Fast Recommendations Using GAI Models

Jean-Philippe Dubus **Christophe Gonzales** **Patrice Perny**
LIP6 - UPMC
104 avenue du président Kennedy, F-75016 Paris
firstname.lastname@lip6.fr

Abstract

This paper deals with Decision-Making in the context of multiattribute utility theory and, more precisely, with the problem of efficiently determining the best alternative w.r.t. an agent's preferences (choice problem). We assume that alternatives are elements of a product set of attributes and that the agent's preferences are represented by a generalized additive decomposable (GAI) utility on this set. Such a function allows an efficient representation of interactions between attributes while preserving some decomposability of the model. GAI utilities can be compiled into graphical structures called GAI networks that can be exploited to solve choice problems using collect/distribute schemes essentially similar to those used in Bayesian networks. In this paper, rather than directly using this scheme on the GAI network for determining the most preferred alternative, we propose to work with another GAI function, acting as an upper-bound on utility values and enhancing the model's decomposability. This method still provides the exact optimal solution but speeds up significantly the search. It proves to be particularly useful when dealing with choice and ranking under constraints and within collective Decision-Making, where GAI nets tend to have a large size. We present an efficient algorithm for determining this new GAI function and provide experimental results highlighting the practical efficiency of our procedure.

1 Introduction

Current works in preference modeling and algorithmic decision theory aim at developing compact preference models achieving a good compromise between descriptive and prescriptive objectives. On the one hand, they aim at proposing new sophisticated models able to capture the diversity of observed decision making behaviors. On the other hand, the models are kept as simple as possible to preserve efficiency both in elicitation and recommendation tasks. This very active flow of research is motivated by the increasing need of decision support systems on the web, to help users in making decisions among a combinatorial sets of possibilities.

In this context, extensive representation of preferences is not feasible and decision procedures cannot resort to exhaustive enumerations of options due to the large size of the domains to be explored. In this paper we consider such situations where the set of items on which recommendations must be done is all or part of a product set of attributes the size of which is huge. Current works on this matter include advances on qualitative preference models (CP-nets [Boutilier *et al.*, 2004a; 2004b; Brafman *et al.*, 2006]). These are naturally suited to applications in which preferences can easily be approximated by almost lexicographic rules on attributes with small domains (e.g. recommender systems to buy books on the web). However, as observed in [Boutilier *et al.*, 2001], when a deeper preference investigation is possible, quantitative representations such as utilities can significantly outperform qualitative models due to their higher descriptive power.

In the literature, different quantitative models based on utilities have been developed to take into account different preference structures. The most popular assumes a special kind of independence among attributes called "mutual preferential independence". It ensures that preferences are representable by an additive utility [Krantz *et al.*, 1971; Bacchus and Grove, 1995]. Such decomposability makes both the elicitation process and the query optimizations very fast and simple. However, in practice, it may fail to hold as it rules out any interaction among attributes. Generalizations have thus been proposed in the literature to increase the descriptive power of additive utilities. Among them, *multilinear utilities* [Keeney and Raiffa, 1993] and GAI (generalized additive independence) decompositions [Braziunas and Boutilier, 2005; Gonzales and Perny, 2004; Bacchus and Grove, 1995] allow quite general interactions between attributes while preserving some decomposability. The latter has been used to endow CP-nets with utilities (UCP-nets) [Boutilier *et al.*, 2001; Brafman *et al.*, 2004].

Studies on GAI utilities have been mainly devoted to their elicitation [Braziunas and Boutilier, 2005; 2007; Gonzales and Perny, 2004; 2005], efficient algorithms for answering queries being similar to those used in the Bayes net or junction tree community [Jensen *et al.*, 1990; Madsen and Jensen, 1999]. However, queries of interest in GAI-based recommender systems are very special: generally, they consist in finding the element most preferred by the user (choice problem), or a list of the top k elements (ranking problem). The specificities of these queries are not taken into account by

the general junction tree-based methods mentioned above. This paper proposes a new algorithm that takes advantage of these particularities, thus significantly increasing the efficiency of the junction tree-based algorithms. This proves to be particularly useful when dealing, e.g., with recommendation tasks under constraints and collective Decision-Making, where GAI nets tend to have a substantial size.

More precisely, Section 2 recalls the basics of GAI utilities and their graphical counterparts, the so-called *GAI networks*. It also briefly recalls how recommendation tasks can be performed using these networks. Section 3 shows how these computations can be significantly sped up by solving a special ranking task within a secondary sparse GAI structure. Section 4 presents benchmarks that illustrate the practical efficiency of our method. Finally, the last section concludes the paper and suggests some perspectives for future research.

2 GAI-Networks Fundamentals

Throughout the paper, $\succsim$ denotes a decision maker's (DM) preference relation (a weak order over some set $\mathcal{X}$). $x \succsim y$ means that x is at least as good as y. $\succ$ refers to the asymmetric part of $\succsim$ and $\sim$ to the symmetric one. In practice, $\mathcal{X}$ is often described by a set of attributes. For simplicity, we assume that $\mathcal{X}$ is the product set of these attributes domains $\prod_{i=1}^{n} X_i$. The combinatorial nature of $\mathcal{X}$ prevents any recommender system to perform an exhaustive search for determining the best element w.r.t. $\succsim$ and the aim of this paper is to present an efficient procedure for its computation. Note that recommendations over subsets $\mathcal{Y}$'s of product set $\mathcal{X}$ can be dealt with simply by imposing that elements $y \in \mathcal{Y}$ have low preference levels. In the sequel, bold letters represent sets of attributes indices, uppercase letters such as A, X_1, denote attributes and their domains (as this is unambiguous) and lowercase letters denote their values: x, x^1 (resp. x_i, x_i^1) are thus values of X (resp. X_i). For any set $\mathbf{Y} \subseteq \{1,\ldots,n\}$, $x_{\mathbf{Y}}$ refers to the projection of $x \in \mathcal{X}$ on $\prod_{i\in\mathbf{Y}} X_i$.

2.1 GAI-Networks' Definition

Under mild hypotheses [Debreu, 1964], $\succsim$ is representable by a utility $u : \mathcal{X} \mapsto \mathbb{R}$ such that $x \succsim y \Leftrightarrow u(x) \geq u(y)$ for all $x, y \in \mathcal{X}$. However, in general, such a function is impractical because: i) as preferences are specific to each individual, utilities must be elicited for each DM, which is impossible due to the combinatorial nature of $\mathcal{X}$; and ii) in recommendation systems with multiple users, storing explicitly for each user the utility of every $x \in \mathcal{X}$ is too prohibitive. Fortunately, DM's preferences usually have a structure induced by independencies among attributes that substantially decreases the elicitation burden and the memory needed to store preferences, thus making the model usable in practice. The simplest case is obtained when preferences are represented by an additive utility $u(x) = \sum_{i=1}^{n} u_i(x_i)$ for any $x = (x_1, ..., x_n) \in \mathcal{X}$ [Wakker, 1989; Keeney and Raiffa, 1993]. This model only requires to elicit and store $u_i(x_i)$ for all $x_i \in X_i$, $i = 1,\ldots,n$. However, such a decomposition is not always convenient because it rules out any interaction between attributes. When DM's preferences are more complex, a more elaborate model is needed as shown next:

$ifnextchar[$

Example 1 Consider a recommender system for train tickets where the set $\mathcal{X}$ of possible configurations for a train ticket is described by three attributes: class $x_1 \in X_1 = \{$first class (c_1^1), second class $(c_1^2)\}$; type of car $x_2 \in X_2 = \{$smoking (s_2), smoking-free $(\bar{s}_2)\}$; and pricing period $x_3 \in X_3 = \{$discount period (d_3), normal period $(n_3)\}$. Suppose that an individual's preferences are as follows:

1) I prefer a smoking-free train car to a smoking one;
2) I prefer to travel in first class during the discount period and in second class during the normal period;
3) Given the option between a second class smoking train car and a first class smoking-free one, I prefer the latter.

These statements are assumed to be *ceteris paribus*, i.e. all else being equal. Such preferences are not representable by a fully additively decomposable utility because $(c_1^1, s_2, d_3) \succ (c_1^2, s_2, d_3) \Rightarrow u_1(c_1^1) > u_1(c_1^2)$ whereas $(c_1^2, s_2, n_3) \succ (c_1^1, s_2, n_3) \Rightarrow u_1(c_1^2) > u_1(c_1^1)$. Actually, the additive decomposition is ruled out by the interactions between attributes X_1 and X_3 (rule 2). However the DM's preferences can be represented by a decomposable utility of the form: $u(x) = u_{1,2}(x_1, x_2) + u_{1,3}(x_1, x_3)$, setting for instance:

$u_{1,2}(c_1^1,s_2)=1$; $u_{1,2}(c_1^1,\bar{s}_2)=3$; $u_{1,2}(c_1^2,s_2)=2$;
$u_{1,2}(c_1^2,\bar{s}_2)=4$; $u_{1,3}(c_1^1,n_3)=1$; $u_{1,3}(c_1^1,d_3)=2$;
$u_{1,3}(c_1^2,n_3)=2$; $u_{1,3}(c_1^2,d_3)=1$.

Note that CP-nets are also unable to represent compactly these preferences due to rule 3. ∎

Such a decomposition over overlapping factors is called a GAI decomposition [Bacchus and Grove, 1995]. It includes additive and multilinear decompositions as special cases. It is also very flexible since it does not make any assumption on the kind of interactions between attributes. GAI decompositions can be defined more formally as follows:

Definition 1 (GAI decomposition) *Let $\mathcal{X} = \prod_{i=1}^{n} X_i$. Let $\mathbf{Z}_1, \ldots, \mathbf{Z}_r$ be some subsets of $\mathbf{N} = \{1,\ldots,n\}$ such that $\mathbf{N} = \cup_{i=1}^{r} \mathbf{Z}_i$. For every i, let $X_{\mathbf{Z}_i} = \prod_{j\in\mathbf{Z}_i} X_j$. Utility $u(\cdot)$ representing $\succsim$ is GAI-decomposable w.r.t. the $X_{\mathbf{Z}_i}$'s iff there exist functions $u_i : X_{\mathbf{Z}_i} \mapsto \mathbb{R}$ such that:*

$$u(x) = \textstyle\sum_{i=1}^{r} u_i(x_{\mathbf{Z}_i}), \text{ for all } x = (x_1, \ldots, x_n) \in \mathcal{X},$$

where $x_{\mathbf{Z}_i}$ denotes the tuple constituted by the x_j's, $j \in \mathbf{Z}_i$.

GAI decompositions can be represented by graphical structures called *GAI networks* [Gonzales and Perny, 2004] which are essentially similar to the junction graphs used for Bayesian networks [Jensen, 1996; Cowell *et al.*, 1999]:

Definition 2 (GAI network) *Let $\mathcal{X} = \prod_{i=1}^{n} X_i$. Let $\mathbf{Z}_1, \ldots, \mathbf{Z}_r$ be some subsets of $\mathbf{N} = \{1,\ldots,n\}$ such that $\bigcup_{i=1}^{r} \mathbf{Z}_i = \mathbf{N}$. Assume that $\succsim$ is representable by a GAI utility $u(x) = \sum_{i=1}^{r} u_i(x_{\mathbf{Z}_i})$ for all $x \in \mathcal{X}$. Then a GAI network representing $u(\cdot)$ is an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, satisfying the following two properties:*

1. *$\mathcal{V} = \{X_{\mathbf{Z}_1}, \ldots, X_{\mathbf{Z}_r}\}$;*
2. *For every $(X_{\mathbf{Z}_i}, X_{\mathbf{Z}_j}) \in \mathcal{E}$, $\mathbf{Z}_i \cap \mathbf{Z}_j \neq \emptyset$. For every pair of nodes $X_{\mathbf{Z}_i}, X_{\mathbf{Z}_j}$ such that $\mathbf{Z}_i \cap \mathbf{Z}_j = \mathbf{T}_{ij} \neq \emptyset$, there exists a path in $\mathcal{G}$ linking $X_{\mathbf{Z}_i}$ and $X_{\mathbf{Z}_j}$ such that all of its nodes contain all the indices of $\mathbf{T}_{ij}$ (Running intersection property).*

Nodes of $\mathcal{V}$ are called cliques*. Every edge $(X_{\mathbf{Z}_i}, X_{\mathbf{Z}_j}) \in \mathcal{E}$ is labeled by $X_{\mathbf{T}_{ij}} = X_{\mathbf{Z}_i \cap \mathbf{Z}_j}$ and is called a* separator.

Cliques are drawn as ellipses and separators as rectangles. In this paper, we shall only be interested in GAI trees. As mentioned in [Gonzales and Perny, 2004], this is not restrictive as general GAI networks can always be compiled into GAI trees. For any GAI decomposition, by Definition 2, the cliques of the GAI network should be the sets of variables of the subutilities. For instance, if $u(a,b,c,d,e,f,g,h) = u_1(a,b,c)+u_2(a,g)+u_3(b,d)+u_4(b,f)+u_5(d,e)+u_6(d,h)$ then the cliques are: ABC, AG, BD, BF, DE and DH (see Fig. 1). By Property 2 of Definition 2, the set of edges of a GAI network can be determined by any algorithm preserving the running intersection property (see the Bayesian network literature on this matter [Cowell *et al.*, 1999]). Within the GAI net, subutilities are stored into their respective clique.

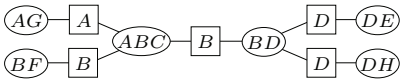

Figure 1: A GAI tree

2.2 Basic Procedures for Choice and Ranking

Finding the optimal choice for the GAI net of Figure 1 is equivalent to solving:

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

This can be computed by eliminating variables one by one using absorption defined below, that is, by maximizing locally within a clique the subutility stored into it and, then, sending the result to its neighbors.

Definition 3 (Absorption of clique $X_{\mathbf{C}_j}$ by $X_{\mathbf{C}_i}$) *Let $\mathbf{S}_{ij} = \mathbf{C}_i \cap \mathbf{C}_j$ and $\mathbf{D}_j = \mathbf{C}_j \backslash \mathbf{S}_{ij}$. Assume that $X_{\mathbf{C}_i}$ and $X_{\mathbf{C}_j}$ contain respectively utilities u_i and u_j. Let $v_{ij}(x_{\mathbf{S}_{ij}}) = \max_{x_{\mathbf{D}_j} \in X_{\mathbf{D}_j}} u_j(x_{\mathbf{S}_{ij}}, x_{\mathbf{D}_j})$ for all $x_{\mathbf{S}_{ij}} \in X_{\mathbf{S}_{ij}}$. Clique $X_{\mathbf{C}_i}$ is said to absorb $X_{\mathbf{C}_j}$ if u_i is substituted by $u'_i = u_i + v_{ij}$.*

Function `Collect`(clique $X_{\mathbf{C}_i}$)
01 **for all** cliques $X_{\mathbf{C}_j}$ adjacent to $X_{\mathbf{C}_i}$ except, if any, that which called `Collect` ($X_{\mathbf{C}_i}$) **do**
02 call `Collect` ($X_{\mathbf{C}_j}$) ; make $X_{\mathbf{C}_i}$ absorb $X_{\mathbf{C}_j}$

After calling `Collect` on any clique $X_{\mathbf{C}_i}$, the max of the subutility stored into $X_{\mathbf{C}_i}$ clearly corresponds to the value of the overall utility for the optimal choice. Now, there just remains to propagate back Argmax's to get the latter.

Definition 4 (Propagation of $x_{\mathbf{C}_i}$ into clique $X_{\mathbf{C}_j}$) *Let $\mathbf{S}_{ij} = \mathbf{C}_i \cap \mathbf{C}_j$ and $\mathbf{D}_j = \mathbf{C}_j \backslash \mathbf{S}_{ij}$. Let $x_{\mathbf{C}_i} \in X_{\mathbf{C}_i}$. Assume that $X_{\mathbf{C}_j}$ contains utilitiy u_j. $x_{\mathbf{C}_j}$ propagates $x_{\mathbf{C}_i}$ to $X_{\mathbf{C}_j}$ if $x_{\mathbf{C}_j} = (x_{\mathbf{S}_{ij}}, Argmax_{x_{\mathbf{D}_j} \in X_{\mathbf{D}_j}} u_j(x_{\mathbf{S}_{ij}}, x_{\mathbf{D}_j}))$.*

Function `Instantiate`(clique $X_{\mathbf{C}_i}$)
01 **if** `Instantiate` ($X_{\mathbf{C}_i}$) was called from clique $X_{\mathbf{C}_j}$
02 **then** Propagate the argmax's value of $X_{\mathbf{C}_j}$ to $X_{\mathbf{C}_i}$
03 **else** let $x_{\mathbf{C}_i}$ be the argmax of the subutility of $X_{\mathbf{C}_i}$
04 **for all** cliques $X_{\mathbf{C}_j}$ adjacent to $X_{\mathbf{C}_i}$ except, if any, that which called `Instantiate` ($X_{\mathbf{C}_i}$) **do** call `Instantiate` ($X_{\mathbf{C}_j}$)

Function `Optimal_choice`(GAI-net)
01 Let $X_{\mathbf{C}_0}$ be any clique in the GAI-net
02 call `Collect` ($X_{\mathbf{C}_0}$) and, then, `Instantiate` ($X_{\mathbf{C}_0}$)
03 $x^* \leftarrow$ the tuple of the argmax's computed by `Instantiate`
04 return the optimal choice x^*

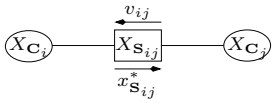

Figure 2: Absorption ($\leftarrow$) and Propagation ($\rightarrow$)

As suggested by [Nilsson, 1998] for finding the k most probable configurations in Bayes nets, determining the k-best elements in a GAI net can be performed by slightly modifying `Instantiate`. First, compute the optimal choice x^*. The next best element differs from x^* by at least the values of the attributes of one clique. Assume that this clique is $X_{\mathbf{C}_0}$, then calling `Instantiate` ($X_{\mathbf{C}_0}$) and preventing $x^*_{\mathbf{C}_0}$ to be chosen leads to the element we look for. Else if it is a neighbor $X_{\mathbf{C}_i}$ of $X_{\mathbf{C}_0}$, then the element we look for can be found by calling `Instantiate` ($X_{\mathbf{C}_i}$) and i) imposing $x_{\mathbf{C}_i \cap \mathbf{C}_0}$ to be equal to $x^*_{\mathbf{C}_i \cap \mathbf{C}_0}$, and ii) preventing calls to `Instantiate` ($X_{\mathbf{C}_0}$). The same process can be applied for all the other cliques. As we do not know actually from which clique `Instantiate` should be called to get the element we look for, we call it from every clique and the second best element is, among all those returned by the calls to `Instantiate`, that which has the maximal utility value. Of course, the same scheme can be applied to get the third best element, and so on till the k-best element. A feature that is important for our algorithm is that all the heavy computations are performed during the `Collect` phase, `Instantiate` having no matrix manipulation to do. As a consequence, the latter is much faster than the former and, even for large k's, ranking the k-best elements is not much more time consuming than computing the optimal choice.

3 An Efficient Exact Choice Procedure

Of course, for "small" GAI nets, we cannot expect to speed up significantly the above ranking and choice procedures. However, when working on large GAI nets, as is often the case when hard constraints are added to the network[1] or when dealing with collective decision making, substantial benefits can be gained using the method described below. The key idea is to approximate the DM's GAI utility —call it u— by another GAI function —call it g— which is much more decomposable than the former (i.e., the cliques of its GAI tree are smaller than those of u's GAI tree), and which acts as an upper-bound on the DM's utility values. Being more decomposable, its GAI net is thus of smaller size and answering choice and ranking queries can be done quickly. Of course, as g is only an upper bound on u, the best element for g is certainly not that for u. However, we show that ranking enough elements with g, we can extract from these elements the best element for u. As ranking involves only one `Collect` phase followed by multiple `Instantiate` calls, which are very light in terms of time consumption, the ranking process with g significantly outperforms answering a choice query with u. As we shall see in the next section, this intuition is confirmed by our experiments.

[1] Hard constraints can be encoded as additional utilities such that feasible tuples are assigned utility 0 and infeasible ones utility $-\infty$.

3.1 Fast Exact Choice Using Upper-Bound g

Let u be a GAI function representing relation $\succsim$ on $\mathcal{X}$, and let $g : \mathcal{X} \mapsto \mathbb{R}$ be a GAI function such that, for all $x \in \mathcal{X}$, $g(x) \geq u(x)$. Assume that $x^1, ..., x^k$ are the k-best elements on $\mathcal{X}$ w.r.t. g. Compute $\hat{x} = \text{Argmax}_{i=1,...,k} u(x^i)$. If $u(\hat{x}) \geq g(x^k)$, then $\hat{x}$ is the optimal choice for u. Indeed, as we rank elements w.r.t. g, for any $k' > k$, $g(x^{k'}) \leq g(x^k)$, and since $g(x) \geq u(x)$ for all $x \in \mathcal{X}$, $u(x^{k'}) \leq g(x^{k'}) \leq g(x^k) \leq u(\hat{x})$. Consequently, the optimal choice for u can be obtained by ranking elements x^i w.r.t. g until the maximum value of the $u(x^j)$'s for all the x^j's found so far, exceeds the value of $g(x^i)$. Recall that rankings are not much more time consuming than optimal choice computations. Hence, if the GAI net related to g is much smaller than that of u, its optimal choice computation is much faster, and the overall response time for g's ranking is much smaller than a direct choice computation using u. Now, the main problem is to find an efficient way for determining a "good" candidate for g.

3.2 Construction of Upper-Bound Function g

Let $\mathcal{X} = \prod_{i=1}^{n} X_i$ and let $\mathbf{Z}_1, \ldots, \mathbf{Z}_r$ be some subsets of $\mathbf{N} = \{1, \ldots, n\}$ such that $\mathbf{N} = \cup_{i=1}^{r} \mathbf{Z}_i$. Consider a GAI function $u = \sum_{i=1}^{r} u_i$, with $u_i : X_{\mathbf{Z}_i} \mapsto \mathbb{R}$. Without additional constraints on the u_i's domains, it may happen that u's GAI net is not singly connected. For instance, if $u(A, B, C, D) = u_1(A, B) + u_2(B, C) + u_3(C, D) + u_4(D, A)$, then the running intersection property requires that the GAI net contains a loop. But a GAI tree is needed to apply the preceding section. Hence, before computing optimal choice with u, the latter must be *compiled* into a GAI tree. This compilation process is similar that of junction trees for Bayes nets [Kjærulff, 1990; van den Eijkhof and Bodlaender, 2002]. First an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is created with $\mathcal{V} = \{X_1, \ldots, X_n\}$ and $\mathcal{E} = \{(X_i, X_j) :$ there exists h such that $\{i, j\} \subseteq \mathbf{Z}_h\}$. Next, $\mathcal{G}$ is triangulated by eliminating one by one the nodes of $\mathcal{V}$. Each time a node, say X_i, is eliminated, edges (fill-ins) are added to $\mathcal{G}$ so that X_i's neighbors form a complete subgraph (a clique) and, then, X_i and its adjacent edges are removed from $\mathcal{G}$. [Rose, 1970] guarantees that, whatever the elimination sequence, the cliques thus created can be linked to form a GAI tree. The compactness of the resulting GAI tree heavily depends on the sequence chosen. However finding the best sequence is out of the scope of this paper and the reader interested may refer for instance to [Kjærulff, 1990; van den Eijkhof and Bodlaender, 2002].

GAI cliques are thus created during the triangulation process. Clearly, the smaller the cliques the faster the choice and ranking procedures, the size of a clique being the product of the domain sizes of its attributes [Nilsson, 1998]. To determine a good upper bound function g, we should thus perform

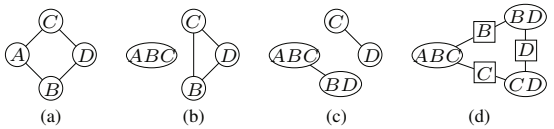

Figure 3: Constraints on edges deletions

a triangulation that ensures that no clique is too large. Assume that, during the triangulation, nodes $X_{i_1}, \ldots, X_{i_p}$ have already been eliminated and that we are currently eliminating $X_{i_{p+1}}$. If the size of the new clique is above a given threshold, then we advocate removing some edges adjacent to $X_{i_{p+1}}$ in order to reduce the size of the clique. The resulting GAI tree will thus represent an approximation of u. When removing edges, care should nevertheless be taken in order to avoid the GAI network to contain loops: consider the triangulation of function u of the preceding paragraph. At the beginning, graph $\mathcal{G}$ is that of Fig.3.a. Eliminating A results in the creation of clique ABC and edge (B, C) (Fig. 3.b). Now, eliminating B creates clique BCD. But if BCD is too large, we may be willing to remove edge (B, C) to reduce its size. The resulting clique would be BD. Finally, removing C and D would create clique CD, thus resulting in the multiply-connected graph of Fig. 3.d. The problem stems from the fact that (B, C) is compulsory for A and its neighbors to form a clique when A is eliminated. Hence, removing (B, C) requires that either B or C be no more a neighbor of A or, equivalently, that either edge (A, C) or (A, B) be removed as well. Of course, if A is not the first node eliminated, then removing edge (A, C) or (A, B) adds similar constraints on edges involved in previous node eliminations. Fortunately, all these constraints can be organized as a set of linear inequalities: assume we eliminate X_i and that X_j and X_k are neighbors of X_i; consider inequality $e_{ij} + e_{ik} \geq e_{jk}$, where e_{ij}, e_{ik}, e_{jk} are Booleans equal to 1 iff their corresponding edge ((X_i, X_j), (X_i, X_k) or (X_j, X_k)) is removed. Then, obviously the edge linking X_i's neighbors, i.e., (X_j, X_k), can be removed only if either (X_i, X_j) or (X_i, X_k) is removed as well. Our system of inequalities is thus constructed incrementally each time a node elimination is performed, adding a new inequality for each pair of nodes adjacent to the one eliminated. Each time we wish to compel an edge removal, we just add to the system that the corresponding Boolean is equal to 1. Clearly, any solution of this system will lead to a triangulated graph and, consequently, to a GAI tree.

Of course, removing an edge from the triangulated graph has a cost: that of approximating the original function u by another function g. As a result, the more the discrepancy between u and g, the higher the number of elements required to be ranked with g. Hence, in addition to our system of inequalities, we should add a cost function measuring how far apart g is from u. Consider a subutility $u_i : X_{\mathbf{Z}_i} \mapsto \mathbb{R}$ of u. In the GAI graph of u, $X_{\mathbf{Z}_i}$ forms a clique. Assume we remove from this clique some edges, hence resulting in m smaller cliques $X_{\mathbf{C}_j}$, each one representing a function, say $g_j : X_{\mathbf{C}_j} \mapsto \mathbb{R}$. Then, u_i is approximated by $\sum_{j=1}^{m} g_j$. In this paper, the discrepancy measure we chose between u_i and $\sum_{j=1}^{m} g_j$ is simply: $\sum_{x_{\mathbf{Z}_i} \in X_{\mathbf{Z}_i}} |\sum_{j=1}^{m} g_j(x_{\mathbf{C}_j}) - u_i(x_{\mathbf{Z}_i})|$. Hence we should find some functions g_j's minimizing this measure while approximating u_i from above. So we should solve the following linear program whose variables are the values of the $g_j(x_{\mathbf{C}_j})$'s for all $x_{\mathbf{C}_j} \in X_{\mathbf{C}_j}$:

$$\begin{array}{l} \min_{\{\text{values } g_j(x_{\mathbf{C}_j})\}} \sum_{x_{\mathbf{Z}_i} \in X_{\mathbf{Z}_i}} [\sum_{j=1}^{m} g_j(x_{\mathbf{C}_j})] - u_i(x_{\mathbf{Z}_i}) \\ \text{s.t. } \sum_{j=1}^{m} g_j(x_{\mathbf{C}_j}) \geq u_i(x_{\mathbf{Z}_i}), \text{ for all } x_{\mathbf{Z}_i} \in X_{\mathbf{Z}_i}. \end{array} \quad (1)$$

Usually, the original GAI subutilities u_i's do not contain

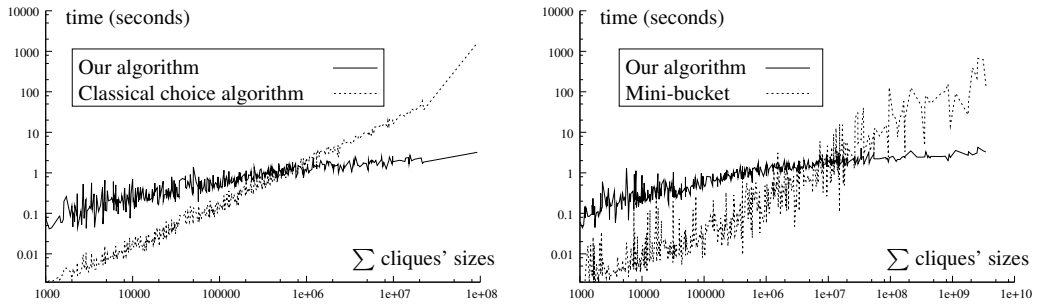

Figure 4: Comparisons with the classical choice algorithm (left) and with Mini-bucket Branch and Bound (right)

many attributes, so optimal g_j's functions can be computed quickly as a preprocess for any subset of edges of cliques $X_{\mathbf{Z}_i}$'s. Now, when removing a given set of edges, the set of linear inequalities described in the preceding paragraph can be used to compute which additional edges need be removed to prevent loops in the resulting GAI graph and, using the g_j's computed in (1), the discrepancy between u and g can be measured. This guides us in selecting, at each node elimination step, the right set of edges to be removed in order to minimize the discrepancy between u and g while preventing cliques sizes to be greater than a given value. Actually, this minimization problem can be completely specified as a linear program with Boolean variables.

3.3 Practical Determination of Function g

In practice, the above method involves solving many linear programs of type (1): one for each set of edges selected to be removed at each node elimination step. Fortunately, this process can be significantly sped-up by approximating the discrepancy measure. The objective function in (1) actually measures the gap between u and g for a given set S of removed edges. If we approximate this measure by the sum of the discrepancies induced separately by each edge in S, then it can be estimated using dynamic programming techniques. Consider a clique $X_{\mathbf{Z}_i}$ of the original GAI function from which we have removed p edges (X_{j_t}, X_{k_t}), $t \in \{1, \ldots, p\}$. Instead of using (1) as the discrepancy measure, for each removed edge (X_{j_t}, X_{k_t}), compute the discrepancy measure in the case where only this edge is removed, i.e., $d_{j_t k_t} = \min_{\{\text{values of } h_t, f_t\}} \sum_{x_{\mathbf{Z}_i} \in X_{\mathbf{Z}_i}} h_t(x_{\mathbf{Z}_i \setminus \{j_t\}}) + f_t(x_{\mathbf{Z}_i \setminus \{k_t\}}) - u_i(x_{\mathbf{Z}_i})$, where $h_t + f_t$ approximates u_i from above, and h_t (resp. f_t) is defined on clique $X_{\mathbf{Z}_i}$ without node X_{j_t} (resp X_{k_t}). We then estimate the discrepancy between u and its approximation g when all the p edges (X_{j_t}, X_{k_t})'s have been removed as $\sum_{t=1}^{p} d_{j_t k_t}$. As such, to each edge of the graph being triangulated can be attached its $d_{j_t k_t}$ value, computed once and for all. During triangulation, if the clique resulting from a node elimination is too large, the edges to be removed to reduce its size can thus be determined minimizing $\sum d_{jk} e_{jk}$ under the constraints of the type $e_{ij} + e_{ik} \geq e_{jk}$ described above, where the e_{jk}'s are the Booleans indicating whether edge (X_j, X_k) is deleted. When d_{jk} is taken into account in this objective function to be minimized, $e_{jk} = 1$, which implies that either $e_{ij} = 1$ or $e_{ik} = 1$. Hence, the resulting graph is triangulated. $\sum d_{jk} e_{jk}$ can be approximated from above using dynamic programming by defining:

$$D_{jk} = \begin{cases} d_{jk} \text{ if there exists no constraint } e_{ij} + e_{ik} \geq e_{jk} \\ d_{jk} + \sum_i \min\{D_{ij}, D_{ik}\} \text{ otherwise.} \end{cases}$$

D_{jk} actually approximates from above $\sum d_{jk} e_{jk}$ as it selects between edges (X_i, X_j) and (X_i, X_k) that which seems, locally, to minimize the discrepancy between u and its approximation. The dynamic programming nature of D_{jk} makes it computable incrementally very quickly during triangulation and avoids needing to solve any Boolean linear program. This is the very process we used in our experiments to determine the GAI tree of the upper-bound function g. It enables a fast computation of this tree and, as we shall see in the next section, its result significantly speeds up the computation of the most preferred element w.r.t. the original GAI utility.

4 Experimentations

To evaluate the performance of our method in practice, we chose to compare it with the basic GAI net' choice algorithm [Gonzales and Perny, 2005] as well as with an adaptation of Mini-Buckets Branch and Bound to GAI networks [Kask and Dechter, 1999]. We performed two different sets of experiments, the difference lying in the maximal number of attributes of product set $\mathcal{X}$: for comparisons with the classical choice algorithm, $\mathcal{X}$ had up to 20 attributes whereas for comparisons with mini-buckets, it could contain up to 40 attributes (in this case, $\mathcal{X}$ could have up to 100 trillion elements). In each experiment, attributes' domain sizes have been chosen randomly between 3 and 5, and GAI subutilities u_i's have been created randomly, with values drawn between 0 and 1000. Fig. 4 summarizes the results of 20000 choice experiments: the X-axis represents the sum of the sizes of the cliques in the GAI net (the size of a clique being the product of the domain sizes of its attributes), whereas the Y-axis represents the response times (in seconds) for solving the choice problem. Note that scales are logarithmic. As expected, when cliques' sizes are large, our algorithm clearly outperforms both the classical algorithm and Mini-Buckets B&B. Actually, experimentally, it tends to be exponentially faster than the classical algorithm. When cliques are small, i.e., their size is smaller than 10^7, the resolution times of the linear programs used in our method cancel the gain induced by using g's network instead of that of u. Note that, even if the sizes of the subutilities representing the DM's preferences are small, the addition of subutilities representing hard constraints may result in GAI trees with large cliques.

5 Conclusion

We showed in this paper that approximating a GAI utility by an appropriate more decomposable function acting as an upper-bound significantly speeds-up the exact determination of optimal tuples. Besides this direct application, this technique will probably also open new possibilities in solving utility-based search problems under constraints. Indeed feasibility constraints can easily be incorporated in a GAI-utility maximization problem, under the form of additional GAI factors. Such factors model new dependencies that might induce, after triangulation of the Markov graph, a significant increase in the size of cliques, with a drastic impact on computation times. Similarly, in collective decision making on combinatorial domains, it might be useful to optimize a multiagent GAI utility function defined as the sum of individual GAI utilities. Since individuals may have different value systems, it may happen that the independence structure significantly varies from an individual to another, thus multiplying the GAI factors of the utility function. In both cases, the possibility of approximating the utility function so as to add artificial decompositions of cliques will significantly speed-up resolution times without departing from optimality, and will increase the size of the problems solved.

For future research, different approximating schemes seem quite promising. In particular, alternative criteria for deciding whether a clique need be approximated should further enhance the efficiency of our approach.

References

[Bacchus and Grove, 1995] F Bacchus and A Grove. Graphical models for preference and utility. In *UAI*, 1995.

[Boutilier *et al.*, 2001] C Boutilier, F Bacchus, and R Brafman. UCP-networks; a directed graphical representation of conditional utilities. In *UAI*, 2001.

[Boutilier *et al.*, 2004a] 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 A.I. Research*, 21:135–191, 2004.

[Boutilier *et al.*, 2004b] C Boutilier, R Brafman, C Domshlak, H Hoos, and D Poole. Preference-based constraint optimization with CP-nets. *Computational Intelligence*, 20, 2004.

[Brafman *et al.*, 2004] R Brafman, C Domshlak, and T Kogan. On generalized additive value-function decomposition. In *UAI*, 2004.

[Brafman *et al.*, 2006] R Brafman, C Domshlak, and S Shimony. On graphical modeling of preference and importance. *Journal of A.I. Research*, 25:389–424, 2006.

[Braziunas and Boutilier, 2005] D Braziunas and C Boutilier. Local utility elicitation in GAI models. In *UAI*, 2005.

[Braziunas and Boutilier, 2007] D Braziunas and C Boutilier. Minimax regret-based elicitation of generalized additive utilities. In *UAI*, pages 25–32, 2007.

[Cowell *et al.*, 1999] R Cowell, A Dawid, S Lauritzen, and D Spiegelhalter. *Probabilistic Networks and Expert Systems*. Statistics for Engineering and Information Science. Springer, 1999.

[Debreu, 1964] G Debreu. Continuity properties of paretian utility. *International Economic Review*, 5:285–293, 1964.

[Gonzales and Perny, 2004] C Gonzales and P Perny. GAI networks for utility elicitation. In *KR*, pages 224–234, 2004.

[Gonzales and Perny, 2005] C Gonzales and P Perny. GAI networks for decision making under certainty. In *IJCAI'05 – Workshop on Advances in Preference Handling*, pages 100–105, 2005.

[Jensen *et al.*, 1990] F.V. Jensen, S.L. Lauritzen, and K.G. Olesen. Bayesian updating in causal probabilistic networks by local computations. *Computational Statistics Quarterly*, 4:269–282, 1990.

[Jensen, 1996] F Jensen. *An introduction to Bayesian Networks*. Taylor and Francis, 1996.

[Kask and Dechter, 1999] K Kask and R Dechter. Branch and bound with mini-bucket heuristics. In *Proc. of IJCAI*, 1999.

[Keeney and Raiffa, 1993] Ralph L Keeney and Howard Raiffa. *Decisions with Multiple Objectives - Preferences and Value Tradeoffs*. Cambridge University Press, 1993.

[Kjærulff, 1990] 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.

[Krantz *et al.*, 1971] D Krantz, R D Luce, P Suppes, and A Tversky. *Foundations of Measurement (Additive and Polynomial Representations)*, volume 1. Academic Press, 1971.

[Madsen and Jensen, 1999] 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.

[Nilsson, 1998] D Nilsson. An efficient algorithm for finding the M most probable configurations in probabilistic expert systems. *Statistics and Computing*, 8(2):159–173, 1998.

[Rose, 1970] D.J. Rose. Triangulated graphs and the elimination process. *J. Math. Analysis and Applications*, 32:597–609, 1970.

[van den Eijkhof and Bodlaender, 2002] F. van den Eijkhof and H. L. Bodlaender. Safe reduction rules for weighted treewidth. In *Proc of the 28th International Workshop on Graph-Theoretic Concepts in Computer Science*, volume 2573 of *LNCS*, pages 176–185. Springer, 2002.

[Wakker, 1989] P Wakker. *Additive Representations of Preferences, A New Foundation of Decision Analysis*. Kluwer, 1989.