Constraint-based Optimization with the Minimax Decision Criterion

Craig Boutilier[1], Relu Patrascu[2], Pascal Poupart[1], and Dale Schuurmans[2]

[1] Dept. of Computer Science, University of Toronto, Toronto, ON, M5S 3H5, CANADA,
cebly,ppoupart@cs.toronto.edu
[2] School of Computer Science, University of Waterloo, Waterloo, ON, N2L 3G1, CANADA,
rpatrascu,dale@cs.uwaterloo.ca

Abstract. In many situations, a set of hard constraints encodes the feasible configurations of some system or product over which users have preferences. We consider the problem of computing a best feasible solution when the user's utilities are partially known. Assuming bounds on utilities, efficient mixed integer linear programs are devised to compute the solution with minimax regret while exploiting generalized additive structure in a user's utility function.

1 Introduction

The problem of interactive decision making has received a fair amount of attention over the years [10, 17], but recently has seen increasing interest within AI as automated decision aids become more prevalent. As has been argued elsewhere [6, 4], there are many situations in which the set of decisions and their dynamics are fixed, while the *utility functions* of different users vary widely. In such a case, some form of utility elicitation must be undertaken in order to capture user preferences to a sufficient degree to allow an (approximately) optimal decision to be taken. Different approaches to this problem have been proposed, including Bayesian methods that quantify uncertainty about preferences probabilistically [7, 4], and methods that simply pose constraints on the set of possible utility functions and refine these incrementally [17, 5, 16].

These issues arise as well in the context of constraint-based optimization problems. For instance, in a car rental scenario, possible configurations are defined by attributes such as automobile size and class, manufacturer, seating and luggage capacity, etc. Available cars are are limited by the configurations offered by manufacturers and stock availability, with hard constraints used to encode infeasible configurations (e.g., no luxury sedans have 4-cylinder engines). Different customers have different preferences for configurations in this restricted decision space [14], and this information must be obtained in an effective way. Typically, categorical preferences are obtained from the customer, and imposed as constraints; but if no feasible solution is found, these constraints are relaxed incrementally.

While interactive preference elicitation has received little attention in the CSP community, optimizing with respect to a given set of preferences over configurations has been studied extensively, with many frameworks proposed for modeling such systems

[15, 3]. Most frameworks can be viewed as adding "soft" constraints that have associated penalties or values that indirectly represent a user's preferences for different configurations. However, modeling preferences as constraints and assuming complete utility information is often problematic (e.g., full utility information may not be required to make an optimal decision, or users may be unwilling to provide full information).

In this paper, we adopt a somewhat different view. We assume a user's preferences are represented directly as a utility function over possible configurations. In the car rental scenario, this utility function can be thought as a measure of the value (not necessarily monetary) of each car from the customer's point of view. Given a utility function and the hard constraints defining the decision space, we have a standard constraint-based optimization problem. 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 the problem of "optimizing" in the presence of partial utility information. Specifically, we assume that bounds on utility function parameters are provided, and consider the problem of finding a feasible solution that minimizes *maximum regret* [11] within the space of feasible utility functions. We show that this minimax problem can be formulated and solved using a set 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* [1] (which includes linear functions and graphical models like UCP-nets [5] as special cases). We derive two solution techniques for solving such structured problems: the first encodes the entire minimax problem as a single MIP using a cost-network to formulate a compact set of constraints; the second gives rise to a MIP with fewer variables combined with an effective constraint generation procedure.

Though our emphasis is on solving problems using the minimax regret criterion, we also briefly discuss how preference elicitation relates to this model. Specifically, we describe methods that can be used to refine utility uncertainty in a way that quickly reduces minimax regret. Throughout, our emphasis is on the compact formulation and solution of the constrained optimization problems as mixed integer programs. While these can be solved using a variety of techniques, including branch-and-bound methods with various constraint propagation techniques, we do not consider specialized methods for solving these MIPs (our experiments, for example, use generic MIP solvers). We leave this investigation to future research.

2 Constraint-based Optimization and Minimax Regret

We begin by describing the basic problem assuming a known utility function to establish background and notation, and then define the minimax regret decision criterion for solving constraint-based decision problems given only incomplete utility information.

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. An assignment $\mathbf{x} \in \mathit{Dom}(\mathbf{X})$ is often referred to as a *state*. For simplicity of presentation, we assume these attributes are boolean, but nothing important depends on this. We also have a set of hard constraints $\mathcal{C}$ over these attributes. Each constraint $\mathcal{C}_\ell$, $\ell = 1, ..., L$, is defined over a set $\mathbf{X}[\ell] \subset \mathbf{X}$, and thus $\mathcal{C}_\ell$ induces a set of legal configurations of attributes in $\mathbf{X}[\ell]$. 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 the legal configurations of X_1, X_2, X_3. We let $\mathit{Feas}(\mathbf{X})$ denote the subset of *feasible states* (i.e., assignments satisfying $\mathcal{C}$).

Suppose we have a known utility function $u : \mathit{Dom}(\mathbf{X}) \rightarrow \mathbf{R}$. Our aim is to find an optimal feasible state $\mathbf{x}^*$; i.e., any

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

For this reason, we sometimes call feasible states *decisions*. This problem can be formulated in an explicit fashion as a (linear) 0-1 integer program:

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

where we have

- variables $I_\mathbf{x}$: for each $\mathbf{x} \in \mathit{Dom}(\mathbf{X})$, $I_\mathbf{x}$ is a boolean variable indicating whether $\mathbf{x}$ is the decision made (i.e., state chosen).
- variables X_i: X_i is a 0-1 variable corresponding to the ith attribute.
- coefficients $u_\mathbf{x}$: for each $\mathbf{x} \in \mathit{Dom}(\mathbf{X})$, 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 $I_\mathbf{x} \leq X_i$; and if X_i is false in $\mathbf{x}$, we constrain $I_\mathbf{x} \leq 1 - X_i$. We denote by $\mathcal{A}$ these constraints.
- constraint set $\mathcal{C}$: we impose each feasibility constraint $\mathcal{C}_\ell$ on the attributes $X_i \in \mathbf{X}[\ell]$. Logical constraints can be written in a natural way as linear constraints [8].

Note that this formulation assures 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.[3]

2.2 Graphical Utility Models

Unfortunately the IP formulation above is not compact since there is one $I_\mathbf{x}$ variable per state and the number of states is exponential in the number of attributes. In such *flat* utility functions, it is not generally possible to formulate the optimization concisely. By constrast, if some structure on the utility function is imposed, say, in the form of a *factored* graphical model, we are then generally able to reduce the number of variables to be linear in the number of parameters of the graphical model. We consider here the GAI (generalized additive independence) model [1] because of its generality (encompassing both linear models [13] and UCP-nets [5] as special cases).[4]

[3] We assume the utility function is non-negative.

[4] 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.

Specifically, assume that our utility function can be written as the sum of K local utility functions, or *factors*, over small sets of variables:

$$u(\mathbf{x}) = \sum_{k \leq K} f^k(\mathbf{x}[k]). \tag{2}$$

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 $\mathbf{X}[k]$. An IP similar to Eq. 1 can be used to solve for the optimal decision in the case of a GAI model:

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

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 instance $\mathbf{x}[k] \in Dom(\mathbf{X}[k])$. 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 constraint set $\mathcal{A}$ as before, and constraints $\mathcal{C}$ is also 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 will be 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 Sec. 2.1.[5]

2.3 Minimax Regret

If the utility function is unknown, then we have a slightly different problem. We cannot maximize expected utility because the utility function is unspecified. However, if we have constraints on the utility function (e.g., in the form of bounds), we can optimize using other criteria. A very natural criterion is *minimax regret* [11, 5, 16]: prefer the (feasible) assignment $\mathbf{x}$ that obtains minimum max-regret, where max-regret is the largest quantity by which one could "regret" choosing action $\mathbf{x}$ (while allowing the utility function to vary within the bounds).

More formally, let $\mathcal{U}$ denote the set of feasible utility functions defined by imposed upper and lower bounds on the utility value $u(\mathbf{x})$ of each state $\mathbf{x}$. The *pairwise regret* of state $\mathbf{x}$ with respect to state $\mathbf{x}'$ over feasible utility set $\mathcal{U}$ is defined as

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

which is the most one could regret choosing $\mathbf{x}$ instead of $\mathbf{x}'$ (e.g., if an adversary could impose any utility function in $\mathcal{U}$). The *maximum regret* of decision $\mathbf{x}$ is:

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

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

[5] Generally, this IP would be solved using some form of search directly on the X_i variables, in which case there would be no need to explicitly represent $I_{\mathbf{x}[k]}$ (state) variables.

The *minimax regret* of feasible utility set $\mathcal{U}$ is:

$$MMR(\mathcal{U}) = \min_{\mathbf{x}} \; MR(\mathbf{x},\mathcal{U}) \tag{7}$$

$$= \min_{\mathbf{x}} \; \max_{\mathbf{x}'} \; \max_{u\in\mathcal{U}} \; u(\mathbf{x}') - u(\mathbf{x}) \tag{8}$$

If the only information we have about a user's utility function is that it lies in the set $\mathcal{U}$, then a decision $\mathbf{x}^*$ that minimizes max-regret (i.e., $MR(\mathbf{x}^*,\mathcal{U}) = MMR(\mathcal{U})$) seems reasonable. Specifically, without distributional information over the set of possible utility functions, choosing (or recommending) $\mathbf{x}^*$ minimizes the worst case loss with respect to possible realizations of the utility function $u \in \mathcal{U}$. Our goal is now to formulate the minimax regret optimization (Eq. 8) in a computationally tractable way.

3 Minimax Regret with Flat Utility Models

If we make no assumptions about the structure of the utility function, Eq. 8 can be interpreted directly as a semi-infinite, quadratic, mixed-integer program (MIP):

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

where we have

1. continuous variables $M_{\mathbf{x}}$ that denote the max regret when state $\mathbf{x}$ is chosen,
2. 0-1 indicator variables $I_{\mathbf{x}}$ that select the optimal state $\mathbf{x}$,
3. utility coefficients $u_{\mathbf{x}}$, which give the utility of each state $\mathbf{x}$ (where, since the utility function u is unkown, there is an infinite number of constraints $M_{\mathbf{x}} \geq u_{\mathbf{x}} - u'_{\mathbf{x}}$; one per utility function $u \in \mathcal{U}$),
4. constraint sets $\mathcal{A}$ and $\mathcal{C}$; the same as before.

The set of constraints on the M_x variables is problematic. First, it is a continuous set, requiring that we enumerate all utility values $u_{\mathbf{x}}$ and $u_{\mathbf{x}'}$ corresponding to any utility function $u \in \mathcal{U}$. Furthermore, it is critical that we restrict our attention to those constraints associated with $\mathbf{x}'$ in the *feasible* set of states (i.e., those satisfying $\mathcal{C}$). Fortunately, we can often tackle this seemingly complex optimization in much simpler stages.

In this paper we consider the case where all utility parameters $u_{\mathbf{x}}$ are independent and have simple upper and lower bounds (e.g., asking standard gamble queries would provide such bounds). 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 $\mathcal{U}$. These assumptions allow us to compute the minimax regret in three simpler stages, which we now describe.[6]

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: $R(\mathbf{x},\mathbf{x}',\mathcal{U}) = u'_{\mathbf{x}}\!\uparrow - u_{\mathbf{x}}\!\downarrow$ if $\mathbf{x} \neq \mathbf{x}'$, and $R(\mathbf{x},\mathbf{x}',\mathcal{U}) = 0$ if $\mathbf{x} = \mathbf{x}'$. 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.

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

Second, using Eq. 5, we can also compute the max regret $MR(\mathbf{x},\mathcal{U})$ 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 pairwise regret:

$$MR(\mathbf{x},\mathcal{U}) = \max_{\mathbf{x}' \in Feas(\mathbf{X}')} r_{\mathbf{x},\mathbf{x}'}. \tag{9}$$

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

$$MR(\mathbf{x},\mathcal{U}) = \max_{\{I_{\mathbf{x}'}, X'_i\}} \sum_{\mathbf{x}'} r_{\mathbf{x},\mathbf{x}'} I_{\mathbf{x}'} \quad \text{subject to } \mathcal{A} \text{ and } \mathcal{C}. \tag{10}$$

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

$$MMR(\mathcal{U}) = \min_{\mathbf{x} \in Feas(\mathbf{X})} m_{\mathbf{x}} \tag{11}$$

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

$$MMR(\mathcal{U}) = \min_{\{I_{\mathbf{x}}, X_i\}} \sum_{\mathbf{x}} m_{\mathbf{x}} I_{\mathbf{x}} \quad \text{subject to } \mathcal{A} \text{ and } \mathcal{C} \tag{12}$$

In this flat model case, the two IPs above are not necessarily practical, since the require one indicator variable per state. However, this reformulation does show that the original quadratic MIP with continuous constraints can be solved in stages using finite, linear IPs. More importantly, these intuitions will next be applied to develop an analogous procedure for graphical utility models.[7]

4 Minimax Regret with Graphical Models

The optimization for flat models is interesting in that it allows us to get a good sense of how minimax regret in a constraint-satisfaction setting works. 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 utilities 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, $k \leq K$, defined over local families $\mathbf{X}[k]$, as described in Sec. 2.2. The parameters of this utility function are denoted by $u_{\mathbf{x}[k]} = f^k(\mathbf{x}[k])$, where

[7] Note that this strategy hinges on the fact that we can independently determine upper and lower bounds on the utility value of each state. If utility values are correlated by more complicated constraints, this strategy may not work.

$\mathbf{x}[k]$ ranges over $Dom(\mathbf{X}[k])$. 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. 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 Sec. 2.3. We now show how to compute each of these quantities in turn.

4.1 Computing Pairwise Regret and Max Regret

As in the unfactored case (Sec. 3), it is straightforward to compute the pairwise regret of any pair of states $\mathbf{x}$ and $\mathbf{x}'$. For each factor f^k and assignment pair $\mathbf{x}[k]$,$\mathbf{x}'[k]$, we define the *local pairwise regret*: $r_{\mathbf{x}[k],\mathbf{x}'[k]} = u_{\mathbf{x}'[k]}\Uparrow - u_{\mathbf{x}[k]}\Downarrow$ when $\mathbf{x}[k] \neq \mathbf{x}'[k]$, and $r_{\mathbf{x}[k],\mathbf{x}'[k]} = 0$ when $\mathbf{x}[k] = \mathbf{x}'[k]$. With factored models, $R(\mathbf{x},\mathbf{x}',\mathcal{U})$ is the sum of of local pairwise regrets:

$$R(\mathbf{x},\mathbf{x}',\mathcal{U}) = \sum_k r_{\mathbf{x}[k],\mathbf{x}'[k]} \tag{13}$$

We can compute max regret $MR(\mathbf{x},\mathcal{U})$ by substituting Eq. 13 into Eq. 5:

$$MR(\mathbf{x},\mathcal{U}) = \max_{\mathbf{x}' \in Feas(\mathbf{X}')} \sum_k r_{\mathbf{x}[k],\mathbf{x}'[k]} \tag{14}$$

which leads to the following IP formulation:

$$MR(\mathbf{x},\mathcal{U}) = \max_{\{I_{\mathbf{x}'[k]},X'_i\}} \sum_k \sum_{\mathbf{x}'[k]} r_{\mathbf{x}[k],\mathbf{x}'[k]} I_{\mathbf{x}'[k]} \;\; \text{subject to } \mathcal{A} \text{ and } \mathcal{C} \tag{15}$$

The above IP differs from its flat counterpart (Eq. 10) in the use of one variable $I_{\mathbf{x}'[k]}$ per utility parameter, and is thus more compact and efficiently solvable.

4.2 Computing Minimax Regret

We can compute minimax regret $MMR(\mathcal{U})$ by substituting Eq. 14 into Eq. 7:

$$MMR(\mathcal{U}) = \min_{\mathbf{x}\in Feas(\mathbf{X})} \max_{\mathbf{x}' \in Feas(\mathbf{X}')} \sum_k r_{\mathbf{x}[k],\mathbf{x}'[k]} \tag{16}$$

which leads to the following MIP formulation:

$$MMR(\mathcal{U}) = \min_{\{I_{\mathbf{x}[k]},X_i\}} \max_{\mathbf{x}' \in Feas(\mathbf{X}')} \sum_k \sum_{\mathbf{x}[k]} r_{\mathbf{x}[k],\mathbf{x}'[k]} I_{\mathbf{x}[k]} \;\; \text{subject to } \mathcal{A} \text{ and } \mathcal{C} \tag{17}$$

$$= \min_{\{I_{\mathbf{x}[k]},X_i,M\}} M$$

$$\text{subject to } \begin{cases} M \geq \sum_k \sum_{\mathbf{x}[k]} r_{\mathbf{x}[k],\mathbf{x}'[k]} I_{\mathbf{x}[k]} \;\; \forall \mathbf{x}' \in Feas(\mathbf{X}') \\ \mathcal{A} \text{ and } \mathcal{C} \end{cases} \tag{18}$$

In Eq. 17, we introduce the variables for the minimization, while in Eq. 18 we transform the minimax program into a min program. The new continuous variable M corresponds to the max regret of any state. In contrast with the flat IP (Eq. 12), 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. 18 has one constraint per feasible state $\mathbf{x}'$. Nevertheless, we can get around the potentially large number of constraints in either of two ways.

Constraint Generation The first technique we consider for dealing with the large number of constraints in Eq. 18 is through the use of *constraint generation*, a common technique in operations research for solving problems with large numbers of constraints (much like cutting plane and column generation methods). This approach proceeds by repeatedly solving the MIP in Eq. 18, 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 $\mathbf{x}$.

Recall that finding the feasible $\mathbf{x}'$ that maximizes $R(\mathbf{x}, \mathbf{x}', \mathcal{U})$ involves solving a single IP given by Eq. 15. We then impose the specific constraint associated with witness $\mathbf{x}'$ and re-solve the MIP in Eq. 18 at the next iteration with this additional constraint. It is not hard to see that if no constraint is violated at the current solution $\mathbf{x}$, then $\mathbf{x}$ is the minimax-optimal configuration. 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 is relatively simple and in practice (as we will see) tends to generate a very small number of constraints [18]. 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.

A Cost Network Formulation A second technique for dealing with the large number of constraints in Eq. 18 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 [12]. The main benefit of the cost network approach is that, in principle, it allows us to formulate a MIP that only expresses a feasible number of constraints.[8]

To formulate a compact constraint system, we first transform the MIP of Eq. 18 into the following equivalent MIP by introducing penalty terms $\rho_{\mathbf{x}[\ell]}$ for each feasibility constraint $\mathcal{C}_\ell$:

$$
\begin{aligned}
\mathit{MMR}(\mathcal{U}) &= \min_{\{I_{\mathbf{x}[k]}, X_i, M\}} M \\
&\quad \text{subject to} \begin{cases} M \geq \sum_k \sum_{\mathbf{x}[k]} r_{\mathbf{x}[k],\mathbf{x}'[k]} I_{\mathbf{x}[k]} + \sum_\ell \rho_{\mathbf{x}'[\ell]} & \forall \mathbf{x}' \in \mathit{Dom}(\mathbf{X}') \\ \mathcal{A} \text{ and } \mathcal{C} \end{cases} \\
&= \min_{\{I_{\mathbf{x}[k]}, X_i, M\}} M \\
&\quad \text{subject to} \begin{cases} M \geq \sum_k R_{\mathbf{x}'[k]} + \sum_\ell \rho_{\mathbf{x}'[\ell]} & \forall \mathbf{x}' \in \mathit{Dom}(\mathbf{X}') \\ R_{\mathbf{x}'[k]} = \sum_{\mathbf{x}[k]} r_{\mathbf{x}[k],\mathbf{x}'[k]} I_{\mathbf{x}[k]} & \forall k, \mathbf{x}'[k] \in \mathit{Dom}(\mathbf{X}'[k]) \\ \mathcal{A} \text{ and } \mathcal{C} \end{cases}
\end{aligned} \tag{19}
$$

[8] We have observed, however, the previous constraint generation approach is usually faster in practice and much easier to implement, even though it lacks the same worst case run-time guarantees.

The MIP of Eq. 18 has one constraint on M per feasible state $\mathbf{x}'$, whereas the MIP of Eq. 19 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 essentially 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.

The above transformation may seem like a step backwards because we have increased the number of constraints, however it will now allow us to rewrite the constraints in a compact form.

Now, 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:

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

This maximization can be computed efficiently by using *variable elimination* [9]; a well-known form of non-serial dynamic programming [2]. The idea is to distribute the $\max$ operator inward over the summations, and then collect the results as new terms which are then successively pulled out.

To illustrate, consider the following simple 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 $\rho^1(x_1)$, $\rho^2(x_3, x_4)$. We then obtain

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

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 $\max$ as follows.

$$\begin{aligned}
&\text{Let } & A_{x'_1,x'_3} &= \max_{x'_4} R_{x'_1,x'_4} + \rho_{x'_3,x'_4} \\
&\text{then let } & A_{x'_1,x'_2} &= \max_{x'_3} R_{x'_2,x'_3} + A_{x'_1,x'_3} \\
&\text{then let } & A_{x'_1} &= \max_{x'_2} R_{x'_1,x'_2} + A_{x'_1,x'_2} \\
&\text{then } & \mathit{GLB} &= \max_{x'_1} \rho_{x'_1} + A_{x'_1}.
\end{aligned}$$

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 variable.

Based on this procedure, we can substitute all the constraints on M with the following compact set of constraints that analytically encodes the greatest lower bound on

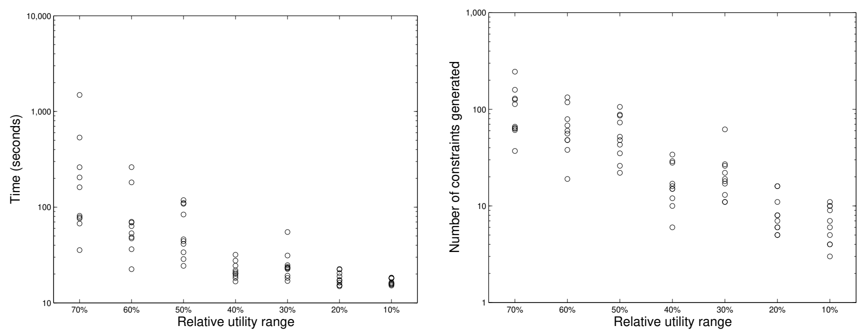

Fig. 1. Car Problem.

M:

$$\begin{aligned} A_{x_1',x_3'} &\geq R_{x_1',x_4'} + \rho_{x_3',x_4'} & \forall x_1', x_3', x_4' \in \mathit{Dom}(X_1', X_3', X_4') \\ A_{x_1',x_2'} &\geq R_{x_2',x_3'} + A_{x_1',x_3'} & \forall x_1', x_2', x_3' \in \mathit{Dom}(X_1', X_2', X_3') \\ A_{x_1'} &\geq R_{x_1',x_2'} + A_{x_1',x_2'} & \forall x_1', x_2' \in \mathit{Dom}(X_1', X_2') \\ M &\geq \rho_{x_1'} + A_{x_1'} & \forall x_1' \in \mathit{Dom}(X_1') \end{aligned}$$

By encoding constraints in this way, the constraint system specified by the MIP in Eq. 19 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 only linear in the number of parameters of the utility function, and the number of auxiliary variables and constraints that are added is locally exponential w.r.t. 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.

5 Empirical Results

To test the plausibility of this approach we implemented the solution strategy outlined above and ran a series of experiments to determine whether graphical structure was sufficient to permit practical solution times. We implemented the constraint generation approach outlined in Sec. 4.2 and used CPLEX as the generic IP solver. Our experiments considered two realistic domains—car rentals and house buying—as well as artificially generated problems. In each case we imposed a factored graphical structure to reduce the required number of utility parameters (upper and lower bounds).

For the house buying 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. 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 only on one, two or three variables. In total, the number of

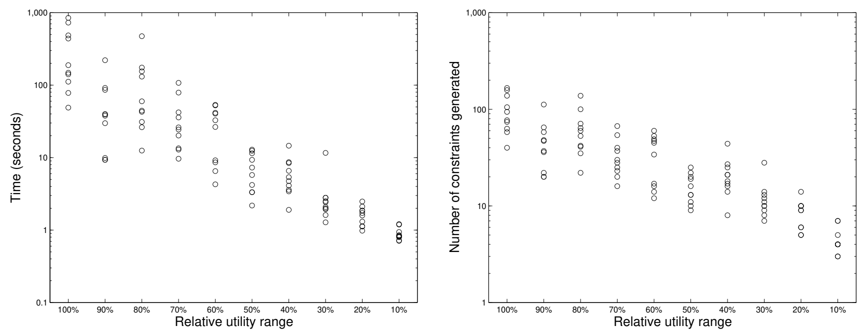

Fig. 2. Real estate problem.

local utility values (utilities for local configurations) was reduced to 160. 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 nonfactored model. The local utility functions represented complementarities and substitutabilities between variables, such as requiring a large yard and a fence to allow a pool, etc.

The rental car 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, etc. 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. Constraints encode infeasible configurations (e.g., no luxury sedans have 4-cylinder engines).

For both the car and real estate problems, we first computed the configuration with minimax regret given manually chosen bounds on the utility functions. The generation technique of Sec. 4.2 took 40 sec for the car problem and 2 sec for the real estate problem. Interestingly, only 7 constraints were generated in finding the minimax-optimal configuration in both the car and real-estate problems (out of the 61,917,360,000 and 47,775,744 constraints, respectively). The structure exhibited by the utility functions of each problem is largely responsible for this small number of required constraints.

In practice, the minimax regret techniques proposed in this paper would normally be interleaved with some preference elicitation technique. 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 an experiment where we randomly set bounds, but with varying degrees of tightness. Figures 1 and 2 show how tightening the bounds decreases the running time exponentially and the number of constraints generated. 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. Figures 1 and 2 show scatterplots of random problems for varying percentages. As those figures suggest, a significant speed up is obtained as elicitation converges to the true utilities. Intuitively, the optimization required to com-

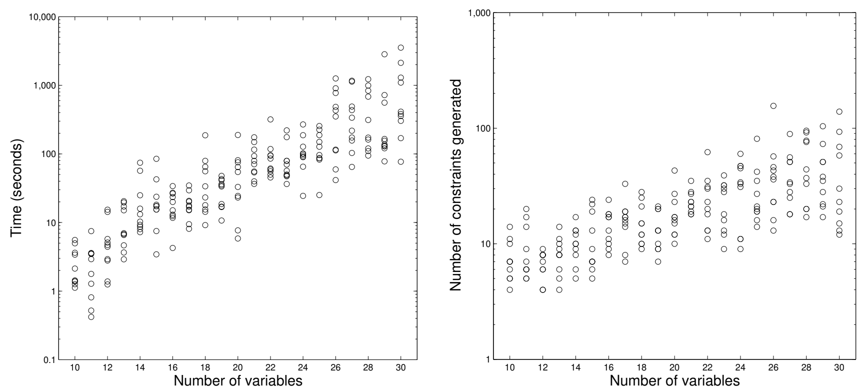

Fig. 3. Artificial random problems – varying sizes

pute 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 artificially generated problems. A set of random problems of varying sizes was constructed by randomly setting the utility bounds and the variables each utility factor depends on. Each utility factor depends on at most 3 variables and each variable has at most 5 values. Figure 3 shows the results as 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 constraints (if they were all enumerated). For problems with 10 variables, only 8 constraints were necessary (out of 278,864) on average and for problems of 30 variables only 47 constraints were necessary (out of 2.8×10^{16}) on average.

We also tested the impact of elicitation on the efficiency of our constraint generation technique in Figure 4. Here, problems of 30 variables and 30 factors were generated randomly while varying the relative range of the utilities w.r.t. some predetermined range. Each factor has at most 3 variables chosen randomly and each variable can take at most 5 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.

6 Concluding Remarks

There are a number of directions in which this work can be extended. Of critical importance is the development of good elicitation strategies that reduce minimax regret quickly. While this work has focused on the computation of minimax-optimal assignments to variables, our current thrust is the incorporation of this approach into querying

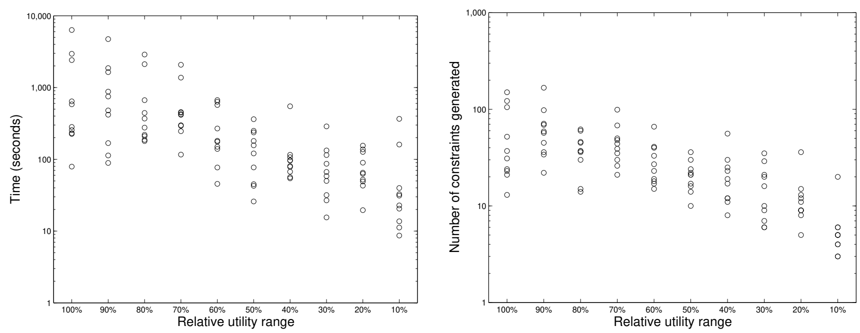

Fig. 4. Artificial random problems – varying relative utility range.

strategies that can be used to tighten only the most relevant utility parameter bounds. We have devised several elicitation strategies that we hope will work well in practice; but these have yet to be implemented (so their performance still needs to be verified). We briefly describe two of these methods.

The *optimistic query method* works as follows: at each iteration we compute the maximax optimal state $\mathbf{x}^*$ (i.e, that with the greatest upper bound on utility). We then query the user about the utility parameters of that factor in such a way that its lower bound on utility is raised to be close (say, within ε) to the upper bound of the state with the second-highest upper bound (which can be computed in a similar way), or its upper bound is reduced to below that of the second state. In either case, we have made progress: in the first case, we have reduced minimax regret to ε; in the second case, we have reduced the regret of every other decision (by an amount equal to the reduction of the upper bound for $\mathbf{x}^*$, less ε).

The *current solution query method* involves computing the minimax optimal state $\mathbf{x}^*$ using one of the methods described, as well as its regret-maximizing "witness" $\mathbf{x}^w$ (i.e., the state an adversary chooses in order to maximize our regret). We then ask queries about the utility parameters at the both states (e.g., asking a midpoint query about each parameter, thus reducing each interval by half). The intuition behind this approach is that gaining tighter information about the current minimax optimal allocation and its witness is the best way to ensure an improvement in regret level (since these are the parameters that play a role in the active constraints at the current solution).

Apart from elicitation, we are also exploring the use of search and constraint-propagation methods for solving the constraint-optimization problems associated with computing minimax regret. 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 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, etc. Finally, we are quite

interested in the possibility of integrating Bayesian methods for reasoning about uncertain utility functions with the constraint-based representation of the decision space.
Acknowledgements: Poupart was supported by a scholarship provided by Precarn Incorporated through the Institute for Robotics and Intelligent Systems.

References

1. F. Bacchus and A. Grove. Graphical models for preference and utility. In *Proc. 11th Conf. on Uncertainty in AI*, pp.3–10, Montreal, 1995.
2. U. Bertele and F. Brioschi. *Nonserial Dynamic Programming*. Academic, Orlando, 1972.
3. S. Bistarelli, U. Montanari, and F. Rossi. Semiring-based constraint satisfaction and optimization. *Journal of the ACM*, 44(2):201–236, 1997.
4. C. Boutilier. A POMDP formulation of preference elicitation problems. In *Proc. 18th National Conf. on AI*, pp.239–246, Edmonton, 2002.
5. C. Boutilier, F. Bacchus, and R. I. Brafman. UCP-Networks: A directed graphical representation of conditional utilities. In *Proc. 17th Conf. on Uncertainty in AI*, pp.56–64, 2001.
6. U. Chajewska, L. Getoor, J. Norman, and Y. Shahar. Utility elicitation as a classification problem. In *Proc. 14th Conf. on Uncertainty in AI*, pp.79–88, Madison, WI, 1998.
7. U. Chajewska, D. Koller, and R. Parr. Making rational decisions using adaptive utility elicitation. In *Proc. 17th National Conf. on AI*, pp.363–369, Austin, TX, 2000.
8. V. Chandru and J. Hooker. *Optimization Methods for Logical Inference*. Wiley, N.Y., 1999.
9. R. Dechter. Bucket elimination: A unifying framework for probabilistic inference. In *Proc. Twelfth Conf. on Uncertainty in AI*, pp.211–219, Portland, OR, 1996.
10. J. S. Dyer. Interactive goal programming. *Management Science*, 19:62–70, 1972.
11. S. French. *Decision Theory*. Halsted Press, New York, 1986.
12. C. Guestrin, D. Koller, and R. Parr. Max-norm projections for factored MDPs. In *Proc. 17th Intl. Joint Conf. on AI*, pp.673–680, Seattle, 2001.
13. R. L. Keeney and H. Raiffa. *Decisions with Multiple Objectives: Preferences and Value Trade-offs*. Wiley, New York, 1976.
14. D. Sabin and R. Weigel. Product configuration frameworks—a survey. *IEEE Intelligent Systems and their Applications*, 13(4):42–49, 1998.
15. T. Schiex, H. Fargie, and G. Verfaillie. Valued constraint satisfaction problems: Hard and easy problems. In *Proc. 14th Intl. Joint Conf. on AI*, pp.631–637, Montreal, 1995.
16. T. Wang and C. Boutilier. Incremental utility elicitation with the minimax regret decision criterion. In *Proc. 18th Intl. Joint Conf. on AI*, to appear, Acapulco, Mexico, 2003.
17. C. C. White, III, A. P. Sage, and S. Dozono. A model of multiattribute decision making and trade-off weight determination under uncertainty. *IEEE Transactions on Systems, Man and Cybernetics*, 14(2):223–229, 1984.
18. D. Schuurmans, R. Patrascu. Direct value-approximation for factored MDPs. In Advances in Neural Information Processing 14, pp. 1579-1586, MIT Press, Cambridge, MA, 2002.