Choix multiattribut à l'aide de réseaux GAI de forte densité

Jean-Philippe Dubus, Christophe Gonzales, Patrice Perny

LIP6 – université Paris 6
104, avenue du Président Kennedy, 75016 Paris, France.
prenom.nom@lip6.fr

Résumé Cet article se situe dans le cadre de la décision multi-attributs et traite de l'optimisation de requêtes de choix, c'est-à-dire de la détermination de l'élément préféré par un décideur parmi un ensemble combinatoire d'alternatives. Nous supposons que celles-ci sont décrites à l'aide de n-uplets de caractéristiques (les attributs). Par ailleurs, les préférences du décideur sur ces n-uplets sont supposées représentables par une fonction d'utilité GAI-décomposable. Une telle fonction exploite des indépendances entre attributs afin de décomposer l'utilité sur l'espace des alternatives en une somme de fonctions définies sur des sous-espaces de tailles réduites, permettant ainsi d'utiliser ce modèle sur de très grands ensembles d'alternatives. Elle offre un bon compromis entre pouvoir descriptif et compacité de la représentation. Les utilités GAI-décomposables sont représentables graphiquement à l'aide de « réseaux GAI ». Lorsqu'elle n'est pas trop dense, cette structure graphique peut être exploitée afin d'obtenir des algorithmes efficaces pour résoudre les problèmes de choix ou de rangement (détermination des M éléments préférés). Toutefois, lorsque les réseaux GAI sont denses, c'est-à-dire lorsqu'il existe de nombreuses dépendances entre attributs, ces algorithmes deviennent inutilisables car leur complexité croît rapidement en fonction de ces dépendances. Nous proposons dans cet article un nouvel algorithme d'inférence capable de résoudre de manière exacte des problèmes de choix sur de tels réseaux GAI. Afin d'en montrer l'intérêt pratique, nous fournissons des résultats d'expérimentations comparant les performances de notre algorithme par rapport aux algorithmes d'inférence classiques.

Mots-Clefs. Décision multiattribut ; Modèles graphiques ; Réseaux GAI ; Optimisation ; Fonctions d'utilité ; Problèmes de choix.

1 Introduction

La théorie de la décision s'attache à modéliser les préférences d'un individu (le *décideur*) afin de pouvoir l'aider à prendre les « meilleures » décisions possibles dans des contextes où la prise de décision est complexe, par exemple lorsque celle-ci intègre la prise en compte de nombreux critères contradictoires. Longtemps cantonnés à des problèmes relevant de l'économétrie [14,20,8], les modèles décisionnels ont trouvé, depuis quelques années, de nombreux débouchés chez le *grand public*, notamment avec les systèmes interactifs d'aide à la décision et les systèmes de recommandation pour le web, ces applications nécessitant de disposer de modèles permettant de manipuler les préférences des utilisateurs afin de fournir des recommandations fondées sur leurs préférences. Ces nouvelles applications ont, elles-mêmes, mis en évidence la nécessité de développer des systèmes de représentation compacte des préférences

réalisant un bon compromis entre deux aspects contradictoires : i) le besoin de modèles suffisamment riches et flexibles pour décrire des comportements de décision sophistiqués ; et ii) la nécessité pratique de maintenir l'effort d'élicitation des préférences à un niveau admissible et aussi le besoin de procédures efficaces pour résoudre des problèmes d'optimisation fondés sur des préférences. Les représentations compactes des préférences ont donc tout naturellement suscité l'intérêt des chercheurs en théorie de la décision, tant en ce qui concerne les modèles qualitatifs de préférences que des modèles quantitatifs. Les premiers ont donné lieu à des modèles graphiques (compacts) tels que les CP-nets (*Conditional Preference networks*) [3,4] ou les TCP-nets [5], et les seconds ont donné lieu à des modèles tels que les UCP-nets [2], les CUI-nets [12] et les réseaux GAI (*Generalized Additive Independence networks*) [15,6].

Les représentations compactes qualitatives telles que les CP-nets ont l'avantage de ne requérir qu'un nombre limité de questions à poser au décideur afin d'éliciter ses préférences. Elles sont donc particulièrement adaptées à des applications telles que des systèmes de recommandation. En revanche, elles ne permettent pas de décrire très finement les préférences et, dès lors qu'une description fine est nécessaire (par exemple dans des applications stratégiques pour l'entreprise), il est préférable d'utiliser des modèles quantitatifs compacts s'appuyant sur des fonctions d'utilité [2], comme par exemple des réseaux GAI. Ce type de modèle nécessite une élicitation des préférences plus approfondie qu'un CP-net, et donc un temps d'élicitation plus long, mais en contrepartie il offre un cadre plus efficace pour manipuler les préférences du décideur et pour l'aider dans sa prise de décision. Dans la suite de cet article, nous nous placerons dans le cadre des modèles quantitatifs, et plus précisément dans un cadre où les préférences du décideur sont représentées par des fonctions d'utilité GAI-décomposables.

Dans ce contexte, les alternatives sur lesquelles portent les préférences sont décrites par un n-uplet de caractéristiques appelées *attributs*. L'espace induit est donc combinatoire et sa taille explose avec le nombre d'attributs. En pratique, on ne peut donc ni décrire en extension les préférences sur cet espace, ni parcourir ses éléments afin d'éliciter les préférences ou d'en déterminer l'élément optimal. Afin de limiter cette explosion combinatoire, une idée répandue consiste à considérer tous les attributs indépendants, ce qui induit une représentation *additive* des préférences, à la fois compacte et efficace [13,22,1]. Malheureusement, il est rare qu'elle décrive finement les préférences car il existe souvent des dépendances entre attributs. On peut toutefois envisager des généralisations des utilités additives qui limitent l'explosion combinatoire tout en ayant un pouvoir descriptif accru, par exemple les utilités multilinéaires [20], et les utilités GAI-décomposables. Ici, nous n'étudierons que ces dernières. Le modèle GAI exploite seulement les indépendances existant entre attributs pour décomposer la fonction d'utilité sur l'espace des alternatives en une somme de sous-utilités sur des sous-espaces de tailles réduites. Ainsi, il permet de décrire des préférences sur des ensembles d'alternatives de grandes dimensions tout en utilisant une taille de stockage raisonnable. Le modèle graphique sous-jacent, le *réseau GAI*, permet, lui, une élicitation efficace des préférences [15,6,16]. La résolution des problèmes de choix avec des réseaux GAI est habituellement réalisée par des algorithmes similaires à l'inférence dans les réseaux bayésiens [27,23,11,16]. Leur complexité est égale à la somme des tailles des sous-espaces sur lesquels portent les sous-utilités. Dès lors que ceux-ci sont de petites tailles, ils sont très performants. En revanche, lorsque certaines sous-fonctions d'utilité contiennent beaucoup d'attributs, les performances de ces algorithmes d'inférence s'effondrent du fait de l'explosion combinatoire ainsi engendrée. L'objet de cet article est donc de proposer un nouvel algorithme permettant de traiter ce type de cas.

Notons que l'inférence dans les réseaux GAI est un problème NP-difficile [7]. Toutefois, l'algorithme que nous présentons dans cet article se révèle en pratique bien plus rapide que les algorithmes classiques et permet de résoudre en un temps raisonnable des problèmes de choix que l'on ne pouvait envisager de résoudre auparavant. En particulier, il permet de résoudre efficacement des problèmes de décision collective dans lesquels chaque individu a des préférences GAI-décomposables et dont le critère à optimiser est non linéaire (min-max regret, max de la norme de Tchebycheff pondérée, etc) [17]. L'idée consiste à approcher par excès la « vraie » fonction d'utilité du décideur à l'aide d'une somme de sous-utilités définies sur des sous-espaces d'attributs de tailles bornées. Ce faisant, cette somme ne représente plus exactement les préférences du décideur. Cela étant dit, nous montrerons que l'on peut résoudre *de manière exacte* le problème de choix d'origine en résolvant un problème de rangement avec cette nouvelle fonction. Les expérimentations montrent que le nouvel algorithme ainsi produit est, en pratique, exponentiellement plus rapide que les algorithmes classiques.

Le plan de l'article est le suivant : nous rappelons dans la section 2 les bases de la décomposition GAI et du modèle graphique associé : le réseau GAI. Nous y décrivons en particulier l'algorithme classique d'inférence habituellement utilisé pour résoudre les problèmes de choix avec réseaux GAI. Dans la section 3, nous proposons notre nouvel algorithme d'inférence et nous montrons, dans la section 4, son efficacité en pratique. Enfin, dans la section 5, nous discutons des limites de cette nouvelle approche ainsi que de quelques extensions possibles.

2 Le modèle GAI

Dans l'article, nous noterons $\mathcal{X}$ l'ensemble des alternatives sur lesquelles portent les préférences du décideur et nous supposerons que chaque objet x de $\mathcal{X}$ peut être décrit par un n-uplet d'attributs $(x_1, \ldots, x_n)$, c'est-à-dire $\mathcal{X} = \prod_{i=1}^{n} X_i$, où X_i représente le $i$ème attribut. Ainsi, si $\mathcal{X}$ correspond à l'ensemble des menus possibles d'un restaurant, les attributs pourront correspondre aux entrées, plats, desserts et vins possibles (voir l'exemple 1 ci-dessous). Nous supposerons en outre que la relation de préférence du décideur $\succsim$ sur $\mathcal{X}$ est un préordre large total (condition nécessaire à ce que $\succsim$ soit représentable par une utilité). Ainsi, $\forall x, y \in \mathcal{X},\ x \succsim y$ signifie que le décideur préfère x à y, ou bien qu'il est indifférent entre x et y. Classiquement, on définit $\succ$ et $\sim$ respectivement comme les parties asymétrique et symétrique de $\succsim$. Dans la suite, il sera courant de projeter un élément de $\mathcal{X}$ sur un sous-ensemble de ses attributs. Pour cela, nous utiliserons la notation suivante : étant donné un sous-ensemble Y des indices des attributs de $\mathcal{X}$, autrement dit $Y \subseteq \{1, \ldots, n\}$, x_Y correspondra à la projection de $x \in \mathcal{X}$ sur les attributs indicés par Y. Ainsi, si $Y = \{i_1, \ldots, i_m\}$, alors $x_Y = (x_{i_1}, \ldots, x_{i_m})$. Lorsque Y est un singleton, $Y = \{i\}$, par abus de notation, on notera $x_i = x_Y = x_{\{i\}}$.

2.1 Décomposition et réseau GAI

Sous certaines hypothèses [10], la relation de préférence $\succsim$ est représentable par une fonction d'utilité, c'est-à-dire une fonction $u : \mathcal{X} \to \mathbb{R}$ telle que $\forall x, y \in \mathcal{X},\ \ x \succsim y \iff u(x) \geq u(y)$. Cette représentation quantitative des préférences est très populaire car, une fois l'utilité u élicitée (déterminée), on peut l'utiliser dans un solveur pour déterminer, par exemple, l'élément préféré du décideur (problème de choix). Toutefois, cette formulation est trop générale pour être utilisable en pratique. Par exemple, si $\mathcal{X}$ est décrit à l'aide de n attributs, chacun

ayant au moins 2 valeurs, $|\mathcal{X}| \geq 2^n$. On devra donc poser des questions au décideur pour déterminer plus de 2^n valeurs de la fonction u, ce qui est irréaliste dès lors que n est grand. Or, en pratique, il est assez rare que les attributs soient tous dépendants les uns des autres. En se fondant sur des hypothèses d'indépendance entre attributs (ou groupes d'attributs), on peut décomposer l'utilité u en une somme de sous-fonctions définies sur des sous-espaces de $\mathcal{X}$ et, par conséquent, de plus petites tailles. La décomposition la plus populaire est la décomposition additive [29,20], qui décompose u comme une somme de fonctions ne dépendant que d'un seul attribut : $u(x) = \sum_{i=1}^{n} u_i(x_i)$. Elle présente l'avantage d'être une représentation très compacte des préférences et de garantir une résolution rapide du problème de choix (en $O(n \max_i\{|X_i|\})$), mais elle présuppose qu'il n'y ait aucune interaction entre les attributs, ce qui n'est pas toujours vérifié en pratique, comme nous le montre l'exemple suivant :

Exemple 1. Considérons un ensemble $\mathcal{X} = (X_1, X_2, X_3)$ de menus dans un restaurant, avec des plats de résistance $X_1 = \{\text{viande}(V), \text{poisson}(P)\}$, des boissons $X_2 = \{\text{vin rouge } (R), \text{vin blanc } (B)\}$ et des desserts $X_3 = \{\text{cake}(C), \text{sorbet}(S)\}$. Supposons que le décideur attache une forte importance à ce que le vin s'accorde avec son plat (donc du vin rouge pour la viande, et du vin blanc pour du poisson), qu'en second lieu, il préfère la viande au poisson, et que, dans une moindre mesure, pour les desserts, il préfère le cake au sorbet après avoir mangé du poisson, et le contraire après la viande (afin d'éviter que le repas soit trop lourd). Ces préférences ne sont pas irrationnelles, mais elles ne s'accordent pas avec une décomposition en utilités additives car $(V, R, C) \succ (P, R, C) \Rightarrow u_1(V) > u_1(P)$ alors que $(P, B, C) \succ (V, B, C) \Rightarrow u_1(P) > u_1(V)$. Dans ce cas, on s'aperçoit que les interactions entre les attributs X_1 et X_2 d'une part, et X_1 et X_3 d'autre part, empêchent la décomposition en utilités additives. Toutefois, le modèle pourrait être décomposé en posant : $u(x) = u_{1,2}(x_1, x_2) + u_{1,3}(x_1, x_3)$. En effet, on pourrait représenter pleinement les préférences du décideur en fixant : $u_{1,2}(V, R) = 6$; $u_{1,2}(P, B) = 4$; $u_{1,2}(V, B) = 2$; $u_{1,2}(P, R) = 0$; $u_{1,3}(V, C) = 0$; $u_{1,3}(V, S) = 1$; $u_{1,3}(P, C) = 1$; $u_{1,3}(P, S) = 0$. On pourrait objecter que cette nouvelle décomposition de u n'est pas compacte puisqu'elle nécessite 8 paramètres, comme celle de u. En réalité, ceci est dû uniquement à la faible taille de l'exemple. Supposons maintenant que chaque attribut ait m valeurs possibles, alors la nouvelle décomposition de u exigera une capacité de stockage de l'ordre de $2m^2$ au lieu de m^3 pour la représentation en extension de u. On obtient donc un gain dès lors que $m > 2$.

Un telle décomposition, avec possibilité d'interaction entre attributs, est appelée une décomposition GAI [13,1]. Elle généralise les notions d'utilités additives et multilinéaires et est, de plus, très flexible puisqu'elle ne fait aucune présupposition sur le type d'interaction liant les attributs. Cette décomposition peut être décrite plus formellement de la façon suivante :

Définition 1 (décomposition GAI). *Soit $\mathcal{X} = \prod_{i=1}^{n} X_i$ et $C_1, \ldots, C_k$ des sous-ensembles de $N = \{1, \ldots, n\}$ tels que $N = \bigcup_{i=1}^{k} C_i$. $\forall i$, on note $X_{C_i} = \prod_{j \in C_i} X_j$. L'utilité u représentant $\succsim$ est dite GAI-décomposable selon les X_{C_i} ssi il existe des fonctions $u_i : X_{C_i} \mapsto \mathbb{R}$ telles que $u(x) = \sum_{i=1}^{k} u_i(x_{C_i})$, $\forall x = (x_1, \ldots, x_n) \in \mathcal{X}$, où x_{C_i} est la projection de x sur X_{C_i}.*

Une décomposition GAI peut être représentée par une structure graphique appelée réseau GAI [15], similaire aux arbres de jonction utilisés pour les réseaux bayésiens [18,9] :

Définition 2 (Arbre de jonction). *Soit $\mathcal{Z} = \{X_1, \ldots, X_n\}$ un ensemble fini et $\mathcal{V} = \{X_{C_1}, \ldots, X_{C_k}\}$ un recouvrement de $\mathcal{Z}$. Un graphe non orienté $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ est un arbre de jonction si et seulement si c'est un arbre tel que :*

1. $\forall (X_{C_i}, X_{C_j}) \in \mathcal{E},\ X_{C_i} \cap X_{C_j} \neq \emptyset$;
2. (***Propriété d'intersection courante***) *Pour tout couple de sommets* (X_{C_i}, X_{C_j}) *tels que* $X_{C_i} \cap X_{C_j} = X_{S_{ij}} \neq \emptyset$, *il existe une chaîne les reliant dans* $\mathcal{G}$ *telle que, pour chaque noeud* X_{C_z} *de cette chaîne,* $X_{S_{ij}} \subseteq X_{C_z}$.

Les noeuds de $\mathcal{V}$ *sont appelés cliques et les arêtes* $(X_{C_i}, X_{C_j}) \in \mathcal{E}$ *sont étiquetées par* $X_{S_{ij}}$ *et sont appelées séparateurs.*

Définition 3 (Réseau GAI). *Soit* $\mathcal{X} = \prod_{i=1}^{n} X_i$ *et* $C_1, \ldots, C_k$ *des sous-ensembles de* $N = \{1, \ldots, n\}$ *tels que* $N = \bigcup_{i=1}^{k} C_i$. *Supposons que* $\succsim$ *soit représentable par une utilité GAI-décomposable vérifiant* $u(x) = \sum_{i=1}^{k} u_i(x_{C_i})$ *pour tout* $x \in \mathcal{X}$. *Alors un réseau GAI représentant* u *est un arbre de jonction dont les nœuds sont* $X_{C_1}, \ldots, X_{C_k}$.

Les cliques sont dessinées sous forme d'ellipses, et les séparateurs sous forme de rectangles. Dans cet article, nous supposerons que le réseau GAI est un arbre. Ceci n'est nullement restrictif car il est toujours possible de se ramener à un réseau de cette forme en effectuant une triangulation de graphe [15]. La définition 3 nous indique que les cliques du réseau GAI doivent être l'ensemble des variables des sous-utilités de la décomposition GAI. Par exemple si $u(A, B, C, D, E, F, G, H) = u_1(A, B, C) + u_2(A, G) + u_3(B, D) + u_4(B, F) + u_5(D, E) + u_6(D, H)$, alors les cliques seront ABC, AG, BD, BF, DE, DH (voir la figure 1). Les séparateurs peuvent être identifiés par un algorithme vérifiant la propriété d'intersection courante (voir à ce sujet la littérature sur les réseaux bayésiens, et plus particulièrement [9]). Au sein du réseau GAI, les sous-utilités sont stockées dans la clique leur correspondant.

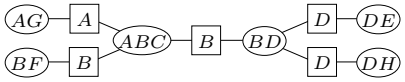

Figure 1. Un arbre GAI

2.2 Procédure de choix

Déterminer l'alternative préférée du décideur lorsque les préférences de ce dernier sont représentées par le réseau GAI de la figure 1 est équivalent à résoudre :

$$\max_{a,b,c,d,e,f,g,h} u_1(a, b, c) + u_2(a, g) + u_3(b, d) + u_4(b, f) + u_5(d, e) + u_6(d, h).$$

Ce problème peut être résolu efficacement en éliminant les variables une à une. D'après la propriété d'intersection courante, les cliques extérieures du réseau GAI, c'est-à-dire les feuilles de ce graphe, ont des variables qui ne se retrouvent dans aucune autre clique. Cela suggère de les éliminer en premier de l'équation ci-dessus, puis de recommencer avec les cliques voisines, et ce jusqu'à avoir parcouru toutes les cliques. On peut donc envisager la résolution du problème de choix par une séquence d'absorptions de cliques par leurs voisines dans le graphe (voir ci-dessous), en partant des cliques extérieures du réseau (les feuilles) et en se déplaçant vers les cliques intérieures. Lors de chaque absorption, on maximise localement la sous-utilité contenue

dans la clique absorbée, et on envoie le résultat à une clique voisine qui l'absorbe en ajoutant ce résultat à sa propre sous-utilité. Lorsqu'il ne reste plus qu'une clique, celle-ci contient alors une fonction d'utilité dont le max correspond au max de l'utilité GAI-décomposée d'origine.

Définition 4 (Absorption de X_{C_j} par X_{C_i}). *Soit $S_{ij} = C_i \cap C_j$ et $D_j = C_j \setminus S_{ij}$. Supposons que X_{C_i} et X_{C_j} contiennent respectivement les sous-utilités u_i et u_j. Soit $v_{ij}(x_{S_{ij}}) = \max_{x_{D_j} \in X_{D_j}} u_j(x_{S_{ij}}, x_{D_j})$ pour tout $x_{S_{ij}} \in X_{S_{ij}}$. On dit que la clique X_{C_i} absorbe la clique X_{C_j} lorsque la clique X_{C_i} substitue sa table d'utilité u_i par $u'_i = u_i + v_{ij}$.*

Fonction `Collecte`(clique X_{C_i})
01 **Pour toutes** cliques X_{C_j} adjacentes à X_{C_i} n'ayant pas encore été appelées **faire**
02 Appeler `Collecte` (X_{C_j}) ; X_{C_i} absorbe X_{C_j}
03 **Fin Pour**

Après avoir appelé la procédure `Collecte` sur une clique X_{C_i} quelconque, il est aisé de voir que la valeur maximale de sa sous-utilité correspond exactement à la valeur maximale de l'utilité globale. Cela nous fournit la projection sur X_{C_i} de l'instanciation optimale de notre problème. Il ne reste plus qu'à la *redistribuer* vers les cliques extérieures pour recomposer l'alternative préférée du décideur. Cette phase de propagation est identique à la phase de `Collecte`, à ceci près qu'elle propage les informations entre les cliques en sens inverse de la collecte, et en utilisant des Argmax à la place des max :

Définition 5 (Redistribution de x_{C_i} dans la clique X_{C_j}). *Soit $S_{ij} = C_i \cap C_j$ et $D_j = C_j \setminus S_{ij}$. Supposons que X_{C_j} contienne la sous-utilité u_j (obtenue à l'issue de la phase de collecte). Soit $x^*_{C_i}$ une valeur de X_{C_i}. On dit que $x^*_{C_j}$ est la redistribution de $x^*_{C_i}$ à X_{C_j} si $x^*_{C_j} = \arg\max_{x_{D_j} \in X_{D_j}} u_j(x^*_{S_{ij}}, x_{D_j})$.*

Fonction `Diffuse`(clique X_{C_i})
01 **Si** `Diffuse` (X_{C_i}) a été appelée par la clique X_{C_j} **Alors**
02 Redistribuer la valeur de l'argmax de X_{C_j} vers X_{C_i}
03 **Sinon**
04 soit $x^*_{C_i}$ l'argmax de la sous-utilité de X_{C_i}
05 **Fin Si**
06 **Pour toutes** cliques X_{C_j} adjacentes à X_{C_i} sauf, si elle existe, celle ayant appelé `Diffuse` (X_{C_i}) **Faire**
07 Appeler `Diffuse` (X_{C_j})
08 **Fin Pour**

Fonction `Choix Optimal`(réseau GAI)
01 Soit X_{C_0} une clique du réseau GAI
02 Appeler `Collecte` (X_{C_0}), puis `Diffuse` (X_{C_0})
03 Soit $x^* \leftarrow$ le n-uplet de l'argmax calculé par `Diffuse`
04 **Retourner** x^*

2.3 Procédure de rangement

Le problème de rangement à l'aide d'un réseau GAI ressemble au problème de choix, que ce soit dans sa formulation aussi bien que dans sa résolution : au lieu de chercher la meilleure

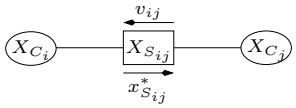

FIGURE 2. Absorption ($\leftarrow$) et Redistribution ($\rightarrow$)

instanciation des attributs (au sens de la maximisation de l'utilité globale), on cherche les M meilleures instanciations. En s'inspirant de [24], il est possible de modifier l'algorithme de choix pour résoudre ce problème. Il suffit tout d'abord de calculer l'élément optimal x^*. On peut affirmer ensuite que le 2ème meilleur élément diffère de x^* sur au moins un attribut, et donc, a fortiori, sur au moins les valeurs des attributs d'une clique. Supposons que cette clique soit X_{C_0}, alors il suffit d'appeler `Diffuse`(X_{C_0}) en empêchant l'élément $x^*_{C_0}$ d'être choisi, pour trouver le 2ème élément préféré du décideur. Sinon, si c'est une clique X_{C_i} voisine de X_{C_0}, il suffit de conserver la valeur optimale $x^*_{C_0}$ dans X_{C_0} et de choisir une nouvelle solution dans cette clique voisine. `Diffuse`(X_{C_i}) permet alors de propager cette nouvelle information vers les autres cliques et, ainsi, de déterminer le 2ème élément préféré. En fait, comme nous ne savons pas a priori quelle est la clique à partir de laquelle les attributs diffèrent de l'optimum, on appelle `Diffuse` successivement en lui passant en paramètre chacune des cliques, et le 2ème élément préféré est celui pour lequel l'appel de `Diffuse` aura renvoyé la valeur d'utilité la plus élevée. On peut recommencer de manière similaire pour trouver le 3ème élément, etc.

Il est important d'avoir en tête que la phase coûteuse des algorithmes de choix et de rangement est la phase de *Collecte*, qui effectue de nombreux calculs tensoriels de matrices. Sa complexité est celle des max effectués à l'intérieur des cliques, autrement dit, elle est en $O(k|\overline{X_C}|)$, où k est le nombre de cliques du réseau GAI et $|\overline{X_C}|$ représente la taille moyenne d'une clique, sachant que la taille d'une clique X_{C_i} est égale à $\prod_{j \in C_i} |X_j|$ (voir [24] pour un exposé détaillé de la complexité des fonctions ci-dessus). En ce qui concerne les phases de diffusions pour obtenir les M meilleurs éléments, si les utilités sont stockées sous la forme de tableaux triés par ordre d'utilité décroissante pour chaque valeur des séparateurs, on montre que l'ensemble des diffusions peut être réalisé en $O(kM\overline{\#X_S} + kM \log kM)$, où $\overline{\#X_S}$ représente le nombre moyen d'attributs dans un séparateur. $\overline{\#X_S}$ est bien inférieur à $|\overline{X_C}|$, donc calculer les M éléments préférés du décideur s'avère à peine plus coûteux que de calculer l'élément préféré. Cette remarque s'avèrera primordiale pour notre algorithme.

2.4 Du graphe markovien à l'arbre GAI

Avant de présenter notre algorithme, il est important d'étudier une structure de graphe dont nous nous servirons : celle des graphes markoviens (nous avons conservé ici le nom donné dans la communauté des réseaux bayésiens) qui entretiennent un rapport très étroit avec les réseaux GAI. Ils représentent les interactions entre attributs selon un autre point de vue.

Définition 6 (Graphe markovien). *Soit $\mathcal{X} = \prod_{i=1}^{n} X_i$ et $C_1, \ldots, C_k$ des sous-ensembles de $N = \{1, \ldots, n\}$ tels que $N = \bigcup_{i=1}^{k} C_i$. Supposons que $\succsim$ soit représentable par une utilité GAI-décomposable $u(x) = \sum_{i=1}^{k} u_i(x_{C_i})$ pour tout $x \in \mathcal{X}$. Alors un graphe markovien représentant u est un graphe non orienté $G^M = (V^M, E^M)$ tel que :*

1. $V^M = \{X_1, \ldots, X_n\}$;

2. $(X_i, X_j) \in E^M \iff \exists z \in \{1, \ldots, k\}$ *tel que* $i, j \in C_z$.

Il est aisé de construire un graphe markovien G^M à partir d'un réseau GAI : il suffit de créer un sommet par attribut, puis d'ajouter des arêtes de manière à ce que chaque clique du réseau GAI forme un sous-graphe complet du graphe markovien. La réciproque est plus complexe : pour transformer un graphe markovien en un arbre GAI, il est nécessaire de trianguler celui-ci. Un graphe G est triangulé si tout cycle de G de longueur supérieure ou égale à 4 possède une corde, c'est-à-dire deux nœuds non adjacents dans le cycle mais adjacents dans G. [25] a montré que toute triangulation de G^M revient à effectuer l'algorithme suivant :

Fonction `Triangulation`$(G^M = (V^M, E^M))$
01 Créer un graphe $G^0 = (V^M, \emptyset)$ et un graphe $G^{M_1} = G^M$
02 **Pour** i variant de 1 à n **faire**
03 Choisir un nœud X_{j_i} dans G^{M_i}
04 Créer deux graphes $G^i = (V^i, E^i) = (V^{i-1}, E^{i-1} \cup \{(X_{i_j}, X_k) \in E^{M_i}\})$ et $G^{M_{i+1}} = G^{M_i}$
05 Ajouter à G^i et $G^{M_{i+1}}$ l'ensemble d'arcs $\{(X_k, X_r) : (X_k, X_{i_j}) \in E^{M_i}$ et $(X_r, X_{i_j}) \in E^{M_i}\}$
06 Supprimer de $G^{M_{i+1}}$ à la fois X_{j_i} et ses arêtes adjacentes
07 **Fin Pour**
08 Retourner le graphe G^n

L'étape 05 fait en sorte que, dans G^i, X_{j_i} et ses voisins forment une clique. Le graphe G^n ainsi obtenu à la fin de l'algorithme est triangulé. Les étapes de 03 à 06 s'appellent une élimination de nœud et les arêtes indiquées sur la ligne 05 s'appellent des « fill-ins ». Comme on peut le constater, le graphe G^n produit par l'algorithme, et en particulier ses cliques, dépendent de la séquence de nœuds choisis à la ligne 03. Le réseau GAI résultant n'est donc pas unique. Bien que l'obtention du réseau GAI optimal (au sens où l'on minimise la taille de la plus grande clique) soit un problème NP-difficile, il existe une littérature riche sur la recherche de bonnes séquences d'élimination [26,21,28]. Nous supposerons par la suite qu'un algorithme de triangulation a été choisi au sein de notre propre algorithme de création d'arbres GAI.

3 Un algorithme efficace pour les réseaux GAI de forte densité

Nous avons vu précédemment que l'algorithme pour la résolution des problèmes de choix et de rangement possède une phase critique (la collecte) dont la complexité est liée à la taille moyenne des cliques et donc, a fortiori, à la taille de la plus grande clique. Les phases de diffusion sont, elles, beaucoup plus rapides. L'algorithme de rangement réalise la collecte une et une seule fois, puis itère sur des phases de diffusion. Notre méthode d'optimisation va s'appuyer sur cette particularité. Elle consiste à résoudre le problème d'optimisation d'une fonction GAI u en procédant à un rangement des solutions avec une fonction $\widehat{u}$ GAI-décomposable construite de manière à borner supérieurement la taille de sa plus grande clique. Il est important de noter que, bien que nous allons nous servir d'un réseau approché pour effectuer nos calculs (avec $\widehat{u}$), la solution trouvée sera bien la solution *optimale* du réseau initial (correspondant à la fonction u). Il ne s'agit donc aucunement d'un algorithme d'approximation.

3.1 De l'optimisation au rangement

Supposons que nous cherchions l'instanciation optimale d'une utilité u GAI-décomposable dont la taille de la plus grande clique rend la phase de collecte irréalisable en un temps

convenable. Notons $x^* = \arg\max_{x\in\mathcal{X}} u(x)$. Nous allons considérer une seconde utilité GAI-décomposable $\widehat{u}$ également définie sur $\mathcal{X}$ et vérifiant les deux propriétés suivantes :

1. $\forall x \in \mathcal{X},\ \widehat{u}(x) \geq u(x)$;
2. la taille de la plus grande clique du réseau GAI représentant $\widehat{u}$ permet d'effectuer une phase de collecte en un temps raisonnable.

La 2ème propriété nous garantit la possibilité de pouvoir itérer l'algorithme de rangement sur $\widehat{u}$ en un temps raisonnable. Ce faisant, imaginons que l'on ait trouvé les M meilleurs éléments $x^1, \ldots, x^M$ selon $\widehat{u}$, et supposons qu'il existe parmi ces élément un x^i tel que $u(x^i) \geq \widehat{u}(x^M)$, alors l'optimum selon u se trouve parmi ces M éléments. En effet, tous les autres éléments x de $\mathcal{X}$ sont tels que $\widehat{u}(x^M) \geq \widehat{u}(x) \geq u(x)$. Or, s'il est complexe d'utiliser le réseau GAI d'origine afin d'en déterminer son élément optimal, il est rapide, étant donné un x^i de calculer $u(x^i)$ puisqu'il suffit de sommer les $u_j(x^i_{C_j})$ dont les valeurs sont accessibles en $O(\#X_{C_j})$. La proposition suivante [17] fournit le point d'arrêt de notre procédure de rangement :

Proposition 1 (Détermination du point d'arrêt). *Soit $u : \mathcal{X} \to \mathbb{R}$ une fonction et $x^* = \arg\max_{x\in\mathcal{X}} u(x)$. Soit $\widehat{u} : \mathcal{X} \to \mathbb{R}$ une utilité GAI-décomposable vérifiant $\widehat{u}(x) \geq u(x)$, $\forall x \in \mathcal{X}$. Soit une suite (x^i) d'éléments de $\mathcal{X}$ vérifiant $\forall i,\ \forall j < i, \widehat{u}(x^i) \leq \widehat{u}(x^j)$. Enfin soit un indice s vérifiant $\widehat{u}(x^s) < \max_{i\in\{1,\ldots,s\}} u(x^i)$. Alors $x^* = \arg\max_{i\in\{1,\ldots,s\}} u(x^i)$.*

Nous avons donc un algorithme simple pour optimiser une fonction u à l'aide d'une utilité $\widehat{u}$ GAI-décomposable constituant une approximation supérieure de u. On sent intuitivement que le nombre d'itérations à effectuer avant d'atteindre le point d'arrêt est lié à la proximité entre ces deux fonctions, nous allons donc maintenant bâtir une fonction $\widehat{u}$ proche de u.

3.2 Construction de l'approximation $\widehat{u}$

Supposons que $\widehat{u}$ soit décomposable sur un ensemble de cliques $Z_1, \ldots, Z_m$, autrement dit $\widehat{u}(x) = \sum_{j=1}^{m} \widehat{u}_j(x_{Z_j})$. Afin d'utiliser notre algorithme de rangement, nous savons que $\widehat{u}$ doit être en tout point de $\mathcal{X}$ supérieure à u. De plus, nous aimerions que, pour chaque x, $\widehat{u}(x)$ soit le plus proche possible de u. On peut caractériser cette proximité comme étant la somme des écarts entre $\widehat{u}(x)$ et $u(x)$ pour tout $x \in \mathcal{X}$. Si l'on désire avoir un $\widehat{u}$ le plus proche possible de u, cela revient donc à résoudre le programme linéaire suivant, dont les variables sont les $\widehat{u}_j(x_{Z_j})$ qui représentent les valeurs prises par les sous-utilités $\widehat{u}_j$ sur les sous-espaces X_{Z_j} :

$$\begin{aligned} &\min_{\{\widehat{u}_j\}} \sum_{x\in\mathcal{X}} \left(\sum_{j=1}^{m} \widehat{u}_j(x_{Z_j}) - \sum_{i=1}^{n} u_i(x_{C_i}) \right) \\ &\text{s.c. } \sum_{j=1}^{m} \widehat{u}_j(x_{Z_j}) \geq \sum_{i=1}^{n} u_i(x_{C_i}),\ \text{pour tout } x \in \mathcal{X}. \end{aligned} \tag{1}$$

Ce programme linéaire a un inconvénient majeur : il y a autant de contraintes que d'éléments dans $\mathcal{X}$. Ce dernier étant un espace combinatoire, on ne peut, en pratique, spécifier toutes les contraintes du problème. En revanche, si l'on suppose que les Z_j sont inclus dans les C_i, alors, on peut raisonnablement approcher le problème (1) de la manière suivante : partitionnons $\{Z_1, \ldots, Z_m\}$ en un ensemble de $\mathcal{Z}_i$, $i = 1, \ldots, k$, tels que $\mathcal{Z}_i \subseteq \{Z_j : Z_j \subseteq C_i\}$. Autrement dit, tout Z_j appartient à un et un seul $\mathcal{Z}_i$ dont les éléments sont tous inclus dans C_i. Alors,

si $\forall i \in \{1,\dots,k\},\ \forall x_{C_i} \in X_{C_i},\ \sum_{j\in\mathcal{Z}_i} \widehat{u}_j(x_{Z_j}) \geq u_i(x_{C_i})$, on a a fortiori $\sum_{j=1}^{m} \widehat{u}_j(x_{Z_j}) \geq \sum_{i=1}^{n} u_i(x_{C_i})\ \forall x \in \mathcal{X}$. De même, afin d'obtenir un problème linéaire de taille raisonnable, nous approcherons la fonction objectif par $\min_{\{\widehat{u}_j\}} \sum_{i=1}^{k} \left(\sum_{x_{C_i}\in X_{C_i}} \left(\sum_{Z_j\in\mathcal{Z}_i} \widehat{u}_j(x_{Z_j}) - u_i(x_{C_i})\right)\right)$. Si l'on résout ce nouveau programme linéaire, nous obtiendrons un $\widehat{u}$ relativement proche de u. Cependant, la résolution de ce problème est encore trop chronophage. Nous allons donc une fois de plus approcher la fonction objectif en minimisant séparément chacun de ses facteurs. Il faut alors résoudre les programmes linéaires PL_i, $i = 1,\dots,k$, suivants :

$$(PL_i) \quad \begin{cases} \min\limits_{\{\widehat{u}_j\in\mathcal{Z}_i\}} \sum\limits_{x_{C_i}\in X_{C_i}} \left(\sum\limits_{Z_j\in\mathcal{Z}_i} \widehat{u}_j(x_{Z_j}) - u_i(x_{C_i})\right) \\ \text{s.c.} \sum\limits_{j\in\mathcal{Z}_i} \widehat{u}_j(x_{Z_j}) \geq u_i(x_{C_i}),\ \text{pour tout } x_{C_i} \in X_{C_i}. \end{cases} \tag{2}$$

Notons que l'on peut encore réduire la taille de ce programme linéaire en observant que si, pour un i donné, certains attributs apparaissent dans tous les X_{Z_j}, $Z_j \in \mathcal{Z}_i$, alors on peut résoudre de manière exacte le problème (2) en résolvant indépendamment, pour chaque valeur de ces attributs, le problème (2) en fixant la valeur de ces attributs. Ainsi, si $C_i = \{B, C, D\}$ et si $\mathcal{Z}_i = \{\{B, C\}, \{B, D\}\}$, alors, pour chaque valeur b de B, on peut résoudre les $\min_{\{\widehat{u}_j\}} \sum_{(c,d)\in C\times D} (\widehat{u}_{bC}(b, c) + \widehat{u}_{bD}(b, d) - u_{bCD}(b, c, d))$ et obtenir la solution exacte de (2). Il reste maintenant à identifier les Z_j, autrement dit à générer un réseau GAI « approché ».

3.3 Génération du réseau GAI « approché »

Soit G le réseau GAI d'origine, dont les cliques sont les X_{C_i}. Soit $G^M = (V^M, E^M)$ le réseau markovien correspondant, $V^M = \{X_1,\dots,X_n\}$. L'idée force pour déterminer un « bon » ensemble de Z_j consiste à trianguler G^M afin d'obtenir un nouvel arbre GAI. Ce faisant, nous risquons de créer de trop grosses cliques (et ainsi d'obtenir un $\widehat{u}$ pour lequel l'algorithme de rangement sera trop chronophage). Nous allons donc modifier les algorithmes de triangulation classiques et, en particulier, celui présenté dans la sous-section 2.4.

Si, à la $i$ème itération, ligne 02, de l'algorithme de triangulation, sous-section 2.4 (donc l'élimination du nœud X_{j_i}), on détecte que la clique ainsi formée dans G^i (ligne 05) est trop grosse, par exemple $|X_{C_i}| = \prod_{j\in C_i} |X_j| > T$, où T représente la taille de la plus grande clique admissible, alors notre algorithme de triangulation supprime des arêtes adjacentes à X_{j_i} dans G^{M_i} lors de la réalisation de la ligne 03. Ainsi, lorsque l'on crée dans G^i la clique formée par X_{j_i} et ses voisins dans G^{M_i}, on obtient une clique plus petite. Bien évidemment, chaque arête ainsi supprimée aura un impact sur l'approximation $\widehat{u}$ ainsi obtenue. En effet, supposons que l'on ait, dans le réseau GAI initial, une sous-utilité $u_i(A, B, C)$, alors si l'on supprime de G^{M_i} l'arête (B, C), on sera obligé d'approcher u_i par la somme de 2 fonctions $\widehat{u}_j(A, B)+\widehat{u}_k(A, C)$. Pour obtenir un « bon » $\widehat{u}$, il suffirait donc, à première vue, de supprimer le plus petit ensemble d'arêtes adjacentes à X_{j_i} qui produise une clique de taille inférieure à T. Malheureusement, ce n'est pas aussi simple car une arête envisagée pour la suppression peut, elle-même, être un fill-in (cf. sous-section 2.4). Dans ce cas, ne supprimer qu'elle seule produit un graphe non triangulé : soit un graphe $G^M = (V^M, E^M)$ avec $V^M = \{A, X_k, B, X_r\}$ et $E^M = \{(A, X_k), (A, X_r)(B, X_k), (B, X_r)\}$. Ce graphe contient précisément un cycle de longueur 4. Si l'on élimine A en premier, sur la ligne 05, on doit ajouter dans G^1 et G^{M_2} le fill-in

(X_k, X_r) (qui suffit pour trianguler le graphe). Si l'on continue l'algorithme en éliminant X_k et, qu'à cette étape, on décide de supprimer l'arête (X_k, X_r), alors, in fine, on obtiendra un graphe G^4 identique au G^M d'origine, qui n'était pas triangulé. La suppression de l'arête (X_k, X_r) doit donc s'accompagner d'autres suppressions d'arêtes pour garantir la triangulation du graphe obtenu. Rappelons que les fills-in (X_k, X_r) de la ligne 05 sont les arêtes joignant des voisins X_k, X_r du nœud X_{j_i} éliminé. Donc, si l'on veut supprimer l'arête (X_k, X_r) tout en garantissant la triangulation, il suffit de faire en sorte que ce ne soit pas un fill-in, autrement dit de faire en sorte que X_k et/ou X_r ne soi(en)t voisin(s) de X_{j_i} ni dans G^i ni dans G^{M_i}. Par conséquent, on devra éliminer dans G^i et G^{M_i} au moins une des deux arêtes (X_{j_i}, X_k) ou (X_{j_i}, X_r). Là encore, la ou les arêtes ainsi supprimées peuvent être des fill-ins et il faut à nouveau supprimer d'autres arêtes, et ainsi de suite. L'identification de cet ensemble d'arêtes repose sur la notion de prédécesseur :

Définition 7 (Prédecesseurs d'une arête). *Soit $G^i = (V^i, E^i)$ un graphe non orienté. Soient trois nœuds $X_k, X_r, X_z \in V^i$ tels que E^i contient les arêtes (X_k, X_r), (X_z, X_k) et (X_z, X_r). X_z est un nœud prédécesseur de (X_k, X_r) si et seulement si X_z est éliminé (dans l'algorithme de triangulation, ligne 03) avant X_k et X_r. On note $\mathrm{Pred}(X_k, X_r)$ l'ensemble des nœuds prédécesseurs de (X_k, X_r). Par ailleurs, on définit l'ensemble des arêtes prédécesseurs de (X_k, X_r) par $\mathcal{P}(X_k, X_r) = \{(X_z, X) \in E^i : X_z \in \mathrm{Pred}(X_k, X_r)$ et $X \in \{X_k, X_r\}\}$. Autrement dit, $\mathcal{P}(X_k, X_r)$ est l'ensemble des arêtes adjacentes à (X_k, X_r) et à un prédécesseur de (X_k, X_r). Enfin, Soit $\mathcal{S}(X_k, X_r) = \{S \subseteq \mathcal{P}(X_k, X_r) : \forall X_z \in \mathrm{Pred}(X_k, X_r),\ \exists X \in \{X_k, X_r\} : (X_z, X) \in S\}$. $\mathcal{S}(X_k, X_r)$ est donc l'ensemble des ensembles d'arêtes tels que le sous graphe induit par X_k, X_r et $\mathrm{Pred}(X_k, X_r)$ est connexe. Par abus de notation, $\forall S \subseteq E^i$, nous noterons $\mathcal{S}(S) = \bigcup_{(X_k, X_r) \in S} \mathcal{S}(X_k, X_r)$.*

On voit aisément que, pour garantir que le graphe obtenu à la fin de l'algorithme de triangulation soit bien triangulé, il suffit, lors de la suppression d'un fill-in (X_k, X_r) de supprimer également un ensemble (quelconque) d'arêtes S^1 appartenant à $\mathcal{S}(X_k, X_r)$. Par ailleurs, la suppression des arêtes de S^1 doit s'accompagner, pour les mêmes raisons, de la suppression d'un ensemble (quelconque) S^2 d'arêtes appartenant à $\mathcal{S}(S^1)$, et ainsi de suite. Plus généralement, on peut démontrer aisément la proposition suivante :

Proposition 2. *Soit l'algorithme de triangulation de la sous-section 2.4. Supposons qu'après la $i-1$ème itération, G^{i-1} soit triangulé et qu'à la ième itération, ligne 03, on supprime un ensemble S d'arêtes adjacentes à X_{i_j} dans G^{M_i}. Soit $G^i = (V^i, E^i)$ le graphe alors obtenu à la fin de cette ième itération. Enfin, soit, dans G^i, des ensembles $S^1 \subseteq \mathcal{S}(S), S^2 \subseteq \mathcal{S}(S^1), \ldots, S^{i-1} \subseteq \mathcal{S}(S^{i-2})$. Alors $G^{i\prime} = (V^i, E^i \backslash \bigcup_{k=1}^{i-1} S^k)$ est un graphe triangulé.*

Il est aisé de déterminer tout ensemble de S^j compatible avec la proposition 2. En effet, définissons, pour toute arête (X_k, X_r) du graphe G^i, un booléen e_{kr} tel que $e_{kr} = 1$ si l'arête (X_k, X_r) doit être supprimée, et $e_{kr} = 0$ sinon. L'ensemble des contraintes imposées par les ensembles $\mathcal{S}(S)$ se traduisent sous la forme (3) ci-dessous et le fait que $e_{kr} = 1$ pour tout $(X_k, X_r) \in S$. Toute solution d'un tel système d'inégalités produit donc un graphe triangulé.

$$e_{zk} + e_{zr} \geq e_{kr} \quad \forall (X_k, X_r) \in E^i,\ \forall X_z \in \mathrm{Pred}(X_k, X_r). \tag{3}$$

Le système ci-dessus a généralement beaucoup de solutions réalisables, qui produisent des fonctions $\widehat{u}$ plus ou moins « proches » de l'utilité u. Il convient donc, lorsque l'on choisit un ensemble d'arêtes S à supprimer, de faire en sorte que celui-ci minimise une fonction de coût

$c : 2^{\mathcal{E}} \mapsto \mathbb{R}$, où $\mathcal{E} = \{(X_k, X_r) \in \{X_1, \ldots, X_n\}^2 : X_k \neq X_r\}$, telle que $c(S)$ représente le coût d'approximation de $\widehat{u}$, c'est-à-dire le coût en terme de rapidité de calcul induit par le fait d'approcher u par $\widehat{u}$. Déterminer la fonction c dépasse l'objet de l'article, et nous décrirons seulement comment nous l'avons utilisée. Supposons que nous connaissions la fonction c. Trouver un ensemble S minimisant c tout en respectant les contraintes (3) nécessite des temps de calcul prohibitifs si c n'a pas de bonnes propriétés mathématiques. Or, la triangulation du réseau markovien doit être réalisée rapidement. C'est pourquoi nous allons approcher c par une fonction $\widehat{c}$ telle que $\widehat{c}(S) = \sum_{(X_k,X_r)\in S} \widehat{c}(\{(X_k, X_r)\})$. Autrement dit, $\widehat{c}(S)$ se décompose comme la somme des coûts induits, individuellement, par chaque arête de S.

Supposons maintenant, qu'à la $i$ème itération de l'algorithme de triangulation, on s'aperçoive que la clique engendrée par X_{j_i} (ligne 03) et ses voisins est trop grosse. On doit sélectionner un ensemble S d'arêtes adjacentes à X_{j_i} tel qu'après suppression de S, la clique obtenue ait une taille inférieure au seuil T. Notons $\{X_{i_1}, \ldots, X_{i_r}\}$ l'ensemble des voisins de X_{j_i} dans G^{M_i}. Alors la taille de la clique formée par X_{j_i} et ses voisins est égale à $|X_{i_j}| \times \prod_{k=1}^{r} |X_{i_k}|^{1-e_{j_i i_k}}$. La contrainte imposée par le seuil T est donc égale (en passant au logarithme) à :

$$\log |X_{i_j}| + \sum_{k=1}^{r} (1 - e_{j_i i_k}) \log |X_{i_k}| \leq \log T. \tag{4}$$

Pour trouver une bonne approximation $\widehat{u}$ de u, on doit donc résoudre le programme linéaire :

$$\begin{array}{l} \min \sum_{(X_k,X_r)\in E^i} \widehat{c}(X_k, X_r) e_{kr} \\ S.C. \begin{cases} \log |X_{i_j}| + \sum_{k=1}^{r} (1 - e_{j_i i_k}) \log |X_{i_k}| \leq \log T \\ e_{zk} + e_{zr} \geq e_{kr} \quad \forall (X_k, X_r) \in E^i, \ \forall X_z \in \mathrm{Pred}(X_k, X_r) \\ e_{kr} \in \{0, 1\} \ \forall (X_k, X_r) \in E^i \end{cases} \end{array} \tag{5}$$

Insistons sur le fait que les contraintes de (5) garantissent que le graphe G^i est triangulé, ce qui garantit également que les cliques de G^i forment un arbre GAI. Par conséquent, approcher la fonction de coût c par $\widehat{c}$ n'a pour incidence que de produire une fonction $\widehat{u}$ sous-optimale en ce sens qu'elle n'approche pas au mieux la fonction u. La résolution des programmes linéaires en variables booléennes (5) peut être chronophage, ce qui peut réduire le gain de notre méthode de résolution de problèmes de choix. Nous avons donc proposé un algorithme glouton pour obtenir une solution réalisable de (5) relativement proche de l'optimum. L'idée consiste à ne pas considérer comme un tout l'ensemble S d'arêtes à supprimer pour que la taille de la clique X_{C_i} soit inférieure au seuil T, mais plutôt à construire S itérativement en y ajoutant les arêtes une à une jusqu'à ce que la contrainte sur la taille de X_{C_i} soit respectée. Bien évidemment, on ajoutera en premier les arêtes induisant un coût d'approximation minimal (en tenant compte des arêtes additionnelles à supprimer pour respecter les contraintes (3)). Pour toute arête (X_k, X_r), nous allons définir ce coût global de la manière suivante :

$$C(X_k, X_r) = \begin{cases} \widehat{c}(X_k, X_r) \text{ s'il n'existe pas de contrainte } e_{zk} + e_{zr} \geq e_{kr} \\ \widehat{c}(X_k, X_r) + \sum_{X_z \in \mathrm{Pred}(X_k,X_r)} \min\{C(X_z, X_k), C(X_z, X_r)\} \text{ sinon.} \end{cases} \tag{6}$$

L'équation (6) s'interprète simplement : le coût global induit par la suppression d'une arête (X_k, X_r) est le coût induit uniquement en supprimant cette arête, $\widehat{c}(X_k, X_r)$, plus le coût induit par l'ensemble S des autres arêtes que l'on a supprimées pour respecter les contraintes (3). Étant donné que ces contraintes imposent uniquement que, pour chaque fill-in (X_k, X_r)

supprimé, une de ses arêtes adjacentes (X_z, X_k) ou (X_z, X_r) le soit également, le coût global de suppression est donc bien $\widehat{c}(X_k, X_r)+$ le coût global de suppression de l'arête adjacente $C(X_z, X_k)$ ou $C(X_z, X_r)$. Nous voulons minimiser les coûts de suppression, nous choisissons donc celle des arêtes adjacentes dont le coût global est minimum. Nos jeux d'essais utilisent ce calcul de coût. Notons qu'il se prête à la programmation dynamique et se calcule très rapidement. Notons également qu'il fournit une solution réalisable de (5), mais pas obligatoirement son optimum car des arêtes peuvent être prises en compte plusieurs fois dans le calcul de C.

La détermination des arêtes à supprimer pour garantir l'obtention d'une « bonne » fonction $\widehat{u}$ est donc simple et rapide : on effectue une triangulation « classique » mais dès que l'on s'apprête à créer une clique de trop grosse taille, on supprime des arêtes du graphe G^{M_i} en sélectionnant celles-ci grâce à l'équation (6). Une fois le graphe entièrement triangulé, nous obtenons les cliques Z_j que nous recherchions. Il ne reste plus qu'à appliquer la sous-section 3.2 pour déterminer optimalement les paramètres de $\widehat{u}$, puis à effectuer l'algorithme de rangement avec $\widehat{u}$ pour obtenir l'élément préféré par le décideur selon u.

4 Expérimentations

Afin d'évaluer notre algorithme, nous avons choisi de le comparer à l'algorithme classique de choix, mais aussi à une version adaptée aux réseaux GAI de Mini-Buckets avec Branch and Bound [19]. Ce dernier permet de résoudre efficacement de manière exacte des problèmes MPE (*most probable explanation*) dans des réseaux bayésiens à forte densité. Ces problèmes étant de nature assez semblables à celui du choix dans les réseaux GAI, Mini-Bucket semblait un bon adversaire pour notre méthode (pour les réseaux bayésiens, c'est actuellement l'algorithme le plus rapide pour résoudre MPE de manière exacte). Nous avons donc réalisé deux types d'expériences : i) des comparaisons avec l'algorithme classique, sur des graphes markoviens pouvant contenir jusqu'à 20 attributs, dont les nombres de modalités sont tirées aléatoirement entre 3 et 5 ; et ii) des comparaisons avec Mini-buckets sur des réseaux markoviens pouvant contenir jusqu'à 40 attributs, eux aussi avec entre 3 et 5 modalités. Nous avons dû limiter le premier type d'expériences à 20 attributs à cause des temps de réponse prohibitifs de l'algorithme classique (l'espace de choix pouvant, pour 40 attributs, atteindre 10^{14} éléments). Dans chaque expérience, les valeurs des sous-fonctions d'utilité ont été choisies aléatoirement entre 0 et 1000. Nous avons ainsi généré environ 20000 réseaux GAI différents et, pour chacun d'eux, résolu un problème de choix. Les résultats des différentes expériences, c'est-à-dire les performances en temps de calcul, sont représentés sur les graphiques de la figure 3 : en abscisse se trouve la somme des tailles des cliques du réseau GAI initial (attention, cette somme est très inférieure à $|\mathcal{X}|$), et en ordonnée le temps de calcul pour résoudre le problème de choix (détermination de l'argmax de u). Notons que les échelles sur les deux axes sont logarithmiques. Ces comparaisons montrent clairement qu'il existe un seuil (environ 10^6 pour l'algorithme classique et 10^7 pour les Mini-Buckets) au delà duquel duquel notre algorithme calcule plus rapidement la solution optimale que les autres algorithmes (d'une manière générale quand il y a plus d'une seconde de calcul). Il semble donc bien adapté pour résoudre les problèmes de choix dans des réseaux GAI peu décomposés. En deçà du seuil, le temps nécessaire à la détermination du réseau GAI approché est trop important et annule le gain de temps obtenu lors de la phase de rangement avec $\widehat{u}$.

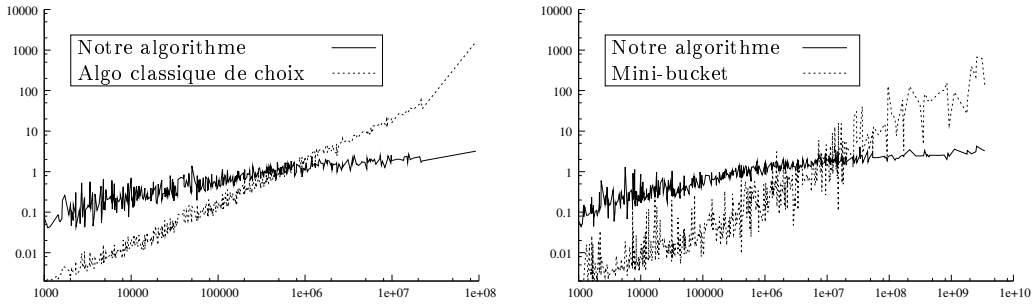

Figure 3. Comparaisons avec l'algorithme classique (gauche) et Mini-bucket Branch and Bound (droite)

5 Conclusion

Nous avons présenté un algorithme efficace de résolution de problèmes de choix avec des réseaux GAI, dans un cadre où les attributs ont de fortes interactions les uns avec les autres. On peut cependant noter que le principe d'optimisation que nous avons proposé se généralise à de nombreux autres problèmes, qu'ils soient ou non liés à des préférences GAI-décomposables. Dans le cadre GAI, par exemple, [17] a montré que l'on peut résoudre efficacement des problèmes de choix en multicritère ou en décision collective (avec des critères d'agrégation non linéaires) en utilisant des réseaux GAI de grandes tailles. Mais notre algorithme peut aussi être utilisé afin de résoudre des problèmes de satisfaction de contraintes (weighted MaxSat, etc) ou bien encore des problèmes MPE (most probable explanation) dans des réseaux bayésiens.

Les algorithmes classiques de triangulation utilisent uniquement des ajouts d'arêtes (les fill-ins). En se fondant sur ce que nous avons présenté, nous pouvons concevoir de nouveaux algorithmes autorisant à la fois l'ajout et la suppression d'arêtes pour se ramener à un graphe triangulable. Comme nous l'avons vu, cette flexibilité permet d'obtenir des algorithmes d'inférence (ici la résolution de problèmes de choix) très efficaces. Les limites actuelles de notre algorithme sont essentiellement liées au choix de l'ensemble d'arêtes à supprimer : dans l'algorithme glouton, la fonction $\widehat{c}$ doit en effet refléter un arbitrage entre la qualité d'approximation de $\widehat{u}$ par rapport à u (de manière à limiter le nombre d'itérations de la procédure de rangement avec $\widehat{u}$) et la faible densité du réseau GAI obtenu (de manière à effectuer la phase de collecte rapidement). Définir un arbitrage optimal n'est pas un problème simple et ouvre donc de nouvelles perspectives de recherche.

Références

1. F Bacchus and A Grove. Graphical models for preference and utility. In *Proc. of UAI*, 1995.
2. C Boutilier, F Bacchus, and R Brafman. UCP-networks : A directed graphical representation of conditional utilities. In *Proc. of UAI*, 2001.
3. C Boutilier, R Brafman, C Domshlak, H Hoos, and D Poole. CP-nets : A tool for representing and reasoning with conditional ceteris paribus preference statements. *Journal of Artificial Intelligence Research*, 21 :135–191, 2004.

4. C Boutilier, R Brafman, C Domshlak, H Hoos, and D Poole. Preference-based constraint optimization with CP-nets. *Computational Intelligence*, 20, 2004.
5. R Brafman and C Domshlak. Introducing variable importance tradeoffs into CP-nets. In *Proc. of UAI*, 2002.
6. D Braziunas and C Boutilier. Local utility elicitation in GAI models. In *Proc. of UAI*, 2005.
7. G F Cooper. The computational complexity of probabilistic inference using Bayesian belief networks. *Artificial Intelligence*, 42(2-3) :393–405, 1990.
8. J Corner and C W Kirkwood. Decision analysis applications in the Operations Research literature, 1970 - 1989. *Operations Research*, 39(2) :206–219, 1991.
9. R Cowell, A Dawid, S Lauritzen, and D Spiegelhalter. *Probabilistic Networks and Expert Systems*. Statistics for Engineering and Information Science. Springer, 1999.
10. G Debreu. Continuity properties of paretian utility. *International Economic Review*, 5 :285–293, 1964.
11. Rina Dechter. Bucket elimination : A unifying framework for reasoning. *Artificial Intelligence*, 113 :41–85, 1999.
12. Y Engel and M P Wellman. CUI networks : A graphical representation for conditional utility independence. In *Proc. of AAAI*, 2006.
13. P C Fishburn. *Utility Theory for Decision Making*. Wiley, 1970.
14. K Golabi, C W Kirkwood, and A Sicherman. Selecting a portfolio of solar energy projects using multiattribute preference theory. *Management Science*, 27 :174–189, 1981.
15. C Gonzales and P Perny. GAI networks for utility elicitation. In *Proc. of KR*, pages 224–234, 2004.
16. C Gonzales, P Perny, and S Queiroz. Réseaux GAI pour la prise de décision. *Revue d'Intelligence Artificielle*, 21(4) :555–587, 2007.
17. C Gonzales, P Perny, and S Queiroz. Preference aggregation with graphical utility models. In *Proc. of AAAI*, pages 1037–1042, 2008.
18. F Jensen. *An introduction to Bayesian Networks*. Taylor and Francis, 1996.
19. K Kask and R Dechter. Branch and bound with mini-bucket heuristics. In *Proc. of IJCAI*, 1999.
20. R L Keeney and H Raiffa. *Decisions with Multiple Objectives - Preferences and Value Tradeoffs*. Cambridge University Press, 1993.
21. U Kjærulff. Triangulation of graphs — algorithms giving small total state space. Technical Report R-90-09, Dept. of Maths and Computer Science, Aalborg University, 1990.
22. D Krantz, R D Luce, P Suppes, and A Tversky. *Foundations of Measurement (Additive and Polynomial Representations)*, volume 1. Academic Press, 1971.
23. A L Madsen and F V Jensen. LAZY propagation : A junction tree inference algorithm based on lazy inference. *Artificial Intelligence*, 113(1–2) :203–245, 1999.
24. D Nilsson. An efficient algorithm for finding the M most probable configurations in probabilistic expert systems. *Statistics and Computing*, 8(2) :159–173, 1998.
25. D J Rose. Triangulated graphs and the elimination process. *Journal of Mathematical Analysis and Applications*, 32 :597–609, 1970.
26. D J Rose, R E Tarjan, and G S Lueker. Algorithmic aspects of vertex elimination on graphs. *SIAM journal on Computing*, 5 :266–283, 1976.
27. G Shafer and P P Shenoy. Probability propagation. *Annals of Mathematics and Artificial Intelligence*, 2 :327–352, 1990.

28. F van den Eijkhof and H L Bodlaender. Safe reduction rules for weighted treewidth. volume 2573 of *Lecture Notes in Computer Science*, pages 176–185. Springer, 2002.
29. P P Wakker. *Additive Representations of Preferences, A New Foundation of Decision Analysis.* Kluwer, 1989.