Assessing Regret-based Preference Elicitation with the UTPREF Recommendation System

Darius Braziunas
Department of Computer Science
University of Toronto, Canada
darius@cs.toronto.edu

Craig Boutilier
Department of Computer Science
University of Toronto, Canada
cebly@cs.toronto.edu

ABSTRACT

Product recommendation and decision support systems must generally develop a model of user preferences by querying or otherwise interacting with a user. Recent approaches to elicitation using *minimax regret* have proven to be very powerful in simulation. In this work, we test both the effectiveness of regret-based elicitation, and user comprehension and acceptance of minimax regret in user studies. We report on a study involving 40 users interacting with the UTPREF Recommendation System, which helps students navigate and find rental accommodation. UTPREF maintains an explicit (but incomplete) generalized additive utility (GAI) model of user preferences, and uses minimax regret for recommendation. We assess the following general questions: How effective is regret-based elicitation in finding optimal or near-optimal products? Do users understand and accept the minimax regret criterion in practice? Do decision-theoretically valid queries for GAI models result in more accurate assessment than simpler, ad hoc queries? On the first two issues, we find that the minimax regret decision criterion is effective, understandable, and intuitively appealing. On the third issue, we find that simple, semantically ambiguous query types perform as well as more demanding, semantically valid queries for GAI models. We also assess the relative difficulty of specific query types.

Categories and Subject Descriptors

H.5.2 [**Information Interfaces and Presentation**]: User Interfaces; I.2.6 [**Artificial Intelligence**]: Learning

General Terms

Experimentation, Human Factors

Keywords

recommender systems, utility elicitation, user studies

1. INTRODUCTION

Product recommendation is a fundamental task in e-commerce. Recommender systems assist users in the navigation of large product spaces and recommend decisions in the presence of many alternatives. Like any decision support system, a recommender system has to elicit, assess, or otherwise estimate a user's preferences or utility function over product alternatives in order to ensure appropriate recommendations are made.

Different approaches to product recommendation adopt a variety of stances on preference assessment. For instance, *critiquing systems* [10, 24] facilitate navigation by allowing the user to request changes to a candidate product description in specific dimensions (e.g., increasing one attribute) and returning products that vary in that dimension but are otherwise similar. Such navigational aids do not usually develop an explicit user preference model (though there are some exceptions [20, 24]). *Collaborative filtering* systems, by contrast, attempt to make explicit recommendations by discovering commonality among user product ratings [15, 9]. While rarely formulated as developing explicit utility models, many of these models can be so interpreted (e.g., matrix factorization methods [21] can be viewed as learning linear user utility models over latent product attributes). Both collaborative filtering and critiquing have found popularity in product recommendation due to the minimal data requirements and burden imposed on users.

Decision-theoretic approaches to preference assessment construct an explicit model of user preferences. For instance, many models have been proposed in which users answer preference queries incrementally, each response posing constraints on the parameters of a user's utility function [26, 5, 22, 23]. Such models offer stronger guarantees on decision quality, but often at the expense of much more intense data requirements. Furthermore, such models are developed or deployed largely for the case of restrictive *additive* utility models, while strategies for query choice are often ad hoc.

Recently, the *minimax regret* has been proposed as an intuitive criterion for decision making in the presence of utility function uncertainty as well as an effective driver of preference elicitation [22, 4]. It has been applied to unstructured product models [25], additive models [5] and generalized additive models [4, 8]. In simulation, it has proven to work extremely well. However, no significant study has explored the effectiveness and intuitive appeal of minimax regret-based elicitation with real users. In this paper, we present the first full-fledged user studies with minimax regret.

We first describe the UTPREF Recommendation System, a fully implemented system that helps students navigate and find rental accommodation (or "apartments") from a university housing database. UTPREF assumes a multiattribute, *generalized additive utility model* of a student's preferences over housing features, and asks queries of several distinct forms about these preferences. Responses impose constraints on the parameters of the student's utility model, and the system uses the minimax regret decision criterion for several purposes: to recommend a rental unit at any point in the interaction; to assess the quality of the recommended unit, specifically bounding how far it is from the (unknown, user-specific) optimal apartment

EC'10, June 7–11, 2010, Cambridge, Massachusetts, USA.

in the database; and to select queries that have the greatest potential to improve the recommendation. UTPREF exploits minimax regret to "prove" that the optimal apartment has been found with very limited information about the user's underlying utility function.

We then present the results of a study in which 40 participants used the UTPREF system under a variety of conditions to find their most preferred apartment from the database. We assess how far the UTPREF-recommended apartment is from the true optimum (using a method discussed below). Our study addresses several important questions. How effective is regret-based elicitation in finding (near-) optimal products? Do users understand the minimax regret criterion and its recommendations in practice? Since UTPREF uses GAI utility models, which are more flexible, but also more complex than commonly used additive models, we are also interested in their value: do they provide for higher quality decisions in practice than an additive approximation. We also assess whether decision-theoretically valid queries for GAI models allow for more accurate estimation of preferences than simpler, ad hoc queries. Finally, we measure the relative difficulty of specific query types for users. Overall, we find that minimax regret is both effective, understandable, and intuitively appealing. We also find some slight evidence that GAI models can provide a more accurate picture of user utility than additive models. However, the complex queries required—at least theoretically—by GAI models seem to be unnecessary in practice and can be replaced by simpler queries for reasons we explain below.

2. MULTIATTRIBUTE PREFERENCES

We begin with a brief overview of the foundations of multiattribute utility functions and utility elicitation.

2.1 Multiattribute utility functions

We consider recommending a product (or some other decision) from a set of available alternatives $\mathbf{X}$ that reflects the preferences of a specific user. For example, in the UTPREF system described below, $\mathbf{X}$ is a database of vacant rental units (hereafter "apartments"). In all but the simplest domains, products will be characterized by a collection of features or *attributes*. We assume N attributes $X_1, \ldots, X_N$, each with finite domains $Dom(X_i)$; e.g., *Bdrm* (number of bedrooms), *Dist* (distance from university), *AT* (apartment type), etc. (We discuss price as a special attribute below.) In multiattribute domains, attribute instantiations define the set of *potential outcomes* $\mathbf{X} = Dom(X_1) \times \cdots \times Dom(X_N)$. Generally, the outcome set will be some subset $\mathbf{X}_D \subseteq \mathbf{X}$ (e.g., instantiations in a database or satisfying product configuration constraints).

The preferences of a user, on whose behalf decisions are made, are captured by a *utility function* $u : \mathbf{X} \mapsto \mathbb{R}$. While a (complete) qualitative preference ordering would suffice to determine the best product, there are several reasons to quantify strength of preference using a utility function in product recommendation. First, strength of preference is important in determining an appropriate tradeoff vs. product price. Second, we will often want to make decisions with incomplete preference information, and require some measure of preference strength to quantify the degree to which a recommendation is suboptimal (we elaborate on this in Sec. 3). A quantitative utility function can be viewed as a representation of (qualitative) preferences over *lotteries* (distributions over outcomes), with one lottery preferred to another if and only if its expected utility is greater [17]. Let $\langle p, \mathbf{x}^\top; 1-p, \mathbf{x}^\bot \rangle$ denote the lottery where the best outcome $\mathbf{x}^\top$ is realized with probability p, and the worst outcome $\mathbf{x}^\bot$ with probability $1-p$. If a user is indifferent between some outcome $\mathbf{x}$ and the *standard lottery* $\langle p, \mathbf{x}^\top; 1-p, \mathbf{x}^\bot \rangle$, then $u(\mathbf{x}) = p$, assuming a normalized $[0, 1]$ scale where $u(\mathbf{x}^\top) = 1$ and $u(\mathbf{x}^\bot) = 0$.

2.2 Additive utilities

Since the size of outcome space is exponential in the number of attributes, specifying the utility of each outcome is infeasible in most settings. User preferences, however, often exhibit internal structure that allows both concise expression and effective elicitation of u. *Additive independence* [17] is commonly assumed in practice, where u can be written as a sum of single-attribute *subutility functions*:[1]

$$u(\mathbf{x}) = \sum_{i=1}^{N} u_i(x_i) = \sum_{i=1}^{N} \lambda_i v_i(x_i).$$

In our rental domain, utility of an apartment $\mathbf{x}$ might be:

$$u(\mathbf{x}) = u_b(\mathbf{x}[\textit{Bdrm}]) + u_d(\mathbf{x}[Dist]) + u_a(\mathbf{x}[\textit{AT}]) + \ldots$$

where u_b is the subutility function for number of bedrooms and $\mathbf{x}[\textit{Bdrm}]$ denotes the value of variable *Bdrm* in $\mathbf{x}$, etc.

The subutility functions $u_i(x_i) = \lambda_i v_i(x_i)$ can be defined as a product of *local value functions (LVFs)* v_i and scaling constants λ_i. This simple factorization allows us to separate the representation of preferences into two components: *local* and *global*. Significantly, LVFs can be defined using local *lotteries* that involve only a single attribute: $v_i(x_i) = p$, where p is the probability at which the user is indifferent between x_i and a local lottery $\langle p, x_i^\top; 1-p, x_i^\bot \rangle$, *ceteris paribus*.[2] Since we can define LVFs independently of other attributes, we can also *assess* them using queries only about values of attribute i. In our example, we can assess relative strength of preferences for different values of *Bdrm*, calibrating these to the best and worst settings of *Bdrm without reference to other attributes*. This has great practical significance, because people have difficulty accounting for more than five or six attributes at a time [14].

The scaling constants λ_i are required to properly calibrate LVFs across attributes. We define these *global* parameters by first introducing the notion of a *reference outcome*, denoted $\mathbf{x}^0 = (x_1^0, \ldots, x_N^0)$. The reference outcome is an arbitrary outcome, though it is common in practice to choose the worst outcome $\mathbf{x}^\bot$ as $\mathbf{x}^0$ (however, any salient outcome, e.g., the user's current apartment, can be used). Let $\mathbf{x}^{\top_i}$ be a full outcome where the i^{th} attribute is set to its best level and other attributes are fixed at their reference levels; $\mathbf{x}^{\bot_i}$ is defined similarly. Then, $\lambda_i = u(\mathbf{x}^{\top_i}) - u(\mathbf{x}^{\bot_i})$. To assess scaling constants λ_i, one must ask queries about $2N$ global outcomes $\mathbf{x}^{\top_i}$ and $\mathbf{x}^{\bot_i}$ for each attribute i. These global outcomes are special because they involve varying only a single feature from the reference outcome. This ease of assessment makes additive utility the model of choice in most practical applications.

Price plays a critical role in product choice, and is typically distinguished as an attribute. We make the standard assumption of *quasilinear utility* in which, overloading u, the utility $u(\mathbf{x}, p)$ of an outcome $\mathbf{x}$ obtained at price p is $u(\mathbf{x}, p) = \alpha u(\mathbf{x}) - p$. Here $u(\mathbf{x})$ is the (price-independent) utility of $\mathbf{x}$ and α is a valuation factor that adjusts u for currency. In quasilinear settings, one can elicit strength of preference directly in terms of "willingness to pay."

2.3 Generalized additive independence

Additive models predominate in practical decision support systems. However, their strong independence assumptions limit their

[1]This decomposition is possible if and only if a user is indifferent between lotteries with the same marginals on each attribute.

[2]$x_i^\top$ and $x_i^\bot$ are the best and worst levels of attribute i. Without loss of generality, we assume $v_i(x_i^\top) = 1$, $v_i(x_i^\bot) = 0$ [17].

applicability. More flexible *generalized additive independence (GAI)* models [12, 1] allow interactions among attributes while maintaining an additive decomposition over (usually small) subsets of attributes, or *factors*. GAI models are fully general and subsume additive models. Suppose a user's strength of preference for *Dist* (distance to university) depends on distance to the nearest subway station *Sub* (e.g., the closer the nearest station, the less value she places on being within walking distance to the university) and on *Bdrm* (an extra bedroom can used as an office, reducing the importance of distance). An additive model cannot accurately reflect such preferences, while a GAI model can:

$$u(\mathbf{x}) = u_1(\mathbf{x}[\textit{Bdrm}, \textit{Dist}]) + u_2(\mathbf{x}[\textit{Dist}, \textit{Sub}]) + u_3(\mathbf{x}[\textit{AT}]) + \dots \quad (1)$$

Note that GAI models allow overlap among the factors.

Formally, we assume M attribute subsets, or *factors* $F_1, \cdots, F_M$, such that $\cup_i F_i = \{X_1, X_2, \dots, X_N\}$. Utility function u is *generalized additively independent* (with respect to the given factor decomposition) if the user is indifferent between any two lotteries whose marginals over these factors are the same. If this GAI condition holds, u can be written as a sum of *subutility functions* on factor outcomes [12] (we use $\mathbf{x}[i]$ to abbreviate $\mathbf{x}[F_i]$):

$$u(\mathbf{x}) = \sum_{i \leq M} u_i(\mathbf{x}[i]). \quad (2)$$

Elicitation with GAI models is generally more complicated than with additive models. Specifically, if factors overlap there are infinitely many valid decompositions of the same utility function in which the subutility functions vary considerably (i.e., not simply through some positive affine transformation): indeed the apparent "local preferences" for factor instantiations can be reversed in two different (valid) representations (we refer to [12, 13, 7] for details). This difficulty has been traditionally dealt with by eliciting GAI utilities using full outcomes [12, 13] with special structure.

In [7], a technique was developed that allows sound elicitation of GAI models using local queries analogous to those used in additive models. A key observation is that the judicious choice of a *conditioning set* shields the influence of other attribute values on local preference over factor instantiations (analogously to a Markov blanket in a probabilistic graphical model). We do not define conditioning sets formally here (see [7]), but in the example partial GAI model in Eq. 1, the conditioning set for factor (*Bdrm*, *Dist*) would consist of a single variable *Sub*. Elicitation then proceeds as follows: we assume a fixed reference outcome $\mathbf{x}^0$; as above, this can be any salient outcome, such as the user's current apartment. Local queries that elicit (relative) strength of preference for instantiations of a specific factor must be conditioned by setting all attributes in that factor's conditioning set to their reference value. In our example, any queries assessing the (local) utility of instantiations of (*Bdrm*, *Dist*) would require the user to assume that *Sub* is set to a fixed reference value (e.g., assume a subway is 500m away). Global calibration across utility factors is accomplished in the same way as in additive models.

In our work, we use a semantically sound parameterization of the GAI function [7, 8] that is well suited to incorporate both local and global information about user preferences:

$$u(\mathbf{x}) = \sum_{i=1}^{M} \sum_{k=1}^{|Dom(F_i)|} C^k_{\mathbf{x}[i]}\ \theta^k_i, \quad (3)$$

where the θ^k_i are the *GAI utility parameters* to be assessed, and the $C^k_{\mathbf{x}[i]}$ are precomputed *structure coefficients* that depend solely on the factor decomposition and choice of reference outcome.

3. RECOMMENDATION USING MINIMAX REGRET

Much work in decision analysis on utility elicitation is focused on eliciting a fairly complete picture of a user's preferences. However, some work in AI [25, 5, 4], decision analysis [22] and conjoint analysis [23] has placed recent emphasis on good decisions with minimal preference information. In this paper, we focus on the use of the *minimax regret* decision criterion for making robust decisions in the presence of incomplete utility information, and as a means of determining suitable user queries to refine our model of user preferences in *relevant* regions of utility space [4, 8]. We first discuss regret-based recommendation, then regret-based elicitation.

3.1 Minimax regret

Assume outcome set $\mathbf{X}_D \subseteq \mathbf{X}$ from which an optimal outcome $\mathbf{x}^*$ must be chosen for a user with utility function u. Suppose further that u is unknown, but that through some process we have determined that it lies in some set $\mathbf{U}$ (e.g., as discussed below, $\mathbf{U}$ may arise through incomplete elicitation of utility parameters) and that a decision must be made knowing only $u \in \mathbf{U}$. Among possible decision criteria in the presence of such utility function uncertainty, *minimax regret* has an especially intuitive appeal.

Definition 1 *Given feasible utility set* $\mathbf{U}$*, define the* pairwise max regret $MR(\mathbf{x}, \mathbf{y}, \mathbf{U})$ *of* $\mathbf{x}, \mathbf{y} \in \mathbf{X}_D$*; the* max regret $MR(\mathbf{x}, \mathbf{U})$ *of* $\mathbf{x} \in \mathbf{X}_D$*; the* minimax regret $MMR(\mathbf{U})$ *of* $\mathbf{U}$*; and the* minimax optimal outcome $\mathbf{x}^*_{\mathbf{U}}$ *as follows:*

$$MR(\mathbf{x}, \mathbf{y}, \mathbf{U}) = \max_{u \in \mathbf{U}} u(\mathbf{y}) - u(\mathbf{x}), \quad (4)$$

$$MR(\mathbf{x}, \mathbf{U}) = \max_{\mathbf{y} \in \mathbf{X}_D} MR(\mathbf{x}, \mathbf{y}; \mathbf{U}), \quad (5)$$

$$MMR(\mathbf{U}) = \min_{\mathbf{x} \in \mathbf{X}_D} MR(\mathbf{x}, \mathbf{U}), \quad (6)$$

$$\mathbf{x}^*_{\mathbf{U}} = \arg\min_{\mathbf{x} \in \mathbf{X}_D} MR(\mathbf{x}, \mathbf{U}). \quad (7)$$

Intuitively, $MR(\mathbf{x}, \mathbf{U})$ is the worst-case loss associated with recommending $\mathbf{x}$, obtained by assuming an adversary will choose the user's utility function u from $\mathbf{U}$ to maximize the difference in utility between the optimal outcome (under u) and $\mathbf{x}$. The minimax optimal configuration $\mathbf{x}^*_{\mathbf{U}}$ minimizes this potential loss. $MR(\mathbf{x}, \mathbf{U})$ bounds the loss associated with $\mathbf{x}$, and is zero iff $\mathbf{x}$ is optimal for all $u \in \mathbf{U}$. Any choice that is not minimax optimal has strictly greater loss than $\mathbf{x}^*_{\mathbf{U}}$ for some $u \in \mathbf{U}$.

Minimax regret is attractive for several reasons. First, it offers robustness with respect to the worst-case loss under any realization of a user's utility function (and provides the tightest bound on such loss).[3] Second, it requires no probabilistic prior, unlike Bayesian methods [11, 3]. Finally, computation of minimax regret is often quite practical, typically much more computationally efficient than probabilistic models. Most natural queries, including those used by the UTPREF system, impose linear constraints on the parameters of a user's utility function, giving rise to a polytope $\mathbf{U}$ over which linear optimization techniques can be applied. In configuration problems, optimization over product space $\mathbf{X}$ is often formulated as a constraint satisfaction problem (CSP) or a mixed integer program (MIP). In such domains, minimax regret computation can be formulated as an MIP, and solved practically for large problems using techniques such as Bender's decomposition and constraint genera-

[3]For a discussion of other criteria typically used in robust optimization, such as maximin [2], and comparison with (and further motivation for) minimax regret with uncertain utility, see [4].

tion [4, 5, 8]. In product databases, search techniques can be used, as we elaborate in the next section.

3.2 Multiattribute product databases

In [4, 8] the authors address the computational issues of efficiently computing minimax regret in GAI utility models when the feasible outcome space $\mathbf{X}_D$ is defined by a set of configuration constraints determining feasible instantiations of domain variables (e.g., product constraints). In this paper, our focus is on domains where the set $\mathbf{X}_D$ is given by a database of multiattribute items, i.e., where feasible configurations are enumerated (e.g., an online product catalog or product database as in our apartment search task). In these settings, the MIP models used for minimax optimization in [4, 8] cannot be applied directly. We briefly describe the search method used in the UTPREF minimax regret optimization in such product databases.

For any two items $\mathbf{x}$ and $\mathbf{y}$, pairwise regret $MR(\mathbf{x}, \mathbf{y}, \mathbf{U})$ can be computed using Eq. 4. The maximum regret $MR(\mathbf{x}, \mathbf{U})$ of $\mathbf{x}$ is determined by considering the pairwise max regret of $\mathbf{x}$ with every other item in the database. To determine the optimal outcome (i.e., that with minimax regret), we compute the max regret of each item in the database, and choose that with the least max regret. However, this approach requires pairwise regret optimization for each pair of items in the database. (We discuss this optimization shortly.)

Instead, we can view minimax regret computation as a game between two players, whose moves (choices) are the $D = |\mathbf{X}_D|$ items in the database; the value of a choice $\mathbf{x}^i$ by the MIN player (recommender system) and choice $\mathbf{x}^j$ by the MAX player (adversary) is $MR(\mathbf{x}^i, \mathbf{x}^j, \mathbf{U})$, the pairwise max regret of choosing item $\mathbf{x}^i$ rather than item $\mathbf{x}^j$. The goal of the MIN player is to choose an item that minimizes the maximum possible value across all MAX choices. The game can be represented either by the $D \times D$ matrix whose i, jth entry is $MR(\mathbf{x}^i, \mathbf{x}^j, \mathbf{U})$, or by the 2-ply *minimax search tree* with the MIN root node, D MAX nodes, and pairwise regret values at the leaves. To compute minimax regret, we find the values of MAX nodes (maximum value of the leaves in a tree or in each row of the matrix), and then choose the MAX node (row) with the smallest value. A complete minimax search takes D^2 steps; however, such complete search is generally not necessary. We can employ known pruning techniques to reduce the number of costly pairwise regret evaluations. In UTPREF, we use beta or alpha-beta pruning on the search tree [18], following the heuristic of favoring the expansion of nodes that correspond to the adversary's best moves. In practice, this allows us to achieve linear (rather than quadratic) performance.

Apart from the size of the search tree, node/score evaluation is critical and requires a pairwise max regret computation at each evaluated leaf node. When user utilities are expressed by a GAI utility function, the space of feasible utilities $\mathbf{U}$ is defined by a set of linear constraints—those dictated by user responses to the queries discussed in the next section—on the utility parameters θ_i^k from Eq. 3. By substituting Eq. 3, pairwise regret can be written as:

$$MR(\mathbf{x}, \mathbf{y}, \mathbf{U}) = \max_{\{\theta_i^k\}} \sum_{i=1}^{M} \sum_{k=1}^{|Dom(F_i)|} (C^k_{\mathbf{y}[i]} - C^k_{\mathbf{x}[i]})\, \theta_i^k, \qquad (8)$$

subject to $\mathbf{U}$ constraints. For many realistic problems, this optimization can be very quickly solved as a linear program with a number of variables equal to the number of GAI model parameters. In the apartment database discussed below, average LP computation time is roughly 0.5ms. and computing the minimax optimal apartment takes roughly 0.2s., far more than fast enough to allow real-time interaction.

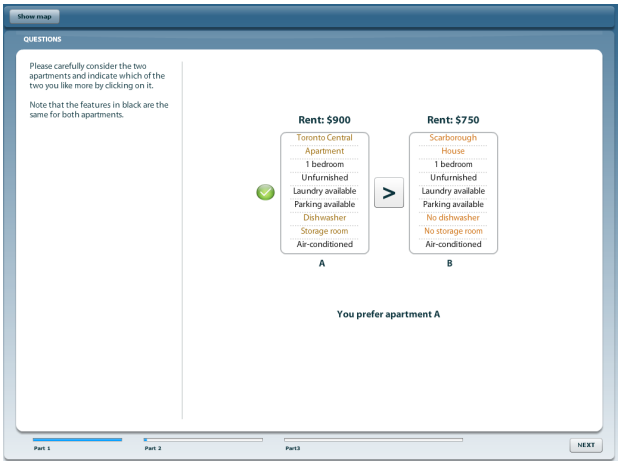


Figure 1: Screenshot from a UTPREF session

3.3 Regret-based elicitation

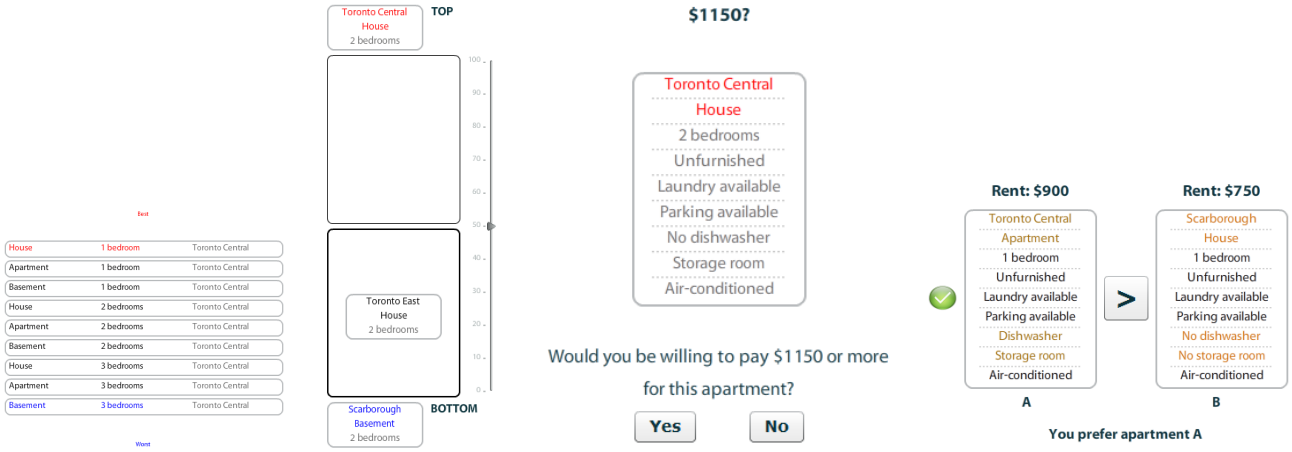
Minimax regret is also an effective means of driving elicitation. Given query set Q, in principle, at each step of elicitation, the system can evaluate each query in Q using a *scoring function* that estimates its potential to reduce minimax regret. A user response imposes constraints on the feasible utility space $\mathbf{U}$, leading to a new decision situation with a reduced minimax regret (minimax regret cannot increase with more information). The process continues until minimax regret reaches some acceptable level, elicitation costs (e.g., the number of queries) become too high, or some other termination criterion is met.

Since computing minimax regret for each response to each query in Q may not be practical in general, query strategies can also rely on heuristic criteria for choosing queries. One approach that has proven effective in simulation is to concentrate on queries that have the potential to constrain utility parameters that are directly involved in the *current solution* of the minimax regret optimization [4, 5, 8]. The current solution is the triple $\langle \mathbf{x}^*, \mathbf{x}^w, \dot{u} \rangle$ consisting of: i) the current regret-minimizing product $\mathbf{x}^*$; ii) the adversarial choice of product, or *witness* $\mathbf{x}^w$ that maximizes the regret of $\mathbf{x}^*$; and iii) the utility function $\dot{u}$ chosen by the adversary. Therefore, $MR(\mathbf{x}^*, \mathbf{x}^w, \dot{u}) = \dot{u}(\mathbf{x}^w) - \dot{u}(\mathbf{x}^*) = MMR(\mathbf{U})$.

Imposing constraints on utility parameters directly involved in the current solution has the greatest potential change in minimax regret, specifically pairwise max regret of the recommended product and the adversarial witness, either by increasing $\dot{u}(\mathbf{x}^*)$ or reducing $\dot{u}(\mathbf{x}^w)$. (More precisely, if a query response does not impact our assessment of one of these parameters, the minimax regret cannot be reduced.) This is the main intuition behind current solution based strategies. We refer to [8] for a more detailed explanation on how queries are rated by a heuristic scoring function that indirectly measures each query's potential to reduce minimax regret by considering the impact on the current solution.

4. UTPREF RECOMMENDATION SYSTEM

The UTPREF recommendation system is a software tool that explicitly models user preferences with a GAI utility function, incrementally acquires preference information through a sequence of queries and responses, and recommends a minimax regret-optimal option to the user.



(a) Local sorting (b) LBQ (c) ABQ (d) GCPQ

Figure 2: Four different query types

4.1 Implementation

UTPREF consists of a back-end recommendation engine, and a graphical user interface. The recommendation engine is written in Python and uses ILOG CPLEX for the solution of linear programs needed to compute pairwise regret. The graphical user interface is written in Adobe Flex and runs inside the Adobe Flash Player either as a stand-alone cross-platform application or a web browser plugin (see Fig. 1 for a sample screenshot of the UTPREF session).

A recommendation session proceeds in two stages. In the preliminary stage A, the system assesses basic user preference information. First, the user is asked to configure a *reference outcome* using drop-down lists of values for each attribute. For example, in the apartment domain, it could be the apartment in which the user currently resides or aspires to: any salient choice will suffice. Then, for each GAI factor, the user is shown a list of the factor's (local) outcomes, together with the conditioning set (if nonempty) with its attributes set to reference values, and is asked to *sort* that list in terms of decreasing preference via a drag-and-drop interface (see Fig. 2(a)). This sorted list provides both the best and the worst factor outcomes (which are used later in local bound queries), as well as a complete ordering of the (local) factor outcomes.

The preliminary preference information acquired in stage A is rarely enough to recommend a good decision. In stage B, the system asks the user a series of queries, until minimax regret drops to an acceptable level. UTPREF employs five different types of queries, described below, and can be configured to implement a variety of elicitation strategies that differ in which query types are used, and what criteria are used to choose a query. At each step, the user can opt to see the recommended option and its max regret level, and then decide whether to continue with further elicitation, or accept the system's recommendation.

4.2 Query types

UTPREF employs several types of GAI queries that can be classified as either *comparison* or *bound* queries; queries are further distinguished by the type of outcomes involved (*local* or *global*). All query types described below have a precise decision-theoretic semantics dictated by the theory of generalized additive independence, and responses to any query impose linear constraints on the GAI model parameters. An equally important characteristic of these query types is their user-friendliness and usefulness in practical applications, as corroborated by the user study results reported in Section 6.

4.2.1 Local queries

Local queries involve only a small subset of all attributes, namely attributes in some GAI factor and the attributes in the factor's conditioning set; the values of remaining attributes do not matter. We use two types of local queries: comparison and sorting queries and bound queries.

A *comparison query* asks the user to compare two outcomes and select the more preferred one. In a *local comparison query* (LCQ), both outcomes are local outcomes and belong to the same factor. In addition, the user is asked to assume that the attributes in the factor's conditioning set are fixed at reference levels. A response to an LCQ induces in a simple inequality constraint between two GAI utility function parameters.

Sorting a list of factor outcomes determines all pairwise relationships between factor outcomes (i.e., it is similar to asking a large set of pairwise LCQs). Fig. 2(a) shows an example of the sorting interface for a factor over *AT* and *Bdrm* attributes; notice that the *Area* attribute, which is the local conditioning set, is fixed at the same reference value for all outcomes (users are alerted to this fact elsewhere on the screen).

A *bound query* asks the user to consider a single outcome, and decide whether its value is greater or less than some specified bound b. In a *local bound query* (LBQ), the outcome is a local factor outcome $\mathbf{x}[i]$ whose value v is between 0 (the value of the worst factor outcome $\mathbf{x}[i]^{\perp}$) and 1 (the value of the best $\mathbf{x}[i]^{\top}$). According to the decision-theoretic semantics, v is a probability at which the user would be indifferent between obtaining outcome $\mathbf{x}[i]$ for sure and a lottery that results in $\mathbf{x}[i]^{\top}$ with probability v, and $\mathbf{x}[i]^{\perp}$ with probability $1 - v$. In UTPREF, we approximate the probabilistic semantics by asking the user to simply consider outcome $\mathbf{x}[i]$ on a scale from 0 to 100, and specify if its value v is greater or less than the bound b. As in other local queries, the attributes in the conditioning set are fixed at reference values. In the screenshot shown in Fig. 2(b), the user has indicated that the value of the *Toronto East, House* outcome is somewhere between 0 and 50 by dragging the outcome in question to the lower bin. The user can also adjust the query "boundary" (in this case 50) by adjusting the slider to provide tighter or looser constraints if they feel comfortable doing

so. A response to an LBQ imposes linear bound constraints that tie together three different GAI utility parameters.

4.2.2 Global queries

Global queries ask a user to consider preferences over full outcomes. UTPREF uses three types of global queries: anchor bound, anchor comparison and global comparison queries.

Anchor bound queries (ABQs) involve factor *anchors*, i.e., the best and worst factor outcomes with all attributes outside the factor (not just the conditioning set) fixed at their reference levels. The user only has to specify whether the anchor utility is greater than the specified bound. In UTPREF, we use a monetary scale to calibrate global outcome utilities. A response to an ABQ leads to a very straightforward bound constraint on the utility parameter $\theta_i^{\top}$ ($\theta_i^{\perp}$) (see Eq. 3), where $\theta_i^{\top}$ ($\theta_i^{\perp}$) is the utility of the top (bottom) anchor in factor F_i. In the screenshot in Fig. 2(c), the user is simply asked "Would you be willing to pay (at least) $1150 for the specified apartment?"

We also use two variations of *global comparison queries (GCQs)*. GCQ and *GCQ queries with price (GCPQs)* ask a user to consider two arbitrary global outcomes, ignoring price in the former case, and accounting for price in the latter. An example GCPQ is shown in Fig. 2(d). The user is simply asked to select the better option. A particular form of GCQ is the *anchor comparison query* (ACQ), in which *both* outcomes to be compared are either top or bottom anchors for some factor. ACQs are likely easier to understand because, unlike general GCQs, most attributes are fixed at reference levels, which are stable and salient. They also lead to linear constraints that involve only two utility parameters whereas responses to general GCQs or GCPQs tie together multiple utility parameters.

4.3 Elicitation strategies

UTPREF system can support a variety of query strategies by restricting the types of queries allowed, and employing different scoring methods. In the user study described below, we test two strategies that are very different from each other both in terms of query types and query scoring methods.

One strategy we investigate is the *all query strategy*. It mixes almost all types of queries (LBQ, ACQ, ABQ and GCQ), and at each step selects a query based on the heuristic scoring function described in [8]. GCQ queries are limited to comparing current solution outcomes (without price). On the other end of the spectrum lies the *GCPQ only strategy*. It relies exclusively on GCPQ queries that ask only about the current solution outcomes (the first one being the minimax optimal outcome, and the second being the adversarial witness). For this strategy, no query scoring function is needed.

5. USER STUDY DESIGN

We conducted a user study with the UTPREF system with three main objectives: 1) to assess overall user comprehension and acceptance of minimax regret-based elicitation; 2) to measure the costs, in terms of time and perceived difficulty, of different query types; and 3) to evaluate the effectiveness of the GAI utility representation as a model of user preferences, investigate the importance of context in local queries in GAI models, and compare different query strategies.

We recruited 40 participants from the University of Toronto who had either searched for rental housing in Toronto in the previous year, or were considering a new rental in the near future. The primary task of participants involved searching for apartments, using the UTPREF system, from a database of Toronto apartments. Each unit was described by nine attributes in addition to its price (monthly rent): area, building type, number of rooms, furnished or not, availability of laundry, parking, dishwasher, storage room, and central air conditioning. Rents ranged from $500 to $1800; area had four values (geographic regions); building type (house, apartment, basement) and number of rooms had three values; and the remaining attributes were binary. The database comprised 100 apartments sampled from a real Toronto housing database. The number of apartments was chosen to be large enough to justify the use of an intelligent search aid, but small enough for a user to evaluate each option in the second phase of the study, as discussed below. (Similar studies [24, 19] used databases of similar size.)

Each user session was divided in two parts. In Part 1, users answered preference queries posed by UTPREF which then recommended an apartment based on user responses. In Part 2, users explicitly rated all apartments in the database and ranked the top few options, allowing us to assess the quality of the recommended apartment relative to the user's explicitly stated preferences.

After a brief introduction and a five-minute demonstration, each user session was controlled by the UTPREF system itself, with minimal researcher oversight. Participants were guided by UTPREF via prompts, task descriptions, and queries. In the first stage of elicitation (Part 1A), participants were asked to configure a reference outcome, and provide an ordering of the local outcomes in each factor using the simple sorting interface (thus also specifying best and worst factor outcomes). In Part 1B, participants responded to a series of local and global queries about their preferences. After ten queries, the system displayed the minimax-optimal apartment and its max regret. The user could then stop the process, or continue answering further queries until minimax regret was reduced to a satisfactory level.

After a short break, participants completed Part 2 of the session, which involved answering a questionnaire about the experience, rating all the apartments in the database, and sorting a small list of apartments in terms of preference. The most demanding task in Part 2 involved rating all 100 apartments in the database (10 screens with 10 apartments in each) on a 0-2 scale, with score semantics: 0, *I do not like this apartment*; 1, *Not sure*; and 2, *I would rent this apartment*. After this rating process, users were presented with a *final list* of 7–12 apartments and asked to sort them in order of preference. The final list was formed by taking the union of, and randomly shuffling, the ten highest user-rated apartments from Part 2, and five apartments with least max regret as determined in Part 1 (because some apartments were in both sets, the final list had fewer than 15 elements). The final list always included the recommended outcome, but it was not distinguished in any way.[4] Finally, users specified an approximate *value difference* (in dollars/monthly rent) between the best and worst options in the final list: users were shown their best and worst options and asked how much the rent of the best option would have to increase to make the user roughly indifferent between the two options.

One main feature of UTPREF is the use of more flexible, but more complex, GAI utility functions to represent user preferences. We believe UTPREF is the first preference elicitation system to employ GAI models (rather than additive or unstructured models). To evaluate the benefits and limitations of using this more complex, but semantically sound, model, we tested UTPREF under three main modeling assumptions.

The first condition, *GAI*, uses a full GAI utility model, reflecting preferential dependence among attributes using GAI factors. Elicitation techniques and local queries are those mandated by the theory of GAI modeling [8]. Utility structure (i.e., GAI factors) was

[4]Because of the break induced by the questionnaire, participants were less likely to precisely recall the recommended apartment.

fixed for all participants in the study. The utility function had two intersecting factors: *area, building type* and *building type, number of bedrooms*; the remaining factors had single attributes.[5] The second condition, *GAI with no local context (GAI-nc)*, was designed to test the sensitivity of GAI elicitation to the use of sound conditioning sets. GAI-nc is identical to GAI, using the same GAI model, except that UTPREF does not display the conditioning context when asking local factor queries (e.g., in the local sorting task or when asking local bound queries). Our aim is to test whether good recommendations can be generated without the use of conditioning context (which, theoretically, is required). The third condition, *ADD* uses a simple additive utility model to represent user preferences. No modification of UTPREF is needed, since additive models are a simple, special case of GAI models which have single-attribute factors and empty conditioning sets.

We used a between-group experimental design, with three randomly selected groups of subjects from the 40 participants were assigned to the conditions GAI, GAI-nc and ADD. In addition, half of the users were asked only GCPQ current solution queries, while the other half interacted with the elicitation strategy that employed all other types of queries (ABQ, ACQ, GCQ and LBQ). This gave six subgroups:

	GAI	GAI no context	Additive
All queries	A	B	C
GCPQ only	D	E	F

All subgroups had 6 participants except subgroups B (7 participants) and C (9 participants).

6. USER STUDY RESULTS

We present the results of the user study in three parts: first we describe overall system performance with respect to recommendation effectiveness, ease of use and user satisfaction; second we discuss query costs (both time and perceived difficulty); and finally we consider performance differences under the different test conditions.

6.1 Overall evaluation

User impressions. Participants were asked to evaluate a variety of aspects of the UTPREF system after the elicitation process, but before manually rating all apartments. The following questions were rated on a 7-point Likert scale, with scores from 1 ("strongly disagree") to 7 ("strongly agree"); we show average and median responses (with std. dev. in parentheses):

Some questions were too hard	1.65	2	(0.65)
My answers reflected my true preferences	6.05	6.5	(1.26)
Some questions were confusing	1.98	2	(1.06)
I fully understood the meaning of each question	6.30	6	(0.71)
I answered some questions carelessly	1.98	2	(1.21)
I understood the task I was asked to do	6.75	7	(0.49)
I found this application easy to use	6.35	7	(0.79)
The task took too much time	2.23	2	(1.01)
I am satisfied with the recommended apartment	5.35	5	(1.19)

The average responses to the last four questions confirm that users understood the minimax-regret criterion and were quite satisfied with system performance. The majority of participants understood the task well, found UTPREF easy to use, and were satisfied with its recommendation. Note that satisfaction with the recommended apartment is not solely a function of recommendation quality, but is strongly influenced by the available units in the database. We examine recommendation quality next.

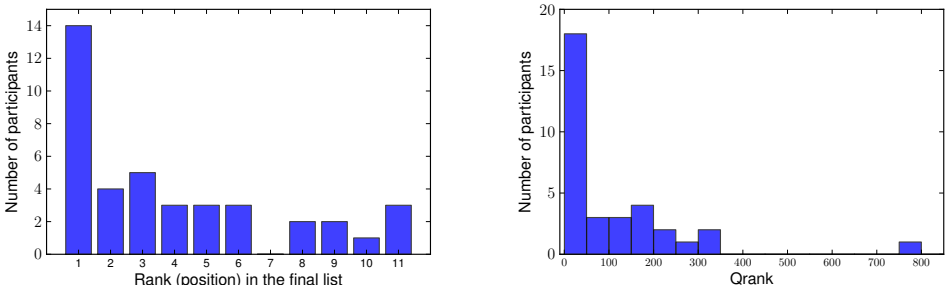
[5]Utility structure obviously depends on the specific user; however, eliciting factor structure was beyond the scope of the current study. The factors were chosen to reflect a reasonable consensus of possible attribute dependencies based on the domain and an informal survey of potential users.

Recommendation quality. Part 2 of the user session was designed to provide a *quantitative* measure of UTPREF's ability to recommend the "right" product. The major task in Part 2 involved rating all apartments in the database and sorting those rated highest (and those with low minimax regret according to UTPREF), thus providing valuable information about the accuracy of the recommended outcome. In the initial rating phase, of 40 participants, 34 assigned the highest rating (2) to the recommended apartment, five rated it 1, and one participant rated it lowest (0). This suggests that the recommended apartment is almost always a very desirable option.

When asked next to sort the final list of 7–12 "very promising" apartments (as described above), users ranked the UTPREF-recommended apartment very highly. The average position (rank) of the recommended apartment was 4.00 (std. dev. 3.32), and the histogram of rankings in Fig. 3(a) shows that 14 users placed the recommended outcome at the top of the sorted list, and majority ranked it among the top three (median 3.0). This too confirms the recommendation accuracy of UTPREF, and is arguably more informative than the coarse 3-point rating of the recommended item.

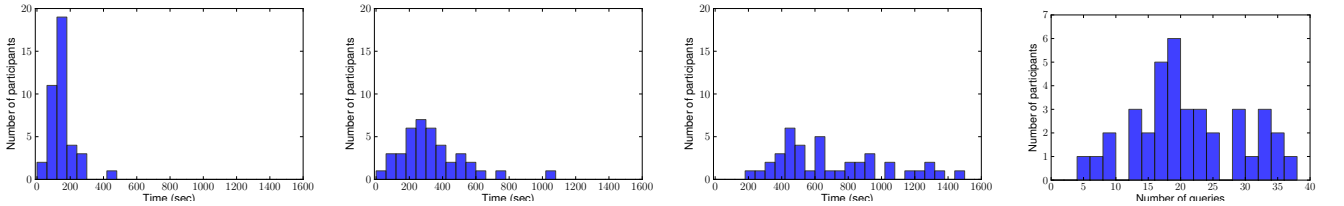
The position of the recommended apartment in the sorted list does not reveal how from optimal it is in terms of *value*. Asking participants to specify the value difference (in dollars) between the best and worst options in the sorted list lets us roughly quantify the difference between the best apartment in the database and UTPREF's recommendation using a *qrank* score, a quantitative analog of rank (the lower the qrank value, the better). The qrank of an outcome is computed by assuming that apartments in the sorted list are evenly distributed in terms of utility (willingness to pay). For example, if the difference between the best and the worst apartments in a sorted list of 11 apartments is $1000, then the qrank of the top outcome is $0, the worst outcome $1000, and the fifth apartment $400. Fig. 3(b) shows the histogram of the qrank of the recommended apartment across all participants. Only six users have a qrank greater (worse) than $200; equivalently, for 34 of 40 participants, the recommended apartment is within $200 (w.r.t. monthly rent) of the optimal apartment, under qrank assumptions. The average qrank is $100.74 (with std. dev. $151.48); and importantly, the median qrank is $44.95. Thus when the recommended apartment is not optimal it is generally quite close to optimal (within $45 for the majority of users).

Time. An important measure of user experience is the duration of the interaction (and the number of queries) required by the recommendation system to find a good product. Fig. 4(a-c) shows how long participants took to complete the UTPREF elicitation session (Parts 1A and 1B), and to rate all items in the database (Part 2). Of special significance is the fact that participants spent more time, on average, searching through all apartments in the database (708 seconds) than completing Parts 1A and 1B combined (145 seconds plus 336 seconds) of the elicitation process. This difference is statistically significant ($p = 0.00009$). This holds despite the fact that by the time participants rated the apartments, they had become quite familiar with domain attributes and range of prices, and had time to explore and verify their own preferences by completing the UTPREF elicitation session (Part 1). In addition, increasing the size of the database directly and proportionately impacts the time required to examine all apartments; in fact, we limited our database



(a) Rank (b) Qrank

Figure 3: Rank and qrank distribution across study participants



(a) Part 1A (b) Part 1B (c) Part 2 (d) Number of queries in Part 1B

Figure 4: Completion times for tasks and number of queries. Part 1A, choosing a reference outcome, and sorting all local outcomes for each factor, is the same for all participants; the duration of Part 1B (answering elicitation queries) depends on the number of queries needed to reduce regret to zero. Part 2, rating all 100 apartments in the database, is the same for all participants.

to 100 apartments because it would be too time-consuming and tedious for users to rate a larger number of options. However, simulation results [8] show that UTPREF can handle much larger databases without a significant increase in the number of queries.

Fig. 4(d) shows the histogram of the number of elicitation queries in Part 1B (which excludes choosing the reference outcome, and sorting local outcomes for each factor). Even though the participants had the option to cut the process short after 10 queries, all but two continued until the regret of the recommended apartment dropped to zero. The average number of queries until termination was 21.70 (std. dev. 8.84), and the median was 19.5, illustrating that the majority of users find a near-optimal apartment in under 20 queries. It is also important to note that most of these queries are not full apartment comparisons, but involve small subsets of attributes.

6.2 Query costs

One of the goals of the user study was to estimate the cognitive costs of different query types. We measured the durations and user ratings for five types of queries described previously: ACQ, GCQ, GCPQ, ABQ, and LBQ (in addition, local comparison queries were implicitly used in the local outcome sorting task, but were not rated by users). At the end of the elicitation session, participants were asked to rate the query types they encountered—"In terms of difficulty, how would you rate the **type** of question shown?"—on a 7-point Likert scale—ranging from 1 ("extremely difficult") to 7 ("extremely easy"). In addition, the amount of time participants spent responding to each query was recorded. The following table shows average rating and response duration for every query type:

Query type	Avg. rating	Avg. duration (sec)
ACQ	5.28	**13.95**
GCQ	**5.88**	15.58
GCPQ	5.18	14.65
ABQ	5.41	14.86
LBQ	4.23	21.27

A one-way ANOVA indicates that the query durations and ratings are significantly different (p=0.00047 and p=0.007, respectively). However there are no significant differences (ratings or duration) among *global* queries; only LBQs exhibit significant differences from every other query (durations differ in a multiple comparison procedure at the 0.05 level, and ratings differ from each other query in a pairwise comparison).

Both the rating and response time can be used as a proxy for the query's cognitive difficulty. We can see that all global queries have very similar average ratings (between "easy" and "very easy") and response times. Somewhat surprising is the lack of significant differentiation between global bound and comparison queries. In general, users found local bound queries to be the most difficult. From our observations and user comments it is clear that one reason was their novelty and occasional confusion about their meaning; better visual design and explanations might make the LBQs easier to answer. On the other hand, the average LBQ rating of 4.23 still does not suggest serious difficulty (4 corresponds to a query being "not difficult"). As far as we are aware, this is the first evaluation and comparison of query costs in the context of preference elicitation systems. This takes a step toward allowing query costs to be traded off against potential query value in future elicitation strategies.

6.3 Comparison of study subgroups

The participants were randomly assigned to one of six subgroups A-F (described above), divided along two axes: Query types—all queries vs. global comparison queries (GCPQ) only; and Utility function structure—GAI vs. GAI with no conditioning context (GAI-nc) vs. additive (ADD). Table 1 shows the average of the various measures discussed above across the six subgroups.

GAI with and without local context. To properly elicit GAI utilities with intersecting factors, local queries have to include certain context attributes. For example (see Fig. 2(b)), a local query

	ALL	A: all, GAI	B: all, GAI-nc	C: all, ADD	D: GCPQ, GAI	E: GCPQ, GAI-nc	F: GCPQ, ADD
Rank of recommended outcome	4.00	3.33	**3.29**	4.56	4.17	4.33	4.17
Qrank	$101	$106	**$27**	$82	$73	$81	$262
Number of queries	21.70	29.50	27.00	24.56	**13.83**	18.00	15.00
Part 1A duration (sec)	145	168	174	**94**	164	181	110
Part 1B duration (sec)	336	401	500	365	**191**	256	259
Database item rating duration (sec)	708	673	564	828	626	734	787

Table 1: Subgroup comparison

about a *house* in *Toronto East* (both building type and area are in Factor 1) also has to specify the reference value of the context attribute *2 bedrooms* (which is in Factor 2), because Factors 1 and 2 intersect. If the context attribute's reference value were different, say *1 bedroom*, user preference for a *house* in *Toronto East* could be very different. To test the importance of local context, local queries for users in subgroups B and E did not include local context (everything else was the same as in subgroups A and D; for subgroups C and F, this distinction is irrelevant, since additive utility functions do not have intersecting factors). The following table compares key performance metrics:

	All queries		GCPQ only	
	A: GAI	B: GAI-nc	D: GAI	E: GAI-nc
Rank	3.33	**3.29**	**4.17**	4.33
Qrank	$106	**$27**	**$73**	$81
Satisfaction	5.00	**5.86**	5.50	**5.67**

There are no obvious, statistically significant patterns that emerge from the comparison of the GAI and GAI-nc groups. Local queries without context attributes seem to perform no worse than proper, semantically sound queries with local context. There are two plausible explanations for this. First, we believe it is likely that users maintain a consistent context themselves (for instance, some salient outcome like their current or desired apartment) without an explicit specification of it, and automatically assume this context when answering local queries. Confirmation of this hypothesis requires further research. Second, the GAI model used in this study is quite simple, with only two overlapping factors. The importance of proper preferential conditioning generally increases with the complexity of the utility model; thus, differences might emerge in more complex, cognitively demanding choice scenarios.

GAI vs. additive utility models. To compare the effectiveness of more flexible GAI models with additive models, we group the GAI and GAI-nc subgroups and compare performance with that of the additive group. The table below compares average Rank, Qrank, and user satisfaction with the recommended apartment for both groups; we show results broken out by query type, and with both query groups combined (as well as p-values in a pairwise comparison):

	Combined			All queries			GCPQ only		
	GAI	ADD	p-	GAI	ADD	p-	GAI	ADD	p-
Subgrp	ABDE	CF	val	AB	C	val	DE	F	val
Rank	**3.76**	4.40	.57	**3.31**	4.56	.38	4.25	**4.17**	.97
Qrank	**$70**	$164	.10	**$63**	$82	.71	**$77**	$262	.07
Satisf.	**5.52**	5.07	.26	**5.46**	4.89	.21	**5.58**	5.33	.68

Although we cannot draw strong, statistically significant conclusions from these results due to the small sample size and high variability, it appears that a more complex GAI utility model leads to better performance w.r.t. recommendation quality than a simple additive utility model. The only statistically significant advantage is in Qrank (at the 0.1 level). The slight advantage of GAI models is maintained across both types of strategies as well as when both strategy results are combined together. We should bear in mind that the GAI utility factorization used in this study is quite simple, and not tailored to the preference structure of individual participants. In more complex domains, the advantage of a more flexible GAI model could be much more pronounced; but the differences in this study are suggestive enough to warrant further exploration.

All queries vs. GCPQ only queries. To explore variation along the query axis, we form the *all queries* group by merging subgroups A,B,C and the *GCPQ only* group by merging subgroups D,E,F. Key performance metrics are shown in the following table (the p-value column shows results from the pairwise comparisons of the first two columns):

	All queries	GCPQ only	p-value
Rank of recommended outcome	**3.82**	4.22	.71
Quantitative rank (qrank)	**$70**	$132	.25
Number of queries	26.68	**15.61**	.000018
Part 1B duration (sec)	418	**235**	.0024

Since global comparison queries concentrate on current solution outcomes, minimax regret tends to decrease very quickly, and on average, the process converges after only 15.61 queries. The all queries strategy uses more queries on average, in large part because it explores a broader utility region by combining both local and global, comparison and bound queries. On the other hand, we suspect that the all queries strategy has some advantage with respect to the final recommendation, whose quality is somewhat better in average rank and qrank (though the difference is not statistically significant): the flexibility of its query space allows it to home in on true utility more precisely.

7. CONCLUSIONS

Explicit user utility modeling with GAI utility functions, semantically sound and user-friendly queries, and minimax regret-based preference elicitation and recommendation are the main features distinguishing UTPREF from other recommendation systems. While very effective in simulated experiments, minimax regret-based elicitation has not been tested before in realistic domains with real users. Our results from the user study with 40 participants are very encouraging. We have demonstrated that minimax regret is an intuitive, comprehensible decision criterion that can be used to drive very effective querying strategies. It results in high-quality recommendations with minimal user preference revelation, and we have seen that GAI utility models perform better than simple additive models with respect to several recommendation quality measures. We have gauged the cognitive costs of different query types, and have shown that simple local queries that omit the local context information perform as well as semantically correct local queries.

There are many possible extensions to our current work. Currently we assume a fixed GAI utility structure for all users. Utility structure personalization is a natural, although non-trivial, next step

in the development of UTPREF (see [6] for some recent steps in this direction). A related issue is product feature elicitation since decision alternatives are not always described in terms of features the user is most comfortable reasoning about. The aim here is to determine the most natural features to describe a product or outcome to allow users to most comfortably assess their preferences. Incorporating cognitive query costs into elicitation strategies would allow us to investigate the tradeoffs between reducing elicitation effort on the part of the user and improving recommendation quality. Finally, incentive issues arising in regret-based elicitation when the elicitor has its own interests in which product is recommended (e.g., a seller eliciting a buyer's preferences in order to recommend a product), or when the elicitor is attempting to facilitate a transaction between multiple parties [16] are of great interest.

8. REFERENCES

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

[2] A. Ben-Tal and A. Nemirovski. Robust solutions of uncertain linear programs. *Operations Research Letters*, 25:1–13, 1999.

[3] C. Boutilier. A POMDP formulation of preference elicitation problems. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence (AAAI-02)*, pages 239–246, Edmonton, 2002.

[4] C. Boutilier, R. Patrascu, P. Poupart, and D. Schuurmans. Constraint-based optimization and utility elicitation using the minimax decision criterion. *Artifical Intelligence*, 170(8–9):686–713, 2006.

[5] C. Boutilier, T. Sandholm, and R. Shields. Eliciting bid taker non-price preferences in (combinatorial) auctions. In *Proceedings of the Nineteenth National Conference on Artificial Intelligence (AAAI-04)*, pages 204–211, San Jose, CA, 2004.

[6] R. Brafman and Y. Engel. Directional decomposition of multiattribute utility functions. In *Proceedings of the 1st Conference on Algorithmic Decision Theory (ADT-09)*, Venice, 2009.

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

[8] D. Braziunas and C. Boutilier. Minimax regret-based elicitation of generalized additive utilities. In *Proceedings of the Twenty-third Conference on Uncertainty in Artificial Intelligence (UAI-07)*, pages 25–32, Vancouver, 2007.

[9] J. S. Breese, D. Heckerman, and C. Kadie. Empirical analyis of predictive algorithms for collaborative filtering. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence (UAI-98)*, pages 43–52, Madison, WI, 1998.

[10] R. Burke. Interactive critiquing for catalog navigation in e-commerce. *Artif. Intell. Rev.*, 18(3-4):245–267, 2002.

[11] U. Chajewska, D. Koller, and R. Parr. Making rational decisions using adaptive utility elicitation. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI-00)*, pages 363–369, Austin, TX, 2000.

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

[13] C. Gonzales and P. Perny. GAI networks for utility elicitation. In *Proceedings of the Ninth International Conference on Principles of Knowledge Representation and Reasoning (KR2004)*, pages 224–234, Whistler, BC, 2004.

[14] P. E. Green and V. Srinivasan. Conjoint analysis in consumer research: Issues and outlook. *Journal of Consumer Research*, 5(2):103–123, September 1978.

[15] J. L. Herlocker, J. A. Konstan, L. G. Terveen, and J. T. Riedl. Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems*, 22(1):53, 2004.

[16] N. Hyafil and C. Boutilier. Mechanism design with partial revelation. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI-07)*, pages 1333–1340, Hyderabad, India, 2007.

[17] R. L. Keeney and H. Raiffa. *Decisions with Multiple Objectives: Preferences and Value Trade-offs*. Wiley, New York, 1976.

[18] J. Pearl. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley, Reading, MA, 1984.

[19] D. Portabella Clotet and M. Rajman. Explicit trade-off and prospective analysis in electronic catalogs. In *ECAI Workshop on Recommender Systems*, Riva del Garda, Italy, August 2006.

[20] J. Reilly, J. Zhang, L. McGinty, P. Pu, and B. Smyth. Evaluating compound critiquing recommenders: a real-user study. In *EC '07: Proceedings of the 8th ACM conference on Electronic commerce*, pages 114–123, New York, NY, USA, 2007. ACM.

[21] J. Rennie and N. Srebro. Fast maximum margin matrix factorization for collaborative prediction. In *Proceedings of the 22nd International Conference on Machine Learning (ICML)*, 2005.

[22] A. Salo and R. P. Hämäläinen. Preference ratios in multiattribute evaluation (PRIME)–elicitation and decision procedures under incomplete information. *IEEE Trans. on Systems, Man and Cybernetics*, 31(6):533–545, 2001.

[23] O. Toubia, J. Hauser, and D. Simester. Polyhedral methods for adaptive choice-based conjoint analysis. *Journal of Marketing Research*, 41:116–131, 2004.

[24] P. Viappiani, B. Faltings, and P. Pu. Preference-based search using example-critiquing with suggestions. *Journal of Artificial Intelligence Research (JAIR)*, 27:465–503, 2006.

[25] T. Wang and C. Boutilier. Incremental utility elicitation with the minimax regret decision criterion. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI-03)*, pages 309–316, Acapulco, 2003.

[26] 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.