du commerce de détail. Cela souligne l'intérêt des règles d'association pour notre problème.

5.3.1 Les règles d'association

Le cadre des règles d'association a été introduit dans la communauté de *data mining* par Agrawal *et al.* (1993).* Ils le définissent formellement comme suit :

Définition 5.1. Soit $\mathcal{I} = \{i_1, i_2, \ldots, i_n\}$ un ensemble de littéraux, nommés *items*. Soit $\mathcal{D}$ un ensemble de transactions, où chaque *transaction* T est un ensemble d'items tel que $T \subseteq \mathcal{I}$. Nous disons qu'une transaction T contient X (où X est un ensemble d'items de $\mathcal{I}$) quand $X \subseteq T$. Une *règle d'association* est une implication du type $X \Rightarrow Y$, où $X \subset \mathcal{I}$, $Y \subset \mathcal{I}$, et $X \cap Y = \emptyset$. Une règle $X \Rightarrow Y$ est vraie dans l'ensemble de transactions $\mathcal{D}$ avec une *confiance* c si $c\%$ des transactions dans $\mathcal{D}$ qui contiennent X contiennent aussi Y. Une règle $X \Rightarrow Y$ a un *support* s dans l'ensemble de transactions $\mathcal{D}$ si $s\%$ des transactions dans $\mathcal{D}$ contiennent $X \cup Y$.

Nous nommerons X et Y, respectivement, la **prémisse** et la **conclusion** de la règle d'association $X \Rightarrow Y$. Remarquons que la confiance mesure le degré de correlation entre la prémisse et la conclusion, tandis que le support mesure la signifiance de cette correlation dans l'ensemble des transactions.

Le problème classique de découverte de règles d'association se pose de la manière suivante : *étant donnés un ensemble de transactions, un niveau minimum de support et un niveau minimum de confiance, trouver toutes les règles d'association qui ont des*

*Des concepts et aspects représentationnels ont été abordés bien avant (Hájek *et al.*, 1966; Hájek et Havránek, 1977), mais sans développer des solutions algorithmiques.

niveaux de support et de confiance au-dessus des niveaux donnés. Voyons maintenant comme nous pouvons implanter le filtrage d'items avec des règles d'association.

5.3.2 Filtrage d'items à l'aide de règles d'association

Supposons que nous disposons des données suivantes :

- Un ensemble d'items touristiques d'une destination $\mathcal{I} = \{i_1, i_2, \ldots, i_n\}$;
- Un ensemble de voyages $\mathcal{D}$ pour cette destination, où chaque élément de $\mathcal{D}$ est un ensemble d'items $T \subseteq \mathcal{I}$.

Il est aisé de voir que nous pouvons modéliser un problème de découverte de règles d'association à partir de ces données, une fois que :

- L'ensemble de littéraux $\mathcal{I}$ est l'ensemble d'items touristiques de la destination ;
- L'ensemble de transactions $\mathcal{D}$ est l'ensemble de voyages pour cette destination ;
- Chaque transaction T est l'ensemble d'items d'un voyage.

Ainsi, une règle $X \Rightarrow Y$ (où $X \subset \mathcal{I}$, $Y \subset \mathcal{I}$ et $X \cap Y = \emptyset$) nous dira qu'une fois que l'on visite tous les items dans X, on visite aussi les items dans Y, avec une confiance c et un support s. Nous pouvons alors utiliser l'ensemble $\mathcal{R}$ de règles obtenues pour filtrer de manière adaptative les items visualisés par l'utilisateur à chaque instant.

Nous commençons par nous focaliser sur les règles d'association obtenues qui possèdent une prémisse vide. Cela signifie que l'ensemble d'items est présent dans plus de $c\%$ des voyages (notons que si la prémisse est vide, $s = c$). Les conclusions de ces règles correspondent donc à des items très populaires, ce que nous appelons « Les Incontournables ». C'est cet ensemble d'items que nous présentons à l'utilisateur au début de son interactions avec le système.

Supposons que l'ensemble initial d'items présentés à l'utilisateur est formé par les items $\{i_1, i_2, i_3, i_4\}$ et que l'utilisateur démontre son intérêt en visiter i_2. Nous ajoutons alors i_2 à son panier de visite $\mathcal{P} = \{i_2\}$. Des règles d'association du type $i_2 \Rightarrow Y$ suggèrent qu'il est probable que l'utilisateur s'intéresse aussi aux items dans Y une fois qu'il est intéressé en i_2. Ainsi, nous allons maintenant afficher aussi les items dans Y. Le processus de sélection de nouveaux items à afficher peut se passer alors de la manière suivante :

- chaque fois que l'utilisateur démontre son intérêt par un item i, nous l'ajoutons à son panier de visite ($\mathcal{P} \leftarrow \mathcal{P} \cup \{i\}$) ;
- pour chaque règle d'association $X \Rightarrow Y$ telle que $X \subseteq \mathcal{P}$ nous affichons les items de Y qui ne sont pas déjà affichés.

De cette manière, nous limitons le nombre d'items affichés, tout en essayant d'avoir disponibles à chaque instant ceux qui ont une probabilité plus élévée d'intéresser l'utilisateur, étant donné les items pour lesquels il a déjà montré de l'intérêt.

5.3.3 La prise en compte de dépendances

Comme nous l'avons mentionné au début du chapitre, il nous semble envisageable que le système soit capable de traiter des dépendances entre les items. Dans *GVisite?*, nous avons implémenté un mécanisme heuristique simple pour identifier automatiquement des dépendances.

Supposons à nouveau que l'ensemble initial d'items présentés à l'utilisateur est formé par les items $\{i_1, i_2, i_3, i_4\}$ et que l'utilisateur exprime de l'intérêt pour visiter i_2. Le panier de visite de l'utilisateur sera alors $\mathcal{P} = \{i_2\}$. Si nous avons les règles $i_2 \Rightarrow i_5$ et $i_2 \Rightarrow i_6$, nous passerons à afficher les items $\{i_1, i_2, i_3, i_4, i_5, i_6\}$. Maintenant, admettons que l'item i_6 soit ajouté au panier de visite de l'utilisateur. Remarquons que cet item a été sélectionné pour être affiché à l'utilisateur grâce à la présence de l'item i_2 dans son panier de visite et de l'existence de la règle d'association $i_2 \Rightarrow i_6$. Nous pouvons penser qu'il est probable qu'il existe des dépendances entre i_2 et i_6. Par exemple, comme nous l'avons mentionné dans la section 5.2, il se peut que ces deux items soient géographiquement proches dans la ville, et l'utilité de visiter tous les deux items soit plus grande que la somme des utilités de visiter un item et pas l'autre. Ainsi, chaque fois qu'un item i est ajouté au panier de visite $\mathcal{P}$ de l'utilisateur, et cet item a été précédemment sélectionné pour affichage grâce à une règle d'association $X \Rightarrow Y$ avec $X \subseteq \mathcal{P}$ et $i \in Y$, nous proposons à l'utilisateur de définir des dépendances entre des items dans X et i. Nous illustrerons ce mécanisme avec un exemple d'interaction avec *GVisite?* dans la section 5.5.4.

À la fin de cette étape, nous avons des sous-ensembles de variables qui intéressent l'utilisateur. Nous pouvons alors les transformer en un ensemble d'arbres GAI, en passant par la triangulation du réseau de Markov induit par les sous-ensembles de variables (cf. la section 1.3.5, à partir de la page 38). Nous pouvons ensuite éliciter les préférences de l'utilisateur à travers une méthode fondée sur un réseau GAI, semblable à celle de Gonzales et Perny (2006) (décrite dans la section 4.4.1), comme nous allons voir dans la prochaine section.

5.4 L'expression et la calibration de préférences

Une fois que nous avons la structure du modèle de préférences, l'utilisateur peut exprimer l'intensité de ses préférences. Nous commençons par les termes indépendants, que nous supposons être directement commensurables. Dans *GVisite?*, l'utilisateur exprime ses préférences sur chacun de ces termes dans une échelle qui varie de 0 à 100.

Il nous reste ensuite une collection d'arbres GAI à éliciter. Comme nous l'avons vu dans la section 4.4.1 (page 112), nous pouvons procéder à l'élicitation d'un arbre GAI en

trois étapes, que nous avons nomées : l'élicitation des utilités dans les cliques ; l'agrégation intra-clique ; et l'agrégation inter-cliques. Ici, comme nous avons une collection d'arbres GAI, une quatrième étape est nécessaire, l'agrégation inter-arbres. Nous verrons chacune de ses étapes dans les sections qui suivent.

5.4.1 Élicitation et agrégation intra-clique

Considérons l'exemple de la figure 5.1. Dans *GVisite?*, le domaine des variables X_i est binaire, et nous utilisons le codage $X_i = 1$ quand l'item representé par X_i sera visité et $X_i = 0$ quand il ne sera pas visité. Nous avons aussi, par définition, $u_i(x^0_{C_i}) = 0$, où x^0 est la configuration où $X_j = 0$, pour toute variable X_j. Comme nous l'avons vu dans le chapitre 4, si l'on fixe la valeur du séparateur B à une valeur arbitraire b, on obtient une utilité additive $u_1(A, b) + u_2(b, C)$. Pour la construction d'une utilité u non transférable (cf. la section 4.4.1, page 112) par rapport à $(x^0, \{AB, BC\})$ nous demandons d'abord des intensités de préférences, dans une échelle de 0 à 100, pour a^1b^0, a^1b^1, b^0c^1, b^1c^0 et b^1c^1 (notons que $u_1(a^0b^0) = u_1(a^0b^1) = u_2(b^0c^0) = 0$, par définition). À ce point, nous arrivons à l'obstacle suivant : pour le problème traité dans *GVisite?*, comme les variables sont binaires et il est fréquent que les cliques possèdent peu de valeurs, souvent nous n'avons pas suffisament de valeurs pour avoir un quadruplet de points qui vérifient l'axiome de connexité des séparateurs. Ainsi, pour « raccorder les morceaux » d'utilités obtenus pour b^0 et b^1 pour u_1, sans avoir besoin de trouver un tel quadruplet de points, au lieu d'utiliser les indifférences correspondantes à l'axiome de connexité des séparateurs, nous supposons qu'il est possible d'obtenir deux nombres réels q et r, tels que :

$$\begin{aligned} ku_1(a^1, b^1) + u_2(b^1, c^0) &= qu(a^1, b^0, c^0) \qquad \text{et} \\ ku_1(a^0, b^1) + u_2(b^1, c^0) &= ru(a^1, b^0, c^0), \end{aligned}$$

on obtient, alors :

$$\begin{aligned} ku_1(a^1, b^1) + ku_2(b^1, c^0) &= qu_1(a^1, b^0) + qu_2(b^0, c^0), \\ ku_1(a^0, b^1) + ku_2(b^1, c^0) &= ru_1(a^1, b^0) + ru_2(b^0, c^0), \end{aligned}$$

d'où, en soustrayant ces deux équations (et puisque $u_1(a^0, b^1) = u_2(b^0, c^0) = 0$) :

$$k = (q - r)\frac{u_1(a^1, b^0)}{u_1(a^1, b^1)}.$$

On peut ainsi multiplier les « morceaux d'utilité » obtenus pour les différentes valeurs de B afin d'obtenir une utilité non transférable u_1 sur $A \times B$.

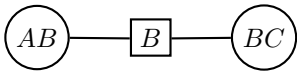


FIG. 5.1 – Élicitation : termes interdépendants

D'une manière générale, si l'on note $D_i = C_i \backslash S_i$, le k à multiplier les termes $u_i(x_{D_i}, z^k_{S_i})$ élicités pour une valeur $z^k_{S_i}$ du séparateur X_{S_i} afin de les mettre dans la même échelle des valeurs élicitées pour la valeur $z^0_{S_i}$ du séparateur correspond à :

$$k = (q - r)\frac{u_i(a^k_{D_i}, z^0_{S_i})}{u_i(a^k_{D_i}, z^k_{S_i}) - u_i(b^k_{D_i}, z^k_{S_i})}, \tag{5.1}$$

une fois que nous avons obtenu q en comparant $(a^k_{D_i}, z^k_{S_i}, x^0_{-C_i})$ avec $(a^k_{D_i}, z^0_{S_i}, x^0_{-C_i})$, et r en comparant $(b^k_{D_i}, z^k_{S_i}, x^0_{-C_i})$ avec $(a^k_{D_i}, z^0_{S_i}, x^0_{-C_i})$.

5.4.2 Agrégation inter-cliques

Revenons à l'exemple de la figure 5.1. Une fois que nous avons obtenu $u_1(A, B)$ et $u_2(B, C)$ séparément, nous pouvons les combiner si l'on connait la valeur de k, tel que $u_1(A, B) + ku_2(B, C)$ forme une utilité. À nouveau, il est fort probable qu'il n'y ait pas de points vérifiant l'axiome de connexité des séparateurs. Toutefois, si nous avons un s tel que l'intensité des préférences sur $a^1b^0c^1$ est s fois celle de $a^0b^0c^1$, alors :

$$u_1(a^1, b^0) + ku_2(b^0, c^1) = s(u_1(a^0, b^0) + ku_2(b^0, c^1))$$

d'où l'on obtient :

$$k = \frac{u_1(a^1, b^0)}{u_2(b^0, c^1)(s - 1)}$$

D'une manière générale, supposons que tous les facteurs $u_p(\cdot)$, $p < i$ ont déjà été agrégés et que l'on veut maintenant agréger le facteur $u_i(\cdot)$. Soit $E_i = D_i \backslash C_1$, $F_i = D_i \cap C_1$ et $G = \{1, \ldots, n\} \backslash (D_1 \cup E_i)$. Soit s tel que l'intensité des préférences sur $(a_{D_1}, a_{E_i}, x^0_E)$ est s fois celle de $(x^0_{D_1}, a_{E_i}, x^0_E)$, alors :

$$u_1(a_{D_1}, x^0_{S_1}) + ku_i(x^0_{F_i}, a_{E_i}, x^0_{S_i}) = s(u_1(x^0_{D_1}, x^0_{S_1}) + ku_i(x^0_{F_i}, a_{E_i}, x^0_{S_i}))$$

d'où l'on obtient :

$$k = \frac{u_1(a_{D_1}, x^0_{S_1})}{u_i(x^0_{F_i}, a_{E_i}, x^0_{S_i})(s - 1)} \tag{5.2}$$

5.4.3 Agrégation inter-arbres

Pour terminer la calibration des préférences, il nous reste encore à exprimer sur la même échelle chacun des éléments de notre forêt de préférences. Les items indépendants étaient déjà exprimés naturellement sur la même échelle. Il nous faut essentiellement calibrer les arbres GAI.

Supposons que Y est un élément déjà calibré (par exemple, un item indépendant), et nous voulons calibrer l'arbre GAI $\mathcal{G}$ (avec une ou plusieurs cliques) sur les variables $\mathcal{X} = \times_{i=1}^{n} X_i$. Si nous pouvons trouver un q tel que l'intensité de préférences sur $(y'x^k)$ est q fois celle de (y^0x^k), avec des utilités pour y' et x^k non nulles, alors :

$$u'(y') + ku(x^k) = kqu(x^k)$$

d'où l'on obtient :

$$k = \frac{u'(y')}{ku(x^k)(q-1)} \tag{5.3}$$

5.5 L'implémentation informatique

Comme nous l'avons signalé au début du chapitre, dans la section 5.2, lorsqu'il s'agit d'une application, l'obtention de données réelles ainsi que sa mise à jour continuelle est un problème qui se pose avant même celui de la construction du modèle de préférences. Dans les sections précédentes, nous avons traité le problème d'élicitation et d'optimisation en utilisant des réseaux GAI, pourtant nous avons jusqu'alors ignoré ce problème d'obtention de données. Pour aborder ce problème, nous avons choisi dans l'implémentation informatique d'utiliser des sources de données extérieures, en créant notre application en tant qu'un **mashup**. Afin d'améliorer sa réactivité, nous avons utilisé le **modèle Ajax** de développement d'applications Web. Ces deux éléments sont détaillés dans la section suivante.

5.5.1 Les mashups et le modèle Ajax

Le terme **mashup** peut être mieux compris si l'on regarde son origine. Ce terme a été emprunté à l'univers musical, où il désigne un genre musical hybride obtenu en mélangeant deux ou plusieurs titres existants dans un nouveau morceau. Selon Geoghegan et Klass (2005), « *A mashup is a song created out of pieces of two or more songs, usually by overlaying the vocal track of one song seamlessly over the music track of another... Mashups are incredible fun and a fascinating way to reexperience some of your favorite tunes.* »

Par analogie, en informatique, le terme mashup désigne un nouveau genre d'applications Web interactives qui exploitent des sources de données externes pour créer des services nouveaux (Merrill, 2006). L'idée d'utiliser des données externes pour créer des nouvelles applications n'est pas nouvelle, par exemple, des sites existent depuis plus d'une dizaine d'années pour comparer le prix d'un produit donnée entre plusieurs distributeurs sur internet*. Toutefois, des développements récents ont à la fois diminué considérablement l'effort nécessaire pour créer des mashups, à travers la mise en place d'interfaces de programmation (API) Web, et ont aussi amélioré l'interactivité de ces applications grâce aux perfectionnements des clients (navigateurs).

La mise en place d'interfaces de programmation (API) Web

Traditionnellement, la seule manière d'accéder au contenu d'un site (que nous nommerons désormais un **fournisseur de contenu**) était par l'intermédiaire de ses pages Web. Or, ces pages sont extrêmement non-structurées et, en général, comportent un grand nombre d'informations sans rapport aucun avec le contenu et malgré cela, mélangées à celui-ci (telles que les instructions pour la mise en page sur l'écran de l'ordinateur). Ainsi, pour obtenir le contenu de ces pages, la construction préalable d'un *parser* adapté au format des pages du fournisseur de contenu est nécessaire. En plus, il n'est pas rare qu'un fournisseur de contenu change le format de ses pages, généralement pour mettre à jour l'apparence de son site. Par conséquent, il est fort probable que le parser ait besoin d'être modifié pour continuer à fonctionner après de tels changements. Cette méthode s'avère donc être un véritable cauchemar de manutention, puisqu'il demande un effort constant de programmation, ne serait-ce que pour accéder au contenu. En outre, des revendications de droits d'auteur de la part du fournisseur de contenu peuvent empêcher l'utilisation de son contenu.

Heureusement, ces dernières années, une nouvelle tendance s'est développée. Dans ce nouveau scénario, les fournisseurs de contenu se sont aperçus qu'en facilitant l'accès à leurs données par des développeurs de mashups, ils pouvaient augmenter leur visibilité sur le Web et ainsi générer plus de revenus, soit grâce à des ventes directes, soit simplement par l'augmentation du trafic sur leur site. Ainsi, plusieurs fournisseurs de contenu ont implanté une API pour un accès structuré à leurs données, par des développeurs. Les questions de droits d'usage ont également été adressées, une fois que les termes d'utilisation du contenu sont clairement explicités par son fournisseur conjointement à l'API Web.

Exemple 5.1. La librairie en ligne *Amazon.com* possède une API pour l'accès à son catalogue de livres. En plus, ils ont mis en œuvre un système pour permettre qu'un tiers puisse recevoir une commission sur la vente d'un livre acheté à partir d'un lien

*Tels que PriceGrabber : `http://www.pricegrabber.com`

placé sur son site. Par exemple, il est possible de créer un forum de discussions sur des livres avec toutes les données sur les livres (telles que le titre, l'auteur, la description), fournies de manière structurée par Amazon, et de placer un lien du type « *acheter ce livre sur Amazon.com* ». Si un utilisateur du forum clique sur ce lien et achète le livre, le développeur du forum recevra une commission.

La mise en œuvre de APIs Web ne se limite pas aux sites marchands, un grand nombre de sites qui proposent des services sur le Web (tels que Google* et Yahoo![†]) proposent désormais des APIs pour accéder à plusieurs de ces services.

Trois alternatives principales se présentent pour exposer le contenu d'un site de manière structurée : **SOAP**, **REST**, et **RSS/Atom**.

SOAP est originaire de la *communauté de génie de logiciel orientée objet*, ce qui se reflète dans le sens original de son acronyme : *Simple Object Access Protocol*. Il permet qu'une API composée par des multiples méthodes (par exemple, `ajouterEmploye()`, `listerEmployes()`) soit décrite par un format XML appelé **WSDL** (de l'anglais, *Web Services Description Language*). Cette API pourra ensuite être appelée de manière distante, par l'échange de messages. Ces messages sont plus souvent transportés par le protocole HTTP, mais d'autres protocoles (comme, par exemple, SMTP) peuvent être utilisés. SOAP a été critiqué par la communauté Internet comme étant trop complexe, et pour ne pas exploiter la puissance et simplicité des standards Internet tels que le HTTP. Les développeurs de SOAP ont admis les critiques au protocole et ont changé le focus des systèmes orientés objet à l'interopérabilité d'échanges de messages. Ainsi, le protocole a été rebaptisé *Services-Oriented Access Protocol*.[‡]

REST (de l'anglais *REpresentational State Transfer*) a été développé par Fielding (2000). Il utilise un style architectural inspiré par l'intention original du Web. Ainsi, REST défend que l'API d'un service doit être définie en termes de manipulations HTTP de documents (essentiellement, `GET`, `POST`, `PUT` et `DELETE`) adressées par des URIs, et non pas dans en termes d'appels de méthodes ayant des paramètres. Dans l'architecture REST, chaque opération doit être autosuffisante : il n'y a pas d'état gardé dans le serveur. Les documents manipulés sont souvent des documents XML, mais d'autres représentations sont parfaitement acceptables. Il est très courant d'utiliser des objets JSON[§] (de l'anglais *JavaScript Object Notation*), un format de données générique et indépendant de langage (pourtant fondé sur la notation des objets JavaScript) pour encoder des informations structurées.

*`http://code.google.com/`

†`http://developer.yahoo.com/`

‡SOAP, XML, WSDL, et HTTP sont des spécifications du *World Wide Web Consortium* (W3C) et peuvent être trouvées sur son site web : `http://www.w3.org/` . SMTP est une spécification de la *Internet Engineering Task Force* (IETF) et peut être trouvée sur son site web : `http://www.ietf.org/` .

§Voir `http://www.json.org/`

RSS désigne une famille de formats XML : *Really Simple Syndication* (RSS 2.0) ; *RDF* Site Summary* (RSS 0.90 et 1.0) ; *Rich Site Summary* (RSS 0.91) ; et on parle aussi souvent de RSS pour désigner également le format Atom†. Ces formats ont été développés pour permettre la **syndication** de contenu Web. Le terme syndication vient de la presse américaine. Dans ce contexte-là, il consiste à vendre le droit de reproduire un contenu ou de transmettre un programme à plusieurs diffuseurs. Ainsi, les syndicats vendent leur production (cartoons, bandes dessinées, chroniques, etc.) à plusieurs media. Par analogie, dans l'informatique, la syndication de contenu Web est une forme de syndication dans laquelle une portion du site d'un fournisseur de contenu est rendue disponible de manière structurée à d'autres sites à travers un **flux RSS** : un document XML dans l'un des formats RSS avec du contenu récemment ajouté (par exemple les derniers articles parus dans le site d'un journal). Pour les recevoir, l'utilisateur doit s'abonner au flux, ce qui lui permet de consulter rapidement les dernières mises à jour, à l'aide d'un lecteur de flux, sans avoir à se rendre sur le site. Les flux RSS sont notamment adaptés à la diffusion d'actualités, soit sous forme de texte, soit dans des formats multimedia (audio et vidéo).

Améliorations des clients

Traditionnellement, l'interaction avec des applications locales possédait un degré de richesse et de réactivité qui semblait hors de portée pour les applications Web. Le modèle classique pour ces dernières, est le suivant : la plupart des actions de l'utilisateur dans l'interface envoie une requête HTTP à un serveur Web. Le serveur exécute le traitement des données et retourne ensuite une page HTML au client. Cette approche ne configure pas une application très réactive. Lorsque le serveur est en train de traiter les données, l'utilisateur est dans l'attente. Et à chaque interaction, l'utilisateur attend un peu plus. Pourquoi l'utilisateur doit-il attendre chaque fois que l'application a besoin d'interagir avec le serveur ? En fait, pourquoi l'utilisateur doit même remarquer que l'application est allée vers le serveur ?

Aujourd'hui, des applications Web, telles que *Google Maps*‡, permettent des interactions comme le défilement d'une carte avec la souris, ou faire un zoom avant ou arrière. Tout se passe presque immédiatement, sans attendre la recharge des pages. Ces applications utilisent un nouveau modèle pour les applications Web, nommé Ajax (de l'anglais *Asynchronous Javascript And XML*, Garrett (2005)).

Le terme Ajax désigne un groupe de techniques utilisées pour créer des applications

*RDF est une spécification du W3C qui définit un modèle formel pour décrire des ressources (normalement des URI) et leurs métadonnées.

†Atom est un standard proposé par l'IETF alors que les autres formats RSS n'ont pas été normalisés par un organisme de standardisation.

‡`http://maps.google.com`

Web interactives (ces applications sont parfois nommées *applications Web 2.0*). La conséquence principale de l'utilisation de ces techniques est l'augmentation de la réactivité et l'interactivité des pages Web atteinte grâce à l'échange *en arrière-plan* de petites quantités de données avec le serveur, de sorte que toute la page Web n'a pas besoin d'être rechargée à chaque fois que l'utilisateur effectue une action.

Ajax est asynchrone dans le sens où des données supplémentaires sont demandées au serveur et chargées en arrière-plan sans perturber l'affichage et le comportement de la page courante. JavaScript est le langage dans lequel les appels de fonction Ajax sont habituellement réalisés. Pour modifier la page Web en exhibition, des appels JavaScript vers le DOM* sont utilisés. Malgré la signification du terme Ajax, il n'y a aucune exigence que le contenu asynchrone soit formaté en XML, tout format peut être utilisé, tel que HTML, texte pur ou JSON.

Ainsi, dans une application Ajax, chaque action de l'utilisateur qu'auparavant générait une requête HTTP prend, à la place, la forme d'un appel local au moteur Ajax (le code, écrit en Javascript, chargé de gérer l'interaction de l'application et qui exécute localement, dans la machine cliente). Toute réponse à une action de l'utilisateur qui ne nécessite pas de contacter le serveur – telles que la simple validation des données, l'édition des données en mémoire, et même certaines actions de navigation – le moteur gère par lui-même. Si le moteur a besoin de quelque chose sur le serveur afin de répondre – par exemple, s'il s'agit d'une soumission de données pour traitement, ou de la récupération des données nouvelles – le moteur fait ces demandes de manière asynchrone, le plus souvent en utilisant XML ou JSON, sans stopper l'interaction de l'utilisateur avec l'application. Une fois que le moteur a de nouvelles données arrivées du serveur, il met à jour l'interface d'utilisateur de manière incrémentale en modifiant la page exhibée à travers le DOM.

Les figures 5.2 et 5.3 comparent le modèle d'application Web classique et le modèle d'application Web Ajax. Dans la figure 5.2 nous remarquons que dans le cas des applications Ajax (5.2b), les interactions avec le serveur ne se limitent pas à des requêtes HTTP répondues par des données HTML+CSS (comme dans 5.2a), même si ces dernières sont toujours possibles. Le modèle Ajax nous permet d'échanger des petites quantités de données avec le serveur. Ces données ne nécessitent pas d'être du type HTML+CSS (et plus couramment il s'agit de données XML/JSON), car au lieu d'être affichées directement, elles seront interprétées par le moteur Ajax, qui ensuite pourra modifier la page affichée (en changeant dynamiquement ses données HTML+CSS à travers des appels JavaScript au DOM).

*De l'anglais *Document Object Model*. Le DOM est une spécification du W3C qui définit une interface pour permettre à des programmes et scripts d'accéder et modifier dynamiquement le contenu, la structure et le style de documents. Dans le cas des applications Web mentionnées, le DOM est implanté par le navigateur pour permettre que du code JavaScript modifie dynamiquement la page Web exhibée à l'écran.

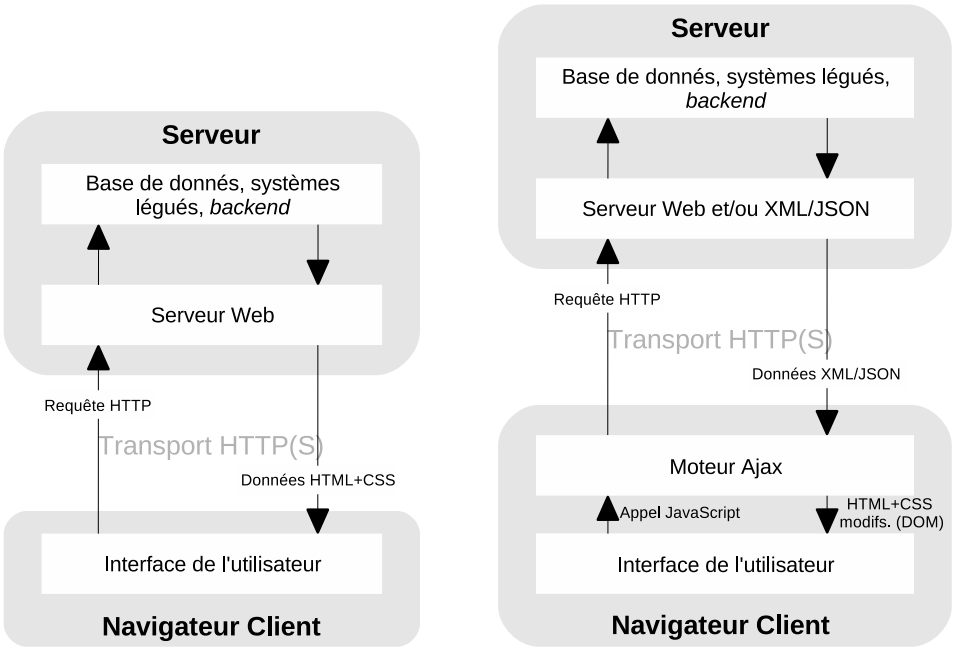


FIG. 5.2 – Le modèle d'application Web classique (a) contre le modèle d'application Web Ajax (b). (Figure adaptée de Garrett (2005))

La figure 5.3 contraste le comportement synchrone propre à une application Web traditionnelle avec le comportement asynchrone d'une application Ajax. Dans la première, les interactions de l'utilisateur déclenchent une requête au serveur et l'utilisateur est contraint d'attendre lorsque le serveur traite les données. Dans la deuxième, les interactions avec le serveur sont gérées par le moteur Ajax en arrière-plan, sans bloquer l'interface de l'application. Dans ce cas, les mises à jour de l'interface de l'utilisateur sont réalisées dynamiquement par le moteur Ajax et peuvent être déclenchées par l'arrivée de données venues du serveur ou par des traitements locaux de données. L'interface de l'utilisateur reste toujours disponible.

Remarquons toutefois que dans le modèle Ajax, nous augmentons la charge de traitement du client. Ainsi, nous devons souligner que l'adoption en masse des applications Ajax n'est possible que grâce à la récente augmentation de la puissance de calcul des ordinateurs clients. Cela est d'autant plus important qu'encore aujourd'hui, les environnements d'exécution JavaScript de tous les principaux navigateurs sont purement interprétés, ce qui entraîne une vitesse d'exécution du code JavaScript (duquel les applications Ajax dépendent largement) très inférieure à celle des codes compilés. Cependant, la prochaine génération de navigateurs Web doit être dotée d'un environment JavaScript couplé à

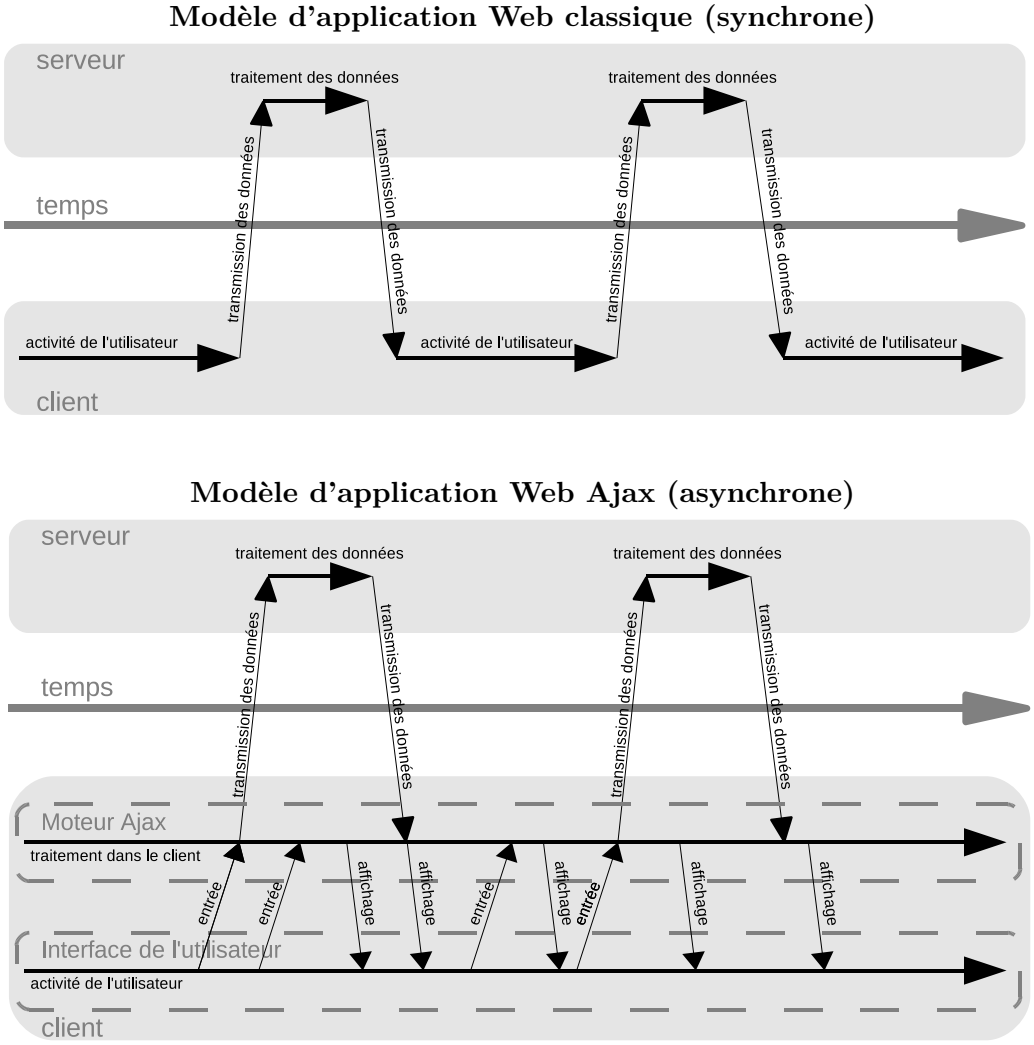


FIG. 5.3 – L'interaction avec une application Web classique comparée à l'interaction avec une application Web Ajax (Figure adaptée de Garrett (2005))

un compilateur *just in time* (JIT)*, ce qui devra augmenter considérablement la vitesse d'exécution des applications Ajax et permettre ainsi l'augmentation de son degré de sophistication.

5.5.2 L'obtention des données

La figure 5.4 montre le début de l'interaction avec *GVisite?*. L'utilisateur commence par renseigner la ville de destination. Dans cet exemple, que nous allons développer tout au long de cette section pour illustrer l'implémentation de *GVisite?*, il s'agit de la ville de Paris. Une fois la ville de destination connue, nous devons obtenir les items

*Par exemple, pour les navigateurs de la famille *Mozilla*, tel que *FireFox*, l'environment JavaScript *Tamarin* (`http://www.mozilla.org/projects/tamarin/`), doté d'un compilateur JIT, est en développement.

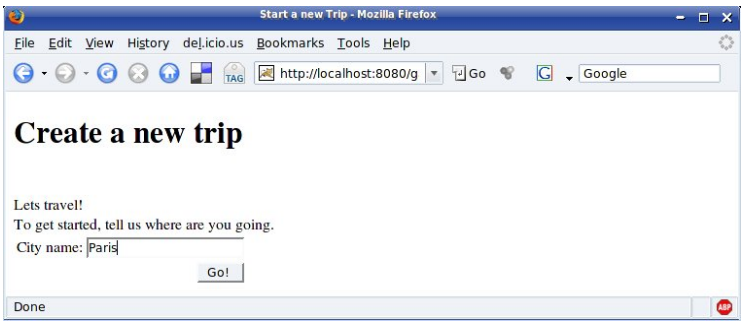


FIG. 5.4 – Définition de la ville de destination

touristiques de cette ville. Nous utilisons les API Web fournis par *Yahoo! Travel Web Services* afin d'obtenir le contenu dans un format structuré. *Yahoo! Travel Web Services* fournit deux API REST (accessibles par la méthode GET du protocole HTTP) : **Trip Search** (`tripSearch`) et **Get Trip** (`getTrip`).*

Le diagramme de séquence UML dans la figure 5.5 montre les étapes du processus d'obtention des items touristiques. D'abord, nous utilisons `tripSearch` pour chercher des plans de voyage qui concernent la ville de destination. Ensuite, pour chaque plan de voyage renvoyé par l'appel à `tripSearch` nous utilisons `getTrip` pour obtenir les détails de ce voyage. Ces détails comprennent les items touristiques visités. Nous ajoutons donc les items touristiques du voyage à un ensemble d'items touristiques de la destination (l'appel à `addTravelItems` dans le diagramme). Ainsi, l'ensemble d'items touristiques de la destination sera l'union des items dans les plans de voyage obtenus.

Malheureusement, `addTravelItems` n'est pas aussi simple à implémenter. Comme nous l'avons mentionné dans le paragraphe précédent, nous voulons avoir un ensemble d'items touristiques de la destination. Ainsi, chaque item touristique doit y paraître une seule fois. Malheureusement, dans les réponses renvoyées par l'API Web utilisée, les items ne sont pas identifiés de manière unique. En effet, chaque item apparaît exactement comme l'utilisateur l'a enregistré. Par exemple, dans un plan de voyage, un voyageur a enregistré pour titre d'un item touristique « Centre Pompidou », dans un autre plan il y a « Musée Pompidou », dans un autre « Centre Georges Pompidou », ou encore « Musée Georges Pompidou », ... Pourtant, tous ces plans de voyage font référence au même item, celui-ci doit donc correspondre à un seul élément dans l'ensemble d'items touristiques de la destination, le « Centre Georges Pompidou ».

Pour régler ce problème, nous avons traité le titre d'un item dans un voyage de la même façon qu'un moteur de recherche traite la chaîne de caractères passée comme terme de recherche : nous la décomposons en *Tokens* et après un algorithme de *scoring*

*Par souci de complétude, les documentations de ces deux API sont dans l'appendice

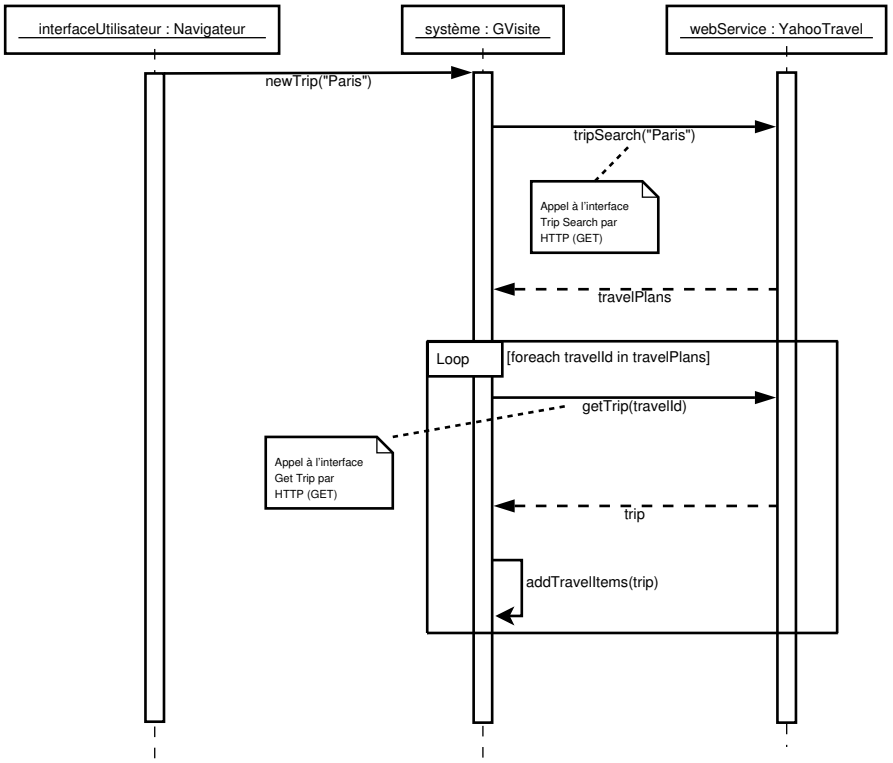


FIG. 5.5 – Récupération des items touristiques pour une ville.

peut calculer la ressemblance entre le terme de recherche et chacun des documents d'une collection. Nous avons utilisé Lucene, un moteur de recherche embarqué de haute performance, écrit en Java et mis à disposition librement, sous licence Apache*.

La fonction `addTravelItems` montre le pseudocode de la méthode utilisée. Pour identifier si un item dans un voyage correspond à un item déjà ajouté à l'ensemble d'items touristiques de la ville de destination, nous commençons par décomposer le titre de l'item dans un ensemble de tokens (ligne 3). Ensuite, nous construisons une requête booléenne avec cet ensemble (lignes 4-7). Lors de la construction de la requête booléenne, chaque token est inséré dans une requête floue (ainsi de petites variations d'écriture pourront être prises en compte)† et est ajouté à la requête booléenne avec l'option SHOULD_APPEAR, ce qui veut dire qu'il n'est pas obligatoire qu'il soit présent dans le résultat (mais en cas positif, le résultat aura un meilleur score). Nous avons utilisé le seuil de similarité 0,5, déterminé de façon expérimentale. Les lignes 13-14 uniformisent les items utilisés pour représenter les items touristiques de la ville dans l'ensemble des voyages (par exemple, nous remplaçons toute variation de « Centre Georges Pompidou »

*Disponible à l'adresse : `http://lucene.apache.org/java/`

†Dans Lucene, la mesure de similarité d'une requête floue est fondée sur la distance d'édition (dite de Levenshtein (1966)).

par cette appellation). Cela sera important pour le filtrage des items.

Entrées : *trip*, un voyage

```
1  Soit I l'ensemble d'items touristiques déjà ajoutés
2  pour chaque item ∈ trip faire
3      Tokens ← Lucene.tokenize(titre de item)
4      Soit B une requête booléenne
5      pour chaque t ∈ Tokens faire
6          B.ajouter(Requête_Floue(t), SHOULD_APPEAR)
7      fin
8      Soit D l'ensemble de titres des items dans I
9      Result ← Lucene.rechercher(B dans D)
10     si meilleur score dans Result < 0.5 alors
11         I ← I ∪ item
12     sinon
13         Soit item' l'item correspondant au meilleur score dans Result
14         trip ← trip − {item} ∪ {item'}
15     fin
16 fin
```

Fonction `addTravelItems`

5.5.3 Le filtrage adaptatif des items

Grâce à la méthode décrite dans la section précédente, nous sommes capables d'obtenir les données nécessaires pour *GVisite?*. Nous avons donc adressé le problème 2 suscité dans la section 5.2 (page 132). Maintenant, passons à l'implementation du filtrage adaptatif des items, comme nous l'avons décrit dans la section 5.3.

Plusieurs algorithmes pour trouver des règles d'association ont été proposés dans la littérature. L'un des plus réussis, souvent utilisé comme référence, est l'algorithme **Apriori** (Agrawal et Srikant, 1994). Dans *GVisite?*, nous avons utilisé une implémentation en C++ d'Apriori (Borgelt et Kruse, 2002; Borgelt, 2003, 2004), librement disponible* sous licence LGPL. Nous exécutons Apriori après `addTravelItems`, avec un facteur de confiance $c = 0.7$ et support $s = 0.1$†. L'ensemble $\mathcal{R}$ de règles obtenues sera ensuite utilisée pour filtrer de manière adaptative les items exhibés sur l'écran à chaque instant (cf. section 5.3).

Dans l'implémentation d'Apriori utilisée, la conclusion est toujours un ensemble unitaire. L'auteur justifie ce choix dans la documentation, de la manière suivante : *The restriction to single item consequents is due to the following considerations: In the first place, association rule mining usually produces too many rules even if one confines*

*À l'adresse `http:/www.borgelt.net/apriori.html`

†Ces valeurs ont été établis de façon expérimentale.

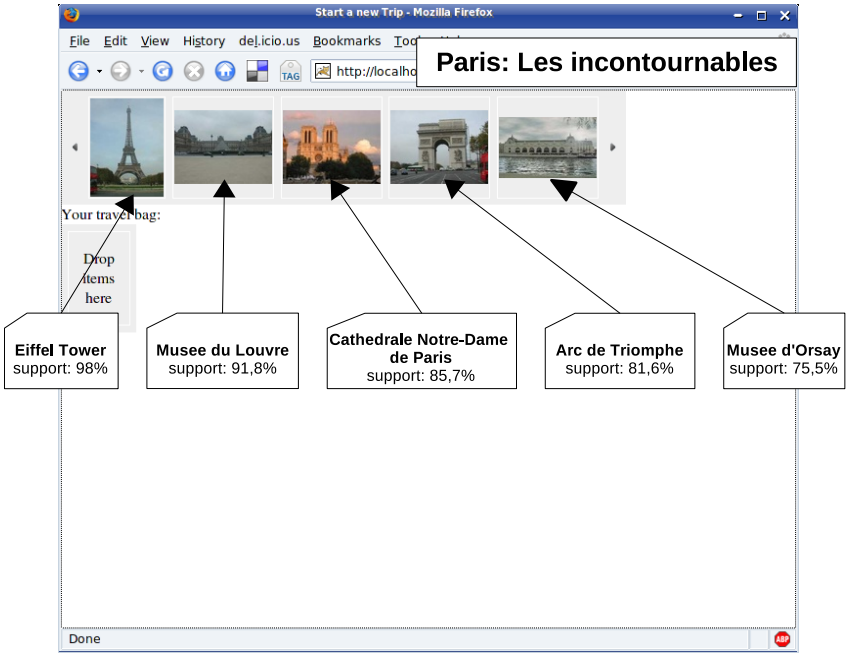


FIG. 5.6 – *Les Incontournables* et le support pour chacune des règles.

oneself to rules with only one item in the consequent. So why should one make the situation worse by allowing more than one item in the consequent? (It merely blows up the output size.) Secondly, I do not know any application in which rules with more than one item in the consequent are of any real use. The reason, in my opinion, is that such more complex rules add almost nothing to the insights about the data set. To understand this, consider the simpler rules that correspond to a rule with multiple items in the consequent, that is, rules having the same antecedent and consequents with only single items from the consequent of the complex rule. All of these rules must necessarily be in the output, because neither their support nor their confidence can be less than that of the more complex rule. That is, if you have a rule a b $\Rightarrow$ *c d, you will necessarily also have the rules a b* $\Rightarrow$ *c and a b* $\Rightarrow$ *d in the output. Of course, these latter two rules together do not say the same as the more complex rule. However, what do you gain from the additional information the more complex rule gives you? How can you use it? And is this little extra information worth having to analyze a much bigger rule set?* La figure 5.6 montre l'écran qui suit celui de la figure 5.4 pour notre voyageur qui va à Paris. Cinq lieux parisiens représentent « Les Incontournables » détectés par *GVisite?* : la Tour Eiffel, le Musée du Louvre, la Cathédrale Notre Dame de Paris, l'Arc de Triomphe et le Musée d'Orsay. Nous avons ajouté à la figure des notes avec le support de chacun des items (les titres des items paraissent dans les notes ajoutées tels qu'ils ont été récupérés par le système).

L'interaction de l'utilisateur avec le système se poursuit à l'aide d'opérations de type glisser-déposer.* À chaque instant, l'utilisateur peut cliquer sur l'un des items affichés, le glisser jusqu'à son « panier de visite » et le déposer. Avec cette opération, l'utilisateur signale qu'il est intéressé en visiter cet item. Dans les coulisses, cette simple opération déclenche une série de traitements.

Le diagramme de séquence UML dans la figure 5.7 montre les traitements déclenchés dans le client et le serveur une fois que l'utilisateur dépose un item, disons « La Cathédrale Notre Dame de Paris », dans son panier de visite. L'interface de l'utilisateur notifie cet évènement au `moteurAjax`, par l'appel d'une méthode (`addItem(item)`, où `item` ici correspond à la cathédrale). Après cela, l'interface redevient immédiatement disponible pour de nouvelles interactions (grâce au comportement asynchrone évoqué dans la section 5.5.1).† De sa part, `moteurAjax` crée une requête (`req`), qui sera transmise en arrière-plan au serveur (notons que `moteurAjax` appelle la méthode asynchrone `addSingleItem(item)` et redevient immédiatement disponible‡). La requête est ensuite transmise au serveur (`système`§), qui appelle une méthode (`newConseqs(item)`) pour découvrir si avec l'ajout d'`item` au panier de visite la prémisse de nouvelles règles d'association est satisfaite. Dans ce cas nous allons afficher les items qui sont les conclusions de ces règles. Dans notre exemple concret, nous avions les règles : « La Cathédrale Notre Dame de Paris » ⇒ « La Sainte Chapelle » ; « La Cathédrale Notre Dame de Paris » ⇒ « Basilique du Sacre-Coeur » ; et « La Cathédrale Notre Dame de Paris » ⇒ « Avenue des Champs-Elysees ». Ainsi, la réponse du système (dans le diagramme, le pair `(consequents, rules)`) contient ces trois nouveaux items et les règles qui ont été utilisées. Une fois que `req` a reçu la réponse, elle notifie `moteurAjax` (dans le diagramme, par l'appel de `newItems(consequents, rules)`), qui de son côté stocke les règles (elles nous seront utiles par la suite) et met à jour l'`interfaceUtilisateur` pour qu'elle affiche ces trois nouveaux items. Ainsi, après avoir ajouté « La Cathédrale Notre Dame de Paris »

*Soulignons que l'obtention de ce niveau d'interactivité a demandé le développement de code pour le client (navigateur). Cela inclut le moteur Ajax et le code qui interagit avec le DOM. Nous avons utilisé la *Yahoo! User Interface Library* (YUI). Cette bibliothèque est un ensemble d'outils et contrôles, écrits en JavaScript, pour faciliter la construction d'applications Web interactives qui utilisent des techniques telles que des appels au DOM et Ajax. La YUI est disponible librement sous une licence du type BSD à l'adresse `http://developer.yahoo.com/yui/` .

†Dans le diagramme, il est possible de visualiser quand des traitements de données sont réalisés dans un objet ; nous avons un rectangle dans sa « ligne de vie ». Au contraire, quand nous avons une ligne pointillée, cela signifie que l'objet est oisif.

‡Le caractère asynchrone de la méthode est affiché dans le diagramme par le style différent de flèches.

§Ceci est le *back end* de *GVisite?*. Nous l'avons implémenté comme une application *multi-tier*, en Java. Nous utilisons JSP et Struts dans le Web-tier et une base de données SQL dans le *tier* de stockage de données (pour la preuve de concept, nous avons utilisé HSQLDB, une petite base de données librement disponible écrite en Java. Toutefois, toute communication avec la BD se fait par moyens d'interfaces standard JDBC). Le *tier* d'application utilise le même code utilisé pour les expérimentations dans les chapitres précédents de la thèse.

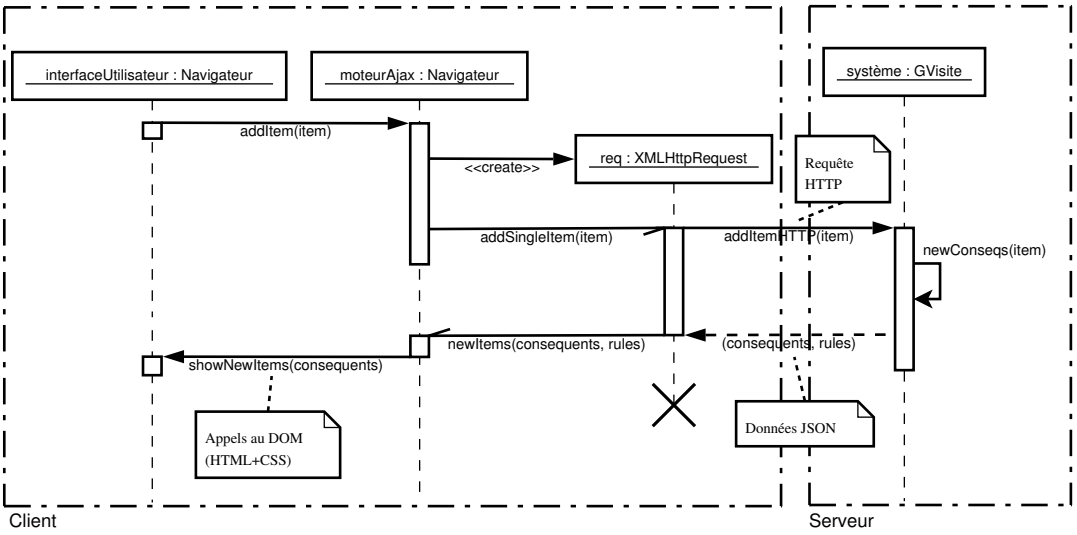


FIG. 5.7 – Nouveau item déposé dans le panier de visite de l'utilisateur

à son panier de visite, l'interface de l'utilisateur est alors celle la figure 5.8 (nous avons ajouté des notes pour souligner les nouveaux items parus).

5.5.4 La prise en compte de dépendances

Poursuivrons l'exemple que nous avons développé jusqu'à maintenant. Au début, nous avons présenté « Les Incontournables » à l'utilisateur : *ces items sont considérés comme étant indépendants*. L'utilisateur a ajouté « La Cathédrale Notre Dame de Paris » à son panier de visite, ce qui a provoqué la parution des items : la Sainte Chapelle, la Basilique du Sacre-Coeur et l'Avenue des Champs-Elysees. Quand le moteur Ajax a reçu ces items pour affichage, il a aussi stocké localement les règles qui ont été utilisées. Ainsi, nous connaissons les prémisses de ces règles, c'est-à-dire, un sous-ensemble des items qui sont dans le panier de visite de l'utilisateur. Supposons maintenant que l'utilisateur clique sur la « Basilique du Sacre-Coeur » et la dépose dans son panier de visite. *GVisite?* considère que la règle d'association responsable de la parution d'un item est un indice que les préférences pour cet item soient dépendantes des items qui apparaissent dans la prémisse de la règle. C'est pourquoi, au lieu de notifier le serveur de l'ajout de l'item au panier de visite (comme dans le diagramme 5.7), le moteur Ajax présente un dialogue où ces interdépendances présumées sont montrées à l'utilisateur. Celui-ci peut enlever des items qui n'influencent pas ses préférences, d'après lui, sur l'item ajouté (en cliquant sur le bouton `Remove` au-dessous de l'item à enlever).

L'utilisateur poursuit l'interaction avec *GVisite?*, il a à la fin ajouté les items indépendants : *Eiffel Tower*, *Musee du Louvre*, *Arc de Triomphe*, *Musee d'Orsay*, *Jardin*

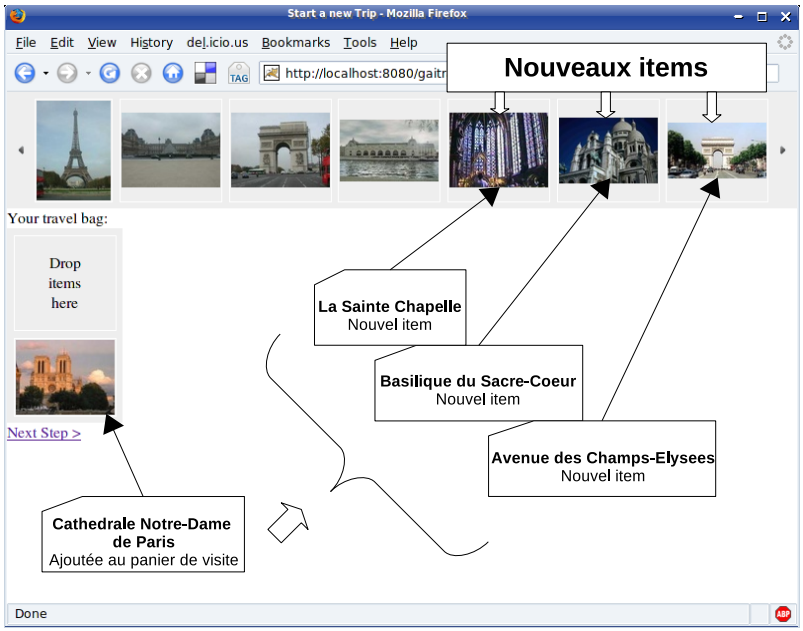


FIG. 5.8 – Interface de l'utilisateur après l'ajout de « La Cathédrale Notre Dame de Paris » à son panier de visite (notes ajoutées dans la figure)

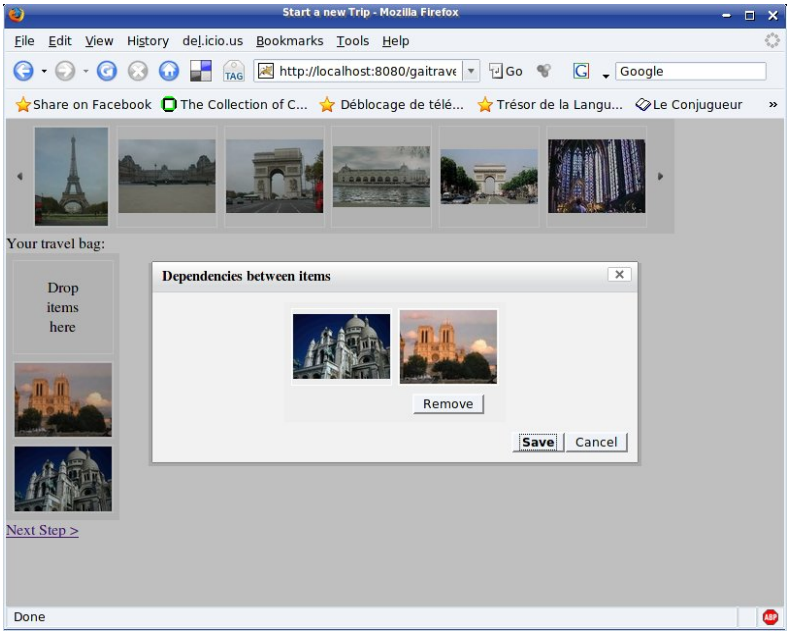


FIG. 5.9 – Items interdépendants

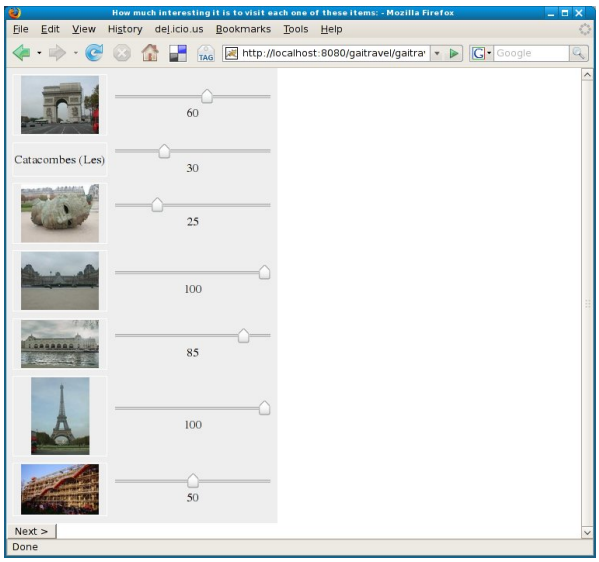


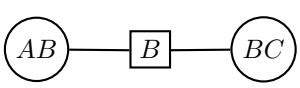
FIG. 5.10 – Préférences pour les items indépendants

des Tuileries, *Les Catacombes* et *Centre Pompidou*. Les n-uplets d'items dépendants ont été : *La Cathédrale Notre Dame de Paris* et *Basilique du Sacre-Coeur* ; *La Cathédrale Notre Dame de Paris* et *La Sainte Chapelle* ; *Place de la Concorde* et *Avenue des Champs-Elysees*.

5.5.5 L'expression et la calibration de préférences

La prochaine étape de l'interaction de l'utilisateur avec *GVisite?* lui permet d'exprimer l'intensité de ses préférences (dans une échelle de 0 à 100) pour les items indépendants (figure 5.10).

Pour les termes interdépendants, considérons la figure 5.11, qui représente les interdépendances entre les items : *La Sainte Chapelle*, *La Cathédrale Notre Dame de Paris* et *Basilique du Sacre-Coeur*.



A	-	La Sainte Chapelle
B	-	La Cathédrale Notre Dame de Paris
C	-	Basilique du Sacre-Coeur

FIG. 5.11 – Élicitation : termes interdépendants

Pour la construction d'une utilité u non transférable par rapport à $(x^0, \{AB, BC\})$ nous demandons d'abord des intensités de préférences, dans une échelle de 0 à 100, pour a^1b^0, a^1b^1, b^0c^1, b^1c^0 et b^1c^1 (voir figure 5.12). La figure 5.13 nous montre comment nous obtenons dans *GVisite?* les nombres q et r, nous permettant ainsi de réaliser l'agrégation intra-clique (cf. section 5.4.1).

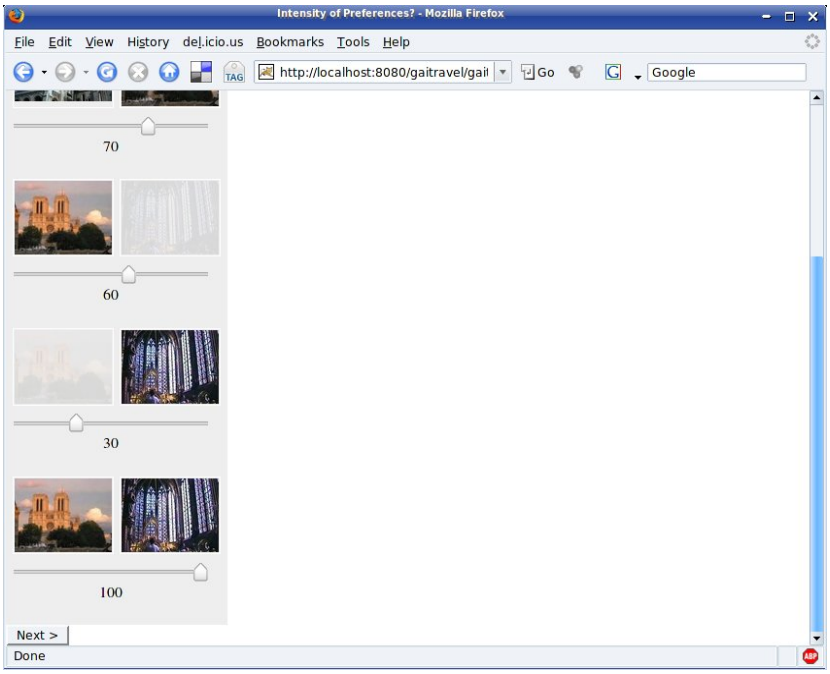


FIG. 5.12 – Intensités de préférences

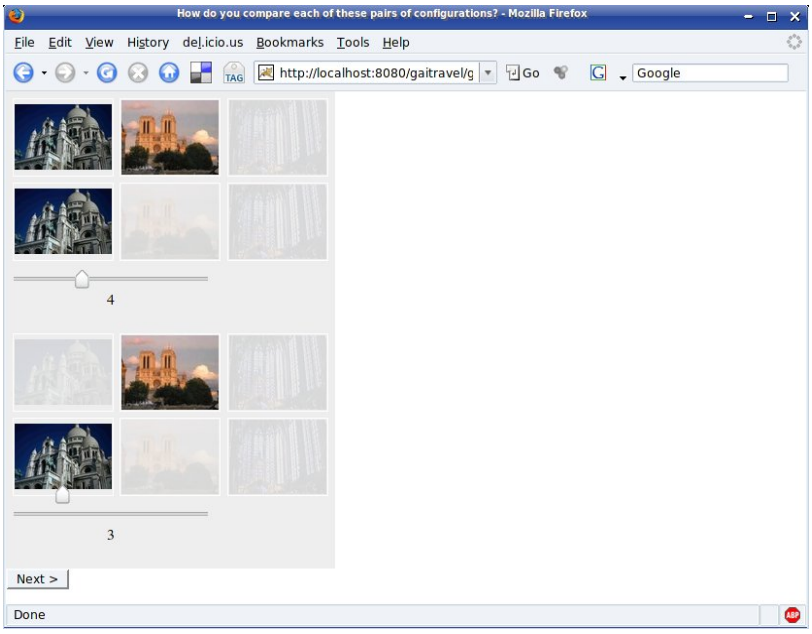


FIG. 5.13 – Obtention de k intra-clique

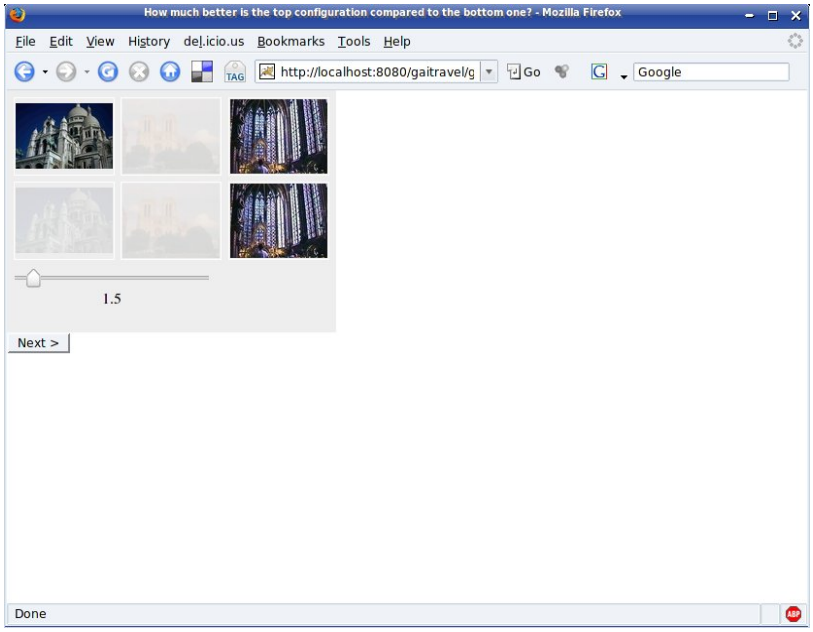


FIG. 5.14 – Comparaison inter-cliques pour un arbre

Ensuite, la figure 5.14 montre l'interface de dialogue que permet d'obtenir le facteur s pour l'agrégation inter-cliques (cf. section 5.4.1) dans *GVisite?*.

Pour terminer la calibration des préférences, il nous reste encore de mettre sur la même échelle chacun des éléments de notre forêt de préférences : les items indépendants ; les arbres GAI, tels que celui de la section précédente ; et les cliques indépendantes, tels que (*Place de la Concorde*, *Avenue des Champs-Elysees*). Pour les premiers, nous les supposons déjà sur la même échelle (voir figure 5.10). Il nous faut toutefois calibrer les deux autres types d'élément, de la façon presentée dans la section 5.4.3.

La figure 5.15 montre cette étape dans *GVisite?*. Dans la partie supérieure de la figure, nous calibrons l'arbre – d'une seule clique– (*Place de la Concorde*, *Avenue des Champs-Elysees*) par rapport à un élément déjà calibré, l'item indépendant *Musee d'Orsay*. Dans la partie inférieure de la figure, nous utilisons le même élément pour calibrer l'arbre formé par *La Sainte Chapelle*, *La Cathédrale Notre Dame de Paris* et *Basilique du Sacre-Coeur* (celle de la figure 5.11).

5.5.6 La détermination des poids

Comme nous l'avons énoncé dans la description du problème (section 5.2), le poids de chaque item correspond au temps que nous envisageons de consacrer à sa visite. Malheureusement, le fournisseur de contenu que nous avons utilisé ne nous offre pas de données précises sur la durée de chaque visite dans les plans de voyage obtenus. Toutefois,

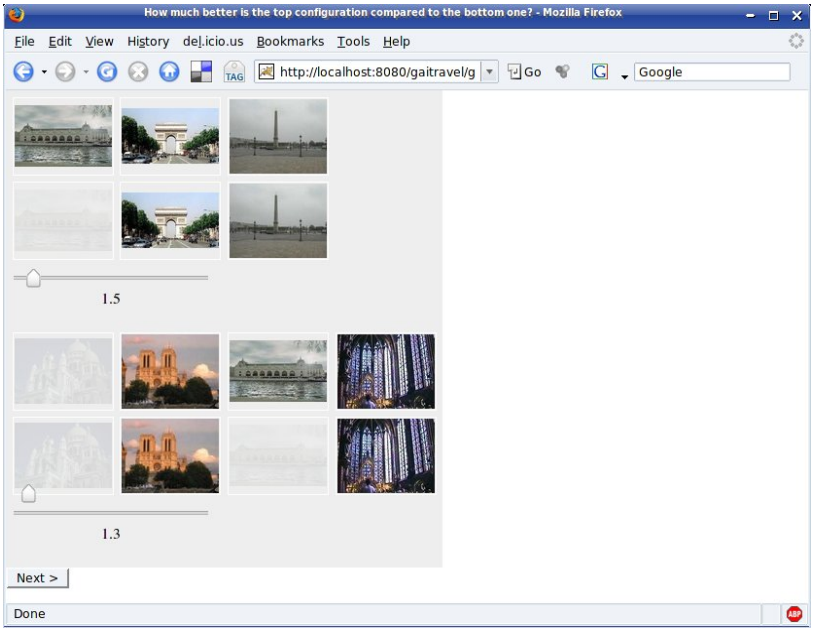


FIG. 5.15 – Calibration inter-arbres

pour chaque plan de voyage nous en connaissons sa durée totale (en nombre de jours). Nous avons ainsi utilisé cette information pour estimer le temps de visite de chaque item.

Supposons que nous avons un plan de voyage d'1 jour où 3 items ont été visités. Sans aucune information additionnelle, notre hypothèse par défaut est de considérer que chaque item a consommé un tiers de la journée. Maintenant supposons que l'un de ces items apparaît aussi dans un autre plan de voyage, de 2 jours, où 4 items ont été visités. Nous pouvons alors estimer le temps de visite moyen pour cet item comme étant $\frac{1/3+2/4}{2} = 5/12$ d'une journée.

Nous procédons ainsi pour estimer le temps de visite pour chacun des items dans le panier de visite de l'utilisateur, c'est-à-dire que nous calculons le temps de visite moyen de chacun de ses items en utilisant tous les plans de voyage (retournés par `tripSearch`) où il apparaît.

La figure 5.16 montre l'écran où nous affichons les temps de visite calculés pour chaque item dans le panier de visite de l'utilisateur. Celui-ci peut les modifier, selon ce qu'il trouve le plus adapté à son rythme de visite (ou simplement accepter les temps suggérés).

5.5.7 Le calcul de la solution

Nous avons finalement tous les éléments d'une instance du NGKP, à l'exception du nombre de jours du voyage, ce que nous obtenons dans l'étape suivante (figure 5.17).

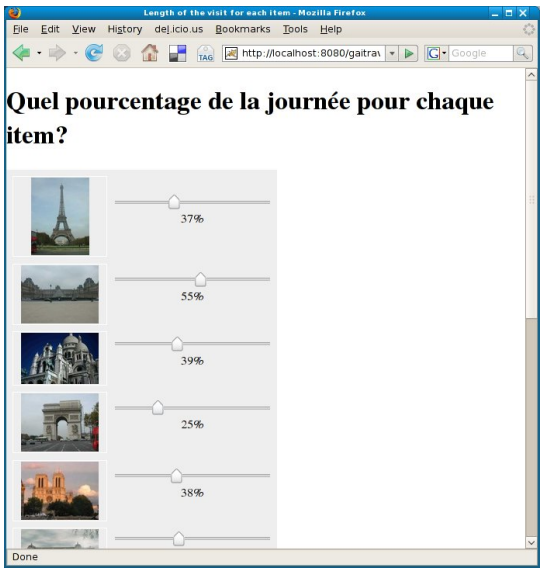


FIG. 5.16 – Temps de visite pour les items

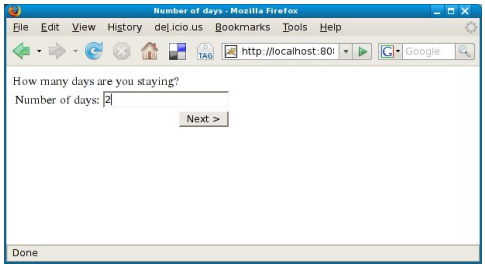


FIG. 5.17 – Durée du voyage

Nous calculons ainsi, en utilisant la méthode décrite dans le chapitre 2 (section 2.5), l'ensemble des items de plus grande utilité dont le temps total de visite ne dépasse pas le nombre de jours du voyage T de l'utilisateur. Formellement, nous résolvons le problème :

$$\begin{aligned} &\text{maximiser la fonction GAI} && u(x_1, ..., x_n) = \sum_{i=1}^{k} u_i(x_{C_i}) \\ &\text{sous la contrainte que} && \sum_{j=1}^{n} t_j x_j \leq T\,, \\ & && x = (x_1, x_2, \ldots, x_n) \in \{0,1\}^n. \end{aligned} \tag{5.4}$$

La solution obtenue est alors affichée sur l'écran de l'utilisateur (figure 5.18).

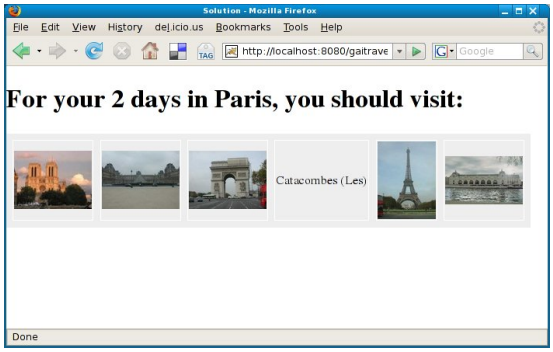


FIG. 5.18 – Résultat final

5.6 Discussion

Dans ce chapitre, nous avons développé une application Web qui utilise des réseaux GAI dans un problème touchant au domaine touristique. Des méthodes inspirées de la théorie de la décision ont été très peu exploitées dans des systèmes de recommandation pratiques. Cette apparente négligence n'est pas due à un manque d'intérêt de la part des développeurs de ces systèmes, mais à la difficulté de trouver l'adéquation entre des méthodes qui sont théoriquement bien fondées et celles qui sont adaptées aux besoins pratiques des décideurs (c'est-à-dire, leurs utilisateurs).

Des modèles graphiques, tels que des GAI-Nets se montrent utiles pour rapprocher ces deux champs, comme nous avons pu le constater en développant *GVisite?*, en tant que preuve de concept pour une application Web capable d'aider un voyageur à choisir ce qu'il doit visiter lors de son passage dans une ville choisie. Cependant, nous avons pu également constater que de nouveaux développements sont toujours nécessaires pour augmenter à la fois l'interactivité et le caractère explorateur d'un tel système. Par exemple, le type de comparaison que l'utilisateur est censé faire dans *GVisite?* peut être considéré comme excessivement précis et irréaliste. Nous pensons que l'introduction de méthodes pour gérer l'imprécision (voir par exemple, Słowiński (1998)), comme des préférences floues pourrait se montrer valuables pour faciliter l'utilisation de tels systèmes.

Pour le problème abordé, plusieurs sophistications sont possibles. Par exemple, il peut être envisagé que le système soit capable d'organiser ce que l'utilisateur ira visiter chaque jour, ce qui revient à un problème à multiples sac-à-dos, où le nombre m de sac-à-dos est celui de jours de visite. Aussi, il est possible de penser à un sac-à-dos multi-critère, où l'utilité est seulement l'un des critères. Par exemple, nous pouvons vouloir minimiser le nombre de billets de transport public utilisés par jour, ou le trajet parcouru à pied quotidiennement, ou encore le temps passé dans les transports publics.

Conclusion

Dans cette thèse, nous nous sommes efforcés de considérer divers problèmes liés à la représentation de préférences modélisables par des décompositions GAI (indépendance additive généralisée). Plus particulièrement, les problèmes considérés surgissent lorsque nous envisageons de traiter des tâches de recommandation dans des systèmes interactifs d'aide à la décision dans des domaines combinatoires de grande taille. L'originalité de notre approche réside surtout dans la prise en compte simultanée d'un aspect de préférences complexes (non-additives) et d'un aspect combinatoire dans un système interactif d'aide à la décision capable de lister les meilleures alternatives pour le décideur. Dans les pages qui suivent, nous réalisons une synthèse de notre contribution, et puis nous évoquons des perspectives de recherche qui pourront être poursuivies dans la suite de nos travaux.

Synthèse

Après avoir réalisé un tour d'horizon des travaux récents en modélisation/représentation de préférences, nous nous sommes focalisés sur les réseaux GAI, un modèle graphique adapté à la représentation de fonctions d'utilité GAI décomposables. Nous avons commençé l'étude par la recherche des k-meilleures alternatives du produit cartésien, un problème de base pour des tâches de recommandation. Nous avons obtenu des bons résultats en développant un algorithme dérivé des techniques issues de la littérature de réseaux bayésiens, pour trouver les m solutions les plus probables. Notre approche est notamment optimisée pour les cas où le k demandé est de grand taille. Grâce à cette efficacité, nous avons pu adapter l'algorithme pour l'utiliser dans une méthode de *branch-and-bound* pour résoudre des instances du problème du sac-à-dos 0-1 avec une fonction non-linéaire.

Cette efficacité pour les k de grande taille nous a permis également de développer une méthode exacte efficace (fondée sur le rangement d'une fonction GAI-décomposable), pour la recherche de la solution de meilleur compromis selon des critères non-linéaires. Cette recherche de compromis peut se poser soit dans le cadre de la décision collective (où

chaque décideur a ses préférences représentées par un réseau GAI), soit dans le cadre de la décision multi-critère (où chaque réseau GAI représente un critère). Parmi les critères non-linéaires de compromis étudiés, nous soulignons les normes de Tchebytcheff (qui peuvent explorer toutes les solutions non-dominées au sens de Pareto) ; et les intégrales de Choquet convexes, qui peuvent représenter un sous-ensemble des majorités floues linguistiques.

Toutefois, avant de pouvoir réaliser des tâches d'optimisation avec le modèle GAI, il est nécessaire de le construire. L'élicitation de modèles GAI est naturellement plus complexe que celle des modèles additifs, une fois que nous avons davantage de dépendances entre les variables. Ainsi, il est nécessaire d'élaborer des méthodes afin d'alléger la charge d'élicitation qui pèse sur le décideur. Souvent, ces méthodes seront adaptées à la tâche pour laquelle nous voulons utiliser le modèle de préférences. Nous avons proposé une méthode heuristique d'élicitation partielle, adaptée à l'élicitation de modèles GAI pour des tâches du type k-meilleures solutions, qui peut être utilisée si nous disposons d'une base préalable de fonctions d'utilité.

Enfin, nous avons développé une application Web comme preuve de concept de l'application des réseaux GAI dans les systèmes de recommandation. Des problèmes classiques pour le développement de telles applications, tel que l'intégration de données non-structurées ont du être traités ici. Nous avons pu modéliser le problème du choix des sites touristiques à visiter dans une ville pendant un court séjour comme un problème de sac-à-dos 0-1 non-décomposable, que nous avons modelisé à l'aide des GAI-Nets et résolu par les méthodes associées développées dans cette thèse.

Perspectives de recherche

L'exploitation de préférences complexes dans des domaines combinatoires est un sujet récent de recherche, où plusieurs développements restent à faire afin de marier des techniques de l'intelligence artificielle, de la recherche opérationnelle et de la théorie de la décision. Notre contribution dans cette thèse a porté essentiellement sur les points suivants :

1. la conception et l'implémentation d'un algorithme efficace pour l'énumération des k-meilleures solutions d'un espace produit selon une fonction GAI, même pour des cas où k est de grand taille ;
2. l'élaboration d'une méthode de recherche de solutions optimales selon des critères non-linéaires de compromis dans le cadre de la décision collective ou multi-critère, quand les préférences sont modélisées par une collection de fonctions GAI ;

3. l'étude de techniques d'élicitation partielle, avec le développement d'une technique adapté à l'élicitation de modèles GAI pour des tâches du type k-meilleures solutions ;
4. l'application de modèles GAI dans une situation décisionnelle concrète.

Les pistes suivantes nous semblent être des suites naturelles à développer pour chacun de ses points :

L'intégration de techniques adaptés aux contraintes valuées : quand des contraintes interdisent certaines instances du produit cartésien, nous pouvons les représenter en tant que des fonctions d'utilité si les interdépendances introduites ne rendent pas la taille des cliques trop importante. Il est important d'introduire l'utilisation des techniques développées pour des CSP valuées (Cooper *et al.*, 2007) pour traiter des cas où cette approche n'est pas possible. Dans ce cadre, il est aussi important de considérer les problèmes algorithmiques causés par le fait que ces contraintes ne sont pas nécessairement binaires (Bessière *et al.*, 2002; Bacchus *et al.*, 2002). En effet, les fonctions GAI-décomposables et les CSP valués sont des cadres formels voisins, même si les problèmes abordés sont souvent différents, les techniques de résolution d'un domaine peuvent servir dans l'autre. Travailler à approcher ces deux cadres formels semble être une voie de recherche intéressante.

Recherche de solutions de compromis pour des structures GAI disparates : quand les structures des fonctions GAI des décideurs/critères sont très différentes, l'arbre GAI agrégé utilisé pour l'énumération peut avoir des cliques très grandes, ce qui empêchera l'utilisation pratique de la méthode développée dans cette thèse. Deux approches peuvent être envisagées pour remédier le problème : toujours dans le cadre des solutions exactes, nous pouvons utiliser une autre fonction GAI plus décomposable que g (l'approximation GAI), si cette nouvelle fonction satisfait toujours la proposition de majoration (page 84). On peut aussi envisager des méthodes approchées avec garantie de performance.

Développement des méthodes d'élicitation : nous considérons que les questions auxquelles l'utilisateur est censé répondre dans des méthodes d'élicitation que nous avons utilisées dans *GVisite?* soit excessivement précises et lourdes pour l'utilisateur. Nous pensons donc que l'introduction de stratégies d'élicitation fondées sur des interactions plus simples, peuvent permettre à l'utilisateur d'exprimer ces préférences de façon plus naturelle.

Extension des applications réelles : les évolutions dans l'élicitation des préférences doivent toujours être soumises à une évaluation réelle, dans l'optique de HCI. En plus, des applications réeles permettent plus naturellement la prise en compte d'un bon nombre de points jusqu'à maintenant ignorés dans le cadre de l'utilisation de modèles GAI. Par exemple : l'évolution de préférences ; dans la décision collective, les questions de la « vie privée » ou de stratégie (est-ce que les membres du groupe ont intérêt à cacher leurs vraies préférences), et de l'évolution des groupes, la manipulation des préférences.

Applications de réseaux GAI dans la théorie des jeux : Dans un système multi-agent, chaque agent a des préférences : des désirs sur les états du monde qu'il souhaite atteindre, et ceux qu'il souhaite éviter. En général, la décision « optimale » pour un individu dépend non seulement de ses propres actions, mais aussi de ce que font les autres agents. Comme chaque agent n'est pas totalement maître de son sort, on dit qu'ils sont en interaction stratégique. La théorie des jeux est un des modèles formels plus adaptés pour l'étude des interactions stratégiques entre agents. Grosso modo, un jeu consiste en un ensemble d'agents (ou joueurs), et pour chaque agent, l'existence d'un ensemble de stratégies possibles et une fonction d'utilité associant une valeur réelle à chaque combinaison possible de stratégies, représentant ses préférences. Classiquement, les fonctions d'utilité sont décrites explicitement dans ces représentations, en énumérant toutes les valeurs pour chaque combinaison de stratégies. Le nombre de valeurs numériques à spécifier, c'est-à-dire le nombre de combinaisons de stratégies possibles, est alors exponentiel en fonction du nombre de joueurs, ce qui rend cette représentation explicite des préférences des joueurs déraisonnable lorsque le nombre de joueurs est grand. Une solution pour répondre à ces besoins consiste à utiliser une forme de représentation compacte de préférences, permettant ainsi de spécifier de façon concise et efficace les interactions entre les agents. Récemment, des CP-Nets ont été utilisés avec succès pour représenter de manière compacte des préférences exprimées comme de formules de la logique propositionnel dans les jeux (Bonzon *et al.*, 2006b,a). Nous pensons que les GAI-Nets peuvent être une bonne alternative pour la représentation compacte des préférences dans les jeux quand celles sont plus naturellement exprimées par des fonctions d'utilité. Des résultats dans ce domaine sont à développer.

Bibliographie

Adams, E. (1965). Elements of a theory of inexact measurement. *Philosophy of Science*, 32:205–228. 106

Agrawal, R., Imielinski, T. et Swami, A. N. (1993). Mining association rules between sets of items in large databases. *In* Buneman, P. et Jajodia, S., éditeurs : *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, pages 207–216, Washington, D.C. 133

Agrawal, R. et Srikant, R. (1994). Fast algorithms for mining association rules in large databases. *In Proceedings of the Twentieth International Conference on Very Large Databases*, pages 487–499, Santiago, Chile. 147

Allais, M. (1953). Le comportment de l'homme rationnel devant le risque : critique des postulats et axiomes de l'école Américaine. *Econometrica*, 21(4):503–546. 28

Arnborg, S., Corneil, D. G. et Proskurowski, A. (1987). Complexity of finding embeddings in a k-tree. *SIAM Journal on Algebraic and Discrete Methods*, 8(2):277–284. 42

Bacchus, F., Chen, X., van Beek, P. et Walsh, T. (2002). Binary vs. non-binary constraints. *Artificial Intelligence*, 140(1/2):1–37. 63, 161

Bacchus, F. et Grove, A. J. (1995). Graphical models for preference and utility. *In* Besnard, P. et Hanks, S., éditeurs : *UAI '95 : Eleventh Annual Conference on Uncertainty in Artificial Intelligence*, pages 3–10, Montreal, Quebec, Canada. Morgan Kaufmann. 3, 22, 28

Barba-Romero, S. et Pomerol, J.-C. (1997). *Decisiones Multicriterio : Fundamentos Teóricos y Utilización Práctica*. Universidad de Alcalá. 10, 11, 14, 109

Baroni, P. et Vicig, P. (2006). On computing the maximum-entropy probability consistent with a capacity. *In Proc. 11th Conf. Information Processing and Management*

of Uncertainty in Knowledge-Based Systems (IPMU'06), pages 1732–1739, Paris, France. 95

Bessière, C., Meseguer, P., Freuder, E. C. et Larrosa, J. (2002). On forward checking for non-binary constraint satisfaction. *Artificial Intelligence*, 141(1/2):205–224. 63, 161

Bonzon, E., Lagasquie-Schiex, M.-C. et Lang, J. (2006a). Compact preference representation for boolean games. *In Ninth Pacific Rim International Conference on Artificial Intelligence (PRICAI'06)*, volume 4099, pages 41–50. Springer-Verlag. 162

Bonzon, E., Lagasquie-Schiex, M.-C., Lang, J. et Zanuttini, B. (2006b). Boolean games revisited. *In 17th European Conference on Artificial Intelligence (ECAI'06)*, pages 265–269. Springer-Verlag. 162

Borgelt, C. (2003). Efficient implementations of apriori and eclat. *In* Goethals, B. et Zaki, M. J., éditeurs : *FIMI'03, Proceedings of the 1st IEEE ICDM Workshop on Frequent Itemset Mining Implementations, Melbourne, Florida, USA, November 19, 2003*, volume 90 de *CEUR Workshop Proceedings*. 147

Borgelt, C. (2004). Recursion pruning for the apriori algorithm. *In* Jr., R. J. B., Goethals, B. et Zaki, M. J., éditeurs : *FIMI'04, Proceedings of the 2nd IEEE ICDM Workshop on Frequent Itemset Mining Implementations, Brighton, UK, November 1, 2004*, volume 126 de *CEUR Workshop Proceedings*. 147

Borgelt, C. et Kruse, R. (2002). Induction of association rules : Apriori implementation. *In Compstat 2002, Proceedings of the 15th Conference on Computational Statistics, Berlin, Germany*, Heidelberg, Germany. Physica Verlag. 147

Boutilier, C. (1994). Toward a logic for qualitative decision theory. *In* Torasso, P., Doyle, J. et Sandewall, E., éditeurs : *Proceedings of the 4th International Conference on Principles of Knowledge Representation and Reasoning*, pages 75–86, Bonn, FRG. Morgan Kaufmann. 28

Boutilier, C. (2002). A POMDP formulation of preference elicitation problems. *In Proceedings of the Eighteenth National Conference on Artificial Intelligence and Fourteenth Conference on Innovative Applications of Artificial Intelligence (AAAI/IAAI-02)*, pages 239–246, Menlo Parc, CA, USA. AAAI Press. 120

Boutilier, C., Bacchus, F. et Brafman, R. I. (2001). UCP-Networks : A directed graphical representation of conditional utilities. *In* Breese, J. et Koller, D., éditeurs :

UAI-01 : Seventeenth Conference on Uncertainty in Artificial Intelligence, pages 56–64, San Francisco, California, USA. Morgan Kaufmann Publishers. 37

Boutilier, C., Brafman, R. I., Domshlak, C., Hoos, H. H. et Poole, D. (2004a). CP-nets : A tool for representing and reasoning with conditional ceteris paribus preference statements. *Journal of Artificial Intelligence Research (JAIR)*, 21:135–191. 3, 25, 32

Boutilier, C., Brafman, R. I., Domshlak, C., Hoos, H. H. et Poole, D. (2004b). Preference-based constrained optimization with CP-nets. *Computational Intelligence*, 20(2):137–157. 32

Boutilier, C., Brafman, R. I., Hoos, H. H. et Poole, D. (1999). Reasoning with conditional ceteris paribus preference statements. *In UAI*, pages 71–80. 28, 31, 32

Boutilier, C., Patrascu, R., Poupart, P. et Schuurmans, D. (2005). Regret-based utility elicitation in constraint-based decision problems. *In* Kaelbling, L. P. et Saffiotti, A., éditeurs : *IJCAI-05, Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence, Edinburgh, Scotland, UK, July 30-August 5, 2005*, pages 929–934. Professional Book Center. 120, 122, 128

Bouyssou, D., Marchant, T., Pirlot, M., Tsoukiàs, A. et Vincke, P. (2006). *Evaluation and Decision Models with Multiple Criteria : Stepping Stones for the Analyst*. Springer, New York, USA. 13

Bouyssou, D. et Pirlot, M. (2005). Conjoint measurement tools for MCDM : A brief introduction. *In* Figueira, J., Greco, S. et Ehrgott, M., éditeurs : *Multiple Criteria Decision Analysis : State of the Art Surveys*, volume 78 de *International Series in Operations Research & Management Science*, pages 73–132. Springer. 18, 22

Brafman, R., Domshlak, C. et Kogan, T. (2004). Compact value-function representations for qualitative preferences. *In Proceedings of the 20th Annual Conference on Uncertainty in Artificial Intelligence (UAI-04)*, pages 51–59, Arlington, Virginia. AUAI Press. 25, 35, 36, 50

Brafman, R. I. et Dimopoulos, Y. (2004). Extended semantics and optimization algorithms for CP-networks. *Computational Intelligence*, 20(2):218–245. 29

Brafman, R. I. et Domshlak, C. (2002). Introducing variable importance tradeoffs into CP-nets. *In* Darwiche, A. et Friedman, N., éditeurs : *UAI '02, Proceedings of the 18th Conference in Uncertainty in Artificial Intelligence, University of Alberta, Edmonton, Alberta, Canada, August 1-4, 2002*, pages 69–76. Morgan Kaufmann. 33

Brafman, R. I., Domshlak, C. et Shimony, S. E. (2006). On graphical modeling of preference and importance. *J. Artif. Intell. Res. (JAIR)*, 25:389–424. 25, 33

Braziunas, D. et Boutilier, C. (2005). Local utility elicitation in GAI models. *In UAI*, pages 42–49. AUAI Press. 110, 111, 128

Braziunas, D. et Boutilier, C. (2006). Preference elicitation and generalized additive utility (nectar paper). *In Proceedings of the Twenty-First National Conference on Artificial Intelligence (AAAI-06)*, Boston, MA. 110

Braziunas, D. et Boutilier, C. (2007). Minimax regret based elicitation of generalized additive utilities. *In Proceedings of the Twenty-third Conference on Uncertainty in Artificial Intelligence (UAI-07)*, Vancouver. 128

Breese, J. S., Heckerman, D. et Kadie, C. M. (1998). Empirical analysis of predictive algorithms for collaborative filtering. *In UAI*, pages 43–52. 124

Bretthauer, K. M. et Shetty, B. (2002). The nonlinear knapsack problem – algorithms and applications. *European Journal of Operational Research*, 138(3):459–472. 64

Cadoli, M. et Donini, F. M. (1997). A survey on knowledge compilation. *AI Commun*, 10(3-4):137–150. 25

Chajewska, U., Getoor, L., Norman, J. et Shahar, Y. (1998). Utility elicitation as a classification problem. *In* Cooper, G. F. et Moral, S., éditeurs : *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence (UAI-98)*, pages 79–88, San Francisco. Morgan Kaufmann. 118

Chajewska, U. et Koller, D. (2000). Utilities as random variables : Density estimation and structure discovery. *In* Boutilier, C. et Goldszmidt, M., éditeurs : *Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence (UAI-00)*, pages 63–71, SF, CA. Morgan Kaufmann Publishers. 121

Chajewska, U., Koller, D. et Parr, R. (2000). Making rational decisions using adaptive utility elicitation. *In Proceedings of the 7th Conference on Artificial Intelligence (AAAI-00) and of the 12th Conference on Innovative Applications of Artificial Intelligence (IAAI-00)*, pages 363–369, Menlo Park, CA. AAAI Press. 118, 128

Chambers, J. M., Cleveland, W. S., Kleiner, B. et Tukey, P. A. (1983). *Graphical methods for data analysis*. The Wadsworth statistics/probability series. Wadsworth. 89

Chateauneuf, A. et Jaffray, J.-Y. (1995). Local Möbius transforms on monotone capacities. *In European Conf. on Symbolic and Quantitative Approaches to Reasoning and Uncertainty*, volume 946 de *LNCS*, pages 115–124. Springer-Verlag. 99

Cooper, M. C., de Givry, S. et Schiex, T. (2007). Optimal soft arc consistency. *In* Veloso, M. M., éditeur : *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007*, pages 68–73. 161

Coste-Marquis, S., Lang, J., Liberatore, P. et Marquis, P. (2004). Expressive power and succinctness of propositional languages for preference representation. *In* Dubois, D., Welty, C. et Williams, M.-A., éditeurs : *Proc. of the 9th International Conference on Principles of Knowledge Representation ans Reasonning (KR'04), Whistler, BC, Canada, 02/06/04-05/06/04*, pages 203–212. AAAI Press. 25

Cowell, R. G., Dawid, A. P., Lauritzen, S. L. et Spiegelhalter, D. J. (1999). *Probabilistic Networks and Expert Systems*. Stat. for Eng. and Information Science. Springer-Verlag. 28, 39, 53

Dantzig, G. (1957). Discrete variable extremum problems. *Operations Research*, 5:226–277. 65

Darwiche, A. et Marquis, P. (2002). A knowledge compilation map. *Journal of Artificial Intelligence Research*, 17:229–264. 25

Darwiche, A. Y. (1999). Compiling knowledge into decomposable negation normal form. *In* Thomas, D., éditeur : *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI-99-Vol1)*, pages 284–289, S.F. Morgan Kaufmann Publishers. 25

de Givry, S., Heras, F., Zytnicki, M. et Larrosa, J. (2005). Existential arc consistency : Getting closer to full arc consistency in weighted CSPs. *In* Kaelbling, L. P. et Saffiotti, A., éditeurs : *IJCAI-05, Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence, Edinburgh, Scotland, UK, July 30-August 5, 2005*, pages 84–89. 53

de Givry, S., Schiex, T. et Verfaillie, G. (2006). Exploiting tree decomposition and soft local consistency in weighted CSP. *In AAAI*. AAAI Press. 62

Dechter, R. (1999). Bucket elimination : A unifying framework for reasoning. *Artificial Intelligence*, 113(1-2):41–85. 50

Domshlak, C., Rossi, F., Venable, K. B. et Walsh, T. (2003). Reasoning about soft constraints and conditional preferences : complexity results and approximation techniques. *In* Gottlob, G. et Walsh, T., éditeurs : *IJCAI-03, Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, Acapulco, Mexico, August 9-15, 2003*, pages 215–220. Morgan Kaufmann. 32

Doyle, J. (2004). Prospects for preferences. *Computational Intelligence*, 20(2):111–136. x, 103, 127

Doyle, J. et Thomason, R. H. (1999). Background to qualitative decision theory. *AI Magazine*, 20(2):55–68. 24

Dubois, D. et Prade, H. (1995). Possibility theory as a basis for qualitative decision theory. *In* Mellish, C. S., éditeur : *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pages 1924–1932, San Mateo. Morgan Kaufmann. 28

Dubois, D., Prade, H. et Sabbadin, R. (2001). Decision-theoretic foundations of qualitative possibility theory. *European Journal of Operational Research*, 128:459–478. 28

Edwards, W. et Fasolo, B. (2001). Decision technology. *Annual Review of Psychology*, 52:581–606. 24

Ellsberg, D. (1961). Risk, ambiguity, and the Savage axioms. *Quartely Journal of Economics*, 75:643–669. 28

Engel, Y. et Wellman, M. P. (2007). Generalized value decomposition and structured multiattribute auctions. *In EC'07 : Proceedings of the 8th ACM conference on Electronic commerce*, pages 227–236, New York, NY, USA. ACM. 3

Fargier, H. et Marquis, P. (2008). Extending the knowledge compilation map : Krom, horn, affine and beyond. *In* Fox, D. et Gomes, C. P., éditeurs : *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI 2008, Chicago, Illinois, USA, July 13-17, 2008*, pages 442–447. AAAI Press. 25

Fielding, R. T. (2000). *Architectural Styles and the Design of Network-based Software Architectures*. Phd thesis, University of California, Irvine, California, USA. 140

Fishburn, P. (1999). Preference structures and their numerical representations. *Theoretical Computer Science*, 217:359–383. 27

Fishburn, P. C. (1967). Interdependence and additivity in multivariate, unidimensional expected utility theory. *International Economic Review*, 8:335–342. 3, 22, 111

Fishburn, P. C. (1970). *Utility Theory for Decision Making*. Wiley, New York. 11, 105

Frankfurt, H. (1971). Freedom of the will and the concept of a person. *Journal of Philosophy*, 68:5–20. 103

Friedman, M. et Savage, L. (1948). The utility analysis of choices involving risk. *Journal of Political Economy*, 56:279–304. 109

Galand, L. et Perny, P. (2007). Search for Choquet-optimal paths under uncertainty. *In UAI'07*, pages 125–132. 94

Galand, L. et Spanjaard, O. (2007). Deux approches complémentaires pour un problème d'arbre couvrant robuste. *In 8ème Congrès de la Société Française de Recherche Opérationnelle et d'Aide à la Décision*, pages 129–137. Presses Universitaires de Grenoble. 97

Garey, M. R. et Johnson, D. S. (1979). *Computers and Intractibility : A Guide to the Theory of NP-Completeness*. Bell Telephone Laboratories, Inc., Murray Hill, NJ. 32

Garrett, J. J. (2005). Ajax : A new approach to web applications. `http://www.adaptivepath.com/ideas/essays/archives/000385.php`. Last visited 04 Feb 2008. x, 141, 143, 144

Geoghegan, M. W. et Klass, D. (2005). *Podcast solutions : the complete guide to podcasting*. Apress, Berkeley, California, USA. 138

Giang, P. H. et Shenoy, P. P. (2001). A comparison of axiomatic approaches to qualitative decision making using possibility theory. *In* Breese, J. et Koller, D., éditeurs : *Proceedings of the Seventeenth Conference on Uncertainty in Artificial Intelligence (UAI-01)*, pages 162–170, San Francisco, CA. Morgan Kaufmann Publishers. 28

Goldsmith, J., Lang, J., Truszczynski, M. et Wilson, N. (2005). The computational complexity of dominance and consistency in CP-nets. *In* Kaelbling, L. P. et Saffiotti, A., éditeurs : *IJCAI-05, Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence, Edinburgh, Scotland, UK, July 30-August 5, 2005*, pages 144–149. Professional Book Center. 32

Gonzales, C. (2003). Additive utility without restricted solvability on every component. *Journal of Mathematical Psychology*, 47:47–65. 106

Gonzales, C. et Jaffray, J.-Y. (1998). Imprecise sampling and direct decision making. *Annals of Operations Research*, 80:285–303. 106

Gonzales, C. et Perny, P. (2004). GAI networks for utility elicitation. *In* Dubois, D., Welty, C. A. et Williams, M.-A., éditeurs : *KR2004 : Principles of Knowledge Representation and Reasoning*, pages 224–233, Whistler, Canada. AAAI Press. 3, 21, 38, 40, 109

Gonzales, C. et Perny, P. (2006). GAI networks : from axiomatization to applications. *In 37th European Mathematical Psychology Group meeting*. 112, 113, 115, 118, 128, 135

Gonzales, C., Perny, P. et Queiroz, S. (2006). Preference aggregation in combinatorial domains using gai-nets. *In* Bouyssou, D., Janowitz, M., Roberts, F. et Tsoukias, A., éditeurs : *Joint DIMACS-LAMSADE workshop on computer science and decision theory*, volume 6, pages 165–179, Paris, France. 77

Gonzales, C., Perny, P. et Queiroz, S. (2007). Réseaux GAI pour la prise de décision. *Revue d'intelligence artificielle*, 21(4):555–587. 38, 47, 101, 112, 113, 115

Gonzales, C., Perny, P. et Queiroz, S. (2008). GAI-networks : Optimization, ranking and collective choice in combinatorial domains. *Foundations of computing and decision sciences*, 32(4):3–24. 47, 77

Grabisch, M. (1995). On equivalence classes of fuzzy connectives : the case of fuzzy integrals. *IEEE Trans. Fuzzy Syst.*, 3(1):96–109. 90

Grabisch, M. (1996). The application of fuzzy integrals in multicriteria decision making. *European Journal of Operational Research*, 89:445–456. 93

Grabisch, M. et Perny, P. (2003). Agrégation multicritère. *In* Bouchon-Meunier, B. et Marsala, C., éditeurs : *Logique floue, principes, aide à la décision*, chapitre 3, pages 81–120. ix, 17

Greco, S., Matarazzo, B. et Słowiński, R. (1999). Rough approximation of a preference relation by dominance relations. *European Journal of Operational Research*, 117:63–83. 25

Hájek, P., Havel, I. et Chyth, M. (1966). The GUHA method of automatic hypotheses determination. *Computing*, 1(4):293–308. 133

Hájek, P. et Havránek, T. (1977). On generation of inductive hypotheses. *International Journal of Man-Machine Studies*, 9(4):415–438. 133

Herlocker, L., J., Konstan, A., J., Borchers, A. et Riedl, J. (1999). An algorithmic framework for performing collaborative filtering. *In 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, Theoretical Models, pages 230–237, Berkeley, California, USA. 2

Herlocker, J. L. (2000). *Understanding and Improving Automated Collaborative Filtering Systems*. Ph.d. thesis, Computer Science Dept., University of Minnesota, USA. 121

Hewett, T., T., Baecker, R., Card, S., Carey, T., Gasen, J., Mantei, M., Perlman, G., Strong, G. et Verplank, W. (1992). ACM SIGCHI curricula for human-computer interaction. 128

Jaffray, J.-Y. (1995). On the maximum-entropy probability which is consistent with a convex capacity. *Int. J. of Uncertainty, Fuzziness and Knowledge-Based Systems*, 3:27–33. 94

Jensen, F., Jensen, F. V. et Dittmer, S. L. (1994). From influence diagrams to junction trees. *In* de Mántaras, R. L. et Poole, D., éditeurs : *UAI '94 : Proceedings of the Tenth Annual Conference on Uncertainty in Artificial Intelligence, July 29-31, 1994, Seattle, Washington, USA*, pages 367–373. Morgan Kaufmann. 41

Jensen, F. V. (1996). *An introduction to Bayesian Networks*. Taylor and Francis, London, UK. 3, 38, 53

Jensen, F. V. et Jensen, F. (1994). Optimal junction trees. *In* de Mántaras, R. L. et Poole, D., éditeurs : *UAI '94 : Proceedings of the Tenth Annual Conference on Uncertainty in Artificial Intelligence, July 29-31, 1994, Seattle, Washington, USA*, pages 360–366. Morgan Kaufmann. 41

Kahneman, D. et Tversky, A. (1979). Prospect theory : An analysis of decision under risk. *Econometrica*, 47(2):263–291. 28

Karim, S. et Heinze, C. (2005). Experiences with the design and implementation of an agent-based autonomous UAV controller. *In AAMAS'05 : Fourth international joint conference on Autonomous agents and multiagent systems*, pages 19–26, New York, NY, USA. ACM Press. 1

Kask, K., Dechter, R., Larrosa, J. et Dechter, A. (2005). Unifying tree decompositions for reasoning in graphical models. *Artificial Intelligence*, 166(1-2):165–193. 50

Keeney, R. L. et Raiffa, H. (1993). *Decisions with Multiple Objectives : Preferences and Value Tradeoffs*. Cambridge University Press, Cambridge, UK. 1, 64

Kim, J. et Kim, S.-K. (2006). A CHIM-based interactive Tchebycheff procedure for multiple objective decision making. *Comput. Oper. Res.*, 33(6):1557–1574. 90

Kjærulff, U. et A/s, J. D. (1990). Triangulation of graphs - algorithms giving small total state space. Rapport technique. 42

Krantz, D., Luce, R. D., Suppes, P. et Tversky, A. (1971). *Foundations of Measurement (Additive and Polynomial Representations)*, volume 1. Academic Press. 12, 105, 106, 108

Larrosa, J., Morancho, E. et Niso, D. (2005). On the practical use of variable elimination in constraint optimization problems : 'still-life' as a case study. *JAIR*, 23:421–440. 62

Larrosa, J. et Schiex, T. (2003). In the quest of the best form of local consistency for weighted CSP. *In* Gottlob, G. et Walsh, T., éditeurs : *IJCAI-03, Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, Acapulco, Mexico, August 9-15, 2003*, pages 239–244. Morgan Kaufmann. 53

Lehmann, D. (1996). Generalized qualitative probability : Savage revisited. *In* Horvitz, E. et Jensen, F., éditeurs : *Proceedings of the 12th Conference on Uncertainty in Artificial Intelligence (UAI-96)*, pages 381–388, San Francisco. Morgan Kaufmann Publishers. 28

Levenshtein, V. I. (1966). Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics Doklady*, 10(8):707–710. Doklady Akademii Nauk SSSR, 163(4) :845–848, 1965. 146

Luce, R. D. (1956). Semiorders and a theory of utility discrimination. *Econometrica*, 24(2):178–191. 10

Markowitz, H. (1952). The utility of wealth. *Journal of Political Economy*, 60(2):151–158. 109

Marquis, P. (1995). Knowledge compilation using theory prime implicates. *In* Mellish, C. S., éditeur : *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pages 837–845, San Mateo. Morgan Kaufmann. 25

McGeachie, M. et Doyle, J. (2004). Utility functions for ceteris paribus preferences. *Computational Intelligence*, 20(2):158–217. 44

Merrill, D. (2006). Mashups : The new breed of web app. `http://www.ibm.com/developerworks/library/x-mashups.html`. Updated 16 Oct 2006, last visited 31 Jan 2008. 139

Mohammed, S. et Ringseis, E. (2001). Cognitive diversity and consensus in group decision making : the role of inputs, processes, and outcomes. *Organizational Behavior and Human Decision Processes*, 85(2):310–335. 81

Nakamura, Y. (2002). Additive utilities on densely ordered sets. *Journal of Mathematical Psychology*, 46(5):515–530. 105

Nguyen, H. et Haddawy, P. (1999). The decision-theoretic interactive video advisor. *In* Laskey, K. B. et Prade, H., éditeurs : *UAI '99 : Fifteenth Conference on Uncertainty in Artificial Intelligence*, pages 494–501, Stockholm, Sweden. Morgan Kaufmann. 2

Nilsson, D. (1998). An efficient algorithm for finding the M most probable configurations in probabilistic expert systems. *Statistics and Computing*, 8(2):159–173. 55, 57

Papadimitriou, C. H. (1994). *Computational Complexity*. Addison-Wesley. 32

Perny, P., Spanjaard, O. et Storme, L.-X. (2006). A decision-theoretic approach to robust optimization in multivalued graphs. *Ann Oper Res*, 147:317–341. 98

Pini, M. S., Rossi, F., Venable, K. B. et Walsh, T. (2005). Aggregating partially ordered preferences : impossibility and possibility results. *In* van der Meyden, R., éditeur : *Proceedings of the 10th Conference on Theoretical Aspects of Rationality and Knowledge (TARK-2005), Singapore, June 10-12, 2005*, pages 193–206. National University of Singapore. 44

Pisinger, D. (1995). *Algorithms for Knapsack Problems*. Ph.D. thesis, Dept. of Computer Science, University of Copenhagen, Denmark. 72, 73

Queiroz, S. (2007a). Adaptive preference elicitation for top-K recommendation tasks using GAI-networks. *In* Devedzic, V., éditeur : *Artificial Intelligence and Applications*, pages 624–629. IASTED/ACTA Press. 101

Queiroz, S. (2007b). Multiperson choquet-compromise search on large combinatorial domains. *In 2nd IEEE International Workshop on Soft Computing Applications SOFA*, pages 187–192, Gyula–Hungary, Oradea–Romania. IEEE Computational Intelligence Society. 77

Queiroz, S., Gonzales, C. et Perny, P. (2007). Décision collective avec des réseaux gai. *In Congrès de la Société Française de Recherche Opérationnelle et d'Aide à la*

Décision FRANCORO V / ROADEF 2007, pages 217–227, Grenoble, France. Presses Universitaires de Grenoble. 77

Ricci, F. et del Missier, F. (2004). Supporting travel decision making through personalized recommendation. *In* Karat, C.-M., Blom, J. O. et Karat, J., éditeurs : *Designing Personalized User Experiences for eCommerce*, Human-Computer Interaction Series, pages 231–252. Kluwer Academic Publishers. 4

Ricci, F., Venturini, A., Cavada, D., Mirzadeh, N., Blaas, D. et Nones, M. (2003). Product recommendation with interactive query management and twofold similarity. *In* Ashley, K. D. et Bridge, D. G., éditeurs : *Case-Based Reasoning Research and Development, 5th International Conference on Case-Based Reasoning, ICCBR*, volume 2689 de *Lecture Notes in Computer Science*, pages 479–493, Trondheim, Norway. Springer. 2

Rish, I. et Dechter, R. (2000). Resolution versus search : Two strategies for SAT. *Journal of Automated Reasoning*, 24(1/2):225–275. 54

Rossi, F., Venable, K. B. et Walsh, T. (2004). mCP nets : Representing and reasoning with preferences of multiple agents. *In* McGuinness, D. L. et Ferguson, G., éditeurs : *AAAI*, pages 729–734, San Jose, California, USA. AAAI Press / The MIT Press. 37, 38

Roy, B. et Bouyssou, D. (1993). *Aide Multicritère à la Décision : Méthodes et Cas*. Economica, Paris, France. 26

Russell, S. et Norvig, P. (2003). *Artificial Intelligence : A Modern Approach*. Prentice Hall, Englewood Cliffs, New Jersey, 2 édition. 86

Sabbadin, R. (1998). *Une Approche Ordinale de la Décision dans l'Incertain : Axiomatisation, Représentation Logique et Application à la Décision Séquentielle*. Thèse de doctorat, Université Paul Sabatier, Toulouse, France. 28

Savage, L. J. (1972). *The Foundations of Statistics*. Dover, New York, USA. 27, 109

Schiex, T., Fargier, H. et Verfaillie, G. (1997). Problèmes de satisfaction de contraintes valués. *Revue d'Intelligence Artificielle*, 11(3):339–373. 59

Selman, B. et Kautz, H. (1996). Knowledge compilation and theory approximation. *Journal of the ACM*, 43(2):193–224. 25

Sen, A. (1997). *On Economic Inequality*. Clarendon Press. 98

Shapley, L. (1971). Cores of convex games. *International Journal of Games Theory*, 1:11–22. 94, 95

Shibata, Y. (1988). On the tree representation of chordal graphs. *Journal of Graph Theory*, 12(3):421–428. 41

Slovic, P. (1975). Choice between equally-valued alternatives. *Journal of Experimental Psychology : Human Perception Performance*, 1(3):280–287. 25

Słowiński, R., éditeur (1998). *Fuzzy sets in decision analysis, operations research and statistics*. Kluwer Academic Publishers, Norwell, MA, USA. 157

Steuer, R. et Choo, E.-U. (1983). An interactive weighted Tchebycheff procedure for multiple objective programming. *Math. Prog.*, 26:326–344. 82, 90

Steuer, R. E., Silverman, J. et Whisman, A. W. (1993). A combined Tchebycheff/aspiration criterion vector interactive multiobjective programming procedure. *Manage. Sci.*, 39(10):1255–1260. 90

Terrioux, C. et Jégou, P. (2003). Bounded backtracking for the valued constraint satisfaction problems. *In* Rossi, F., éditeur : *Principles and Practice of Constraint Programming - CP 2003, 9th International Conference, CP 2003, Kinsale, Ireland, September 29 - October 3, 2003, Proceedings*, volume 2833 de *Lecture Notes in Computer Science*, pages 709–723. Springer. 62

Torra, V. (1998). On some relationships between the WOWA operator and the Choquet integral. *In Proc. 7th Conf. Information Processing and Management of Uncertainty in Knowledge-Based Systems (IPMU'98)*, pages 818–824, Paris, France. 93

von Neumann, J. et Morgenstern, O. (1944). *Theory of games and economic behavior*. Princeton University Press, Princeton, New Jersey, first édition. 109

Wakker, P. P. (1988). The algebraic versus the topological approach to additive representations. *Journal of Mathematical Psychology*, 32:421–435. 106

Wakker, P. P. (1989). *Additive Representations of Preferences, A New Foundation of Decision Analysis*. Kluwer. 105

Wakker, P. P. (2001). Testing and characterizing properties of nonadditive measures through violations of the sure-thing principle. *Econometrica*, 69(4):1039–1059. 94

Weng, P. (2005). Qualitative decision making under possibilistic uncertainty : Toward more discriminating criteria. *In UAI*, pages 615–622. AUAI Press. 28

Wierzbicki, A. (1986). On the completeness and constructiveness of parametric characterizations to vector optimization problems. *OR Spektrum*, 8:73–87. 82

Wilson, N. (2004). Extending CP-nets with stronger conditional preference statements. *In* McGuinness, D. L. et Ferguson, G., éditeurs : *AAAI*, pages 735–741, San Jose, California, USA. AAAI Press / The MIT Press. 33

Yager, R. R. (1988). On ordered weighted averaging aggregation operators in multicriteria decision making. *IEEE Trans. Syst., Man, Cybern.*, 18:183–190. 90, 93

Zadeh, L. A. (1983). A computational approach to fuzzy quantifiers in natural languages. *Computing and Mathematics with Applications*, 9:149–184. 92

Zhang, N. L. et Poole, D. (1994). A simple approach to Bayesian network computations. *In Proceedings of the 10th Canadian conference on artificial intelligence, Banff, Alberta, Canada, May 1994*, pages 171–178. 50

Index

A-1 La méthode de la transformée inverse

La méthode de la transformée inverse est une méthode informatique utilisée pour générer une suite de nombres aléatoires selon une distribution donnée, à partir de l'expression de la fonction de répartition de cette distribution. Ainsi, on adresse le problème suivant :

- Soit X une variable aléatoire dont la distribution est décrite par la fonction de répartition $F(x)$; on désire obtenir une suite de réalisations de X.

Cette méthode s'appuie sur la propriété suivante :

- Soit U une variable aléatoire distribuée uniformément sur l'intervalle $[0, 1]$, et F une fonction de répartition inversible quelconque. Alors, la variable aléatoire $X = F^{-1}(U)$ est distribuée selon F.

Ainsi, nous pouvons générer des réalisations de X de la manière suivante :

1. soit u un nombre aléatoire généré selon la distribution uniforme sur l'intervalle $[0, 1]$;
2. calculer $F^{-1}(u) = x_{\text{choisi}}$
3. x_{choisi} est une réalisation de X.

Considérons maintenant la génération des réalisations de X distribuée selon une loi normale F, pourtant tronquée par par les limites l_{up} et l_{down} (comme c'est le cas de la section 4.6 du chapitre 4). Nous procédons de la manière suivante :

1. soit u un nombre aléatoire généré selon la distribution uniforme sur l'intervalle $[0, 1]$;
2. soit $f_{up} = F(l_{up})$ et $f_{down} = F(l_{down})$
3. soit $u' = f_{down} + (f_{up} - f_{down}) \times u$
4. calculer $F^{-1}(u') = x_{\text{choisi}}$
5. x_{choisi} est une réalisation de la loi normale tronquée.

Notons que la loi normale n'est pas inversible analytiquement, mais des implémentations numériques efficaces sont disponibles dans les librairies de programmation numérique*.

*Dans notre implémentation, nous avons utilisé JSci, une librairie scientifique pour le langage Java, librement disponible (sous licence LGPL) à l'adresse `http://jsci.sourceforge.net/`

Travel Web Services: Trip Search*

The Yahoo! Travel Trip Search service enables you to search for public trip plans created with the Yahoo! Travel Trip Planner through a REST-like API. With this web service you can get a list of trip plans based on specific criteria including Yahoo! ID of the user who created the plan. Only trip plans that have been defined as public can be retrieved using Yahoo! Travel Trip Search. Use the information from this search to retrieve more information about a plan with the Get Trip service.

To use the Get Trips service will need an application ID to identify your application.

REST Request

The Yahoo! Travel Trip Search request (tripSearch) follows standard HTTP GET syntax. See constructing REST queries for details.

The base URL for tripSearch is:

```
http://travel.yahooapis.com/TripService/V1.1/tripSearch?
```

In this section Request Summary describes the available request parameters; Request Parameters shows a table summarizing those parameters.

Request Summary

The Yahoo! Travel service enables you to search for trip plans created through Yahoo! Travel Trip Planner. The only required parameter is `appid`, which indicates your application ID. You can use YahooDemo as a sample appid for testing.

You can search for public trip plans containing a specific keyword with the `query` parameter. The trip plan's title, description and tags are searched:

```
http://travel.yahooapis.com/TripService/V1.1/tripSearch?appid=YahooDemo&query=paris
```

Your request may return a large number of trip plan results; use the `results` and `start` parameters to determine how many results you get back and from which result you want to start the display. By default, the response returns ten results, starting from result 1. The `results` parameter indicates how many trip plan results to return. So, for example, to return twenty results per request, use this request:

```
http://travel.yahooapis.com/TripService/V1.1/tripSearch?appid=YahooDemo&query=paris&results=20
```

Use the start parameter to start the block of results from some result number other than 1. So, for example, the previous request returned the first 20 trip plan results. To get the next 20 in the list, use the `start` parameter with the value 21:

```
http://travel.yahooapis.com/TripService/V1.1/tripSearch?appid=YahooDemo&query=paris&results=20&start=21
```

Note that the trip plans are ordered by trip ID so you can ensure that each block of trip plans are unique. The total number of trip plans available to request is in the totalResultsAvailable attribute of the ResultSet element of the response.

By default, Yahoo! Travel returns an XML document in response to this request. See Response for information on the elements contained in this document. If you use the `output` parameter

---

* Excerpts from http://developer.yahoo.com/travel/tripservice/V1.1/tripSearch.html

with the value `json`, the output is in JSON (JavaScript Object Notation) format. The names of the objects are the same as with the XML elements. If in addition to the output parameter you also use the callback parameter with the name of a callback function, the JSON output is wrapped in that function (that is, `my_function(`*`json_text`*`)`).

Request Parameters

Parameter	Value	Description
appid	string (required)	An identifier for your application.
query	string	Search keyword(s). Public trip plans that contain the given text in the title, description or tags are returned.
results	integer	The number of trip plans to return. The default value is 10; the maximum allowed value is 50.
start	integer	The result from which to start the block of trip plan results. By default the results start from 1.
output	string	The output format. The default is XML. If `output=json`, the results are returned in JSON format. If `output=php`, the results will be returned in Serialized PHP format.
callback	string	The name of the callback function to wrap around the JSON data. The following characters are allowed: A-Z a-z 0-9 . [] and _. If `output=json` has not been requested, this parameter is ignored.

Response

The Yahoo! Travel Trip Search REST response conforms to XML 1.0. The schema document for this response is located at
http://travel.yahooapis.com/TripService/V1.1/TripSearchResponse.xsd

Top-Level Elements

Element	Description
xml	The Yahoo! Travel Trip Search conforms to XML 1.0. No child elements.
ResultSet	Parent element for the all results (element). Child element: tripplan Attributes: • totalResultsAvailable: The total number of results found from the query (which may be larger than the number of results in this response. • totalResultsReturned: The number of results in this response. This value is determined by the results parameter. • firstResultPosition: The position of the first result in the total number of responses, for response pagination. This value is determined by the start parameter
Result	Parent element for individual trip plan results (element). Child elements: Author, Title, Summary, Destinations, CreateDate, UpdateDate, Duration, Image, Geocode, Url Attribute: id: The ID of the trip plan.

Result Elements

The Result element contains information about each trip plan and its contents.

Element	Description
Result	Parent element for the trip plan (element). Child elements: Author, Title, Summary, Destinations, CreateDate, UpdateDate, Duration, Image, Geocode, Url Attribute: id: The ID of the trip plan.
Author	The Yahoo! user who created this trip plan (string).
Title	The title of the trip plan, for example "Paris Vacation in June" (string).
Summary	A summary of the items available in the trip plan (string).
Destinations	One or more destinations for the trip plan, separated by commas (string).
CreateDate	The date this trip plan was created, in Unix timestamp format (integer).
UpdateDate	The date this trip plan was most recently updated, in Unix timestamp format (integer).
Duration	The number of days this trip plan covers (integer).
Image	The image used to identify this trip plan. See Image Elements for element descriptions (element). Child elements: Url, Width, Height
Geocode	The location for the destination for this trip plan. See Geocode Elements for element descriptions (element).
Url	The URL of the trip plan on Yahoo! Travel Trip Search (string).

Image Elements

The image element describes the image or icon used to identify the trip plan. The image element can be contained inside the Result element.

Element	Description
Image	The parent element for the image (element). Child elements: Url, Width, Height
Url	The URL of the image (string).
Width	The width of the image, in pixels (integer).
Height	The height of the image, in pixels (integer).

Geocode Elements

The geocode element describes the location of a destination or an item.

Element	Description
Geocode	The parent element for the location (element). Child elements: Latitude, Longitude Attribute: precision: the precision of the geocoder, or "not available".
Latitude	The latitude of the location (float).
Longitude	The longitude of the location (float).

Examples

To get a list of trip plans containing the word "mardi gras":

```
http://travel.yahooapis.com/TripService/V1.1/tripSearch?appid=YahooDemo&query=mardi%20gras
```

To get a list of trip plans containing the word "mardi gras", and get JSON output wrapped in a callback function called `get_plan`:

```
http://travel.yahooapis.com/TripService/V1.1/tripSearch?appid=YahooDemo&query=mardi%20gras&output=json&callback=get_plan
```

A sample XML response is shown below:

```
<?xml version="1.0" encoding="utf-8" ?>
      <ResultSet firstResultPosition="1" totalResultsAvailable="218"
totalResultsReturned="2" xmlns="urn:yahoo:travel"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="urn:yahoo:travel
http://travel.yahooapis.com/TripService/V1.1/TripSearchResponse.xsd">

            <Result id="397415">
                  <Author>YDNgeek</Author>
                  <Title>Paris, France</Title>
                  <Summary>Things to do: Musee du Louvre, Cathedrale
                  </Summary>
                  <Destinations>Paris</Destinations>
                  <CreateDate>1134572151</CreateDate>
                  <Duration>7</Duration>
                  <Image>
                        <Url>http://us.i1.yimg.com/us.yimg.com/i/travel/tg/
                        lp/a9/100x100_a963e6ad7aa4caee31c99370f7a60aaa.jpg
                        </Url>
                        <Height>65</Height>
                        <Width>100</Width>
                  </Image>

            <Url>http://travel.yahoo.com/trip/?pid=1234&action=view</Url>
            </Result>
            <Result id="398901">
                  <Author />
                  <Title>Paris</Title>
                  <Summary>Things to do: Cathedrale Notre-Dame de Par...
                  Hotel: Le Meridien Etoile</Summary>
                  <Destinations>Paris</Destinations>
                  <CreateDate>1134617729</CreateDate>
                  <Duration>7</Duration>
                  <Image>
                        <Url>http://us.i1.yimg.com/us.yimg.com/i/travel/tg/
                        lp/93/100x100_93021ee4843ab4aa53b50b56dcecbfd.jpg<
                        /Url>
                        <Height>100</Height>
                        <Width>67</Width>
                  </Image>

            <Url>http://travel.yahoo.com/trip/?pid=4567&action=view</Url>
            </Result>
      </ResultSet>
```

Travel Web Services: Get Trip*

The Yahoo! Travel Get Trip service enables you to get detailed information about a trip created with the Yahoo! Travel Trip Planner through a REST-like API. With this web service you can get information about a public trip plan including its description, destinations and all the items associated with that trip.

REST Request

The Yahoo! Travel Get Trip request (getTrip) follows standard HTTP GET syntax.

The base URL for getTrip is:

```
http://travel.yahooapis.com/TripService/V1.1/getTrip?
```

In this section Request Summary describes the available request parameters; Request Parameters shows a table summarizing those parameters.

Request Summary

For the getTrip REST request there are four parameters:

- `appid` for the application ID.
- `id` for the ID of the trip plan to be retrieved.
- `output` for the output type
- `callback` to specify a callback function

The `appid` is your application ID -- an identifier for your application -- and is a required parameter. You can use YahooDemo as a sample appid for testing.

The `id` is the ID of the trip plan, created in Yahoo! Travel Trip Planner, that you want to retrieve. You can find out the ID of the trip to retrieve using the Trip Search Web Service. Note that the trip plan you are retrieving must have been defined to be a public trip plan.

By default, Yahoo! Travel returns an XML document in response to this request. See Response for information on the elements contained in this document. If you use the `output` parameter with the value `json`, the output is in JSON (JavaScript Object Notation) format. The names of the objects are the same as with the XML elements. If in addition to the output parameter you also use the `callback` parameter with the name of a callback function, the JSON output is wrapped in that function (that is, `my_function(`*`json_text`*`)`).

Request Parameters

Parameter	Value	Description
appid	string (required)	An identifier for your application.
id	integer (required)	The ID of the trip plan to retrieve. You can find out the ID of the trip you want to find using The Trip Search Web Service.
output	string	The output format. The default is XML. If `output=json`, the results are returned in JSON format. If `output=php`, the results will be returned in Serialized PHP format.
callback	string	The name of the callback function to wrap around the JSON data. The following characters are allowed: A-Z a-z 0-9 . [] and _. If

* Excerpts from http://developer.yahoo.com/travel/tripservice/V1.1/getTrip.html

	`output=json` has not been requested, this parameter is ignored.

Response

The Yahoo! Travel Get Trip REST response conforms to XML 1.0. The schema document for this response is located at http://travel.yahooapis.com/TripService/V1.1/GetTripResponse.xsd

Top-Level Elements

Element	Description
xml	The Yahoo! Travel Get Trip conforms to XML 1.0. No child elements.
Result	Parent element for the result. Child elements: Author, Title, Summary, CreateDate, UpdateDate, Duration, Image, Tag, Destination

Result Elements

The result element contains information about the trip plan and its contents.

Element	Description
Result	The parent element for the trip plan (element). Attribute: id: The trip plan ID. Child elements: Author, Title, Summary, CreateDate, UpdateDate, Duration, Image, Tag, Destination
Author	The user who created this trip plan. (string)
Title	The title of the trip plan, for example "Paris Vacation in June " (string).
Summary	The overall description of the trip plan, for example "Two weeks in Paris June 15-30 " (string).
CreateDate	The date this trip plan was created, in Unix timestamp format (integer).
UpdateDate	The date this trip plan was most recently updated, in Unix timestamp format (integer).
Duration	The number of days this trip plan covers (integer).
Image	The image used to identify this trip plan. See Image Elements for element descriptions (element). Child elements: Url, Width, Height
Tag	A tag associated with this result. If the result contains multiple tags there are multiple tag elements. (string).
Destination	The parent element for trip plan destinations and items. If the result contains multiple destinations there are multiple Destination elements. See Destination Elements for element descriptions. Child elements of Destination: Title, Geocode, AirportCode, WeatherCode, Item

Destination Elements

Destinations represent places to go on the trip. Each destination element describes a destination for the trip plan, including its name, geocode location, airport code, as well as specific items (places to stay, things to do, and so on) while at that destination). A trip plan can contain multiple Destination elements.

Element	Description
Destination	The parent element for a destination (element). Child elements: Title, Geocode, AirportCode, WeatherCode, Item
Title	The title of the destination, for example "Paris" (string).

Geocode	The location for the destination. See Geocode Elements for element descriptions (element).
AirportCode	The airport code for the destination. (string)
WeatherCode	The weather location ID code for the destination. (string). You can use this code to look up the current weather conditions and forecast for the destination on Yahoo! Weather.
Item	One or more scheduled things to do, places to stay, and so on, for the destination. See Item Elements for element descriptions. Child elements of item: Title, Description, Category, Image, Geocode, Address, Phone, Url, Note, UserRating, Tag

Item Elements

Items represent different activities and points of interest for a destination: hotels, things to do, and so on. Each item has a category, an address, URL, image, and other elements representing that item.

Element	Description
Item	An individual item for this trip plan (element). Child Elements: Title, Description, Category, Image, Geocode, Address, Phone, Url, Note, UserRating, Tag Attributes: id: The unique ID of the item
Title	The title for this item (string)
Summary	A summary of the item (string)
Category	The category of this item. Possible categories are Transportation, City, Entertainment, Flight, Hotel, Other, Restaurant, Shopping, Things to do (string).
Image	The image used to identify this item. See Image Elements for element descriptions (element). Child elements: Url, Width, Height
Geocode	The location associated with this item, if available. See Geocode Elements for element descriptions (element).
Address	The address associated with this item, if available. See Address Elements for element descriptions (element).
Phone	The phone number associated with this item, if available (string).
Url	A URL associated with this item. If there are multiple URLs for the item there will be multiple URL elements. (string)
Note	A note associated with the item (string)
UserRating	The user rating for this item, if available (integer)
Tag	A tag associated with this item. If the item contains multiple tags there are multiple tag elements. (string).

Image Elements

The image element describes the image or icon used to identify the trip plan or item. The image element can be contained inside the Result or Item elements.

Element	Description
Image	The parent element for the image (element). Child elements: Url, Width, Height
Url	The URL of the image (string).
Width	The width of the image, in pixels (integer).
Height	The height of the image, in pixels (integer).

Geocode Elements

The geocode element describes the location of a destination or an item.

Element	Description
Geocode	The parent element for the location (element). Child elements: Latitude, Longitude Attribute: precision: the precision of the geocoder, or "not available".
Latitude	The latitude of the location (float).
Longitude	The longitude of the location (float).

Address Elements

The address element describes an address for an item.

Element	Description
Address	The parent element for the address (element). Child elements: Address1, Address2, City, State, Country, PostalCode, Neighborhood
Address1	Street address (string)
Address2	Second line of street address, if available (string)
City	City name (string).
State	Two-digit state code (for US addresses) (string).
Country	Country name (string).
PostalCode	Postal or ZIP code, if available (integer).
Neighborhood	Neighborhood of the address, if available (string).

Examples

To retrieve a trip plan with the ID of 507115:

http://travel.yahooapis.com/TripService/V1.1/getTrip?appid=YahooDemo&id=507115

To retrieve a trip plan with the ID of 507115, and get JSON output (instead of XML):

http://travel.yahooapis.com/TripService/V1.1/getTrip?appid=YahooDemo&id=507115&output=json

To retrieve a trip plan with the ID of 507115, and get JSON output wrapped in a callback function named `blasmo`:

http://travel.yahooapis.com/TripService/V1.1/getTrip?appid=YahooDemo&id=507115&output=json&callback=blasmo

A sample XML response is shown below:

```
<?xml version="1.0" encoding="utf-8" ?>
<Result id="393091" xmlns="urn:yahoo:travel"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="urn:yahoo:travel
http://travel.yahooapis.com/TripsService/V1.1/GetTripResponse.xsd">
       <Author>YDNgeek</Author>
       <Title>Trip Home to Boston</Title>
       <Summary>Home again</Summary>
       <CreateDate>1134068274</CreateDate>
       <UpdateDate>1134341463</UpdateDate>
       <Duration>1</Duration>
       <Image>
           <Url>http://us.i1.yimg.com/us.yimg.com/i/travel/hg/tcy_mmh/95/75x75_
          95622245ec1f09cc85167f26cb8bcffa.jpg</Url>
          <Height>50</Height>
          <Width>75</Width>
```

```
</Image>
<Destination>
       <Title>Boston</Title>
       <Geocode precision="not available">
              <Latitude>42.358028</Latitude>
              <Longitude>-71.060417</Longitude>
       </Geocode>
       <AirportCode>bos</AirportCode>
       <WeatherCode>USMA0046</WeatherCode>
       <Item id="567416">
              <Title>The Westin Copley Place Boston</Title>
              <Summary>Historic Back Bay overlooking Copley Square in
              downtown Boston</Summary>
              <Category>Hotel</Category>
              <Image>
                     <Url>http://us.i1.yimg.com/us.yimg.com/i/travel/hg/t
                     cy_mmh/95/75x75_95622245ec1f09cc85167f26cb8bcffa.jpg
                     </Url>
                     <Height>50</Height>
                     <Width>75</Width>
              </Image>
              <Geocode precision="not available">
                     <Latitude>42.348770</Latitude>
                     <Longitude>-71.077324</Longitude>
              </Geocode>
              <Address>
                     <Address1>10 Huntington Ave</Address1>
                     <Address2 />
                     <City>Boston</City>
                     <State>MA</State>
                     <Country>United States</Country>
                     <PostalCode>02116-5707</PostalCode>
                     <Neighborhood>Back Bay</Neighborhood>
              </Address>
              <Phone>617-262-9600</Phone>
              <Url>http://travel.yahoo.com/p-hotel-322871-
              the_westin_copley_place_boston-i</Url>
              <Tag>foo</Tag>
              <Tag>bar</Tag>
              <Tag>test</Tag>
       </Item>
</Destination>
```

Vu :
Le Président
M.

Vu :
Les Examinateurs
MM.