GAI-NETWORKS: OPTIMIZATION, RANKING AND COLLECTIVE CHOICE IN COMBINATORIAL DOMAINS

Christophe GONZALES * Patrice PERNY * Sergio QUEIROZ *

Abstract. This paper deals with preference representation and decision-making problems in the context of multiattribute utility theory. We focus on the generalized additive decomposable utility model (GAI) which allows interactions between attributes while preserving some decomposability. We present procedures to deal with the problem of optimization (choice) and ranking of multiattribute items. We also address multiperson decision problems and compromise search using weighted Tchebycheff distances. These procedures are all based on GAI networks, a graphical model used to represent GAI utilities. Results of numerical experiments highlight the practical efficiency of our procedures.

Keywords:
GAI networks, graphical models, optimization, ranking, compromise search

1 Introduction

The development of decision support systems and web recommender systems has stressed the need for models that can handle users' preferences and perform preference-based recommendation tasks. In this respect, current works in preference modeling and decision theory aim at developing compact preference models achieving a good trade-off between two conflicting aspects: i) the need for models flexible enough to describe sophisticated decision behaviors; and ii) the practical necessity of keeping the elicitation effort at an admissible level as well as the need for fast procedures to solve preference-based optimization problems. As an example, let us mention interactive decision support systems on the web where the preferred solution must be found among a combinatorial set of possibilities. This kind of application motivates the current interest for qualitative preference models and compact representations

*Laboratoire d'Informatique de Paris 6, UPMC. 104 Av. du Président Kennedy, 75016 Paris, France. E-mail: firstname.lastname@lip6.fr

like CP-nets [3] and mCP-nets [19], and their extension to the multiagent case. Such models are deliberately simple and flexible enough to be integrated efficiently in interactive recommendation systems; the preferences of the agents must be captured using only a few questions so as to perform a fast preference-based search over the possible items.

In other applications (e.g. configuration system, fair allocation of resources, combinatorial auctions [11]), more time can be spent in the elicitation stage in order to get a finer description of preferences. In such cases, utilities can significantly outperform qualitative models due to their higher descriptive power [2]. Moreover, the use of cardinal utilities in the multiagent setting allows us to escape the framework of Arrow's impossibility theorem which considerably restricts aggregation possibilities [18].

In the literature, different quantitative models based on utilities have been developed to take into account different preference structures. The most widely used model assumes a special kind of independence among attributes called "mutual preferential independence" which ensures that the preferences are representable by an additive utility [15]. Such decomposability makes the elicitation process very fast and simple. However, in practice, preferential independence may fail to hold as it rules out any interaction among attributes. Some generalizations have thus been investigated. For instance *utility independence* on every attribute leads to a more sophisticated form called *multilinear utilities* [1]. Those are more general than additive utilities but still cannot cope with many kinds of interactions among attributes. To increase the descriptive power of such models, GAI (generalized additive independence) decompositions have been introduced by [12], that allow more general interactions between attributes [1] while preserving some decomposability. Such decompositions have been used to endow CP-nets with utility functions (UCP-nets) both under uncertainty [2] and under certainty [5].

In the same direction [13, 6] propose general procedures to assess GAI utilities in decision under risk. They consist in sequences of questions involving simple lotteries that capture efficiently the basic features of the agent's attitude with respect to risk. These elicitation processes are guided by GAI networks, a graphical model introduced in [13].

In this paper, we address the problems of optimization (choice of the best element), ranking, and the determination of a good compromise solution for a group of individuals. We assume here that the alternatives to be compared belong to a product set the size of which prevents exhaustive enumeration. Our aim is to show the potential of GAI-networks to provide a compact representation of preferences as well as to perform efficiently preference-based recommendation tasks. The paper is organized as follows: In Section 2, we recall some elements about GAI models and GAI-Networks. Section 3 deals with the optimization problem, that will be the first stage of the ranking problem, presented in Section 4. Section 5 considers the problem of looking for a compromise solution between individuals and provides an efficient algorithm to find a good compromise solution according to a classic non-linear criterion in multiobjective search: the weighted Tchebytcheff norm. Sections 4 and 5 both present numerical results that indicate the efficiency of our algorithms.

2 GAI Networks

Before describing GAI networks, we shall introduce some notations. Throughout the paper, $\succsim$ denotes a decision maker's (DM) preference relation, which is assumed to be a weak order, over some set $\mathcal{X}$. Proposition "$x \succsim y$" 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 the domains of these attributes, although extensions to general subsets are possible [7]. In the rest of the paper, we adopt the following notation:

- uppercase letters (possibly subscripted) such as A, B, X_1 denote attributes as well as their domains (as this is unambiguous and it simplifies the notation);
- unless otherwise mentioned, lowercase letters such as a, b', x_1 denote attribute values;
- subscripted attributes X_i are characterized by their index. Their values are also identified by this index. For instance, x_i, y_i, z_i^3 represent values of X_i;
- non-subscripted attributes A, B are characterized by their letter. Their values are also identified by this letter. For instance, a, a', a^3 represent values of A;
- By abuse of notation, for any set $Y \subset \{X_1, \ldots, X_n\}$, x_Y (resp. x_{-Y}) will refer to the projection of $x \in \mathcal{X}$ on $\times_{X_i \in Y} X_i$ (resp. $\times_{X_i \notin Y} X_i$).

2.1 Motivation

Under mild hypotheses [10], it can be shown that $\succsim$ is representable by a utility, i.e., there exists a function $u : \mathcal{X} \mapsto \mathbb{R}$ such that $x \succsim y \Leftrightarrow u(x) \geq u(y)$ for all $x, y \in \mathcal{X}$. As preferences are specific to each individual, utilities must be elicited for each DM, which is impossible due to the combinatorial nature of $\mathcal{X}$. Moreover, in a recommendation system with multiple regular users, storing explicitly for each user the utility of every element of $\mathcal{X}$ is prohibitive.

Fortunately, DM's preferences usually have an underlying structure induced by independences among attributes that substantially decreases the elicitation burden and the memory needed to store preferences. The simplest case is obtained when preferences over $\mathcal{X} = X_1 \times \ldots \times X_n$ are represented by an additive utility $u(x) = \sum_{i=1}^{n} u_i(x_i)$ for any $x = (x_1, \ldots, x_n) \in \mathcal{X}$. This model only requires to elicit and store $u_i(x_i')$ for any $x_i' \in X_i$, $i = 1, \ldots, n$. Unfortunately, such decomposition is not always appropriate because it rules out interactions between attributes. When DM's preferences are more complex, a more elaborate model is needed, as is shown below:

Example 1 : Consider a set $\mathcal{X}$ of menus $x = (x_1, x_2, x_3)$, with main course $x_1 \in X_1 = \{\text{meat } (m_1), \text{ fish } (f_1)\}$, drink $x_2 \in X_2 = \{\text{red wine } (r_2), \text{ white wine } (w_2)\}$ and dessert $x_3 \in X_3 = \{\text{cake } (c_3), \text{ ice cream } (i_3)\}$.

Assume that an individual has the following preferences:

1. Matching the main course with the wine (red wine for meat, white wine for fish) is my most important choice criterion;
2. At a lower level of priority, meat is preferred to fish;
3. I prefer cake to ice cream when the main course is fish but the opposite when the main course is meat (the combination cake + meat being too heavy).

Given the interaction between attributes X_1 and X_2, and between X_1 and X_3, these preferences cannot be represented by a fully decomposable additive utility function of the form $u_1(x_1) + u_2(x_2) + u_3(x_3)$. Indeed, if such an additive utility existed, $(m_1, r_2, i_3) \succ (f_1, r_2, i_3)$ —which matches the above three rules— would induce that $u_1(m_1) + u_2(r_2) + u_3(i_3) > u_1(f_1) + u_2(r_2) + u_3(i_3)$ and, hence that $u_1(m_1) + u_2(w_2) + u_3(i_3) > u_1(f_1) + u_2(w_2) + u_3(i_3)$. However, the last inequality is equivalent to proposition "$(m_1, w_2, i_3) \succ (f_1, w_2, i_3)$", which contradicts the above first rule. It is however interesting to remark that preferences can still be represented by a decomposable utility of the form: $u(x) = u_{1,2}(x_1, x_2) + u_{1,3}(x_1, x_3)$ by setting, for instance:

$$\begin{array}{llll} u_{1,2}(m_1, r_2) = 6 & u_{1,2}(f_1, w_2) = 4 & u_{1,2}(m_1, w_2) = 2 & u_{1,2}(f_1, r_2) = 0 \\ u_{1,3}(m_1, c_3) = 0 & u_{1,3}(m_1, i_3) = 1 & u_{1,3}(f_1, c_3) = 1 & u_{1,3}(f_1, i_3) = 0. \end{array}$$

In this case, the utilities of the 2^3 possible menus $x^{(i)}$ will be:

$$\begin{array}{ll} u(x^{(1)}) = u(m_1, r_2, c_3) = 6 & u(x^{(2)}) = u(m_1, r_2, i_3) = 7 \\ u(x^{(3)}) = u(m_1, w_2, c_3) = 2 & u(x^{(4)}) = u(m_1, w_2, i_3) = 3 \\ u(x^{(5)}) = u(f_1, r_2, c_3) = 1 & u(x^{(6)}) = u(f_1, r_2, i_3) = 0 \\ u(x^{(7)}) = u(f_1, w_2, c_3) = 5 & u(x^{(8)}) = u(f_1, w_2, i_3) = 4; \end{array}$$

which induce the following order, consistent with the individual's preferences:

$$x^{(2)} \succ x^{(1)} \succ x^{(7)} \succ x^{(8)} \succ x^{(4)} \succ x^{(3)} \succ x^{(5)} \succ x^{(6)}.$$

It could be objected that $u(x)$ is not so compact since it requires as many values as an extensive representation of u. But this is only due to the small size of our toy example. Note that, if m denotes the domain size of each one of the 3 attributes in this decomposition, only $2m^2$ numbers are needed to store the utility instead of m^3, a gain as soon as $m > 2$. ♦

Such a decomposition over overlapping factors is called a GAI decomposition. It includes additive and multilinear decompositions as special cases, but it is much more flexible as it does not make any assumption on the kind of interactions between attributes. GAI decompositions were introduced in [12] and brought into the realm of the AI community by [1].

2.2 Definition

GAI decompositions can be defined more formally as follows:

Definition 1 *Let $X = \times_{i=1}^{n} X_i$. Let $C_1, \ldots, C_k$ be subsets of $N = \{1, \ldots, n\}$ such that $N = \bigcup_{i=1}^{k} C_i$. For all i, let $X_{C_i} = \{X_j : j \in C_i\}$; in other words, X_{C_i} is the product set of the attributes whose indices belong to C_i. A utility function $u(\cdot)$ representing $\succsim$ over X is GAI-decomposable w.r.t. the X_{C_i}'s iff there exist functions $u_i : X_{C_i} \mapsto \mathbb{R}$ such that:*

$$u(x_1, ..., x_n) = \sum_{i=1}^{k} u_i(x_{C_i}), \quad \textit{for all } x = (x_1, \ldots, x_n) \in \mathcal{X},$$

where x_{C_i} is the tuple formed by the x_j's, $j \in C_i$.

For instance, $u(a, b, c, d, e, f, g) = u_1(a, b) + u_2(b, c, d) + u_3(c, e) + u_4(b, d, f) + u_5(b, g)$ is a GAI-decomposable utility, with $X_{C_1} = \{A, B\}$, $X_{C_2} = \{B, C, D\}$, $X_{C_3} = \{C, E\}$, $X_{C_4} = \{B, D, F\}$ and $X_{C_5} = \{B, G\}$. GAI decompositions can be represented by graphical structures we call *GAI networks* [13] which are essentially similar to junction graphs used in the Bayesian network literature [14, 8]:

Definition 2 *Let $u(x_1, \ldots, x_n) = \sum_{i=1}^{k} u_i(x_{C_i})$ be a GAI utility function over $\mathcal{X}$. A GAI network representing $u(\cdot)$ is an undirected graph $\mathcal{G} = (\mathcal{C}, \mathcal{E})$ satisfying the following three properties:*

> *Property 1: $\mathcal{C} = \{X_{C_1}, \ldots, X_{C_k}\}$. Vertices X_{C_i}'s are called* cliques. *To each vertex X_{C_i} is associated the corresponding factor u_i from the utility function u;*
>
> *Property 2: $(X_{C_i}, X_{C_j}) \in \mathcal{E} \Rightarrow C_i \cap C_j \neq \emptyset$. Edges (X_{C_i}, X_{C_j})'s are labeled by $X_{T_{ij}}$, where $T_{ij} = C_i \cap C_j$. $X_{T_{ij}}$ is called a* separator*;*
>
> *Property 3: for all X_{C_i}, X_{C_j} such that $C_i \cap C_j = T_{ij} \neq \emptyset$, there exists a path between X_{C_i} and X_{C_j} in $\mathcal{G}$ such that for every clique X_{C_k} in this path $T_{ij} \subseteq C_k$ (running intersection property).*

Cliques are usually drawn as ellipses and separators as rectangles. For any GAI decomposition, by Definition 2, the cliques of the GAI network should be the sets of variables of the subutilities. The edges in the network represent the intersections between subsets of attributes. As the intersections are commutative, the GAI network is an undirected graph. Note that this contrasts with UCP-nets, where the relationships between vertices in the network correspond to conditional dependencies, thus justifying the use of directed graphs for UCP-nets.

In this paper, we shall only be interested in GAI trees. As mentioned in [13], this is not restrictive as general GAI networks can always be compiled into GAI trees. 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 [8]). Figure 1 shows a GAI network for the example given just below Definition 1.

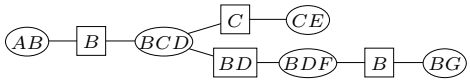

Figure 1: A GAI tree

2.3 CP-nets, TCP-nets and GAI-nets

One of the most distinguishable features of CP-nets [3] is their ability to represent *ceteris paribus* preferences over attribute values (possibly conditioned by the values of some other attributes). In this way, by using CP-nets in Example 1, we may formulate preference statements in one of the following formats:

- an unconditional preference statement, for example "I prefer meat to fish". We write this compactly as $m_1 \succ f_1$, meaning that for all $x_2 \in X_2, x_3 \in X_3$, $(m_1, x_2, x_3) \succ (f_1, x_2, x_3)$.
- a conditional preference statement, for example "when I eat meat, I prefer red wine to white wine but it is the contrary when I eat fish". We write this compactly as $m_1 : r_2 \succ w_2$ and $f_1 : w_2 \succ r_2$, meaning that for all $x_3 \in X_3$, $(m_1, r_2, x_3) \succ (m_1, w_2, x_3)$ and $(f_1, w_2, x_3) \succ (f_1, r_2, x_3)$.

Assuming that preferences are transitive, other preferences can be deduced by taking the transitive closure of the *ceteris paribus* preferences. However, not all preferences can be obtained in this way. For instance, preferences like $(m_1, w_2, i_3) \succ (f_1, r_2, c_3)$ cannot be obtained. In order to discuss the descriptive potential of CP-nets and GAI-nets, consider the following example:

Example 2 : In a recommender system for train tickets, the set $\mathcal{X}$ of the 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 $x_3 \in X_3 =$ {blue period (b_3), red period (r_3)}. Suppose that an individual's preferences are as follows:

- I prefer to travel in second class;
- I prefer a smoking-free train car to a smoking one;
- Given the option between a second class smoking train car and a first class smoking-free one, I prefer the latter;[3]
- I prefer traveling during the blue period, no matter the other conditions.

♦

These preferences may be represented by a GAI utility $u(x) = u_{1,2}(x_1, x_2) + u_3(x_3)$ with subutility values: $u_{1,2}(c_1^2, \bar{s}_2) = 3$, $u_{1,2}(c_1^1, \bar{s}_2) = 2$, $u_{1,2}(c_1^2, s_2) = 1$, $u_{1,2}(c_1^1, s_2) =$

[3]These first three preference statements are assumed to be *ceteris paribus*, i.e. all else being equal.

0, $u_3(b_3) = 4$, $u_3(r_3) = 0$. They result in the following utilities for the 2^3 possible configurations $x^{(i)}$:

$$\begin{array}{lll} u(x^{(1)}) = u(c_1^2, \bar{s}_2, b_3) = 7; & u(x^{(2)}) = u(c_1^2, \bar{s}_2, r_3) = 3; & u(x^{(3)}) = u(c_1^1, \bar{s}_2, b_3) = 6; \\ u(x^{(4)}) = u(c_1^1, \bar{s}_2, r_3) = 2; & u(x^{(5)}) = u(c_1^2, s_2, b_3) = 5; & u(x^{(6)}) = u(c_1^2, s_2, r_3) = 1; \\ u(x^{(7)}) = u(c_1^1, s_2, b_3) = 4; & u(x^{(8)}) = u(c_1^1, s_2, r_3) = 0; & \end{array}$$

which induce the following preference ordering:

$$x^{(1)} \succ x^{(3)} \succ x^{(5)} \succ x^{(7)} \succ x^{(2)} \succ x^{(4)} \succ x^{(6)} \succ x^{(8)}.$$

This GAI model is completely consistent with the stated preferences. Using CP-nets, we can easily represent the first two statements (all we need is writing $c_1^2 \succ c_1^1$ and $\bar{s}_2 \succ s_2$) but we cannot add the last two into the model. Indeed, the third statement reveals an interaction between attributes X_1 and X_2, therefore preventing their decomposability. Hence, in order to obtain preference ordering $(c_1^1, \bar{s}_2, x_3) \succ (c_1^2, s_2, x_3)$ in a CP-net where $\bar{s}_2 \succ s_2$ we should add the conditional statement $\bar{s}_2 : c_1^1 \succ c_1^2$ but this is inconsistent with the unconditional preference $c_1^2 \succ c_1^1$ expressed by the individual. Similarly, the consequences of the last statement cannot be obtained by simply adding $b_3 \succ r_3$. Actually, given that $\bar{s}_2 \succ s_2$, $c_1^2 \succ c_1^1$ and $b_3 \succ r_3$, we are unable to identify preferences over configurations like (c_1^1, s_2, x_3) and $(c_1^2, \bar{s}_2, x_3)$.

The descriptive limits of CP-nets have motivated the introduction of a richer formalism that allows different importance levels for the attributes. This idea has given birth to TCP nets [4] which introduce "trade-offs" into CP-nets. They allow expressing statements about the relative importance of different attributes like, for instance, X_1 is more important than attribute X_2 (denoted by $1 \triangleright 2$). Additional preferences can now be derived from these statements. For instance, $(c_1^1, s_2, x_3) \succ (c_1^2, \bar{s}_2, x_3)$ can be deduced from the TCP-net, the undecidability mentioned in the preceding paragraph being solved by the fact that X_1 is declared to be more important than X_2. This additional sophistication addresses only partially the descriptive limitations previously mentioned. Indeed, assume now that the domain of variable X_1 has 5 elements $c_1^1, c_1^2, c_1^3, c_1^4, c_1^5$ corresponding to decreasing fare levels in both pricing periods. It would be natural that an individual had the following preferences:

$$(c_1^4, \bar{s}_2, x_3) \succ (c_1^5, s_2, x_3) \quad \text{and} \quad (c_1^1, \bar{s}_2, x_3) \prec (c_1^5, s_2, x_3).$$

These preferences may be explained by the fact that the decision maker is willing to pay a small amount of money (an increase from c_1^5 to c_1^4) to benefit from a smoking-free train car. However, the fare's difference between tickets from class c_1^5 and c_1^1 being too large, the decision maker would prefer the low-fare smoking car to the high-fare smoking-free car. In a TCP-net, the first of these two preferences would require the statement $1 \triangleright 2$ but this would induce the preference $(c_1^1, \bar{s}_2, x_3) \succ (c_1^5, s_2, x_3)$, an undesired collateral effect. This shows a descriptive limit intrinsic to TCP-nets: it is not possible to represent different degrees of preference between the attribute values.

We can now highlight the most distinguishable features of GAI networks:

- the descriptive power of GAI networks is not limited to preferences *ceteris paribus*. They can describe every total weak order over a product set, which is not the case for a CP-net or a TCP-net.
- the local utility tables in a GAI network allow us to express the intensity of preferences for the attribute values, which is not possible with a CP-net or a TCP-net.

However, note that unless utility functions map to a partially ordered set (a case we do not consider in this paper), GAI-nets cannot be used to describe partial preference structures, whereas CP-nets are able to represent some of them. In this respect, these preference representations are complementary. In addition, the preference structures representable by CP-nets being simpler than those representable by GAI-nets, the former are easier to elicit than the latter. In practice, the choice of the appropriate representation depends highly on factors such as how much information on the user's preferences is available, the level of sophistication the recommendations need to reach, or how much time the user is willing to spend to elicit her preferences.

3 Optimal Choice

Several types of queries are of interest for recommendation tasks based on preferences represented by GAI networks, in particular:

- *global choice queries*: find the preferred tuple x^* over $\mathcal{X}$;
- *constrained choice queries*: find the preferred tuple x^* over $\mathcal{X}$ given that some attributes are fixed at some specific values;
- *comparison queries*: find which tuple among a given pair $(x, y) \in \mathcal{X} \times \mathcal{X}$ is preferred by the DM;
- *ranking queries*: give a list of the k preferred tuples in $\mathcal{X}$.

Note that the second type of query is a special case of the first one. The third type of query is not critical from a computational point of view as a comparison query for a given pair (x, y) can be solved simply by computing and comparing $u(x)$ and $u(y)$. But the combinatorial nature of $\mathcal{X}$ prevents exhaustive pairwise comparisons, and so we need to address the choice and ranking queries. Determining the preferred tuple will be used as a preliminary step in the ranking procedure and is addressed in this section. Then in Section 4 we describe a general ranking procedure. The key idea is to take advantage of the structure of the GAI network to decompose the query problem into a sequence of local optimizations, hence keeping the computational cost of the overall ranking task at a very admissible level. For the clarity of the presentation, the procedure is first introduced on a small example and, then, a general algorithm is derived.

Example 3 : Consider a global choice query performed over a feasible set $\mathcal{X} = A \times B \times C \times D \times E \times F \times G$ with $A = \{a^0, a^1, a^2\}$, $B = \{b^0, b^1\}$, $C = \{c^0, c^1\}$, $D = \{d^0, d^1\}$, $E = \{e^0, e^1, e^2\}$, $F = \{f^0, f^1\}$, $G = \{g^0, g^1\}$. The DM's preferences are represented by a utility defined, for any tuple (a, b, c, d, e, f, g), by:

$$u(a, b, c, d, e, f, g) = u_1(a, b) + u_2(c, e) + u_3(b, c, d) + u_4(b, d, f) + u_5(b, g),$$

where the u_i's are given in Figure 2. Remark that utility u is completely characterized by only 32 integers whereas storing u in extension requires $|\mathcal{X}| = 288$ integers. When attributes are continuous, the tables of Figure 2 are substituted by functions providing an analytical representation of u. Figure 1 depicts the GAI network representing u's decomposition.

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

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

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

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

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

Figure 2: Utility tables for $u(\cdot)$

Finding the most preferred item is equivalent to solving $\max_{a,b,c,d,e,f,g} u_1(a, b) + u_2(c, e) + u_3(b, c, d) + u_4(b, d, f) + u_5(b, g)$. This can be efficiently performed by exploiting the properties below:

1. the max over variables $X_1, \ldots, X_n$ of $u(X_1, \ldots, X_n)$, can be decomposed as $\max_{X_1} \max_{X_2} \ldots \max_{X_n} u(X_1, \ldots, X_n)$, and the order in which the max's are performed is not important;
2. if $u(X_1, \ldots, X_n)$ can be decomposed as $f() + g()$ where $f()$ does not depend on X_i, then $\max_{X_i}[f() + g()] = f() + \max_{X_i} g()$;
3. in a GAI-net, the running intersection ensures that a variable contained in an outer clique X_C (i.e. a clique with at most one neighbor) but not contained in X_C's neighbor cannot appear in the rest of the net.

Properties 2 and 3 suggest computing the max recursively by first maximizing over the variables contained only in the outer cliques as only one factor is involved in these computations, then adding the result to the factor of their adjacent clique, remove these outer cliques and iterate until all cliques have been removed. In our example, this corresponds to solving the expression:

$$\begin{aligned} \max_{b,c,d}[\ & u_3(b, c, d) + \max_f[u_4(b, d, f) + \max_g u_5(b, g)] + [\max_e u_2(c, e)] \\ & + [\max_a u_1(a, b)]] \end{aligned} \tag{1}$$

by performing the following operations:

1. on clique AB, compute $u_1^*(b) = \max_{a \in A} u_1(a, b)$ for all $b \in B$;
2. on clique CE, compute $u_2^*(c) = \max_{e \in E} u_2(c, e)$ for all $c \in C$;
3. on clique BG, compute $u_5^*(b) = \max_{g \in G} u_5(b, g)$ for all $b \in B$;
4. on clique BDF, substitute $u_4(b, d, f)$ by $u_4(b, d, f) + u_5^*(b)$ for all $(b, d, f) \in B \times D \times F$. Then, compute $u_4^*(b, d) = \max_{f \in F} u_4(b, d, f)$ for all $(b, d) \in B \times D$;
5. on clique BCD, substitute $u_3(b, c, d)$ by $u_3(b, c, d) + u_1^*(b) + u_2^*(c) + u_4^*(b, d)$ for all $(b, c, d) \in B \times C \times D$. Then, compute $\max_{b,c,d} u_3(b, c, d)$, the maximal utility in the GAI-net (34, in the example).

The contents of the u_i^*'s and u_i's after substitution are given in Figure 3. At the end of step 5, we have precisely computed the value of Eq (1), and thus that of the optimum of the utility function (here 34).

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

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

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

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

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

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

Figure 3: Contents of the u_i^* and u_i after the substitutions

The above computations allow a fast determination of the optimal choices: first note that 34, the optimal value computed at step 5, corresponds to $u_3(b, c, d) = (b^1, c^0, d^0)$. Hence, at the optimal choice, $(B, C, D) = (b^1, c^0, d^0)$. Now $u_4^*(b^1, d^0)$, computed at step 4, was equal to $u_4(b^1, d^0, f^1) = 14$, hence at the optimal choice $F = f^1$. Then, on step 3, $u_5^*(b^1) = 6$ was equal to $u_5(b^1, g^0)$. Hence at the optimal choice $G = g^0$. Thus, had we saved in the 5 steps described above, the argmax's of the u_i^*'s, that is, had we computed:

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

then it would have been sufficient to iterate back from step 5 to step 1, and extract from the M_i^*'s the optimal values of the attributes conditionally to the values of the attributes selected previously. In our example, this would correspond to:

- *Step 5back:* $\text{Argmax}_{b,c,d} u_3(b,c,d) = (b^1, c^0, d^0)$;
- *Step 4back:* $M_4^*(b^1, d^0) = \text{Argmax}_{f \in F} u_4(b^1, d^0, f) = f^1$;
- *Step 3back:* $M_5^*(b^1) = \text{Argmax}_{g \in G} u_5(b^1, g) = g^0$;
- *Step 2back:* $M_2^*(c^0) = \text{Argmax}_{e \in E} u_2(c^0, e) = e^0$;
- *Step 1back:* $M_1^*(b^1) = \text{Argmax}_{a \in A} u_1(a, b^1) = a^2$.

The optimal choice is thus $x^* = (a^2, b^1, c^0, d^0, e^0, f^1, g^0)$. ♦

The whole process can be performed using function `Optimal_choice` below, which is essentially similar to that used for most probable explanations in Bayesian networks [14, 8]. It uses function `Collect` to compute Steps 1 to 5, and then, `Instantiate`, to perform Steps 5back to 1back. Note that, in function `Optimal_choice`, the value x_{C_k} passed as third argument to function `Instantiate` is an arbitrary value of clique X_{C_k}.

Function `Optimal_choice`(GAI-net)
01 Let X_{C_k} be any clique in the GAI-net
02 call `Collect`(X_{C_k}, X_{C_k}) and, then, call `Instantiate`$(X_{C_k}, X_{C_k}, x_{C_k}, \emptyset)$
03 return the $x^*_{C_i}$'s computed by `Instantiate`, which, together, constitute the optimal choice x^*

Function `Collect` performs the collect of data from X_{C_i}'s neighbors toward clique X_{C_i}, while avoiding clique X_{C_r}. This prevents from looping infinitely on lines 01–06. Note that, to perform the computations of Step 1 through Step 5, in this precise order, the recursive calls to function `Collect` must be done in the reverse order w.r.t. Steps 1–5, that is, `Collect` should first be called passing BCD as first argument, which, in turn, should call `Collect` of BDF, CE and AB, and `Collect` of BDF should call `Collect` of BG.

Function `Collect`(X_{C_i}, X_{C_r})
01 **for all** cliques $X_{C_j} \in \{\text{cliques adjacent to } X_{C_i}\} \backslash \{X_{C_r}\}$ **do**
02 call `Collect` (X_{C_j}, X_{C_i})
03 **for all** $x_{C_i} \in X_{C_i}$ **do**
04 $u_i(x_{C_i}) \leftarrow u_i(x_{C_i}) + u_j^*(x_{S_j})$, where x_{S_j} is the projection of x_{C_i} on $X_{C_i \cap C_j}$
05 **done**
06 **done**
07 **if** $X_{C_i} \neq X_{C_r}$ **then**
08 **for all** $x_{C_i \cap C_r} \in X_{C_i \cap C_r}$ **do**
09 $u_i^*(x_{C_i \cap C_r}) \leftarrow \max_{x_{C_i \backslash C_r}} u_i(x_{C_i})$
10 **done**
11 **endif**

Function `Instantiate` performs the instantiation of the optimal values to the attributes. As for `Collect`, its second argument prevents infinite loops 08–10. Argument $x^*_{C_r}$ contains the optimal values of the attributes in X_{C_r}. Finally, *Forbidden* is

a set of forbidden configurations of clique X_{C_i} that will be useful in the next section. Of course, as Steps 5back to 1back instantiate cliques in the reverse order w.r.t. the order in which cliques are examined in Steps 1 to 5, function `Instantiate` should perform its recursive exploration of the GAI-net in the same order as function `Collect` performed its own recursive calls.

Function `Instantiate`(X_{C_i}, X_{C_r}, $x^*_{C_r}$, *Forbidden*)
01 **if** $X_{C_i} = X_{C_r}$ **then**
02 $\quad x^*_{C_i} \leftarrow \text{Argmax}\{u_i(x_{C_i}) : x_{C_i} \in X_{C_i} \setminus \mathit{Forbidden}\}$
03 **else**
04 $\quad S_i \leftarrow C_i \cap C_r$; $D_i \leftarrow C_i \backslash S_i$
05 $\quad x^*_{S_i} \leftarrow$ the projection of $x^*_{C_r}$ over X_{S_i}
06 $\quad x^*_{C_i} \leftarrow \text{Argmax}\{u_i(x^*_{S_i}, x_{D_i}) : x_{D_i} \in X_{D_i}$ and $(x^*_{S_i}, x_{D_i}) \notin \mathit{Forbidden}\}$
07 **endif**
08 **for all** cliques X_{C_j} in {cliques adjacent to X_{C_i}}$\backslash\{X_{C_r}\}$ **do**
09 $\quad$ call `Instantiate` (X_{C_j}, X_{C_i}, $x^*_{C_i}$, $\emptyset$)
10 **done**
11 return the values x^*_i's of the attributes found at the optimum

The computational complexity of the whole process is equal to the sum of the sizes of the cliques in the network, where the size of a clique is the product of the domain sizes of the variables contained in the clique. As a matter of fact, function `Collect` is called only once per clique X_{C_i}; each clique X_{C_i} computes on lines 08–10 a message to be sent to one of its separators, thus requiring $O(|X_{C_i}|)$ operations. Such messages are added to the factor stored into the clique on lines 03-05, again requiring $O(|X_{C_i}|)$ operations. Hence the complexity of function `Collect` is equal to the sum of the sizes of the cliques in the network. Similarly, function `Instantiate` is also called once per clique X_{C_i}. When *Forbidden* $= \emptyset$, finding the Argmax on line 02 can be performed in $O(1)$ provided each time a max is computed on line 09 of function `Collect`, the value of the related tuple is kept in an Argmax table. When *Forbidden* $= \emptyset$, computing the projection $x^*_{S_i}$ of $x^*_{C_r}$ on line 05 and the Argmax on line 06 both require $O(|S_i|)$ operations, where $|S_i|$ is the number of variables contained in separator X_{S_i}. Consequently, the complexity of function `Instantiate` is equal to the sum of the number of variables stored in the separators. As these are smaller than the domain sizes of the variables contained in the cliques, the overall complexity of the choice procedure is that of `Collect`, hence the sum of the sizes of the cliques.

4 Ranking

Consider again the example of the preceding section and assume that `Optimal_choice` has returned tuple $x^* = (a^2, b^1, c^0, d^0, e^0, f^1, g^0)$. Then, the next best tuple, say x^2, differs from x^* by at least one attribute or, equivalently, there exists at least one clique X_{C_i} such that the projection of x^2 on X_{C_i} differs from that of x^*. As we do not know in which X_{C_i} the difference occurs, we cover all possibilities by dividing the space of remaining alternatives according to the partition scheme proposed in [17].

Set 1: $(B, C, D) \neq (b^1, c^0, d^0)$
Set 2: $(B, C, D) = (b^1, c^0, d^0)$ and $(B, D, F) \neq (b^1, d^0, f^1)$
Set 3: $(B, C, D, F) = (b^1, c^0, d^0, f^1)$ and $(B, G) \neq (b^1, g^0)$
Set 4: $(B, C, D, F, G) = (b^1, c^0, d^0, f^1, g^0)$ and $(C, E) \neq (c^0, e^0)$
Set 5: $(B, C, D, E, F, G) = (b^1, c^0, d^0, e^0, f^1, g^0)$ and $(A, B) \neq (a^2, b^1)$

Note that this splitting scheme is heavily related to the collect phase in that the order in which the cliques with new forbidden tuples are examined corresponds precisely to the order in which they are examined by the `Collect` function. Now, finding the optimal choice in Set 1 is equivalent to solving:

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

This suggests using function `Collect` as in the preceding subsection and then calling function `Instantiate`, taking care to prevent tuple (b^1, c^0, d^0) to be chosen in step 5back. This amounts to calling `Instantiate`$(BCD, BCD, (b^1, c^0, d^0), \{(b^1, c^0, d^0)\})$. Similarly, finding the optimal choice in Set 2 is equivalent to solving:

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

This is achieved by first calling function `Collect` as previously and, then, performing `Instantiate`$(BDF, BCD, (b^1, c^0, d^0), \{(b^1, d^0, f^1)\})$. Clearly, the last call will instantiate only cliques BDF and BG, as the other cliques cannot be reached by the function because, due to the fact that BCD is passed as second argument, line 08 of `Instantiate` will prevent any recursive call passing through clique BCD. To constitute the tuple we look for, we will just have to assign to the attributes of these "unreachable" cliques the values they had at the optimal solution. Thus `Instantiate`$(BDF, BCD, (b^1, c^0, d^0), \{(b^1, d^0, f^1)\})$ is precisely what is needed to determine the optimal choice in Set 2. Of course, this generalizes to the other sets and, in our example, this results in:

Set 1: `Instantiate`$(BCD, BCD, (b^1, c^0, d^0), \{(b^1, c^0, d^0)\})$, $u(x) = 33$,
$x = (a^0, b^0, c^0, d^1, e^0, f^1, g^1)$,
Set 2: `Instantiate`$(BDF, BCD, (b^1, c^0, d^0), \{(b^1, d^0, f^1)\})$, $u(x) = 31$,
$x = (a^2, b^1, c^0, d^0, e^0, f^0, g^0)$,
Set 3: `Instantiate`$(BG, BDF, (b^1, d^0, f^1), \{(b^1, g^0)\})$, $u(x) = 32$,
$x = (a^2, b^1, c^0, d^0, e^0, f^1, g^1)$,
Set 4: `Instantiate`$(CE, BCD, (b^1, c^0, d^0), \{(c^0, e^0)\})$, $u(x) = 33$,
$x = (a^2, b^1, c^0, d^0, e^2, f^1, g^0)$,
Set 5: `Instantiate`$(AB, BCD, (b^1, c^0, d^0), \{(a^2, b^1)\})$, $u(x) = 30$
$x = (a^1, b^1, c^0, d^0, e^0, f^1, g^0)$.

The second preferred tuple is thus the optimal choice of Set 1 or that of Set 4. Assume we select the former. Then the next tuple, x^3, is the preferred tuple that is different from both $(a^2, b^1, c^0, d^0, e^0, f^1, g^0)$ and $(a^0, b^0, c^0, d^1, e^0, f^1, g^1)$. It can be

obtained using the same process. As x^2 is in Set 1, we should substitute Set 1 by the sets below to exclude x^2:

Set 1.1: $(B, C, D) \notin \{(b^1, c^0, d^0), (b^0, c^0, d^1)\}$
Set 1.2: $(B, C, D) = (b^0, c^0, d^1)$ and $(B, D, F) \neq (b^0, d^1, f^1)$
Set 1.3: $(B, C, D, F) = (b^0, c^0, d^1, f^1)$ and $(B, G) \neq (b^0, g^1)$
Set 1.4: $(B, C, D, F, G) = (b^0, c^0, d^1, f^1, g^1)$ and $(C, E) \neq (c^0, e^0)$
Set 1.5: $(B, C, D, E, F, G) = (b^0, c^0, d^1, e^0, f^1, g^1)$ and $(A, B) \neq (a^0, b^0)$

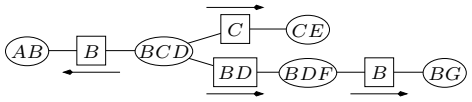

Figure 4: Propagation performed by `Instantiate`

and, then, iterate the same process. Note that whenever `Instantiate` is called, it instantiates the attributes toward the "outer" cliques of the GAI tree, that is, on Figure 4, it can easily be shown that `Instantiate` modifies only the values of the cliques that can be reached following the arrows. For instance, after calling `Instantiate`$(BDF, BCD, (b^1, c^0, d^0), \{(b^1, d^0, f^1)\})$, only the values of BDF and BG can change. Similarly, after calling `Instantiate`$(AB, BCD, (b^1, c^0, d^0), \{(a^2, b^1)\})$, only the value of clique AB can change. As a consequence, the splitting algorithm need only create one new set for each clique encountered following the arrows of Figure 4. Note that, on this figure, to each arrow from a clique X_{C_j} to another clique X_{C_i} corresponds, during the collect phase, a call by `Collect`$(X_{C_j}, \cdot)$ to `Collect`(X_{C_i}, X_{C_j}). It is convenient to have a special notation to represent these arrows: let $X_{C_{n(i)}}$ denote the clique at the tail of the arrow directed toward clique X_{C_i}. As propagations of instantiations are performed in a unique way, each set of the current space partition can be defined uniquely and unambiguously as a quadruple of the form: $(X_{C_i}, \mathit{Forbidden}_{C_i}, z^*, u^*)$, where X_{C_i} corresponds to the clique on which exclusion constraints are set, $\mathit{Forbidden}_{C_i}$ denotes the set of $|C_i|$-tuples already forbidden for the clique X_{C_i}, z^* is an optimal tuple in the set we consider, and u^* is the value of the utility at z^*. For instance, Sets 1 to 5 above can be represented as:

Set 1: $(BCD,\ \{(b^1, c^0, d^0)\},\ (a^0, b^0, c^0, d^1, e^0, f^1, g^1),\ 33)$,
Set 2: $(BDF,\ \{(b^1, d^0, f^1)\},\ (a^2, b^1, c^0, d^0, e^0, f^0, g^0),\ 31)$,
Set 3: $(BG,\ \{(b^1, g^0)\},\ (a^2, b^1, c^0, d^0, e^0, f^1, g^1),\ 32)$,
Set 4: $(CE,\ \{(c^0, e^0)\},\ (a^2, b^1, c^0, d^0, e^2, f^1, g^0),\ 33)$,
Set 5: $(AB,\ \{(a^2, b^1)\},\ (a^1, b^1, c^0, d^0, e^0, f^1, g^0),\ 30)$.

The slicing of the above sets can be performed by the function `Split` below: when a set needs be splitted, as for instance Set 1, function `Split` must be called passing as parameters the clique X_{C_i} on which new forbidden constraints are added, the set of all forbidden $|C_i|$-tuples of this clique, the optimal tuple of the set, and a Boolean set to `true`. This Boolean is needed only to make a difference between the first call to function `Split` and the other recursive calls.

Function `Split`(X_{C_i}, $Forbidden_{C_i}$, y^*, *is_root*)
01 **if** *is_root* = true **then**
02 $\quad Forbidden_{C_i} \leftarrow Forbidden_{C_i} \cup \{y^*_{C_i}\}$
03 **else**
04 $\quad Forbidden_{C_i} \leftarrow \{y^*_{C_i}\}$
05 **endif**
06 $z^* \leftarrow$ `Instantiate`$(X_{C_i}, X_{C_{n(i)}}, y^*_{C_{n(i)}}, Forbidden_{C_i})$
07 complete the missing attributes in z^* with their values in y^* to form a tuple on $\mathcal{X}$
08 $u^* \leftarrow u(z^*)$
09 $\mathcal{S} \leftarrow \{(X_{C_i}, Forbidden_{C_i}, z^*, u^*)\}$
10 **for all** cliques X_{C_j} in {cliques adjacent to X_{C_i}}\{$X_{C_{n(i)}}$} **do**
11 $\quad \mathcal{S} \leftarrow \mathcal{S} \cup$ `Split`$(X_{C_j}, \emptyset, y^*, \text{false})$
12 **done**
13 return $\mathcal{S}$

Using function `Split`, we can now define our ranking procedure `k-best`:

Function `k-best`($\mathcal{G}$, K)
01 $x^* \leftarrow$ `Optimal_choice`$(\mathcal{G})$
02 $\mathcal{S} \leftarrow$ `Split`$(X_{C_k}, \emptyset, x^*, \text{true})$
03 $i \leftarrow 1$
04 **while** $i < K$ and $\mathcal{S} \neq \emptyset$ **do**
05 $\quad S \leftarrow$ the element $(X_{C_j}, Forbidden_{C_j}, z^*, u^*)$ of $\mathcal{S}$ with the highest u^* value
06 $\quad x^i \leftarrow z^*$
07 $\quad \mathcal{S} \leftarrow \mathcal{S} \cup$ `Split`$(X_{C_j}, Forbidden_{C_j}, z^*, \text{true}) \setminus \{S\}$
08 $\quad i \leftarrow i + 1$
08 **done**
09 return $(x^*, x^1, \ldots, x^{i-1})$

4.1 Integrating constraints

Until now, we have considered that all the configurations in the Cartesian product $\mathcal{X}$ were possible. However, in many situations, some configurations are not allowed or simply not available. These constraints may be directly integrated into the GAI model by adding new subutility factors with utility value 0 for possible configurations and $-\infty$ otherwise. As such, the utility function plays the role of a member function in a soft CSP setting [16, 20]: the preferences may be seen as soft constraints and the feasibility constraints as hard constraints. Suppose that the pairs (a^0, e^1), (a^1, e^2), (a^2, e^1) and (a^2, e^2) are impossible. In this case we should add utility factor $u_6(a, e)$ defined by:

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

Similarly, if the pairs (e^0, f^0) and (e^1, f^1) are also forbidden, we should add utility factor $u_7(e, f)$ defined by:

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

After adding these new utility factors specifying the feasibility constraints, we need to recompile the GAI-net as it may no longer be a tree (if it is to satisfy the running intersection property). This is achieved by first constructing the Markov field of the utility, i.e., a graph in which i) the nodes correspond to the attributes, and ii) each pair of attributes belonging to some subutility factor are linked by an edge. This network is triangulated and the result is used to construct a join tree (as in the Bayes net literature [14]). The latter is the new GAI-net in which computations are performed. Figure 5 shows the new GAI-tree for our example after adding the utility factors u_6 and u_7.

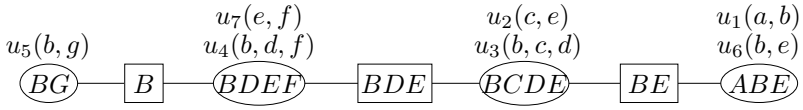

Figure 5: A GAI-Tree after integrating constraints

Remark that u_6 and u_7 create new dependencies between variables. This results in a tree with cliques larger than before, thus requiring more space to store the utility tables corresponding to the cliques. However it is possible to use a "*lazy*" implementation, in which the original utilities that should be in the larger cliques are not merged but stored in their original form using linked lists (in our example, the utility factors corresponding to each clique are pointed out in Figure 5). In this case, the storage space used by the newly triangulated GAI-net is roughly equivalent to that of the original net. However, optimal choice and ranking computations may be considerably longer than what they would be without the addition of the constraints.

The next section presents some experimental results for the determination of optimal choice and ranking with GAI-nets.

4.2 Experimental Results

To test the choice algorithms, computation times have been recorded for various randomly generated instances, both for the best element (choice) and for the top 50 and 100 elements (ranking). These instances were divided into 5 classes, $K_1, \ldots, K_5$, described in the table below and characterized by the number of attributes, the size of the attribute domains, the number of constraints in the problem, the size of the decomposition (i.e., the number of subutility factors in the GAI decomposition), and the arity of the subutility factors (and constraints):

	Attr. nb.	Dom. Size	Constr. nb.	Size of decomp.	Arity of sub.
K_1	40	2	5	5	4
K_2	35	5	30	30	2
K_3	15	10	20	20	2
K_4	30	5	5	5	4
K_5	10	10	5	5	3

In each class, 50 instances were randomly generated with various tightness indices (i.e., the ratio of admissible tuples by constraint). For each category, every instance was first processed without including the constraints into the GAI network, so as to measure the computation time on a complete product set. Then, in a second stage, constraints were included into the network, that is, additional utility factors representing the constraints were added to the GAI-decomposition, a new GAI-net was then derived after triangulation of the corresponding Markov network, and the query was answered using this new GAI-net. Our experiments were performed with a Java program on a 2.4GHz PC-computer. The average times over the 50 instances are summarized in the table below. For comparison, this table also shows the execution times for finding the best element using *Toolbar*[4]. Toolbar is a freely available reference software (implemented in C language) in the soft constraint domain [9]. However, Toolbar does not allow us to enumerate the k-best solutions. For this reason we only used it to find the best element (note that the obvious approach of introducing a constraint over all attributes that forbids an already generated solution was infeasible because it caused a tremendous increase in memory requirement).

Time	Without constraints				With constraints			
(in ms)	TBBest	Best	Top 50	Top 100	TBBest	Best	Top 50	Top 100
K_1	6153	8	44	82	2047	28	64	102
K_2	< 10	2	8	17	< 10	65	100	133
K_3	< 10	4	9	14	< 10	1272	1334	1365
K_4	35510	40	65	82	11257	2553	2647	2702
K_5	60	625	719	780	72	7858	7971	8062

As can be seen, solving choice or ranking problems on full product sets by GAI network-based algorithms is very efficient. When constraints are added, computation times remain satisfactory despite the additional complexity. In particular, it is remarkable that ranking elements from the 2nd to the 100th position is only marginally more time consuming than obtaining only the best element (this will be useful for the ranking approach for compromise search in the next section). The increase in computation times observed after adding n-ary constraints is due to the additional inter-attributes dependencies induced by them. The impact of the new inter-attributes dependencies during the triangulation stage amounts in worst cases to merge previously distinct cliques into one new big clique. The increase in the sizes of the cliques then results in an increase of the computation times for solving choice and ranking problems. However, in practical applications the arity of subutility factors is usually small (≤ 3 in most cases); the computation time may thus depend essentially on the constraint arity.

[4] `http://carlit.toulouse.inra.fr/cgi-bin/awki.cgi/ToolBarIntro`

5 Collective Choice

We consider now a multiperson decision making (MPDM) problem involving a set $\mathcal{A} = \{1, \ldots, m\}$ of agents. We assume that, for each agent $j \in \mathcal{A}$, a GAI utility u^j representing her preferences over $\mathcal{X}$ has been elicited. For simplicity, we assume that each agent uses the same absolute utility scale, so as to have commensurability. Then, a classical way of defining the best compromise solution for the group of agents is to define an overall utility $u(x)$ which gives, for any $x \in \mathcal{X}$, the value of x for the group. Thus we consider $u(x) = h(u^1(x), \ldots, u^m(x))$ where h is an aggregation function implicitly defining the type of compromise sought in $\mathcal{X}$. The best compromise solution is obtained by optimizing u over $\mathcal{X}$. When h is non-decreasing in each component, u-optimal solutions are weakly Pareto-optimal (i.e. there is no other solution improving the satisfaction of all the agents). Moreover, if h is strictly increasing in each component then u-optimal solutions are Pareto-optimal (i.e. there is no other solution improving the satisfaction of an agent without decreasing the satisfaction of another).

If h is linear, then u is the sum of GAI utilities and, as such, is itself a GAI utility. Thus the problem reduces to a monoagent decision problem with a GAI utility. However, linear aggregation functions are not good candidates as they may lead to choose a solution having a very ill-balanced utility profile. For instance, consider a problem with 3 agents and assume that $\mathcal{X} = \{x, y, z, w\}$ with:

$$\begin{array}{llll} u^1(x) = 0, & u^1(y) = 100, & u^1(z) = 100, & u^1(w) = 65, \\ u^2(x) = 100, & u^2(y) = 0, & u^2(z) = 100, & u^2(w) = 65, \\ u^3(x) = 100, & u^3(y) = 100, & u^3(z) = 0, & u^3(w) = 65. \end{array}$$

All solutions except w are unacceptable for at least one agent. Thus w is the only possible compromise solution; yet it cannot be obtained by maximizing a linear combination (with positive coefficients) of agent utilities. To find better compromise solutions, we shall use the weighted Tchebycheff norm, a standard scalarizing function allowing to reach any compromise solution within the set of Pareto optima [21]:

$$u(x) = \max_{j \in \mathcal{A}} \left\{ \omega_j (u^j(x^j) - u^j(x)) \right\}, \tag{2}$$

where $x^j = \text{Argmax}_{x \in \mathcal{X}} u^j(x)$. This criterion is minimized over $\mathcal{X}$. It represents the distance (w.r.t. a weighted Tchebycheff norm) between two utility profiles: $(u^1(x), \ldots, u^n(x))$ obtained with solution x, and the ideal utility profile $(u^1(x^1), \ldots, u^m(x^m))$ corresponding to a fictitious ideal situation (generally not feasible) in which all agents are optimally satisfied simultaneously. This ideal point is an upper bound of the set of Pareto non-dominated solutions. Coefficient ω_j is a weight attached to agent j. It makes it possible to modulate the importance of agents and to control the type of compromise. As mentioned above, as the Tchebycheff aggregation function is only non-decreasing in its arguments (i.e., it is not strictly increasing), in general, minimizing it over $\mathcal{X}$ only produces weak Pareto optimal solutions. However, as shown in [22], any Pareto optimal solution can be obtained by optimizing the Tchebycheff criterion with appropriate choices of ω_j.

5.1 The ranking approach for compromise search

The determination of the best compromise solution w.r.t. function u (Eq. (2)) is NP-hard as soon as there are $n \geq 3$ attributes, $m \geq 2$ agents, each having a GAI utility function including at least one factor of size greater than or equal to 3. This can be proved using a reduction from 3-SAT. Indeed, consider an instance of 3-SAT with n variables and m clauses. To each variable, we associate a Boolean attribute X_i and to any clause C_j over variables we associate an agent with Boolean function u^j. For instance $C_j = x \vee y \vee \neg z$ will be represented by function $u^j(x, y, z) = 1-(1-x)(1-y)z$. Hence, choosing $\omega_j = 1$ for all j's, the optimal value of the Tchebycheff optimization problem over $\mathcal{X} = X_1 \times \cdots \times X_n$ with functions $u^1, \ldots, u^m$ is 1 if and only if the initial 3-SAT problem is feasible. This shows the complexity of the search for a good compromise solution. To overcome the problem and be able to optimize such a non-decomposable function u on $\mathcal{X}$, we suggest resorting to a ranking approach based on the following 3-stage procedure:

Step 1: scalarization. we reformulate the problem as a monoagent problem, using an overall criterion $v(x) = 1/m \sum_j \omega_j(u^j(x^j) - u^j(x))$ defined as a linear combination of individual utilities. Such a function is easier to optimize than u since, as the sum of GAI functions, it is also a GAI function. Similarly to u, v is to be minimized.

Step 2: ranking. we enumerate the solutions of $\mathcal{X}$ by increasing value of v. Here, we use the ranking algorithm presented in section 4 which takes advantage of the decomposability of v.

Step 3: stopping condition. we need to stop enumeration as early as possible due to the size of set $\mathcal{X}$. This can be done efficiently using the following proposition.

Proposition 1 *Let $x^{(1)}, ..., x^{(k)}$ be the ordered sequence of k-best solutions generated during Step 2. If $v(x^{(k)}) \geq u(\hat{x}^{(k)})$ where $\hat{x}^{(k)} = Argmin_{i=1,\ldots,k} u(x^{(i)})$, then $\hat{x}^{(k)}$ is optimal for u, i.e. $u(\hat{x}^{(k)}) = \min_{x \in \mathcal{X}} u(x)$.*

Proof. Note that $u(x) \geq v(x)$ for all $x \in \mathcal{X}$. Hence, for all $i > k$, we have, by construction, $u(x^{(i)}) \geq v(x^{(i)}) \geq v(x^{(k)})$. Since $v(x^{(k)}) \geq u(\hat{x}^{(k)})$ by hypothesis, we get $u(x^{(i)}) \geq u(\hat{x}^{(k)})$ which shows that no solution found after step k in the ranking can improve the current best solution $\hat{x}^{(k)}$. □

Hence we can stop the enumeration whenever k is such that $v(x^{(k)}) \geq u(\hat{x}^{(k)})$.

5.2 Experimental Results

To evaluate our approach in practice, we have performed experiments on various instances of the MPDM problem. We have recorded computation times and the number of solutions generated before returning the optimal compromise solution using the Tchebycheff criterion. The experiments were performed on a 2.4GHz PC with a Java program.

To run the experiments, we generated synthetic data for GAI-decomposable preferences. All GAI decompositions involved 20 variables, with 10 subutilities $u_i(x_{C_i})$ of

domain size $|x_{C_i}|$ randomly drawn between 2 to 4. It does not seem realistic to consider higher-order interactions as far as human preference modeling is concerned (such complex interaction might actually be very difficult to assess in practice). Each u_i's domain variables were randomly selected from the set of all variables. For variables that were not selected in any subutility function, we created unary subutilities. Next, we created 5 different utility functions for the structure previously generated, representing the preferences of 5 agents. For each subutility function u_i^j of an agent j, we first generated its maximum value $\max(u_i^j)$, in the interval $[0, 1]$. Then we uniformly generated the utility values for all configurations of u_i^j in the interval $[0, \max(u_i^j)]$. This gave us 5 different GAI-decomposable utility functions with the same structure. We generated test data for variables of domain sizes 2, 5 and 10, resulting in problems with 2^{20}, 5^{20} and 10^{20} possible configurations respectively.

The average results (t: times in ms and $\#gen$: number of generated solutions) over 100 runs are summarized below (values within brackets are standard deviations):

Domain size	Tchebycheff criterion			
	t(ms)		$\#gen$	
2	37	(19)	1233	(863)
5	267	(178)	7268	(7040)
10	741	(384)	19956	(16338)

As can be seen, we obtained average times ranging from 0.03 to 0.7 seconds, depending on attribute domain size. Fortunately, the number of elements that need be enumerated before returning the solution increases at a much lower rate than the problem size. For instance, from 20 attributes of domain size 5 to 20 attributes of domain size 10, the number of configurations is multiplied by over 10^6 while, at the same time, the average number of solutions enumerated increases by a factor lower than 3. We also ran experiments where each agent had a different GAI decomposable preference structure. In these cases, to generate the aggregated GAI network we triangulated the Markov graph induced by the subutilities of all the agents. The more the discrepancy between the agents structures, the larger the cliques, and the less efficient our algorithm. Whenever the GAI network structures were very different, it turned out to be impossible to conduct the ranking procedure due to the large amount of memory required to fill the cliques. However, there are many practical situations where interacting attributes are almost identical for all agents, the difference between individual utilities being mainly due to discrepancies in utility values.

6 Conclusion

In this paper we have shown how GAI-networks could be used not only to efficiently perform individual recommendations (choice and ranking) on combinatorial sets, but also to solve collective recommendation requests for multiagent decision problems. The proposed procedure allows the determination of various types of compromise solutions and remains very efficient provided the number of agents is not too important. It might be used in many real-world situations like preference-based design of

an holidays-trip for a group, preference-based configuration of a car for the family, or for content-based movie recommendation tasks for a group of friends.

It is clear that for problems where there are hard constraints of great arity (e.g., knapsack problems), the solution consisting in including constraints as utility factors is not appropriate. In these cases it would be interesting to separate the hard constraints from the preferences, in order to keep the size of cliques manageable. The hard constraints should then be treated separately using tools developed in the CSP and operations research communities.

References

[1] F. Bacchus and A. Grove. Graphical models for preference and utility. In *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence (UAI'95)*, pages 3–10, Montreal, Canada, 1995.

[2] C. Boutilier, F. Bacchus, and R. Brafman. UCP-networks: A directed graphical representation of conditional utilities. In *Proceedings of the Seventeenth Conference on Uncertainty in Artificial Intelligence (UAI'01)*, pages 56–64, 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] R. Brafman and C. Domshlak. Introducing variable importance tradeoffs into CP-nets. In *Proceedings of the Eighteenth Conference on Uncertainty in Artificial Intelligence (UAI'02)*, pages 69–76, 2002.

[5] R. Brafman, C. Domshlak, and T. Kogan. Compact value-function representations for qualitative preferences. In *Proceedings of the Twentieth Conference on Uncertainty in Artificial Intelligence (UAI'04)*, pages 51–58, 2004.

[6] D. Braziunas and C. Boutilier. Local utility elicitation in GAI models. In *Proceedings of the Twenty First Conference on Uncertainty in Artificial Intelligence (UAI'05)*, pages 42–49, 2005.

[7] A. Chateauneuf and P. Wakker. From local to global additive representation. *Journal of Mathematical Economics*, 22:523–545, 1993.

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

[9] S. de Givry, M. Zytnicki, F. Heras, and J. Larrosa. Existential arc consistency: Getting closer to full arc consistency in weighted CSPs. In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI'05)*, pages 84–89, 2005.

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

[11] Y. Engel and M.P. Wellman. Generalized value decomposition and structured multiattribute auctions. In *Proceedings of the Eighth ACM Conference on Electronic Commerce (EC'07)*, pages 227–236. ACM Press, 2007.

[12] P. C. Fishburn. Interdependence and additivity in multivariate, unidimensional expected utility theory. *International Economic Review*, 8:335–342, 1967.

[13] C. Gonzales and P. Perny. GAI networks for utility elicitation. In *Proccedings of the Ninth International Conference on the Principles of Knowledge Representation and Reasoning (KR'04)*, pages 224–234, Whistler, BC, Canada, 2004.

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

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

[16] J. Larrosa, P. Meseguer, and T. Schiex. Soft constraint processing. In *Nineteenth International Joint Conference on Artificial Intelligence (IJCAI'05) Tutorial*, 2005.

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

[18] M.S. Pini, F. Rossi, K.B. Venable, and T. Toby Walsh. Aggregating partially ordered preferences: impossibility and possibility results. In *Proceedings of the Tenth Conference on Theoretical Aspects of Rationality and Knowledge (TARK'05)*, volume 10, pages 193–206, 2005.

[19] F. Rossi, K.B. Venable, and T. Toby Walsh. mCP nets: Representing and reasoning with preferences of multiple agents. In *Proceedings of the Nineteenth National Conference on Artificial Intelligence (AAAI'04)*, pages 729–734, 2004.

[20] T. Schiex, H. Fargier, and G. Verfaillie. Problèmes de satisfaction de contraintes valués. *Revue d'Intelligence Artificielle*, 11(3):339–373, 1997.

[21] R.E. Steuer and E.-U. Choo. An interactive weighted Tchebycheff procedure for multiple objective programming. *Mathematical Programming*, 26:326–344, 1983.

[22] A.P. Wierzbicki. On the completeness and constructiveness of parametric characterizations to vector optimization problems. *OR Spektrum*, 8:73–87, 1986.