Available online at www.sciencedirect.com
SCIENCE DIRECT®
Artificial Intelligence 170 (2006) 686–713
Artificial Intelligence
www.elsevier.com/locate/artint

Constraint-based optimization and utility elicitation using the minimax decision criterion ☆

Craig Boutilier [a,*], Relu Patrascu [a,1], Pascal Poupart [b,2], Dale Schuurmans [c,1]

[a] *Department of Computer Science, University of Toronto, Toronto, ON, M5S 3H5, Canada*
[b] *School of Computer Science, University of Waterloo, Waterloo, ON, Canada*
[c] *Department of Computing Science, University of Alberta, Edmonton, AB, T6G 2E8, Canada*

Received 15 August 2005; received in revised form 27 February 2006; accepted 27 February 2006

Available online 30 March 2006

Abstract

In many situations, a set of hard constraints encodes the feasible configurations of some system or product over which multiple users have distinct preferences. However, making suitable decisions requires that the preferences of a specific user for different configurations be articulated or *elicited*, something generally acknowledged to be onerous. We address two problems associated with preference elicitation: computing a best feasible solution when the user's utilities are imprecisely specified; and developing useful elicitation procedures that reduce utility uncertainty, with minimal user interaction, to a point where (approximately) optimal decisions can be made. Our main contributions are threefold. First, we propose the use of minimax regret as a suitable decision criterion for decision making in the presence of such utility function uncertainty. Second, we devise several different procedures, all relying on mixed integer linear programs, that can be used to compute minimax regret and regret-optimizing solutions effectively. In particular, our methods exploit generalized additive structure in a user's utility function to ensure tractable computation. Third, we propose various elicitation methods that can be used to refine utility uncertainty in such a way as to quickly (i.e., with as few questions as possible) reduce minimax regret. Empirical study suggests that several of these methods are quite successful in minimizing the number of user queries, while remaining computationally practical so as to admit real-time user interaction.

Keywords: Decision theory; Constraint satisfaction; Optimization; Preference elicitation; Imprecise utility; Minimax regret

☆ Parts of this article appeared in [C. Boutilier, R. Patrascu, P. Poupart, D. Schuurmans, Constraint-based optimization with the minimax decision criterion, in: Proc. of the Ninth Conference on the Principles and Practice of Constraint Programming (CP-2003), Kinsale, Ireland, 2003, pp. 168–182]; and [C. Boutilier, R. Patrascu, P. Poupart, D. Schuurmans, Regret-based utility elicitation in constraint-based decision problems, in: Proc. of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI-05), Edinburgh, Scotland, 2005, pp. 1293–1299].

* Corresponding author.
E-mail addresses: cebly@cs.toronto.edu (C. Boutilier), relu@cs.toronto.edu (R. Patrascu), ppoupart@cs.uwaterloo.ca (P. Poupart), dale@cs.ualberta.ca (D. Schuurmans).
URLs: http://www.cs.toronto.edu/~cebly(C. Boutilier), http://www.cs.toronto.edu/~relu(R. Patrascu), http://www.cs.uwaterloo.ca/~ppoupart(P. Poupart), http://www.cs.ualberta.ca/~dale(D. Schuurmans).

1 Part of this work was completed while the author was at the School of Computer Science, University of Waterloo.

2 Part of this work was completed while the author was at the Department of Computer Science, University of Toronto.

0004-3702/$ – see front matter
doi:10.1016/j.artint.2006.02.003

1. Introduction

The development of automated decision support software is a key focus within decision analysis [21,43,52] and artificial intelligence [9,10,16,17]. In the application of such tools, there are many situations in which the set of decisions and their effects (i.e., induced distribution over outcomes) are fixed, while the *utility functions* of different users vary widely. Developing systems that make or recommend decisions for a number of different users requires accounting for such differences in preferences. Several classes of methods have been employed by decision-support systems to "tune" their behavior appropriately, including inference or induction of user preferences based on observed behavior [28,33] or the similarity of a user's behavior to that of others (e.g., as in collaborative filtering [31]). Such behavior-based methods often require considerable data before strong conclusions can be drawn about a user's preferences (hence before good decisions can be recommended).

In many circumstances, direct *preference elicitation* may be undertaken in order to capture specific user preferences to a sufficient degree to allow an (approximately) optimal decision to be taken. In preference elicitation, the user is queried about her preferences directly. Different approaches to this problem have been proposed, including Bayesian methods that quantify uncertainty about preferences probabilistically [10,17,27], and methods that simply pose constraints on the set of possible utility functions and refine these incrementally [14,50,52].

In this paper, we will focus on direct preference elicitation for constraint-based optimization problems (COPs). COPs provide a natural framework for specifying and solving many decision problems. For example, configuration tasks [42] can naturally be viewed as consisting of a set of hard constraints (options available to a customer) and a utility function (reflecting customer preferences). While much work in the constraint-satisfaction literature has considered indirectly modeling preferences as hard constraints (with suitable relaxation techniques), more direct modeling of utility functions has come to be recognized as both natural and computationally effective. The direct or indirect modeling of multi-attribute utility functions has increasingly been incorporated into constraint optimization software.[3] Soft-constraint frameworks [8,46] that associate values with the satisfaction or violation of various constraints can also be seen as implicitly reflecting a user utility function.

However, the requirement of complete utility information demanded by a COP is often problematic. For instance, users may have neither the ability nor the patience to provide full utility information to a system. Furthermore, in many if not most instances, an optimal decision (or some approximation thereof) can be determined with a very partial specification of the user's utility function. As such, it is imperative that preference elicitation procedures be designed that focus on the relevant aspects of the problem. Preferences for unrealizable or infeasible outcomes are not (directly) relevant to decision making in a particular context; nor are precise preferences needed among outcomes that are provably dominated by others given the partial information at hand. Finally, though one could refine knowledge of a user's utility function with increased interaction, the elicitation effort needed to reach an optimal decision may not be worth the improvement in decision quality: often a near-optimal decision can be made with only a fraction of the information needed to make the optimal choice. Ultimately, it is the impact on decision quality that should guide elicitation effort [10,13,17,50].

The preference elicitation problem lies at the heart of considerable work in multiattribute utility theory [30,35,51] and the theory of consumer choice (such as conjoint analysis [29,48]), though our approach will differ considerably from classic models. Unfortunately, scant attention has been paid to preference elicitation in the constraint-satisfaction and optimization literature. Only recently has the problem of elicitation of objective functions been given due consideration [38,39].[4] While interactive preference elicitation has received little attention, optimizing with respect to a given set of preferences over configurations has been studied extensively. Branch-and-bound methods are commonly used for optimization in conjunction with constraint propagation techniques. A number of frameworks have also been proposed for modeling such systems using "soft constraints" of various types [8,46], each with an associated penalty or value that indirectly represent a user's preferences for different configurations.

In this paper, we adopt a somewhat different perspective from the usual soft constraints formalisms: we assume a user's preferences are represented directly as a utility function over possible configurations. Given a utility function and the hard constraints defining the decision space, we have a standard constraint-based optimization problem.

[3] For example, see www.ilog.com.

[4] Related, but of a decidedly different character is work on constraint acquisition [37]; more closely tied is work on learning soft constraints [41].

However, as argued earlier, it is unrealistic to expect users to express their utility functions with complete precision, nor will we generally require full utility information to make good decisions. Thus we are motivated to consider two problems, namely, the problem of "optimizing" in the presence of partial utility information, and the problem of effectively eliciting the most relevant utility information.

With respect to optimization in the presence of imprecise utility information, we suppose that a set of bounds (or more general linear constraints) on utility function parameters are provided (these constraints will arise as the result of the elicitation procedures we consider). We then consider the problem of finding a feasible solution that minimizes *maximum regret* [4,24,32,34,45] within the space of feasible utility functions. This is the solution we would regret the least should an adversary choose the user's utility function in a way that is consistent with our knowledge of the user's preferences. In a very strong sense, this minimizes the worst-case loss the user could experience as a result of our recommendation. We show that this minimax problem can be formulated and solved using a series of linear integer programs (IPs) and mixed integer programs (MIPs) in the case where utility functions have no structure.

In practice, some utility structure is necessary if we expect to solve problems of realistic size. We therefore also consider problems where utility functions can be expressed using a *generalized additive form* [3,22,23], which includes linear utility functions [30] and factored (or graphical) models like UCP-nets [12] as special cases. We derive two solution techniques for solving such structured problems: the first gives rise to a MIP with fewer variables, combined with an effective constraint generation procedure; the second encodes the entire minimax problem as a single MIP using a cost-network to formulate a compact set of constraints. The former method is shown to be especially effective.

If our knowledge of the utility parameters is loose enough, minimax regret may be unacceptably high, in which case we would like to query the user for additional information about her utility function. In this work we consider *bound queries*—a local form of *standard gamble queries* [24] that provide tighter upper or lower bounds on the utility parameters—and *comparison queries*, that present outcomes to the user for ranking. However, we focus on bound queries in our experiments. We develop several new strategies for eliciting bound information, strategies whose aim is to reduce the worst-case error (i.e., get guaranteed improvement in decision quality) with as few queries as possible. Our first strategy, *halve largest gap* (HLG), provides the best theoretical guarantees—it works by providing uniform uncertainty reduction over the entire utility space. The HLG strategy is similar to heuristically motivated polyhedral methods in conjoint analysis, used in product design and marketing [29,48]. In fact, HLG can be viewed as a special case of the method of [48] in which our polyhedra are hyper-rectangles. Our second strategy, *current solution* (CS), is more heuristic in nature, and focuses attention on *relevant* aspects of the utility function. Our empirical results show that this strategy works much better in practice than HLG, and does indeed distinguish relevant from irrelevant queries. Furthermore, its ability to discern good queries is also largely unaffected by approximation: the anytime nature of minimax computation allows time bounds to be used to ensure real-time response with little impact on the elicitation effort required. We also introduce several additional strategies which capture some of the same intuitions as HLG and CS, but with different computational procedures (and complexity). Among these, the *optimistic-pessimistic (OP)* method works almost as well as CS, having much lower computational demands, but without providing the same strong guarantees on decision quality.

The remainder of the paper is organized as follows. In Section 2 we briefly review constraint-based optimization with factored utility models. We define and motivate minimax regret for decision making with imprecisely specified utility functions in Section 3. In Section 4 we describe several methods for computing minimax regret for COPs, and evaluate one such method empirically. We also suggest several computational shortcuts well-suited to the interactive elicitation context. We define a number of elicitation strategies in Section 5 and provide empirical comparisons of these strategies, both computationally and with respect to number of queries required to reach an optimal solution or an acceptable level of regret. We conclude in Section 6 with a discussion of future research directions.

2. Constraint-based optimization and factored utility models

We begin by describing the basic problem of constraint-based optimization assuming a known utility function and also describe the use of factored utility models in COPs. This will establish background and notation for the remainder of the paper.

2.1. Optimization with known utility functions

We assume a finite set of attributes $\mathbf{X} = \{X_1, X_2, \ldots, X_N\}$ with finite domains $Dom(X_i)$. An assignment $\mathbf{x} \in Dom(\mathbf{X}) = \Pi_i Dom(X_i)$ is often referred to as a *state*. For simplicity of presentation, we assume these attributes are Boolean, but nothing important depends on this (indeed, our experiments will involve primarily non-Boolean attributes). We assume a set of hard constraints $\mathcal{C}$ over these attributes. Each constraint $\mathcal{C}_\ell$, $\ell = 1, \ldots, L$, is defined over a set $\mathbf{X}[\ell] \subseteq \mathbf{X}$, and thus induces a set of legal configurations of attributes in $\mathbf{X}[\ell]$. More formally, $\mathcal{C}_\ell$ can be viewed as the subset of $Dom(\mathbf{X}[\ell])$ from which all feasible configurations must be constructed. We assume that the constraints $\mathcal{C}_\ell$ are represented in some logical form and can be expressed compactly; for example, we might write

$$(X_1 \wedge X_2) \supset \neg X_3$$

to denote that all legal configurations of Boolean variables X_1, X_2, X_3 are such that X_3 must be false if X_1 and X_2 are both true. We let $Feas(\mathbf{X})$ denote the subset of *feasible states* (i.e., assignments satisfying $\mathcal{C}$). The *constraint satisfaction problem* is that of finding a feasible assignment to a specific set of constraints. While the set of states $Dom(\mathbf{X})$ is exponential in N (the number of variables), logically expressed constraints allow us to specify the feasible set $Feas(\mathbf{X})$ compactly, and makes explicit problem structure that can be exploited to (often) effectively solve CSPs. We refer to Dechter [20] for a detailed overview of models and methods for constraint satisfaction.

The *constraint graph* for a given set of constraints is the undirected graph whose nodes are attributes and whose edges connect any two attributes that occur in the same constraint. This graph has useful properties that can be used to determine the worst-case complexity of various algorithms for constraint-satisfaction and constraint-based optimization [20]. Fig. 1 illustrates a small constraint graph over four variables induced by the logical constraints shown.

Our focus here will not be on constraint satisfaction, but rather on *constraint-based optimization problems* (*COPs*).[5] Suppose we have a known non-negative *utility function* $u : Dom(\mathbf{X}) \rightarrow \mathbf{R}^+$ which ranks all states (whether feasible or not) according to their degree of desirability.[6] Our aim is to find an optimal feasible state $\mathbf{x}^*$; that is, any

$$\mathbf{x}^* \in \arg \max_{\mathbf{x} \in Feas(\mathbf{X})} u(\mathbf{x}).$$

Since choices are restricted to feasible states, we sometimes call feasible states *decisions*. Without making any assumptions regarding the nature of the utility function (e.g., with regard to structure, independence, compactness, etc.) we can formulate the COP in an "explicit" fashion as a (linear) 0–1 integer program:

$$\max_{\{I_\mathbf{x}, X_i\}} \sum_{\mathbf{x}} u_\mathbf{x} I_\mathbf{x} \quad \text{subject to } \mathcal{A} \text{ and } \mathcal{C}, \tag{1}$$

where we have:

- variables $I_\mathbf{x}$: for each $\mathbf{x} \in Dom(\mathbf{X})$, $I_\mathbf{x}$ is a Boolean variable indicating whether $\mathbf{x}$ is the decision made (i.e., the feasible state chosen). In other words, $I_\mathbf{x} = 1$ iff $\mathbf{x}$ is the solution to the COP.

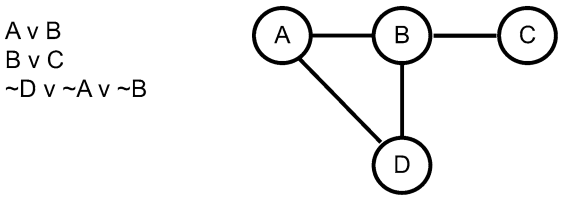

Fig. 1. An example constraint graph induced by the logical constraints shown to the left.

[5] We use the term *constraint-based optimization* (often called constraint optimization) to refer to discrete combinatorial optimization problems with explicit logical constraints over variable instantiations, as in CSPs, to distinguish these from more general *constrained optimization* problems with arbitrary (e.g., continuous) variables and general functional constraints.

[6] For ease of presentation, we assume the utility function has been normalized to be non-negative. Nothing critical depends on this, as such normalization is always possible [23].

- variables X_i: X_i is a Boolean variable corresponding to the ith attribute. In other words, variables $X_i = 1$ iff feature X_i is true in the solution to the COP.[7]
- coefficients $u_{\mathbf{x}}$: for each $\mathbf{x} \in Dom(\mathbf{X})$, constant $u_{\mathbf{x}}$ denotes the (known) utility of state $\mathbf{x}$.
- constraint set $\mathcal{A}$: for each variable $I_{\mathbf{x}}$, we impose a constraint that relates it to its corresponding variable assignment. Specifically, for each X_i: if X_i is true in $\mathbf{x}$, we constrain $I_{\mathbf{x}} \leqslant X_i$; and if X_i is false in $\mathbf{x}$, we constrain $I_{\mathbf{x}} \leqslant 1 - X_i$. We use $\mathcal{A}$ to denote this set of *state definition constraints*.
- constraint set $\mathcal{C}$: we impose each feasibility constraint $\mathcal{C}_\ell$ on the attributes $X_i \in \mathbf{X}[\ell]$. Logical constraints on the variables X_i can be written in a natural way as linear constraints [18], so we omit details.[8]

Note that this formulation ensures that, if there is a feasible solution (given the constraints $\mathcal{A}$ and $\mathcal{C}$), then exactly one $I_{\mathbf{x}}$ will be non-zero.

As an example of the interplay between the constraints in $\mathcal{A}$ and $\mathcal{C}$, consider the example in Fig. 1. The following constraints will be present in the set $\mathcal{A}$ for the state $abc\bar{d}$ (with analogous constraints for the other 15 instantiations of the four Boolean variables):

$$I_{abc\bar{d}} \leqslant A; \quad I_{abc\bar{d}} \leqslant B; \quad I_{abc\bar{d}} \leqslant C; \quad I_{abc\bar{d}} \leqslant 1 - D;$$

while the constraint set $\mathcal{C}$, capturing the three domain constraints in the figure are:

$$A + B \geqslant 1; \quad B + C \geqslant 1; \quad A + B + C \leqslant 2.$$

2.2. Factored utility models

Even with logically specified constraints, solving a COP in the manner above is not usually feasible, since the utility function is not specified concisely. As a result, the IP formulation above is not practical, since there is one $I_{\mathbf{x}}$ variable per state and the number of states is exponential in the number of attributes. With such *flat* utility functions, it is not generally possible to formulate the optimization problem concisely: indeed, if there is no (structural) relationship between the utility of different states, little can be done but to enumerate feasible states to ensure that an optimal solution is found. In contrast, if some structure is imposed on the utility function, say, in the form of a *factored* (or graphical) model, we are then often able to reduce the number of variables to be linear in the number of parameters of the factored model.

We consider here *generalized additive independence* (*GAI*) models [3,22], a natural, but flexible and fully expressive generalization of additive (or linear) utility models.[9] GAI is appealing because of its generality and expressiveness; for instance, it encompasses both linear models [30] and UCP-nets [12] as special cases,[10] but can capture any utility function. The advantage of structured utility models, and GAI specifically, is that the constraint-based optimization can be formulated and (typically) solved without explicit enumeration of states. While we focus on the GAI model, other compact structured utility models can be exploited in similar fashion (e.g., see the many such models proposed by Keeney and Raiffa [30]).

The GAI model assumes that our utility function can be written as the sum of K local utility functions, or *factors*, over small sets of attributes:

$$u(\mathbf{x}) = \sum_{k \leqslant K} f_k(\mathbf{x}[k]). \tag{2}$$

Here each function f_k depends only on a local family of attributes $\mathbf{X}[k] \subset \mathbf{X}$. We denote by $\mathbf{x}[k]$ the restriction of state $\mathbf{x}$ to the attributes in $\mathbf{X}[k]$. Again we assume that each function f_k is non-negative. For example, if $\mathbf{X} = \{A, B, C, D\}$ we might decompose a utility function as follows:

$$u(ABCD) = f_1(A, B) + f_2(B, C).$$

[7] If features are non-Boolean, then X_i simply needs to be a general integer variable ranging over $Dom(X_i)$.

[8] Again, generalizing the form of these constraints to generalized logical constraints involving non-Boolean attributes is straightforward.

[9] Fishburn [22] introduced the model, using the term *interdependent value additivity*; Bacchus and Grove [3] dubbed the same concept GAI, which seems to be more commonly used in the AI literature currently.

[10] For example, UCP-nets encompass GAI with some additional restrictions. Hence any algorithm for GAI models automatically applies to UCP-nets, though one might be able to exploit the structure of UCP-nets for *additional* computational gain.

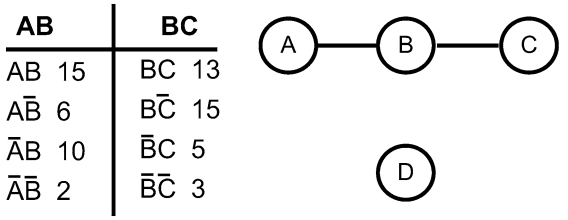

Fig. 2. An example utility graph induced by the two utility factors shown to the left.

Fig. 2 illustrates a possible instantiation of the two factors. Note that a user with such a utility function exhibits no preference for the different values of variable D.

The conditions under which a GAI model provides an accurate representation of a utility function were defined by Fishburn [22,23]. We provide the intuitions here. Let P be some probability distribution over $Dom(\mathbf{X})$, interpreted as a gamble or lottery presented to the decision maker. For instance, P may correspond to the distribution over outcomes induced by some decision she could take. Fishburn showed that the decision maker's utility function u could be written in GAI form over factors f_k iff she is indifferent between any pair of gambles P and Q over $Dom(\mathbf{X})$ whose marginals over each subset of variables $\mathbf{X}[k]$ ($k \leqslant K$) are identical. Note that GAI models are completely general (since any utility function can be represented trivially using a single factor consisting of all variables). Furthermore, linear models are a special case in which we have a singleton factor for each variable.

We refer to a pair $\langle \mathcal{C}, \{f_k\colon k \leqslant K\}\rangle$ as a *structured COP*, where $\mathcal{C}$ is a set of feasibility constraints and $\{f_k\colon k \leqslant K\}$ is a set of utility factors. An IP similar to Eq. (1) can be used to solve for the optimal decision in the case of a GAI model:

$$\max_{\{I_{\mathbf{x}[k]}, X_i\}} \sum_{k \leqslant K} \sum_{\mathbf{x}[k] \in Dom(\mathbf{X}[k])} u_{\mathbf{x}[k]} I_{\mathbf{x}[k]} \quad \text{subject to } \mathcal{A} \text{ and } \mathcal{C}. \tag{3}$$

Instead of one variable $I_{\mathbf{x}}$ per state, we now have a set of *local state variables* $I_{\mathbf{x}[k]}$ for each family k and each instantiation $\mathbf{x}[k] \in Dom(\mathbf{X}[k])$ of the variables in that family. Similarly, we have one associated constant coefficient $u_{\mathbf{x}[k]}$ denoting $f_k(\mathbf{x}[k])$. $I_{\mathbf{x}[k]}$ is true iff the assignment to $\mathbf{X}[k]$ is $\mathbf{x}[k]$. Each $I_{\mathbf{x}[k]}$ is related logically to the attributes $X \in \mathbf{X}[k]$ by (local) state definition constraints $\mathcal{A}$ as before, and the usual feasibility constraints $\mathcal{C}$ are also imposed as above.

Notice that the number of variables and constraints in this IP (excluding the exogenous feasibility constraints $\mathcal{C}$) is now *linear* in the number of parameters of the underlying utility model, which itself is linear in the number of attributes $|\mathbf{X}|$ if we assume that the size of each utility factor f_k is bounded. This compares favorably with the exponential size of the IP for unfactored utility models in Section 2.1.

This formulation of COPs is more or less the same as several other models of COPs found in the constraint satisfaction literature. For example, the notion of a *cost network* is often used to represent objective functions by assuming a similar form of factored decomposition of the objective function [20]. Specifically, let the *utility graph* be defined in the same fashion as the constraint graph, but with edges connecting attributes that occur in the same utility factor. This utility graph can be viewed as a cost network. Fig. 2 illustrates the utility graph over variables $\{A, B, C, D\}$ induced by the utility function decomposition described above (corresponding to the two utility factors AB and BC shown at the left of the figure).

Similarly, certain soft constraint formalisms can be used to represent COPs [8]. Every constraint is assigned a cost corresponding to a penalty incurred if that constraint is not satisfied. Hard constraints in $\mathcal{C}$ are assigned an infinite cost, and constraints corresponding to configurations of local utility factors are given a finite cost corresponding to their (negative) utility. The goal is to find a minimum cost state, with the infinite costs associated with hard constraints ensuring a preference for any feasible solution over any infeasible solution.

The IP formulation of structured COPs in Eq. (3) can be solved using off-the-shelf solution software for general IPs. Various special purposes algorithms that rely on the use of dynamic programming (e.g., the *variable elimination* algorithm) or constraint satisfaction techniques can also be used to solve these problems; we refer to Dechter [20] for a discussion of various algorithmic approaches to COPs developed in the constraint satisfaction literature. While many of these could be adapted to address the problems discussed in the paper, we will focus on IP formulations and their direct solution using standard IP software.

We remark here on the use of utility functions rather than (qualitative) preference rankings in this work. In a deterministic setting, such as in the constraint-based framework adopted here, one does not need utility functions to make decisions. Instead an ordinal preference ranking will suffice [30] since, without uncertainty, strength of preference information is not needed to assess tradeoffs: complete ordinal preference information will dictate which feasible outcome is preferred. However, in large multiattribute domains, even with factored preference models, full elicitation will often be time-consuming and unnecessary. As discussed, our aim is to make decisions with incomplete preference information. If we make a decision that is potentially suboptimal, we cannot be sure of its quality unless we have some information about the strength of preference of this decision (at least relative to the optimal decision). Simply knowing that one decision "may be preferred" to another does not give us enough information to know whether additional preference information should be elicited if we are content with making good, rather than, optimal decisions. It is impossible to say *how good* a non-optimal decision actually is without some quantitative utility information.

3. Minimax regret

In many circumstances, we are faced with a COP in which a user may not have fully articulated her utility function over configurations of attributes. This arises naturally when distinct configurations must be produced for users with different preferences, with some form of utility elicitation used to extract only a partial expression of these preferences. It will frequently be the case that we must make a decision before a complete utility function can be specified. For instance, users may have neither the ability nor the patience to provide full utility information to a system before requiring a decision to be recommended. Furthermore, in many if not most instances, an optimal decision (or some approximation thereof) can be determined with a very partial specification of the user's utility function. This will become evident in our preference elicitation framework and the models we consider in this paper.

If the utility function is unknown, then we have a slightly different problem than the standard COP. We cannot maximize utility (or expected utility in stochastic decision problems) because the utility function is incompletely specified. However, we will often have constraints on the utility function, either initial information about plausible utility values, or more refined constraints based on the results of utility elicitation with a specific user. For example, these might be bounds on the parameters of the utility model, or possibly more general constraints (as we discuss below). Given such a set of possible utility functions (namely those consistent with these constraints), we must adopt some suitable decision criterion for optimization, knowing only that the user's utility function lies within this set.

In the paper we propose the use of *minimax regret* [12,24,32,43,45,50] as a natural decision criterion for imprecise COPs. We first define the notion of minimax regret and then provide some motivation for its use as a suitable criterion in this setting.

3.1. Minimax regret in COPs

Minimax regret is a very natural criterion for decision making with imprecise or incompletely specified utility functions. It requires that one adopt the (feasible) assignment $\mathbf{x}$ with minimum *max regret*, where max regret is the largest amount by which one could *regret* making decision $\mathbf{x}$ (while allowing the utility function to vary within the bounds). It has been suggested as an alternative to classical expected utility theory. Specifically, it has been proposed as a means for accounting for uncertainty over possible states of nature (or the outcomes of decisions) [4,32,34,44], both when probabilistic information is unavailable, and as a descriptive theory of human decision making that explains certain common violations of von Neumann–Morgenstern [49] axioms. However, only recently has it been considered as a means for dealing with the utility function uncertainty a decision system may possess regarding a user's preferences [12,14,43,50]. It is this formulation we present here.

Formally, let $\mathbf{U}$ denote the set of feasible utility functions, reflecting our partial knowledge of the user's preferences. The set $\mathbf{U}$ may be finite; but more commonly it will be continuous, defined by bounds (or constraints) on (sets of) utility values $u(\mathbf{x})$ for various states. We refer to a pair $\langle \mathcal{C}, \mathbf{U} \rangle$ as an *imprecise COP*, where $\mathcal{C}$ is a set of feasibility constraints and $\mathbf{U}$ is the set of feasible utility functions. In the case where $\mathbf{U}$ is defined by a finite set of linear constraints $\mathcal{U}$, we sometimes abuse terminology by speaking of $\langle \mathcal{C}, \mathcal{U} \rangle$ as an imprecise COP.

We define minimax regret in stages:

Definition 1. The *pairwise regret* of state $\mathbf{x}$ with respect to state $\mathbf{x}'$ over feasible utility set $\mathbf{U}$ is defined as

$$R(\mathbf{x}, \mathbf{x}', \mathbf{U}) = \max_{u \in \mathbf{U}} u(\mathbf{x}') - u(\mathbf{x}). \tag{4}$$

Intuitively, $\mathbf{U}$ represents the knowledge a decision support system has of the user's preferences. $R(\mathbf{x}, \mathbf{x}', \mathbf{U})$ is the most the system could regret choosing $\mathbf{x}$ instead of $\mathbf{x}'$, if an adversary could impose any utility function in $\mathbf{U}$ on the user. In other words, if the system were forced to choose between $\mathbf{x}$ and $\mathbf{x}'$, then this corresponds to the worst-case loss associated with choosing $\mathbf{x}$ rather than $\mathbf{x}'$ with respect to possible realizations of $u \in \mathbf{U}$.

Definition 2. The *maximum regret* of decision $\mathbf{x}$ is:

$$MR(\mathbf{x}, \mathbf{U}) = \max_{\mathbf{x}' \in Feas(\mathbf{X})} R(\mathbf{x}, \mathbf{x}', \mathbf{U}) \tag{5}$$

$$= \max_{u \in \mathbf{U}} \max_{\mathbf{x}' \in Feas(\mathbf{X})} u(\mathbf{x}') - u(\mathbf{x}). \tag{6}$$

Since the goal of our decision support system is to make the optimal choice with respect to the user's true utility function, $MR(\mathbf{x}, \mathbf{U})$ is the most the system could regret choosing $\mathbf{x}$; that is, it is the worst-case loss associated with choosing $\mathbf{x}$ in the presence of an adversary who could choose the user's utility function to maximize the difference between $\mathbf{x}$ and acting optimally.[11]

Definition 3. The *minimax regret* of feasible utility set $\mathbf{U}$ is:

$$MMR(\mathbf{U}) = \min_{\mathbf{x} \in Feas(\mathbf{X})} MR(\mathbf{x}, \mathbf{U}) \tag{7}$$

$$= \min_{\mathbf{x} \in Feas(\mathbf{X})} \max_{u \in \mathbf{U}} \max_{\mathbf{x}' \in Feas(\mathbf{X})} u(\mathbf{x}') - u(\mathbf{x}). \tag{8}$$

A *minimax-optimal* decision $\mathbf{x}^*$ is any decision that minimizes max regret:

$$\mathbf{x}^* \in \arg \min_{\mathbf{x} \in Feas(\mathbf{X})} MR(\mathbf{x}, \mathbf{U}).$$

If the only information we have about a user's utility function is that it lies in the set $\mathbf{U}$, then a decision $\mathbf{x}^*$ that minimizes max regret is very intuitive as we elaborate below. Specifically, without distributional information over the set of possible utility functions, choosing (or recommending) a minimax-optimal decision $\mathbf{x}^*$ minimizes the worst-case loss with respect to possible realizations of the utility function $u \in \mathbf{U}$. Our goal of course is to formulate the minimax regret optimization (Eq. (8)) in a computationally tractable way. We address this in Section 4.

To illustrate minimax regret, consider the example illustrated in Figs. 1 and 2; but suppose that the precise utility values associated with these factors are unknown, and instead replaced with the upper and lower bounds shown in Fig. 3. The problem admits five feasible states, and the pairwise max regret $R(\mathbf{x}, \mathbf{x}', \mathbf{U})$ for each pair of states (with $\mathbf{x}$ along columns and $\mathbf{x}'$ along the rows) is shown in Table 1. The max regret of each feasible state is shown in the final column, from which we see that state $AB\overline{C}$ is the minimax optimal decision.

AB	BC
AB [9,17]	BC [2,4]
$A\bar{B}$ [5,8]	$B\bar{C}$ [0,10]
$\bar{A}B$ [10,11]	$\bar{B}C$ [4,8]
$\bar{A}\bar{B}$ [0,3]	$\bar{B}\bar{C}$ [0,5]

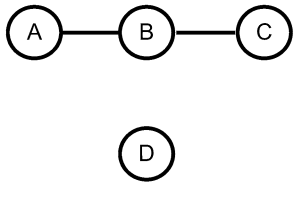

Fig. 3. Imprecisely specified utility factors (with upper and lower bounds provided for each parameter.

[11] Note that the $\mathbf{x}'$ chosen by the adversary for a specific u will always be the optimal decision under u: any other choice would give the adversary lesser utility and thus reduce regret.

Table 1

	ABC	$AB\overline{C}$	$A\overline{B}C$	$\overline{A}BC$	$\overline{A}B\overline{C}$	Max regret
ABC	0	8	5	2	10	10
$AB\overline{C}$	4	0	7	6	2	7
$A\overline{B}C$	12	18	0	6	12	18
$\overline{A}BC$	7	15	6	4	0	15
$\overline{A}B\overline{C}$	15	7	6	4	0	15

3.2. Motivation for minimax regret

As mentioned, minimax regret has been widely studied (and critiqued) as a means for decision making under uncertainty. It has been studied primarily as a means for making decisions when a decision maker is unwilling or unable to quantify her uncertainty over possible states of nature, or as a means of explaining violations of classical axioms of expected utility theory. Savage [44] introduced the notion though could not provide a "categorical" defense of its use. The use of regret, including the incorporation of feelings of regret (and its opposite, "rejoicing") into expected utility theory, has been proposed as a means of accounting for the manner in which people violate the axioms of expected utility theory in practice [4,34]. Difficulties with the use of this decision criterion include the fact that the minimax regret criterion does not satisfy the principle of irrelevant alternatives [24,44], and regret theory fails to satisfy a reasonable notion of stochastic dominance [40]. It can also be argued that, from a Bayesian perspective, a decision maker might as well construct their own subjective assessment of possible states of nature. For this reason, minimax regret is often viewed as too cautious a criterion.

The perspective we adopt here is somewhat different. We do not adopt minimax regret as a means of accounting for a decision maker's personal feelings of regret. Rather we define regret with respect to our *decision system's uncertainty* with respect to the user's true utility function. It seems incontrovertible that there will generally be some utility function uncertainty on the part of any system designed to make decisions on behalf of users. The only issue is how this uncertainty is represented and reasoned with.

Naturally, Bayesian methods may be entirely appropriate in some circumstances: if one can quantify uncertainty over possible utility functions probabilistically, then one can take expectations over this density to determine the *expected* expected utility of a decision [10,11,17]. However, there are many circumstances in which the Bayesian approach is inappropriate. First, it can often be very difficult to develop reasonable priors over the utility functions of a wide class of users. Furthermore, representing priors over such complex entities as utility functions is fraught with difficulty and inevitably requires computational approximations in inference (this is made abundantly clear in recent Bayesian approaches to preference elicitation [10,17]). As a consequence, the value of such "normatively correct" models is undermined in practice. Often bounds on the parameters of utility functions are generally much easier to come by, much easier to maintain, and lend themselves to much more computationally manageable algorithms as we will see in this paper. In addition, max regret provides an upper bound on the expected loss when probabilistic information is known. As we will see below, minimax regret is a very effective driver of preference elicitation, so concerns about its pessimistic nature are largely unfounded. We will see that with relatively few queries, max regret can be reduced to very low levels (often to the point of offering provably optimal solutions). Though we don't pursue this approach here, when probabilistic information is available, it can be combined rather effectively with minimax computation [50].

Finally, it is worth noting that making a recommendation whose utility is near optimal *in expectation*, as is the case in Bayesian models of preference elicitation, is often of cold comfort to a user when the decision made is *actually* very far from optimal. While minimax regret provides a worst-case bound on the loss in decision quality arising from utility function uncertainty (even in cases where distributional information is available), Bayesian methods cannot typically provide such a bound. In some contexts, such as procurement, this has been reported as a source of contention with clients using automated preference elicitation [14]. The argument is often made that users do not want to "leave money on the table" (even if the odds are low); if any money is left on the table, they want guarantees (as opposed to probabilistic assurances) that the amount they could have saved through further preference elicitation is limited.

Recently, a considerable amount of work in *robust optimization* has adopted the minimax regret decision criterion [1,2,32].[12] This work addresses combinatorial optimization problems with data uncertainty (e.g., shortest path problems or facility location with uncertain parameters) and find "robust deviation decisions" that minimize max regret. While the perspective in this work is somewhat different than that adopted in ours, the models and methods are quite similar. Our formulation is specific to the constraint-based optimization setting, but more importantly we focus on how minimax regret can be used to drive the process of elicitation, a problem not addressed systematically in the robust optimization literature. Our techniques for preference elicitation could in fact be adapted for problems in robust optimization as a means to drive the reduction in data uncertainty.

4. Computing minimax regret in COPs

We address the computational problem of computing minimax optimal decisions in several stages. We initially assume upper and lower bounds on utility parameters and discuss procedures for minimax computation for this form of uncertainty. We begin in Section 4.1 by formulating minimax regret in flat (unfactored) utility models to develop intuitions used in the factored case. In Section 4.2 we discuss the computation of maximum regret in factored utility models, and propose two procedures for dealing with minimax regret. We evaluate one of these methods empirically in Section 4.3. Finally, in Section 4.4 we propose a generalization for the minimax problem in the case where the feasible utility set is defined by arbitrary linear constraints on parameters of the utility model.

4.1. Minimax regret with flat utility models

If we make no assumptions about the structure of the utility function, nor any assumptions about the nature of the feasible utility set $\mathbf{U}$, the optimization problem defined in Eq. (8) can be posed directly as a semi-infinite, quadratic, mixed-integer program (MIP):

$$\min_{\{M_{\mathbf{x}}, I_{\mathbf{x}}, X_i\}} \sum_{\mathbf{x}} M_{\mathbf{x}} I_{\mathbf{x}} \quad \text{subject to} \begin{cases} M_{\mathbf{x}} \geqslant u_{\mathbf{x}'} - u_{\mathbf{x}} & \forall \mathbf{x} \in \mathbf{X},\ \forall \mathbf{x}' \in \mathit{Feas}(\mathbf{X}),\ \forall u \in \mathbf{U}, \\ \mathcal{A} \text{ and } \mathcal{C}, \end{cases}$$

where we have:

- variables $M_{\mathbf{x}}$: for each $\mathbf{x}$, $M_{\mathbf{x}}$ is a real-valued variable denoting the max regret when decision $\mathbf{x}$ is made (i.e., when that state is chosen).
- variables $I_{\mathbf{x}}$: for each $\mathbf{x}$, $I_{\mathbf{x}}$ is a Boolean variable indicating whether $\mathbf{x}$ is the decision made.
- coefficients $u_{\mathbf{x}}$: for each $u \in \mathbf{U}$ and each state $\mathbf{x}$, $u_{\mathbf{x}}$ denotes the utility of $\mathbf{x}$ given utility function u.
- state definition constraints $\mathcal{A}$ and feasibility constraints $\mathcal{C}$ (defined as above).

Direct solution of this MIP is problematic, specifically because of the set of constraints on the $M_{\mathbf{x}}$ variables. First, if $\mathbf{U}$ is continuous (the typical case we consider here), then the set of constraints of the form $M_{\mathbf{x}} \geqslant u_{\mathbf{x}'} - u_{\mathbf{x}}$ is also continuous, since it requires that we "enumerate" all utility values $u_{\mathbf{x}}$ and $u_{\mathbf{x}'}$ corresponding to any utility function $u \in \mathbf{U}$. Furthermore, it is critical that we restrict our attention to those constraints associated with $\mathbf{x}'$ in the *feasible* set (i.e., those states satisfying $\mathcal{C}$). Fortunately, we can often tackle this seemingly complex optimization in much simpler stages if we make some very natural assumptions regarding the nature of the feasible utility space and utility function structure.

We begin by considering the case where our imprecise knowledge regarding all utility parameters $u_{\mathbf{x}}$ is independent and represented by simple upper and lower bounds. For example, asking standard gamble queries, as discussed further in Section 5.1, provides precisely such bounds on utility values [24]. Specifically, we assume an upper bound $u_{\mathbf{x}}\uparrow$ and a lower bound $u_{\mathbf{x}}\downarrow$ on each $u_{\mathbf{x}}$, thus defining the feasible utility set $\mathbf{U}$ to be a hyper-rectangle. This assumption allows us to compute the minimax regret in three simpler stages, which we now describe.[13]

[12] The term "robust optimization" has a number of different interpretations (see for example the work of Ben-Tal and Nemirovski [5]) of which minimax regret is one of the less common.

[13] This transformation essentially reduces the *semi-infinite quadratic* MIP to a *finite linear* IP.

First, we note that the pairwise regret for an ordered pair of states can be easily computed since each $u_{\mathbf{x}}$ is bounded by an upper and lower bound:

$$R(\mathbf{x}, \mathbf{x}', \mathbf{U}) = \begin{cases} u'_{\mathbf{x}}\uparrow - u_{\mathbf{x}}\downarrow & \text{when } \mathbf{x} \neq \mathbf{x}', \\ 0 & \text{when } \mathbf{x} = \mathbf{x}'. \end{cases} \tag{9}$$

Let $r_{\mathbf{x},\mathbf{x}'}$ denote this pairwise regret value for each $\mathbf{x}$, $\mathbf{x}'$, which we now assume has been pre-computed for all pairs.

Second, using Eq. (5), we can also compute the max regret $MR(\mathbf{x}, \mathbf{U})$ of any state $\mathbf{x}$ based on the pre-computed pairwise regret values $r_{\mathbf{x},\mathbf{x}'}$. Specifically, we can enumerate all feasible states $\mathbf{x}'$, retaining the largest (pre-computed) pairwise regret:

$$MR(\mathbf{x}, \mathbf{U}) = \max_{\mathbf{x}' \in Feas(\mathbf{X})} r_{\mathbf{x},\mathbf{x}'}. \tag{10}$$

Alternatively, we can search through feasible states "implicitly" with the following IP:

$$MR(\mathbf{x}, \mathbf{U}) = \max_{\{I_{\mathbf{x}'}, X'_i\}} \sum_{\mathbf{x}'} r_{\mathbf{x},\mathbf{x}'} I_{\mathbf{x}'} \quad \text{subject to } \mathcal{A} \text{ and } \mathcal{C}. \tag{11}$$

Third, let $m_{\mathbf{x}}$ denote the value of $MR(\mathbf{x}, \mathbf{U})$. With the max regret terms $m_{\mathbf{x}} = MR(\mathbf{x}, \mathbf{U})$ in hand, we can compute the minimax regret $MMR(\mathbf{U})$ readily. We simply enumerate all feasible states $\mathbf{x}$ and retain the one with the smallest (precomputed) max regret value $m_{\mathbf{x}}$:

$$MMR(\mathbf{U}) = \min_{\mathbf{x} \in Feas(\mathbf{X})} m_{\mathbf{x}}. \tag{12}$$

Again, this enumeration may be done implicitly using the following IP:

$$MMR(\mathbf{U}) = \min_{\{I_{\mathbf{x}}, X_i\}} \sum_{\mathbf{x}} m_{\mathbf{x}} I_{\mathbf{x}} \quad \text{subject to } \mathcal{A} \text{ and } \mathcal{C}. \tag{13}$$

In this flat model case, the two IPs above are not necessarily practical, since they require one indicator variable per state. However, this reformulation does show that the original quadratic MIP with a continuous set of constraints can be solved in stages using finite, linear IPs. More importantly, these intuitions will next be applied to develop an analogous procedure for factored utility models.

Note that the strategy above hinges on the fact that we have independently determined upper and lower bounds on the utility value of each state. If utility values are correlated by more complicated constraints, this strategy will not generally work. In particular, *comparison queries* in which a user is asked which of two states is preferred induce linear constraints on the entire set of utility parameters, thus preventing exploitation of independent upper and lower bounds. We discuss formulations that allow us to deal with such feasible utility sets in Section 4.4. However, we initially focus on the case of independent bounds.

4.2. Minimax regret with factored utility models

The optimization for flat models is interesting in that it allows us to get a good sense of how minimax regret works in a constraint-satisfaction setting. From a practical perspective, however, the above model has little to commend it. By solving IPs with one $I_{\mathbf{x}}$ variable per state, we have lost all of the advantage of using a compact and natural constraint-based approach to problem modeling. As we have seen when optimizing with known utility functions, if there is no *a priori* structure in the utility function, there is very little one can do but enumerate (feasible) states. On the other hand, when the problem structure allows for modeling via factored utility functions the optimization becomes more practical. We now show how much of this practicality remains when our goal is to compute the minimax-optimal state given uncertainty in a *factored* utility function represented as a graphical model.

Assume a set of factors f_k, $k \leqslant K$, defined over local families $\mathbf{X}[k]$, as described in Section 2.2. The parameters of this utility function are denoted by $u_{\mathbf{x}[k]} = f_k(\mathbf{x}[k])$, where $\mathbf{x}[k]$ ranges over $Dom(\mathbf{X}[k])$. We use the term *imprecise structured COP* to describe an imprecise COP $\langle \mathcal{C}, \mathcal{U} \rangle$ where the feasible utility set $\mathbf{U}$ is defined by a set of constraints $\mathcal{U}$ over the parameters $u_{\mathbf{x}[k]}$ of a factored utility model $\{f_k\colon\ k \leqslant K\}$.

As in the flat-model case, we assume upper and lower bounds on each of these parameters, which we denote by $u_{\mathbf{x}[k]}\uparrow$ and $u_{\mathbf{x}[k]}\downarrow$, respectively. Hence the range of each utility factor f_k for a given assignment $\mathbf{x}[k]$ corresponds to an interval. By defining $u(\mathbf{x})$ as in Eq. (2), pairwise regret, max regret and minimax regret are all defined in the

same manner outlined in Section 3. We now show how to compute each of these quantities in turn by generalizing the intuitions developed for flat models.

4.2.1. Computing pairwise regret and max regret

As in the unfactored case (Section 4.1), it is straightforward to compute the pairwise regret of any pair of states $\mathbf{x}$ and $\mathbf{x}'$. For each factor f_k and pair of local assignments $\mathbf{x}[k]$, $\mathbf{x}'[k]$, we define the *local pairwise regret*:

$$r_{\mathbf{x}[k],\mathbf{x}'[k]} = \begin{cases} u_{\mathbf{x}'[k]}\uparrow - u_{\mathbf{x}[k]}\downarrow & \text{when } \mathbf{x}[k] \neq \mathbf{x}'[k], \\ 0 & \text{when } \mathbf{x}[k] = \mathbf{x}'[k]. \end{cases}$$

With factored models it is not hard to see from Eqs. (2) and (9) that $R(\mathbf{x}, \mathbf{x}', \mathbf{U})$ is simply the sum of local pairwise regrets:

$$R(\mathbf{x}, \mathbf{x}', \mathbf{U}) = \sum_k r_{\mathbf{x}[k],\mathbf{x}'[k]}. \tag{14}$$

We can compute max regret $MR(\mathbf{x}, \mathbf{U})$ by substituting Eq. (14) into Eq. (5):

$$MR(\mathbf{x}, \mathbf{U}) = \max_{\mathbf{x}' \in Feas(\mathbf{X})} \sum_k r_{\mathbf{x}[k],\mathbf{x}'[k]}, \tag{15}$$

which leads to the following IP formulation:

$$MR(\mathbf{x}, \mathbf{U}) = \max_{\{I_{\mathbf{x}'[k]}, X'_i\}} \sum_k \sum_{\mathbf{x}'[k]} r_{\mathbf{x}[k],\mathbf{x}'[k]} I_{\mathbf{x}'[k]} \quad \text{subject to } \mathcal{A} \text{ and } \mathcal{C}. \tag{16}$$

The above IP differs from its flat counterpart (Eq. (11)) in the use of one indicator variable $I_{\mathbf{x}'[k]}$ per utility parameter rather than one per state, and is thus much more compact and efficiently solvable. Indeed, the size of the IP in terms of the number of variables and constraints (excluding exogenously determined feasibility constraints $\mathcal{C}$) is linear in the size of the underlying factored utility model.

4.2.2. Computing minimax regret: constraint generation

We can compute minimax regret $MMR(\mathbf{U})$ by substituting Eq. (15) into Eq. (7):

$$MMR(\mathbf{U}) = \min_{\mathbf{x} \in Feas(\mathbf{X})} \max_{\mathbf{x}' \in Feas(\mathbf{X})} \sum_k r_{\mathbf{x}[k],\mathbf{x}'[k]} \tag{17}$$

which leads to the following MIP formulation:

$$MMR(\mathbf{U}) = \min_{\{I_{\mathbf{x}[k]}, X_i\}} \max_{\mathbf{x}' \in Feas(\mathbf{X})} \sum_k \sum_{\mathbf{x}[k]} r_{\mathbf{x}[k],\mathbf{x}'[k]} I_{\mathbf{x}[k]} \quad \text{subject to } \mathcal{A} \text{ and } \mathcal{C} \tag{18}$$

$$= \min_{\{I_{\mathbf{x}[k]}, X_i, M\}} M \qquad \text{subject to } \begin{cases} M \geqslant \sum_k \sum_{\mathbf{x}[k]} r_{\mathbf{x}[k],\mathbf{x}'[k]} I_{\mathbf{x}[k]} & \forall \mathbf{x}' \in Feas(\mathbf{X}), \\ \mathcal{A} \text{ and } \mathcal{C}. \end{cases} \tag{19}$$

In Eq. (18), we introduce the variables for the minimization, while in Eq. (19) we transform the minimax program into a min program. The new real-valued variable M corresponds to the max regret of the minimax-optimal solution. In contrast with the flat IP (Eq. (13)), this MIP has a number of $I_{\mathbf{x}[k]}$ variables that is linear in the number of utility parameters. However, this MIP is not generally compact because Eq. (19) has one constraint per feasible state $\mathbf{x}'$. Nevertheless, we can get around the potentially large number of constraints in either of two ways.

The first technique we consider for dealing with the large number of constraints in Eq. (19) is *constraint generation*, a common technique in operations research for solving problems with large numbers of constraints. Our approach can be viewed as a form of Benders' decomposition [6,36]. This approach proceeds by repeatedly solving the MIP in Eq. (19), but using only a subset of the constraints on M associated with the feasible states $\mathbf{x}'$. At the first iteration, all constraints on M are ignored. At each iteration, we obtain a solution indicating some decision $\mathbf{x}$ with purported minimax regret; however, since certain unexpressed constraints may be violated, we cannot be content with this solution. Thus, we look for the unexpressed constraint on M that is maximally violated by the current solution. This involves finding a *witness* $\mathbf{x}'$ that maximizes regret w.r.t. the current solution $\mathbf{x}$; that is, a decision $\mathbf{x}'$ (and, implicitly, a utility function) that an adversary would choose to cause a user to regret $\mathbf{x}$ the most.

Recall that finding the feasible $\mathbf{x}'$ that maximizes $R(\mathbf{x}, \mathbf{x}', \mathbf{U})$ involves solving a single IP given by Eq. (16). We then impose the specific constraint associated with witness $\mathbf{x}'$ and re-solve the MIP in Eq. (19) at the next iteration with this additional constraint. Formally, we have the following procedure:

(1) Let $Gen = \{\mathbf{x}'\}$ for some arbitrary feasible $\mathbf{x}'$.
(2) Solve the MIP in Eq. (19) using the constraints corresponding to states in *Gen*. Let $\mathbf{x}^*$ be the MIP solution with objective value m^*.
(3) Compute the max regret of state $\mathbf{x}^*$ using the IP in Eq. (16), producing a solution with regret level r^* and witness (adversarial state) $\mathbf{x}''$. If $r^* > m^*$, then add $\mathbf{x}''$ to *Gen* and repeat from Step 2; otherwise (if $r^* = m^*$), terminate with minimax-optimal solution x^* (with regret level m^*).

Intuitively, when we solve the MIP in Step 2 using only the constraints in *Gen*, we are computing minimax regret against a *restricted adversary*: the adversary is only allowed to use choices $\mathbf{x}' \in Gen$ in order to make us regret our solution $\mathbf{x}^*$ to the MIP. As such, this solution provides a lower bound on true minimax regret (i.e., the solution that would have been obtained were a completely unrestricted adversary considered).

When we compute the true max regret r^* of $\mathbf{x}^*$ in Step 3, we also obtain an upper bound on minimax regret (since we can always attain max regret of r^* simply by stopping and recommending solution $\mathbf{x}^*$). It is not hard to see that if $r^* = m^*$, then no constraint is violated at the current solution $\mathbf{x}^*$ (and our upper and lower bounds on minimax regret coincide); so $\mathbf{x}^*$ is the minimax-optimal configuration at this point. The procedure is finite and guaranteed to arrive at the optimal solution. The constraint generation routine is not guaranteed to finish before it has the full set of constraints, but it is relatively simple and (as we will see) tends to generate a very small number of constraints. Thus in practice we solve this very large MIP using a series of small MIPs, each with a small number of variables and a set of active constraints that is also, typically, very small.

Since minimax regret will be computed between elicitation queries, it is critical that minimax regret be estimated in a relatively short period of time (e.g., five seconds for certain applications, five minutes for others, possibly several hours for very high stakes applications). With this in mind, several improvements can be made to speed up minimax regret computation. For instance, it is often sufficient to find a feasible (instead of optimal) configuration $\mathbf{x}$ for the MIP in Eq. (19) for each newly generated constraint. Intuitively, as long as the feasible $\mathbf{x}$ allows us to find a violated constraint—constraint generation continues to make progress. Hence, instead of waiting a long time for an optimal $\mathbf{x}$, we can stop the MIP solver as soon as we find a feasible solution for which a violated constraint exists. Of course, at the last iteration, when there are no violated constraints, we have no choice but to wait for the optimal $\mathbf{x}$.

Minimax regret can also be estimated more quickly—to allow for the real-time response needed for interactive optimization—by exploiting the anytime nature of the computation to simply stop early. Since minimax regret is computed incrementally by generating constraints, early stopping has the effect that some violated constraints may not have been generated. As a result the solution provides us with a lower bound on minimax regret. We can terminate early based on a fixed number of iterations (constraints), a fixed amount of computation time, or by terminating when bounds on the solution are tight enough. Apart from this lower bound, we can also obtain an upper bound on minimax regret by computing the max regret of the $\mathbf{x}$ found for the last minimax MIP solved. Note that we may need to explicitly compute this since Step (3) of our procedure may not be invoked if we terminate based only on the number of iterations rather than testing for constraint violation.

Approximation can be very appealing if real-time interactive response is required. The anytime flavor of the algorithm means that these lower and upper bounds are often tight enough to provide elicitation guidance of similar quality to that obtained from computing minimax regret exactly.

Although the full interaction of minimax regret computation with elicitation is explored in Section 5, as a precursor to that discussion, we mention another strategy for accelerating computation which directly influences the querying process. We have observed, unsurprisingly, that the minimax regret problem solved after receiving a response to one query is very similar to that solved before posing the query. As such, one can "seed" the minimax procedure invoked after a query with the constraints generated at the previous step. In this way, typically, only a few extra constraints are generated during each minimax computation. Given that the running time of minimax regret is dominated by constraint generation, this effectively amortizes the cost of minimax computation over a number of queries.

4.2.3. Computing minimax regret: a cost network formulation

A second technique for dealing with the large number of constraints in Eq. (19) is to use a cost network to generate a *compact* set of constraints that effectively summarizes this set. This type of approach has been used recently, for example, to solve Markov decision processes [26]. The main benefit of the cost network approach is that, in principle, it allows us to formulate a MIP with a feasible number of constraints (as elaborated below). We have observed, however, the constraint generation approach described above is usually much faster in practice and much easier to implement, even though it lacks the same worst-case run-time guarantees. Indeed, this same fact has been observed in the context of MDPs [47]. It is for this reason that we emphasize (and only experiment with) the constraint generation algorithm. However, we sketch the cost network formulation for completeness.

To formulate a compact constraint system, we first transform the MIP of Eq. (19) into the following equivalent MIP by introducing penalty terms $\rho_{\mathbf{x}[\ell]}$ for each feasibility constraint $\mathcal{C}_\ell$:

$$\begin{aligned} MMR(\mathbf{U}) &= \min_{\{I_{\mathbf{x}[k]}, X_i, M\}} M \quad \text{s.t.} \quad \begin{cases} M \geqslant \sum_k \sum_{\mathbf{x}[k]} r_{\mathbf{x}[k],\mathbf{x}'[k]} I_{\mathbf{x}[k]} + \sum_\ell \rho_{\mathbf{x}'[\ell]} & \forall \mathbf{x}' \in Dom(\mathbf{X}) \\ \mathcal{A} \text{ and } \mathcal{C} \end{cases} \\ &= \min_{\{I_{\mathbf{x}[k]}, X_i, M\}} M \quad \text{s.t.} \quad \begin{cases} M \geqslant \sum_k R_{\mathbf{x}'[k]} + \sum_\ell \rho_{\mathbf{x}'[\ell]} & \forall \mathbf{x}' \in Dom(\mathbf{X}) \\ R_{\mathbf{x}'[k]} = \sum_{\mathbf{x}[k]} r_{\mathbf{x}[k],\mathbf{x}'[k]} I_{\mathbf{x}[k]} & \forall k, \mathbf{x}'[k] \in Dom(\mathbf{X}[k]) \\ \mathcal{A} \text{ and } \mathcal{C}. \end{cases} \end{aligned} \tag{20}$$

The MIP of Eq. (19) has one constraint on M per feasible state $\mathbf{x}'$, whereas the MIP of Eq. (20) has one constraint per state $\mathbf{x}'$ (whether feasible or not). Therefore, to effectively maintain the feasibility constraints on $\mathbf{x}'$, we add penalty terms $\rho_{\mathbf{x}'[\ell]}$ that make a constraint on M meaningless when its corresponding state $\mathbf{x}'$ is infeasible. This is achieved by defining a local penalty function $\rho^\ell(\mathbf{x}'[\ell])$ for each logical constraint $\mathcal{C}_\ell$ that returns $-\infty$ when $\mathbf{x}'[\ell]$ violates $\mathcal{C}_\ell$ and 0 otherwise.

This transformation has, unfortunately, increased the number of constraints. However, it in fact allows us to rewrite the constraints in a much more compact form, as follows. Instead of enumerating all constraints on M, we analytically construct the constraint that provides the *greatest lower bound*, while simply ignoring the others. This greatest lower bound *GLB* is computed by taking the max of all constraints on M:

$$\begin{aligned} GLB &= \max_{\mathbf{x}'} \sum_k R_{\mathbf{x}'[k]} + \sum_\ell \rho_{\mathbf{x}'[\ell]} \\ &= \max_{x'_1} \max_{x'_2} \ldots \max_{x'_N} \sum_k R_{\mathbf{x}'[k]} + \sum_\ell \rho_{\mathbf{x}'[\ell]}. \end{aligned}$$

This maximization can be computed efficiently by using *variable elimination* [19], a well-known form of non-serial dynamic programming [7]. The idea is to distribute the max operator inward over the summations, and then collect the results as new terms which are successively pulled out. We illustrate its workings by means of an example.

Suppose we have the attributes X_1, X_2, X_3, X_4, a utility function decomposed into the factors $f_1(x_1, x_2)$, $f_2(x_2, x_3)$, $f_3(x_1, x_4)$ and two logical constraints with associated penalty functions $\rho^1(x_1)$ and $\rho^2(x_3, x_4)$. We then obtain

$$\begin{aligned} GLB &= \max_{x'_1} \max_{x'_2} \max_{x'_3} \max_{x'_4} R_{x'_1,x'_2} + R_{x'_2,x'_3} + R_{x'_1,x'_4} + \rho_{x'_1} + \rho_{x'_3,x'_4} \\ &= \max_{x'_1} [\rho_{x'_1} + \max_{x'_2} [R_{x'_1,x'_2} + \max_{x'_3} [R_{x'_2,x'_3} + \max_{x'_4} [R_{x'_1,x'_4} + \rho_{x'_3,x'_4}]]]] \end{aligned}$$

by distributing the individual max operators inward over the summations. To compute the *GLB*, we successively formulate new terms that summarize the result of completing each max in turn, as follows:

Let $A_{x'_1,x'_3} = \max_{x'_4} R_{x'_1,x'_4} + \rho_{x'_3,x'_4}$.

Let $A_{x'_1,x'_2} = \max_{x'_3} R_{x'_2,x'_3} + A_{x'_1,x'_3}$.

Let $A_{x'_1} = \max_{x'_2} R_{x'_1,x'_2} + A_{x'_1,x'_2}$.

Let $GLB = \max_{x'_1} \rho_{x'_1} + A_{x'_1}$.

Notice that this incremental procedure can be substantially faster than enumerating all states $\mathbf{x}'$. In fact the complexity of each step is only exponential in the local subset of attributes that indexes each auxiliary A variable.

Based on this procedure, we can substitute all the constraints on M in the MIP in Eq. (20) with the following compact set of constraints that analytically encodes the greatest lower bound on M:

$$
\begin{aligned}
&A_{x'_1,x'_3} \geqslant R_{x'_1,x'_4} + \rho_{x'_3,x'_4} \quad \forall x'_1, x'_3, x'_4 \in Dom(X_1, X_3, X_4),\\
&A_{x'_1,x'_2} \geqslant R_{x'_2,x'_3} + A_{x'_1,x'_3} \quad \forall x'_1, x'_2, x'_3 \in Dom(X_1, X_2, X_3),\\
&A_{x'_1} \geqslant R_{x'_1,x'_2} + A_{x'_1,x'_2} \quad \forall x'_1, x'_2 \in Dom(X_1, X_2),\\
&M \geqslant \rho_{x'_1} + A_{x'_1} \quad \forall x'_1 \in Dom(X_1).
\end{aligned}
$$

By encoding constraints in this way, the constraint system specified by the MIP in Eq. (20) can be generally encoded with a small number of variables and constraints. Overall we obtain a MIP where: the number of $I_{\mathbf{x}}$ variables is linear in the number of parameters of the utility function; and the number of auxiliary variables (the A variables in our example) and constraints that are added is locally exponential with respect to the largest subset of attributes indexing some auxiliary variable. In practice, since this largest subset is often very small compared to the set of all attributes, the resulting MIP encoding is compact and readily solvable. In particular, let the *joint graph* be the union of the constraint graph and the utility graph (e.g., the union of the graphs in Figs. 1 and 2). The complexity of this algorithm and hence the size of the resultant set of constraints is determined directly by the properties of variable elimination, and as such depends on the order in which the variables in $\mathbf{X}$ are eliminated. More precisely, it is exponential in the *tree width* of the joint graph induced by the elimination ordering [19]. Often this tree width is very small, thus rendering the algorithm only locally exponential [19].

4.3. Empirical results

To test the plausibility of minimax regret computation, we implemented the constraint generation strategy outlined above and ran a series of experiments to determine whether factored structure was sufficient to permit practical solution times. We implemented the constraint generation approach outlined in Section 4.2 and used CPLEX 9.0 as the generic IP solver.[14] Our experiments considered two realistic domains—car rentals and real estate—as well as randomly generated synthetic problems. In each case we imposed a factored structure to reduce the required number of utility parameters (upper and lower bounds).

For the real-estate problem, we modeled the domain with 20 (multivalued) variables that specify various attributes of single family dwellings that are normally relevant to making a purchase decision. The variables we used included: square footage, age, size of yard, garage, number of bedrooms, etc. Variables have domains with sizes ranging from two to four values. In total, there were 47,775,744 possible configurations of the variables. We then used a factored utility model consisting of 29 local factors, each defined on only one, two or three variables. In total, there were 160 utility parameters (i.e., utilities for local configurations). Therefore a total of 320 upper and lower bounds had to be specified, a significant reduction over the nearly 10^8 values that would have been required using a unfactored model. The local utility functions represented complementarities and substitutabilities in the utility function, such as requiring a large yard and a fence to allow a pool, sacrificing a large yard if the house happens to be near a park, etc.

The car-rental problem features 26 multi-valued variables encoding attributes relevant to consumers considering a car rental, such as: automobile size and class, manufacturer, rental agency, seating and luggage capacity, safety features (air bags, ABS, etc.), and so on. The size of the domain of the variables varies from 2 to 9. The total number of possible variable configurations is 61,917,360,000. There are 36 local utility factors, each defined on at most five variables, giving rise to 435 utility parameters. Constraints encode infeasible configurations (e.g., no luxury sedans have four-cylinder engines).

For both the car-rental and real-estate problems, we first computed the configuration with minimax regret given manually chosen bounds on the utility functions. The constraint generation technique of Section 4.2 took 15 seconds for the car-rental problem and 0.48 seconds for the real-estate problem. It is interesting to note that only 63 constraints

[14] These experiments were performed on 3.0 GHz PCs.

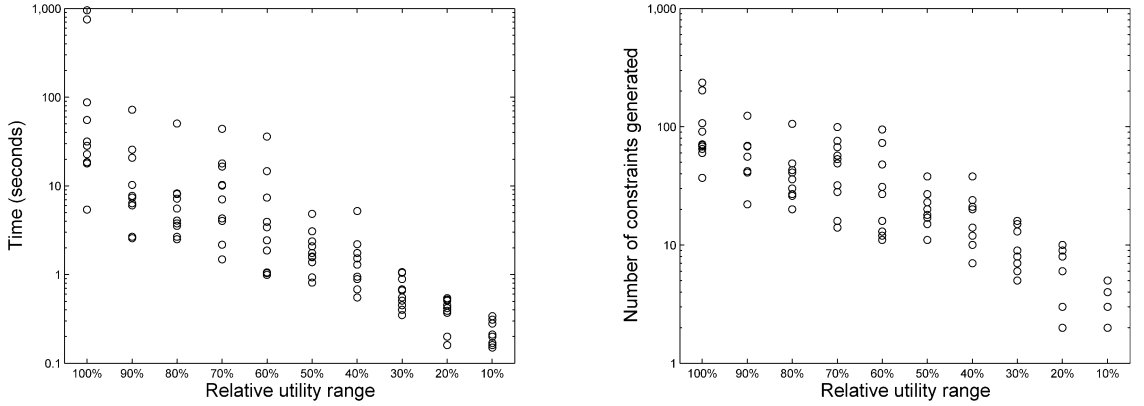

Fig. 4. Computation time (left) and number of constraints generated (right) for minimax regret on real-estate problem (48 million configurations) as a function of the tightness of the utility bounds.

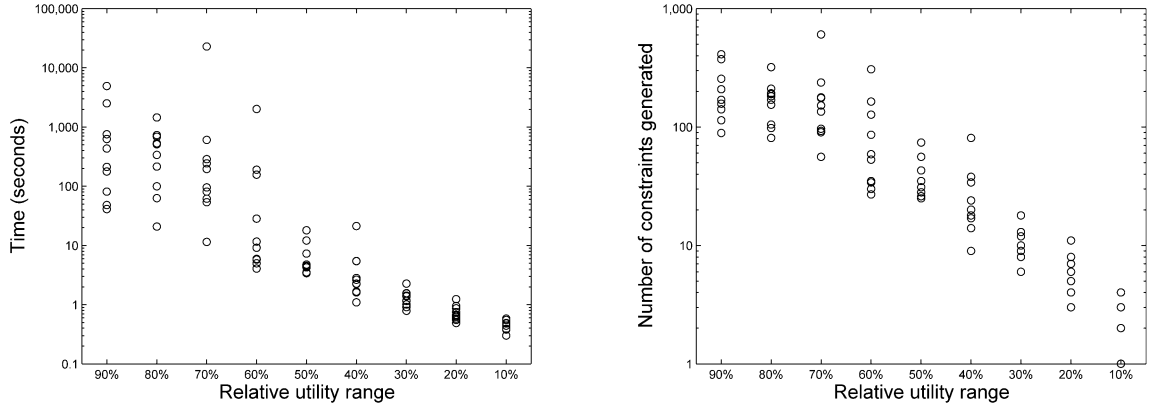

Fig. 5. Computation time (left) and number of constraints generated (right) for minimax regret on car-rental problem (62 billion configurations) as a function of the tightness of the utility bounds.

(out of 61,917,360,000 possible constraints) for the car-rental problem and seven constraints (out of 47,775,744 possible constraints) for the real-estate problem were generated in the search for the minimax optimal configuration. The structure exhibited by the utility functions of each problem is largely responsible for this small number of required constraints.

In practice, minimax regret computation will be interleaved with some preference elicitation technique (as we discuss in Section 5). As the bounds on utility parameters get tighter, we would like to know the impact on the running time of our constraint generation algorithm. To that effect, we carried out an experiment where we randomly set bounds, but with varying degrees of tightness. Initial utility gaps (i.e., difference between upper and lower bounds) ranged from 0 to 100. Figs. 4 and 5 show how tightening the bounds decreases both the running time and the number of constraints generated in an exponential fashion. For this experiment, bounds on utility were generated at random, but the difference between the upper and lower bounds of any utility was capped at a fixed percentage of some predetermined range. Intuitively, as preferences are elicited, the values will shrink relative to the initial range.

Figs. 4 and 5 show scatterplots of computation time and number of constraints for ten random problem instances generated for each of a number of increasingly tight relative utility ranges. As those figures suggest, a significant speed up is obtained as elicitation converges to the true utilities. Intuitively, the optimization required to compute minimax regret benefits from tighter bounds since some configurations emerge as clearly dominant, which in turn requires the generation of fewer constraints.

We carried out a second experiment with synthetic problems. A set of random problems of varying sizes was constructed by randomly setting the utility bounds as well as the variables on which each utility factor depends. Each utility factor depends on at most three variables and each variable has at most five values. Fig. 6 shows the results as

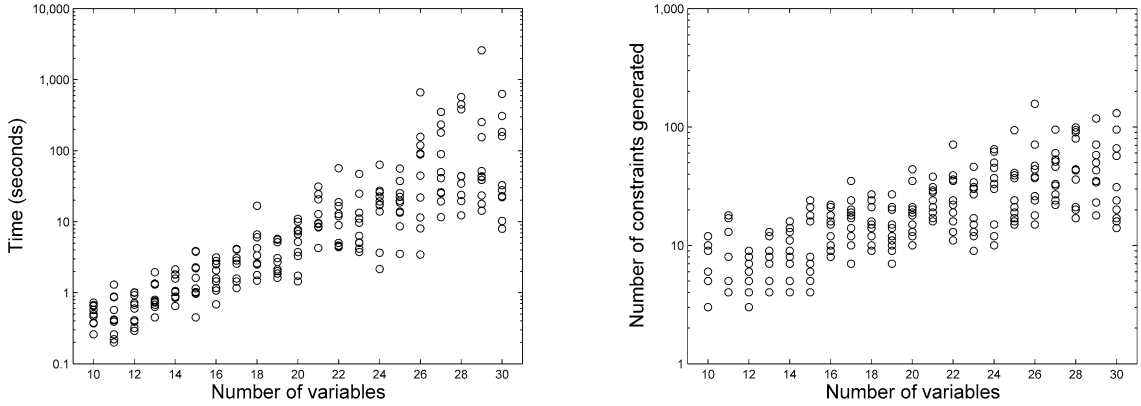

Fig. 6. Computation time (left) and number of constraints generated (right) for artificial random problems as a function of problem size (number of variables and factors).

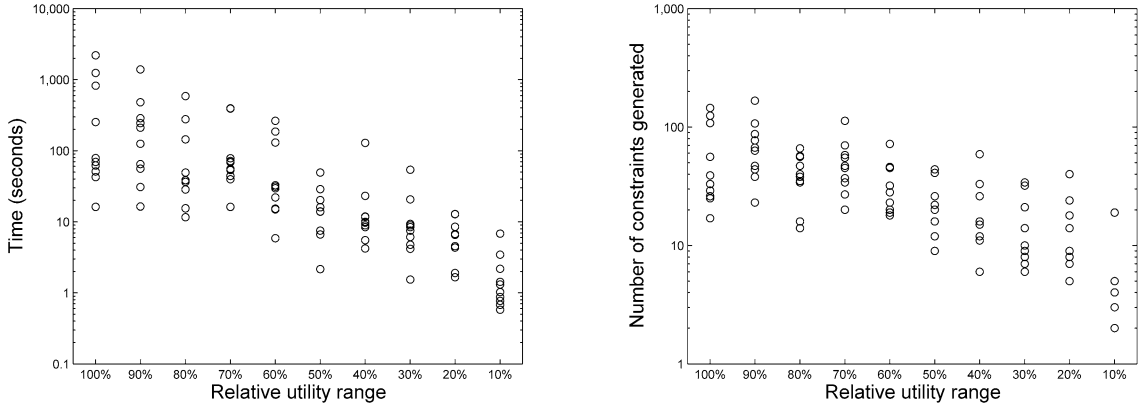

Fig. 7. Computation time (left) and number of constraints generated (right) for minimax regret on artificial random problems (30 variables, 30 factors) as a function of the tightness of the utility bounds.

we vary the number of variables and factors (the number of factors is always the same as the number of variables). The running time and the number of constraints generated increases exponentially with the size of the problem. Note however that the number of constraints generated is still a tiny fraction of the total number of possible constraints. For problems with 10 variables, only 7 constraints were necessary (out of 278,864) on average; and for problems with 30 variables, only 47 constraints were necessary (out of 2.8×10^{16}) on average.

We also tested the impact of the relative tightness of utility bounds on the efficiency of our constraint generation technique, with results shown in Fig. 7. Here, problems of 30 variables and 30 factors were generated randomly while varying the relative range of the utilities with respect to some predetermined range. Each factor has at most three variables chosen randomly and each variable can take at most five values. Once again, as the bounds get tighter, some configurations emerge as clearly dominant, which allows an exponential reduction in the running time as well as the number of required constraints.

Finally, we illustrate the anytime properties of our algorithm. In Fig. 8 we show the lower bound on minimax regret as a function of computation time. Each data point corresponds to one additional generated constraint. As we can see, the constraint generation algorithm has very good anytime properties, approaching the true minimax regret level very quickly as a function of time and number of constraints. This is due to two factors. First, as a function of the number of constraints generated, minimax regret lower bounds increase much more quickly early on, thus exhibiting the desired anytime behavior. Second, solution time with smaller numbers of constraints tends to be considerably less than with larger numbers of constraints. This enhances the anytime profile with respect to time (providing a much steeper increase than one would see if plotting the bound with respect to number of constraints generated). For example, in the rental car problem shown, the first ten constraints are generated in 1.7 s, the first twenty in 4.6 s, the first thirty in

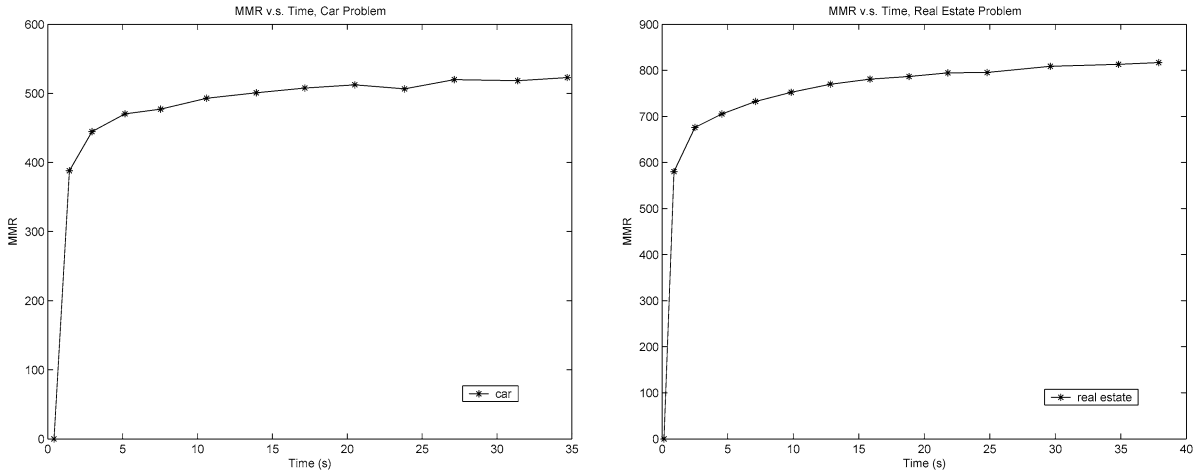

Fig. 8. Lower bound on minimax regret until convergence as a function of computation time. Results for Rental Car (left) and Real Estate (right), both at 70% of utility range. Solution quality is plotted for the solutions generated by adding one additional constraint. Data points are marked for every fifth constraint generated.

8.9 s, and so on until convergence to the true minimax regret after 57 constraints (24.62 s). This anytime property has important implications for real-time preference elicitation as we discuss below.

4.4. Minimax regret with linear utility constraints

The computational methods above exploit the existence of upper and lower bounds on utility parameters. While some types of queries used in elicitation allow one to maintain such independent bounds, other forms of queries (e.g., comparison queries) impose arbitrary linear constraints on these utility parameters, demanding new methods for computing minimax regret. We now develop an IP-based procedure for solving COPs with linear constraints on the utility parameters.

Suppose we have an imprecise structured COP $\langle \mathcal{C}, \mathcal{U} \rangle$ where $\mathbf{U}$ is a polytope defined by a finite set of (arbitrary) linear *utility constraints* $\mathcal{U}$ over the parameters $u_{\mathbf{x}[k]}$ of a factored utility model $\{f_k\colon\ k \leqslant K\}$. Thus we relax the assumption that the constraints $\mathcal{U}$ take the form of bounds.

Computing the max regret of a state $\mathbf{x}$ can no longer rely on the existence of local pairwise regrets as in Eq. (16). However, we can reformulate the problem somewhat differently to allow this to be solved linearly even when the constraints $\mathcal{U}$ take this more general form. First, we can recast the computation of max regret as a quadratic optimization:

$$MR(\mathbf{x}, \mathbf{U}) = \max_{\{I_{\mathbf{x}'[k]}, X'_i, U_{\mathbf{x}[k]}\}} \sum_k \left[\sum_{\mathbf{x}'[k]} U_{\mathbf{x}'[k]} I_{\mathbf{x}'[k]} \right] - U_{\mathbf{x}[k]} \quad \text{subject to } \mathcal{A},\ \mathcal{C},\ \text{and } \mathcal{U}. \tag{21}$$

We have introduced one real-valued variable $U_{\mathbf{x}[k]}$ (denoted $U_{\mathbf{x}'[k]}$ when referring to specific adversarial local states $\mathbf{x}'$) for each utility parameter $u_{\mathbf{x}[k]}$, reflecting their unknown nature; these are constrained by $\mathcal{U}$. The presence of the $U_{\mathbf{x}'[k]}$ variables renders the optimization quadratic. However, this can be reformulated by the introduction of new real-valued variables $Y_{\mathbf{x}'[k]}$ for each such utility variable. Intuitively, $Y_{\mathbf{x}'[k]}$ denotes the product $U_{\mathbf{x}'[k]} I_{\mathbf{x}'[k]}$. This product can be defined by assuming (loose) upper bounds $u_{\mathbf{x}[k]}\uparrow$ on the utility parameters. Specifically, we rewrite Eq. (21) as follows:

$$MR(\mathbf{x}, \mathbf{U}) = \max_{\{I_{\mathbf{x}'[k]}, X'_i, U_{\mathbf{x}[k]}, Y_{\mathbf{x}'[k]}\}} \sum_k \left(\sum_{\mathbf{x}'[k]} Y_{\mathbf{x}'[k]} \right) - U_{\mathbf{x}[k]} \quad \text{subject to } \begin{cases} Y_{\mathbf{x}'[k]} \leqslant I_{\mathbf{x}'[k]} u_{\mathbf{x}'[k]}\uparrow \quad \forall k, \mathbf{x}'[k], \\ Y_{\mathbf{x}'[k]} \leqslant U_{\mathbf{x}'[k]} \quad \forall k, \mathbf{x}'[k], \\ \mathcal{A}, \mathcal{C} \text{ and } \mathcal{U}. \end{cases} \tag{22}$$

The constraints on $Y_{\mathbf{x}'[k]}$, together with the fact that the objective aims to maximize its value, ensure that it takes the value zero if $I_{\mathbf{x}'[k]} = 0$ and takes the value $U_{\mathbf{x}'[k]}$ otherwise.[15]

With the ability to compute max regret by solving a MIP, we can use a variant of the constraint generation procedure described in Section 4.2. Notice that the solution to the MIP in Eq. (22) produces a witness $\mathbf{x}'$ as well as a specific utility function in which each variable $U_{\mathbf{x}[k]}$ is set to the value of utility parameter $u_{\mathbf{x}[k]}$ that maximizes the regret of the state in question. Notice also that the state $\mathbf{x}'$ must be the optimal feasible state for the chosen utility function $u \in \mathbf{U}$ (otherwise regret could be made even higher).

We can then express minimax regret in a way similar to Eqs. (18) and (19).

$$\begin{aligned} MMR(\mathbf{U}) &= \min_{\{I_{\mathbf{x}[k]}, X_i\}} \max_{\mathbf{x}' \in Feas(\mathbf{X}'), u \in \mathbf{U}} \sum_k \left(u_{\mathbf{x}'[k]} - \sum_{\mathbf{x}[k]} u_{\mathbf{x}[k]} I_{\mathbf{x}[k]} \right) \quad \text{s.t.} \quad \mathcal{A} \text{ and } \mathcal{C} \\ &= \min_{\{I_{\mathbf{x}[k]}, X_i\}} M \quad \text{s.t.} \quad \begin{cases} M \geqslant \sum_k \left(u_{\mathbf{x}'[k]} - \sum_{\mathbf{x}[k]} u_{\mathbf{x}[k]} I_{\mathbf{x}[k]} \right) & \forall \mathbf{x}' \in Feas(\mathbf{X}'), u \in \mathbf{U}, \\ \mathcal{A} \text{ and } \mathcal{C}. \end{cases} \end{aligned} \tag{23}$$

We can use the max regret computation described in Eq. (22) to generate constraints iteratively as required for Eq. (23).

5. Elicitation strategies

While the use of minimax regret provides a useful way of handling imprecise utility information, the initial bounds on utility parameters provided by users are unlikely to be tight enough to admit configurations with provably low regret. Instead, we imagine an interactive process in which the decision software queries the user for further information about her utility function—refining bounds or constraints on the parameters—until minimax regret, given the current constraints, reaches an acceptable level τ.[16] We can summarize the general form of the interactive elicitation procedure as follows:

(1) Compute minimax regret *mmr*.
(2) Repeat until $mmr < \tau$:
  (a) Ask query q.
  (b) Update the constraints $\mathcal{U}$ over utility parameters to reflect the response to q.
  (c) Recompute *mmr* with respect to new constraint set $\mathcal{U}$.

We begin by discussing bound queries, the primary type of query that we consider here, then describe a number of elicitation strategies using bound queries. Throughout most of this section we assume some imprecise structured COP problem $\langle \mathcal{C}, \mathcal{U} \rangle$ where $\mathcal{U}$ is specified by a factored utility model $\{f_k\colon\ k \leqslant K\}$ with upper and lower bounds on its parameters. However, we will also discuss comparison queries, and hence $\mathcal{U}$ in which arbitrary linear constraints are present, in Section 5.6.

There are a number of important issues regarding user interaction that will need to be addressed in the development of any interactive decision support software. The perspective we adopt here is rather rigid and assumes users can (somewhat comfortably) answer the types of queries we pose. We do not consider issues of framing, preference construction or exploration, or other issues surrounding the mode of interaction. Nor we do consider users who may express inconsistent preferences (indeed, none of our strategies will ever ask a query that can be responded to inconsistently). However, we believe the core of our elicitation techniques can certainly be incorporated into the larger context in which these important issues are addressed. For a discussion of some of these issues in the context of constraint-based optimization, see the work of Pu, Faltings, and Torrens [39].

[15] Note that this relies on the fact that each $U_{\mathbf{x}'[k]}$ is non-negative. If we allow negative local utility, these constraints can be generalized by exploiting a (loose) lower bound on $U_{\mathbf{x}'[k]}$ as well.

[16] We could insist that regret reaches zero (i.e., that we have a provably optimal solution), or stop when regret reaches a point where further improvement is outweighed by the cost of additional interaction.

5.1. Bound queries

Bound queries form the primary class of queries we consider, in which we ask the user whether one of her utility parameters lies above a certain value. A positive response raises the lower bound on that parameter, while a negative response lowers the upper bound: in both cases, uncertainty is reduced.

While users often have difficulty assessing numerical parameters, they are typically better at comparing outcomes [24,30]. Fortunately, a bound query can be viewed as a local form of a *standard gamble query* (*SGQ*), commonly used in decision analysis; these in fact ask for comparisons. An SGQ for a specific state $\mathbf{x}$ asks the user if she prefers $\mathbf{x}$ to a gamble in which the best outcome $\mathbf{x}_\top$ occurs with probability l and the worst $\mathbf{x}_\bot$ occurs with probability $1-l$ [30]. A positive response puts a lower bound on the utility of $\mathbf{x}$, and a negative response puts an upper bound. Calibration is attained by the use of common best and worst outcomes across all queries (and numerical assessment is restricted to evaluating probabilities). Thus a bound query "Is $u(\mathbf{x}) > q$?" can be cast as a standard gamble query: "Do you prefer $\mathbf{x}$ to a gamble in which $\mathbf{x}_\top$ is obtained with probability q and $\mathbf{x}_\bot$ is obtained with probability $1-q$?"[17]

For instance, ignoring factorization, one might ask in the car rental domain: "Would you prefer CAR27 or a gamble in which your received CARB with probability l and CARW with probability $1-l$?" Here CAR27 is the specific *complete* outcome of interest, while CARB and CARW are the best and worst possible car configurations, respectively (these need not be feasible in general). Of course, given the factorization of the model, we would prefer not to focus a user's attention on complete outcomes, but rather take advantage of the utility independence inherent in the GAI model to elicit information about local outcomes. As a consequence, we will ask analogous bound queries on local factors.

Our general elicitation procedure when restricted to bound queries takes the following form:

(1) Compute minimax regret *mmr*.
(2) Repeat until $mmr < \tau$:
  (a) Ask a bound query "Is $u_{\mathbf{x}[k]} \leqslant q$?" of some utility parameter $u_{\mathbf{x}[k]}$.
  (b) If $u_{\mathbf{x}[k]} \leqslant q$ then reduce upper bound $u_{\mathbf{x}[k]}\uparrow$ to q. Otherwise raise lower bound $u_{\mathbf{x}[k]}\downarrow$ to q.
  (c) Recompute *mmr* using the new bounds.

While we focus on bound queries, other forms of queries are quite natural. For example, comparison queries ask if one state $\mathbf{x}$ is preferred to another $\mathbf{x}'$ and are discussed further in Section 5.6. Hierarchical structuring of attributes is another avenue that could be considered in posing queries, though we leave this for future research within our model.

The foundations of bound queries can be made precise using results of Fishburn [22]. Roughly speaking, we require calibration across factors in the GAI model in order to be sure that the stated comparisons are meaningful. Gonzales and Perny [25] provide a specific procedure for (full) elicitation in GAI networks that relies on asking queries over complete outcomes, while Braziunas and Boutilier [15] provide a algorithm that allows for local queries (over small subsets of attributes). Bound queries in our framework can be supplemented with a small number of additional calibration queries as suggested in [15] if one requires calibration across factors for the user. Alternatively, if the scales associated with each of the GAI factors is obviously calibrated (e.g., the "utilities" refer to the monetary amount the user is willing to pay for a specific combination of attributes), then bound queries can be used directly without need for additional calibration. We refer to [15,25] for further details.

Several of our bound query strategies rely on the following definitions.

Definition 4. Let $\langle \mathcal{C}, \mathcal{U} \rangle$ be an imprecise COP problem. An *optimistic state* $\mathbf{x}^o$, a *pessimistic state* $\mathbf{x}^p$, and a *most uncertain state* $\mathbf{x}^{mu}$ are any states satisfying (respectively):

$$\mathbf{x}^o \in \arg \max_{\mathbf{x} \in Feas(\mathbf{X})} \max_{u \in \mathbf{U}} u(\mathbf{x}), \qquad \mathbf{x}^p \in \arg \max_{\mathbf{x} \in Feas(\mathbf{X})} \min_{u \in \mathbf{U}} u(\mathbf{x}), \qquad \mathbf{x}^{mu} \in \arg \max_{\mathbf{x} \in Feas(\mathbf{X})} \max_{u,u' \in \mathbf{U}} u(\mathbf{x}) - u'(\mathbf{x}).$$

An optimistic state is a feasible state with the greatest upper bound on utility. A pessimistic state has the greatest lower bound on utility. A most uncertain state has the greatest difference between its upper and lower bounds. Each

[17] If the user is nearly indifferent to the two alternatives, they may be tempted to respond "I don't know". This can be handled by imposing a quantitative interpretation on "near indifference" and imposing a constraint that makes these two utilities "close".

of these states can be computed in a single optimization by setting the parameters of the utility model to their upper bounds, their lower bounds, or their difference, and solving the corresponding (precise) COP problem.

5.2. The halve largest gap strategy

The first query strategy we consider is the *halve largest gap* (*HLG*) *strategy*. It asks a query at the midpoint of the interval of the parameter $\mathbf{x}[k]$ with the largest gap between its upper and lower bounds. This is motivated by theoretical considerations, based on simple worst-case bounds on minimax regret.

Definition 5. Define the *gap* of a utility parameter $u_{\mathbf{x}[k]}$, the *span* of factor f_k and *maxspan* of our utility model as follows:

$$gap(\mathbf{x}[k]) = u_{\mathbf{x}[k]}\uparrow - u_{\mathbf{x}[k]}\downarrow, \tag{24}$$

$$span(f_k) = \max_{\mathbf{x}[k]\in Dom(\mathbf{X}[k])} gap(\mathbf{x}[k]), \tag{25}$$

$$maxspan(\mathbf{U}) = \sum_k span(f_k). \tag{26}$$

The quantity *maxspan* measures the largest difference between the upper and lower utility bound, regardless of feasibility. We can show that this quantity bounds minimax regret:

Proposition 1. *For any* $\langle \mathcal{C}, \mathcal{U}\rangle$, $MMR(\mathbf{U}) \leqslant maxspan(\mathbf{U})$.

Proof. By definition of minimax regret, we have $MMR(\mathbf{U}) \leqslant MR(\mathbf{x}^o, \mathbf{U})$. For any optimistic state $\mathbf{x}^o$ and any alternative state $\mathbf{x}$ we must have that $u\uparrow(\mathbf{x}) - u\downarrow(\mathbf{x}^o) \leqslant u\uparrow(\mathbf{x}^o) - u\downarrow(\mathbf{x}^o) \leqslant maxspan(\mathbf{U})$ (i.e., the difference between the upper and lower bounds of $\mathbf{x}$ and $\mathbf{x}^o$, respectively, is bounded by $maxspan(\mathbf{U})$, since the upper bound of $\mathbf{x}$ cannot exceed that of $\mathbf{x}^o$, and the lower bound of $\mathbf{x}^o$ can be no less than $u\uparrow(\mathbf{x}^o) - maxspan(\mathbf{U})$). Thus, $MR(\mathbf{x}^o, \mathbf{U}) \leqslant maxspan(\mathbf{U})$. The result follows immediately. □

We note that the definition of *maxspan* can be tightened in two ways. (a) One could account for logical consistency across utility factors (e.g., if X occurs in two factors, we cannot have a total utility span for a single state that instantiates the span in one factor with X true, and the span in the other with X false). Computing this tighter definition of span requires some minor optimization to find the logically consistent state with maximum span, but is otherwise straightforward. (b) One could make this tighter still by restricting attention to feasible states (w.r.t. $\mathcal{C}$); in other words, maxspan would be defined as the "span" of any most uncertain state $\mathbf{x}^{mu}$. The result still holds with these tighter definitions. However, the current definition requires no optimization to assess.

The relationship between *maxspan* and minimax regret suggests an obvious query strategy, the HLG method, in which a bound query is asked of the local state $\mathbf{x}[k]$ with the largest utility gap, at the midway point of its interval, $(u_{\mathbf{x}[k]}\uparrow - u_{\mathbf{x}[k]}\downarrow)/2$. This method guarantees reasonably rapid reduction in max regret:

Proposition 2. *Let* $\mathbf{U}$ *be an uncertain utility model with* n *parameters and let* $m = maxspan(\mathbf{U})$. *After* $n\log(m/\varepsilon)$ *queries in the HLG strategy, minimax regret is no greater than* ε.

Proof. Given a utility model with n parameters with a specific initial set of gaps, the largest gap among all states must be reduced by at least half after n queries according to the HLG strategy (with this bound being tight only if the largest initial gap is no more than twice that of the smallest initial gap). Thus after kn queries, leading to an updated feasible utility set $\mathbf{U}'$, we have $maxspan(\mathbf{U}') \leqslant 2^{-k}m$. The result then follows by application of Proposition 1. □

In the worst case, there are classes of utility functions for which the bound is tight, so sets $\mathbf{U}$ and configuration constraints $\mathcal{C}$ exist that ensure regret will never be reduced to zero in finitely many queries. For example, if we have a linear utility function over $X_1, \ldots, X_n$ with

$$u(\mathbf{X}) = f_1(X_1) + \cdots + f_n(X_n),$$

with each local utility parameter having the same gap g and no feasibility constraints, then minimax regret can be reduced no more quickly than this.[18]

This strategy is similar to heuristically motivated polyhedral methods in conjoint analysis used in product design and marketing [29,48]. In fact, HLG can be viewed as a special case of the polyhedral method of [48] in which our polyhedra are hyper-rectangles.

5.3. The current solution strategy

While HLG allows one to provide strong worst-case guarantees on regret improvement, it is "undirected" in that considerations of feasibility play no role in determining which queries to ask. An alternative strategy is to focus attention on parameters that participate in defining minimax regret, namely, the minimax optimal $\mathbf{x}^*$ and the adversarial witness $\mathbf{x}^w$ for the current feasible utility set $\mathbf{U}$ (recall that the witness $\mathbf{x}^w$ maximizes the regret of $\mathbf{x}^*$). The *current solution* (*CS*) *query strategy* asks about the utility parameter in the set $\{\mathbf{x}^*[k]\colon k \leqslant K\} \cup \{\mathbf{x}^w[k]\colon k \leqslant K\}$ with largest $gap(\mathbf{x}[k])$ and queries the midpoint of the corresponding utility interval. Intuitively, should the answer to a query raise the lower bound on some $u_{\mathbf{x}^*[k]}$ or lower the upper bound on some $u_{\mathbf{x}^w[k]}$, then the pairwise regret $R(\mathbf{x}^*, \mathbf{x}^w, \mathbf{U})$ will be reduced, and usually minimax regret will be reduced as well. Of course, if the answer lowers the upper bound on some $u_{\mathbf{x}^*[k]}$ or raises the lower bound on some $u_{\mathbf{x}^w[k]}$, then pairwise regret $R(\mathbf{x}^*, \mathbf{x}^w, \mathbf{U})$ remains unchanged and minimax regret is not guaranteed to be reduced (though it may).

We have also experimented with a variant of the CS strategy in which regret is computed approximately to ensure fast interactive response in the querying process. This can be done by imposing a time bound on the solution algorithm for computing minimax regret, exploiting the anytime nature of the method described in Section 4.2. While we can't be sure we have the minimax optimal solution with early termination, the solution may be good enough to guide the querying process. Furthermore, since we can compute the max regret of the anytime solution, we have an upper bound on minimax regret which can be used as a natural termination criterion.

5.4. Alternative strategies

Finally, we consider several other strategies, which we describe briefly. The *optimistic query strategy* computes an optimistic state $\mathbf{x}^o$ and queries (at the midpoint of the interval) the utility parameter in $\mathbf{x}^o$ with the largest gap. Intuitively, an optimistic $\mathbf{x}^o$ is a useful adversarial choice, so refining information about it can help reduce regret. The *pessimistic query strategy* is analogous, relying on the intuition that a pessimistic choice is useful in preventing the adversary from making us regret our decision too much. The *optimistic-pessimistic* (*OP*) *strategy* combines the two intuitions: it chooses the parameter with largest gap among both states. These strategies are computationally appealing since they require solving only a standard COP, not a full-fledged minimax optimization.[19]

The *most uncertain state* (*MUS*) *strategy* is a variant of HLG that accounts for feasibility: we compute a most uncertain state $\mathbf{x}^{mu}$ and query (at the midpoint) the parameter in $\mathbf{x}^{mu}$ with the largest gap. Finally, the *second-best* (*SB*) *strategy* is based on the following intuition: suppose we have the optimistic state $\mathbf{x}^o$ and the second-best optimistic state $\mathbf{x}^{2o}$ (i.e., that state with the second-highest upper bound—this is computable with a single optimization). If we could ask a query which reduced the upper bound utility of $\mathbf{x}^o$ to lower than that of $\mathbf{x}^{2o}$, we ensure that regret is reduced (since the adversary can no longer attain this most optimistic value); if the lower bound of $\mathbf{x}^o$ were raised to the level of $\mathbf{x}^{2o}$'s upper bound, then we could terminate—knowing that $\mathbf{x}^o$ is *optimal*. Thus we would like to query $\mathbf{x}^o$ at $\mathbf{x}^{2o}$'s upper bound: a negative response will reduce regret, a positive response ensures $\mathbf{x}^o$ is optimal. Unfortunately, this cannot be implemented directly, since we can only query local parameters, but the strategy can be approximated for factored models by "distributing" this query across the different parameters and asking a set of queries.

The *myopically optimal* (*MY*) *strategy* computes the average regret reduction of the midpoint query for *each* utility parameter by solving the minimax optimization problem for each response to each query; it then asks the query with

[18] The bound is not generally tight if there is overlap in factors. But the bound is tight if *maxspan* is defined to account for logical consistency.

[19] Even termination can be determined heuristically, for example, by computing the max regret of the optimistic state after each query, or doing minimax optimization after every q queries.

the largest regret reduction averaged over both possible answers, *yes* and *no*. For large problems, this approach is computationally infeasible, but we test it on small problems to see how the other methods compare.[20]

5.5. Empirical results

To test the effectiveness of the various query strategies, we ran a series of elicitation experiments on a variety of problems. For each problem we tested the following elicitation strategies: halve largest gap (HLG), current solution (CS), current solution with a computation-time bound of five seconds per query (CS-5), optimistic-pessimistic (OP), second-best (SB), and most uncertain state (MUS). In addition, on problems small enough to permit it, we also compared these strategies to the much more computationally demanding myopically optimal method (MY).

We implemented the constraint generation approach outlined in Section 4.2 and used CPLEX 9.0 as the generic IP solver.[21] Our experiments considered two realistic domains—car rental and real estate—as well as randomly generated synthetic problems, as described in Section 4.3, with a factored structure sufficient to admit practical solution.

First, we experimented with a set of small synthetic problems. We did this to allow comparison of all of our proposed heuristics with the computationally demanding MY strategy. Fig. 9 reports the average minimax regret over 45 small synthetic problems constructed by randomly setting the utility bounds and the variables on which each utility factor depends. Each problem has ten attributes that can take at most four values and ten factors that depend on at most three attributes. We simulate user responses by drawing a random utility function u for each trial, consistent with the bounds, representing a specific user's preferences. Responses to queries are generated using u, assuming that the user accurately answers all queries relative to the specific utility function u.

Results are shown for two cases: first, for utility parameters drawn from a uniform distribution over the corresponding interval; and second, for parameters drawn from a truncated Gaussian distribution centered at the midpoint of the corresponding interval and truncated at the endpoints of that interval. This second regime reflects the fact that, given some initial unquantified uncertainty about a utility parameter, a user is somewhat more likely to have a true parameter value nearer the middle of the range. However, this probabilistic information is used only to generate "simulated users", and is not exploited by the elicitation algorithms.[22]

In the case of both the uniform and truncated Gaussian distributions, we observe that the OP, CS and CS-5 elicitation strategies decrease minimax regret at a rate very close to MY. This suggests that OP, CS and CS-5 are computationally feasible, yet promising alternatives to the computationally prohibitive MY strategy.

We report on further experiments using all strategies except MY (excluded for computational reasons) with larger synthetic problems, the real-estate problem and the car-rental problem. All results are averaged over 45 trials and

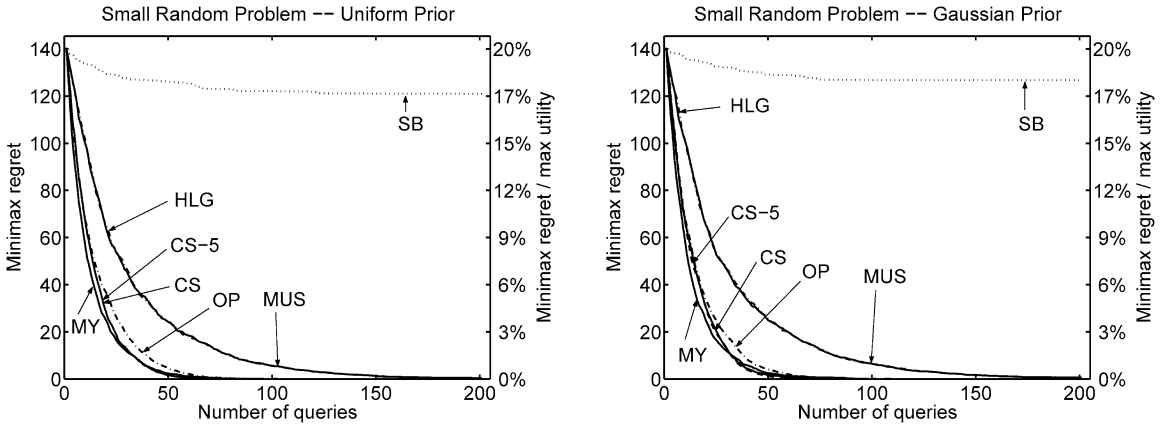

Fig. 9. Average max regret on small random problems (45 instances) as a function of number of queries given uniform (left) and Gaussian (right) distributed utilities.

[20] By doing lookahead of k stages of this type, we could in fact compute the optimal query plan of k-steps; however, doing so is infeasible for all but the smallest problems and small values of k.

[21] These simulations were performed on 3.0 GHz PCs.

[22] All experiments show a reasonably small variance so we exclude error bars for legibility.

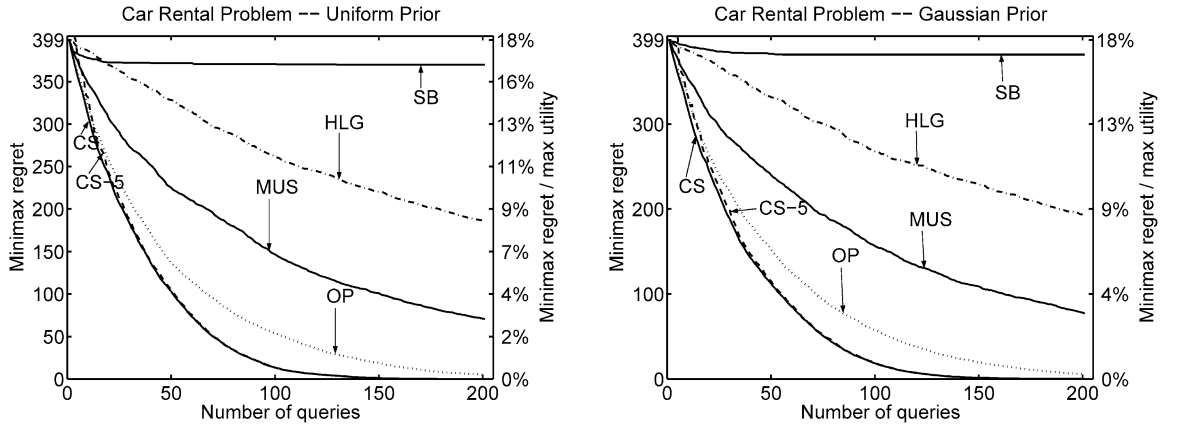

Fig. 10. Average max regret on car-rental problem (45 instances) as a function of number of queries given uniform (left) and Gaussian (right) distributed utilities.

use the same regime described above, involving both uniform and truncated Gaussian priors to generate users. Performance of the various query strategies on the car rental problem is depicted in Fig. 10, showing average minimax regret as a function of the number of queries. Initial utility bounds are set to give minimax regret of roughly 18% of the optimal solution.

Both CS and CS-5 perform extremely well: regret is reduced to almost zero within 160 queries on average. Though this may seem like a lot of queries, recall that the problem is itself large and the utility model has 150 parameters. We intentionally choose problems this large to push the computational boundaries of regret-based elicitation. Furthermore, while 160 queries may be large for typical consumer choice problems, it is more than reasonable for high stakes configuration applications. More importantly, these methods show excellent anytime performance: after only 80 queries, average minimax regret has dropped from 18% to under 2%.

Interestingly, the time bound of five seconds imposed by CS-5, while leading to approximately minimax optimal solutions, does not affect query quality: the approximate solutions give rise to queries that are virtually as effective as those generated by the optimal solutions. This demonstrates the importance of the anytime properties of our constraint generation procedure discussed in the previous section. The CS strategy requires on average 83 seconds per query, compared to the five seconds needed by CS-5. The OP strategy works very well too, and requires less computation time (0.1 s per query) since it does not need to solve minimax problems (except to verify termination "periodically", which is not reflected in the reported query computation time). However, both OP and CS-5 are fast enough to be used interactively on problems of this size. MUS, HLG, and SB do not work nearly as well, with SB essentially stalling because of the slow progress made in reducing the upper bounds of the optimistic state.

Note the HLG performs poorly since it fails to account for the feasibility of options, thus directing its attention to parts of utility space for which no product exists (hence polyhedral methods alone [29,48] will not offer reasonable elicitation in our setting). MUS significantly outperforms HLG for just this reason.

The real-estate problem was also tested, with query performance shown in Fig. 11, using the same regime as above. Again, both CS and CS-5 perform best, and the time bound of CS-5 has no effect on the quality of the CS strategy. Interestingly, OP performs almost identically to these, with somewhat lower computational cost.[23] Each of these methods reduces minimax regret from 40% of optimal to under 5% in about 120 queries. As above, SB fails to make progress, while HLG and MUS provide reasonable performance. Note that HLG requires no optimization nor any significant computation (except to test for termination).

Finally, we tested the query strategies on larger randomly generated problems (with 25 variables of domain size no more than four, and 20 utility factors with no more than three variables each). Results are shown in Fig. 12. The same performance patterns as in the real-estate problem emerge, with CS, CS-5 and OP all performing much better than the others. Although OP performs slightly better than CS/CS-5, the difference is not statistically significant.

[23] CS takes 14 seconds per query, CS-5 takes five seconds, and OP 0.1 seconds. Though we haven't experimented with this, we expect CS would work equally well on this problem with a much tighter time bound than five seconds.

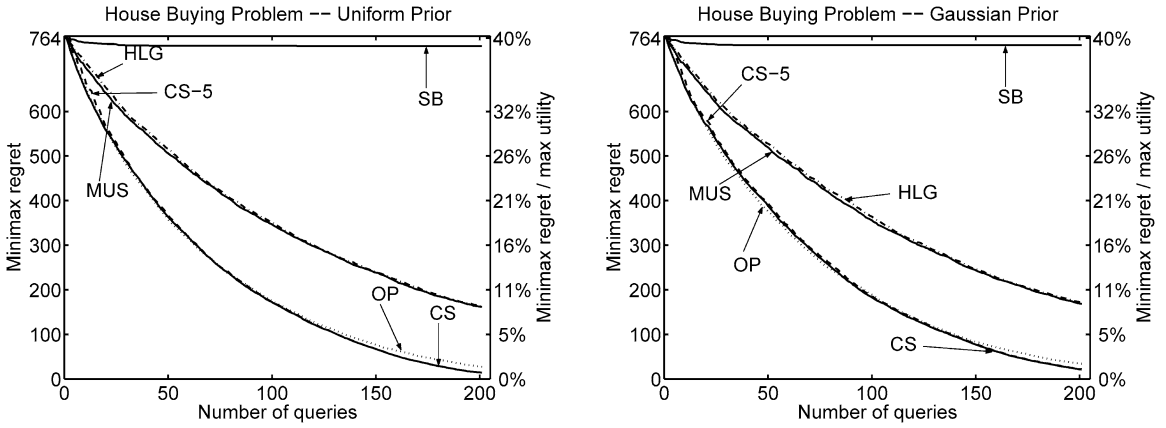

Fig. 11. Average max regret on real-estate problem (45 instances) as a function of number of queries given uniform (left) and Gaussian (right) distributed utilities.

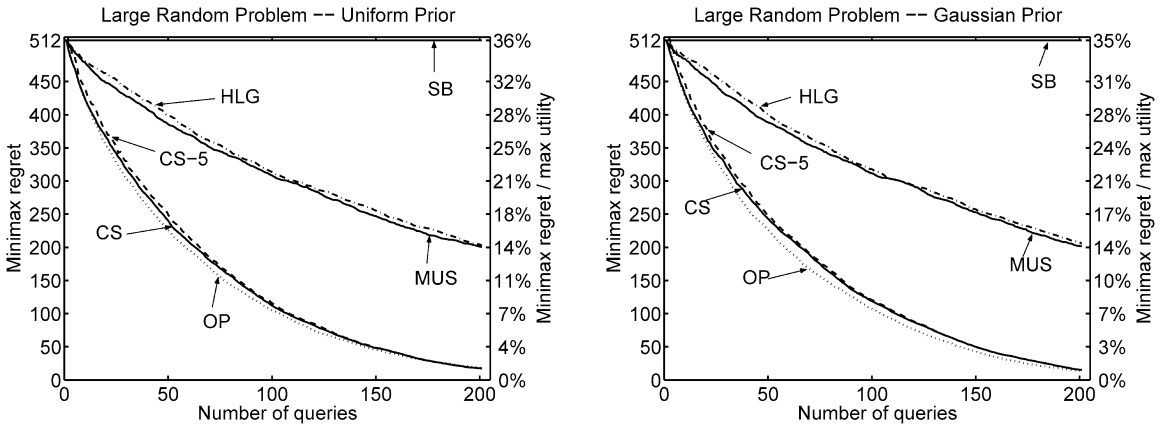

Fig. 12. Average max regret on large random problems (45 instances) as a function of number of queries given uniform (left) and Gaussian (right) distributed utilities.

5.6. Comparison queries

Comparison queries provide a natural alternative to bound queries in many situations. A comparison query takes the form "Do you prefer $\mathbf{x}$ to $\mathbf{x}'$?" A positive response implies that $u_{\mathbf{x}} > u_{\mathbf{x}'}$. If the utility model is factored, this corresponds to the following linear constraint:

$$\sum_k u_{\mathbf{x}[k]} > \sum_k u_{\mathbf{x}'[k]}.$$

A negative response imposes the complementary constraint.

Given a collection of linear constraints $\mathcal{U}$ imposed by responses to a sequence of comparison queries, the minimax optimal decision can be computed using the constraint generation procedure described in Section 4.4. Our generic elicitation algorithm can then be used to ask specific comparison queries until minimax regret reaches an acceptable level. Though we do not experiment with specific comparison query strategies here, we expect that a modification of the current solution (CS) strategy proposed for bound queries would work especially well with comparison queries. More precisely, suppose that given the current constraints $\mathcal{U}$, the minimax optimal solution is computed to be $\mathbf{x}^*$ with adversarial witness $\mathbf{x}^w$. The CS query strategy for comparison queries requires that we ask the user to compare $\mathbf{x}^*$ and $\mathbf{x}^w$. Should the user prefer $\mathbf{x}^*$, this rules out the adversary's chosen utility function from the feasible set $\mathbf{U}$, thus ensuring a reduction in the pairwise regret $R(\mathbf{x}^*, \mathbf{x}^w, \mathbf{U})$ to zero, and usually reducing minimax regret as well. If $\mathbf{x}^w$ is preferred, this does not rule out the adversary's chosen utility function, nor is it guaranteed to reduce regret, but

generally imposes a fairly strong constraint on **U**. Given the success of this strategy with respect to bound queries, and the success of related strategies in other domains [13,14], we expect the CS strategy to perform quite well.

Unlike the case of bound queries, where it is quite clear (due to the focus on gaps in specific parameters) that a user cannot provide a response that is inconsistent with prior responses, it is not obvious that a user cannot be inconsistent in responding to comparison queries. However, the CS strategy for bound queries does indeed ensure consistency. Unless minimax regret is zero (in which case the process would terminate), there must be some utility function in the current feasible set $\mathcal{U}$ for which $\mathbf{x}^w$ is preferred to $\mathbf{x}^*$. Furthermore, there must be some utility function for which $\mathbf{x}^*$ is preferred to $\mathbf{x}^w$; otherwise, $\mathbf{x}^w$ would have regret no greater that of $\mathbf{x}^*$ under all $u \in \mathcal{U}$, and would thus be minimax optimal as well (also implying that, since it is the witness, that minimax regret is zero).

6. Concluding remarks

Preference elicitation techniques for constraint-based optimization problems are critical to the development of interactive decision software. We have begun to address several important issues in this regard, specifically, how one should make decisions in the presence of (non-probabilistic) utility function uncertainty, and elicitation strategies that improve decision quality with minimal interaction. We have developed techniques for computing minimax optimal decisions in constraint-based decision problems when a user's utility function is only partially specified in the form of upper and lower bounds on utility parameters, or arbitrary linear constraints on such parameters. While the corresponding optimizations are potentially complex, we derived methods whereby they could be solved effectively using a sequence of MIPs. Furthermore, we showed how structure in the utility model could be exploited to ensure that the resulting IPs are compact or could be solved using an effective constraint generation procedure. Experiments with utility uncertainty specified by parameter bounds demonstrated the practicality of these techniques.

We also developed a number of query strategies for eliciting bounds on the parameters of utility models for the purpose of solving imprecise COPs. The most promising of these strategies, CS and OP, perform extremely well, requiring very few queries (relative to the model size) to provide dramatic reductions in regret. We have shown that using approximation of minimax regret reduces interactive computation time to levels required for real-time response without a noticeable effect on the performance of CS. OP also can be executed in real-time, since it does not require the same intensive minimax computation.

There are a number of directions in which this work can be extended. For example, the use of search and constraint-propagation methods for solving the COPs associated with computing minimax regret is of great interest. Our goal in this paper was to provide a precise formulation of these computational problems as integer programs and use off-the-shelf software to solve them. We expect that constraint-based optimization techniques that are specifically directed toward these problems should prove fruitful. Along these lines, we hope to develop deeper connections to existing work on soft constraints, valued-CSPs, and related frameworks.

Experimental validation of our suggested approach for comparison queries is an important next step, as is the development of new query strategies. In practice, we can often assume or develop prior distributional information over utilities. Rather than asking queries at midpoints of intervals, we could optimize the query point using probabilistic (value of information) computation, while using (distribution-free) regret to make decisions [50]. We are quite interested in the possibility of integrating Bayesian methods for reasoning about uncertain utility functions [10,17,27] with the constraint-based representation of the decision space. Finally, while optimal (non-myopic) strategies could be found (in principle) by solving prohibitively large continuous MDPs, it would nevertheless be interesting to explore non-myopic heuristics, and investigate the extent to which lookahead information can improve the reduction in minimax regret.

Naturally, we would like to consider additional query types, as well as alternative means for structuring outcomes and interactions to ease the cognitive burden on users. Developing means to improve robustness of our methods to the types of errors users typically make is also critical if tools such as those proposed here are to find widespread use. As such, user studies with our models will be required in order to assess the naturalness of such interaction models and to further refine our techniques to make them more understandable and intuitive for users.

An important question left unaddressed is that of eliciting the structure of a GAI model. Our work here assumes that the GAI factorization has been given and elicits only parameters of this model. We are currently exploring the application of techniques from decision analysis and elicitation of graphical models to the automated elicitation of GAI model structure.

Acknowledgements

This research was supported by the Institute for Robotics and Intelligent Systems (IRIS) and the Natural Sciences and Engineering Research Council (NSERC). Poupart was supported by a scholarship provided by Precarn Incorporated through IRIS. Thanks to the anonymous referees for their helpful suggestions.

References

[1] I. Averbakh, Minmax regret solutions for minimax optimization problems with uncertainty, Operations Research Letters 27 (2000) 57–65.
[2] I. Averbakh, V. Lebedev, On the complexity of minmax regret linear programming, European Journal of Operational Research 160 (1) (2005) 227–231.
[3] F. Bacchus, A. Grove, Graphical models for preference and utility, in: Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence (UAI-95), Montreal, 1995, pp. 3–10.
[4] D.E. Bell, Regret in decision making under uncertainty, Operations Research 30 (1982) 961–981.
[5] A. Ben-Tal, A. Nemirovski, Robust solutions of uncertain linear programs, Operations Research Letters 25 (1999) 1–13.
[6] J.F. Benders, Partitioning procedures for solving mixed-variables programming problems, Numerische Mathematik 4 (1962) 238–252.
[7] U. Bertele, F. Brioschi, Nonserial Dynamic Programming, Academic Press, Orlando, FL, 1972.
[8] S. Bistarelli, U. Montanari, F. Rossi, Semiring-based constraint satisfaction and optimization, Journal of the ACM 44 (2) (1997) 201–236.
[9] J. Blythe, Visual exploration and incremental utility elicitation, in: Proceedings of the Eighteenth National Conference on Artificial Intelligence (AAAI-02), Edmonton, 2002, pp. 526–532.
[10] C. Boutilier, A POMDP formulation of preference elicitation problems, in: Proceedings of the Eighteenth National Conference on Artificial Intelligence (AAAI-02), Edmonton, 2002, pp. 239–246.
[11] C. Boutilier, On the foundations of *expected* expected utility, in: Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI-03), Acapulco, 2003, pp. 285–290.
[12] C. Boutilier, F. Bacchus, R.I. Brafman, UCP-Networks: A directed graphical representation of conditional utilities, in: Proceedings of the Seventeenth Conference on Uncertainty in Artificial Intelligence (UAI-01), Seattle, WA, 2001, pp. 56–64.
[13] C. Boutilier, R. Das, J.O. Kephart, G. Tesauro, W.E. Walsh, Cooperative negotiation in autonomic systems using incremental utility elicitation, in: Proceedings of the Nineteenth Conference on Uncertainty in Artificial Intelligence (UAI-03), Acapulco, 2003, pp. 89–97.
[14] C. Boutilier, T. Sandholm, R. Shields, Eliciting bid taker non-price preferences in (combinatorial) auctions, in: Proceedings of the Nineteenth National Conference on Artificial Intelligence (AAAI-04), San Jose, CA, 2004, pp. 204–211.
[15] D. Braziunas, C. Boutilier, Local utility elicitation in GAI models, in: Proceedings of the Twenty-first Conference on Uncertainty in Artificial Intelligence (UAI-05), Edinburgh, 2005, pp. 42–49.
[16] U. Chajewska, L. Getoor, J. Norman, Y. Shahar, Utility elicitation as a classification problem, in: Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence (UAI-98), Madison, WI, 1998, pp. 79-88.
[17] U. Chajewska, D. Koller, R. Parr, Making rational decisions using adaptive utility elicitation, in: Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI-00), Austin, TX, 2000, pp. 363–369.
[18] V. Chandru, J.N. Hooker, Optimization Methods for Logical Inference, Wiley, New York, 1999.
[19] R. Dechter, Bucket elimination: A unifying framework for probabilistic inference, in: Proceedings of the Twelfth Conference on Uncertainty in Artificial Intelligence (UAI-96), Portland, OR, 1996, pp. 211–219.
[20] R. Dechter, Constraint Processing, Morgan Kaufmann, San Francisco, CA, 2003.
[21] J.S. Dyer, Interactive goal programming, Management Science 19 (1972) 62–70.
[22] P.C. Fishburn, Interdependence and additivity in multivariate, unidimensional expected utility theory, International Economic Review 8 (1967) 335–342.
[23] P.C. Fishburn, Utility Theory for Decision Making, Wiley, New York, 1970.
[24] S. French, Decision Theory, Halsted Press, New York, 1986.
[25] C. Gonzales, P. Perny, GAI networks for utility elicitation, in: Proceedings of the Ninth International Conference on Principles of Knowledge Representation and Reasoning (KR2004), Whistler, BC, 2004, pp. 224–234.
[26] C. Guestrin, D. Koller, R. Parr, Max-norm projections for factored MDPs, in: Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI-01), Seattle, WA, 2001, pp. 673–680.
[27] H.A. Holloway, C.C. White III, Question selection for multiattribute decision-aiding, European Journal of Operational Research 148 (2003) 525–543.
[28] E. Horvitz, J. Breese, D. Heckerman, D. Hovel, K. Rommelse, The lumiere project: Bayesian user modeling for inferring goals and needs of software users, in: Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence (UAI-98), Madison, WI, 1998, pp. 256–265.
[29] V.S. Iyengar, J. Lee, M. Campbell, Q-Eval: Evaluating multiple attribute items using queries, in: Proceedings of the Third ACM Conference on Electronic Commerce, Tampa, FL, 2001, pp. 144–153.
[30] R.L. Keeney, H. Raiffa, Decisions with Multiple Objectives: Preferences and Value Trade-offs, Wiley, New York, 1976.
[31] J.A. Konstan, B.N. Miller, D. Maltz, J.L. Herlocker, L.R. Gordon, J. Riedl, Grouplens: Applying collaborative filtering to usenet news, Communications of the ACM 40 (3) (1997) 77–87.
[32] P. Kouvelis, G. Yu, Robust Discrete Optimization and Its Applications, Kluwer, Dordrecht, 1997.

[33] G. Lee, S. Bauer, P. Faratin, J. Wroclawski, Learning user preferences for wireless services provisioning. in: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-04), New York, 2004, pp. 480–487.
[34] G. Loomes, R. Sugden, Regret theory: An alternative theory of rational choice under uncertainty, Economic Journal 92 (1982) 805–824.
[35] A.M. Mármol, J. Puerto, F.R. Fernández, The use of partial information on weights in multicriteria decision problems, Journal of Multicriteria Decision Analysis 7 (1998) 322–329.
[36] G.L. Nemhauser, L.A. Wolsey, Integer Programming and Combinatorial Optimization, Wiley, New York, 1988.
[37] B. O'Sullivan, E. Freuder, S. O'Connell, Interactive constraint acquisition, in: CP-2001 Workshop on User Interaction in Constraint Processing, Paphos, Cyprus, 2001.
[38] P. Pu, B. Faltings, Decision tradeoff using example-critiquing and constraint programming, Constraints 9 (4) (2004) 289–310.
[39] P. Pu, B. Faltings, M. Torrens, User-involved preference elicitation, in: IJCAI-03 Workshop on Configuration, Acapulco, 2003.
[40] J.C. Quiggan, Stochastic dominance in regret theory, The Review of Economic Studies 57 (3) (1990) 503–511.
[41] F. Rossi, A. Sperduti, K.B. Venable, L. Khatib, P.H. Morris, R.A. Morris, Learning and solving soft temporal constraints: An experimental study, in: Proceedings of the Eighth International Conference on Principles and Practice of Constraint Programming, Ithaca, NY, 2002, pp. 249–263.
[42] D. Sabin, R. Weigel, Product configuration frameworks—a survey, IEEE Intelligent Systems and their Applications 13 (4) (1998) 42–49.
[43] A. Salo, R.P. Hämäläinen, Preference ratios in multiattribute evaluation (PRIME)—elicitation and decision procedures under incomplete information, IEEE Transaction on Systems, Man and Cybernetics 31 (6) (2001) 533–545.
[44] L.J. Savage, The Foundations of Statistics, Wiley, New York, 1954.
[45] L.J. Savage, The theory of statistical decision, Journal of the American Statistical Association 46 (1986) 55–67.
[46] T. Schiex, H. Fargier, G. Verfaillie, Valued constraint satisfaction problems: Hard and easy problems, in: Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95), Montreal, 1995, pp. 631–637.
[47] D. Schuurmans, R. Patrascu, Direct value approximation for factored MDPs, in: Advances in Neural Information Processing Systems 14 (NIPS-2001), Vancouver, 2001, pp. 1579–1586.
[48] O. Toubia, J. Hauser, D. Simester, Polyhedral methods for adaptive choice-based conjoint analysis, Technical Report 4285-03, Sloan School of Management, MIT, Cambridge, 2003.
[49] J. von Neumann, O. Morgenstern, Theory of Games and Economic Behavior, Princeton University Press, Princeton, NJ, 1944.
[50] T. Wang, C. Boutilier, Incremental utility elicitation with the minimax regret decision criterion, in: Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI-03), Acapulco, 2003, pp. 309–316.
[51] M. Weber, Decision making with incomplete information, European Journal of Operational Research 28 (1987) 44–57.
[52] C.C. White III, A.P. Sage, S. Dozono, A model of multiattribute decisionmaking and trade-off weight determination under uncertainty, IEEE Transactions on Systems, Man and Cybernetics 14 (2) (1984) 223–229.