Regret-based Optimal Recommendation Sets in Conversational Recommender Systems

Paolo Viappiani
Department of Computer Science
University of Toronto
Toronto, ON, Canada
paolo@cs.toronto.edu

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

ABSTRACT

Current conversational recommender systems are unable to offer guarantees on the quality of their recommendations due to a lack of principled user utility models. We develop an approach to recommender systems that incorporates an explicit utility model into the recommendation process in a decision-theoretically sound fashion. The system maintains explicit constraints on user utility based on preferences revealed by the user's actions. We investigate a new decision criterion, *setwise minimax regret (SMR)*, for constructing optimal recommendation sets: we develop algorithms for computing SMR, and prove that SMR determines choice sets for queries that are myopically optimal. This provides a natural basis for generating compound critiques in conversational recommender systems. Our simulation results suggest that this utility-theoretically sound approach to user modeling allows much more effective navigation of a product space than traditional approaches based on, for example, heuristic utility models and product similarity measures.

Categories and Subject Descriptors

I.2 [**Artificial Intelligence**]: Miscellaneous; H.5.2 [**User Interfaces**]: interaction styles

General Terms

Algorithms, Human Factors

Keywords

recommender systems, preference elicitation, critiquing, minimax regret

1. INTRODUCTION

Recommender systems can help users navigate product spaces and make decisions involving very large sets of alternatives. *Conversational* recommender systems rely on mixed-initiative interactions, with both the user and the system taking an active role in the navigation process. User feedback can be entered in many forms, for instance, as direct answers to queries about preferred products, or *critique* of the options displayed by the system (e.g., by choosing one of several options, or altering the features of an option).

Many conversational systems use some notion of the *top-k items* based on an estimate of a user's preferences: the k (estimated) highest rated items are presented either as the current recommendation, or offered to the user for critique [6, 22]. However, such systems often offer very little "choice" to the user, since these products are likely to be similar, and potentially far from optimal if preference estimates are crude. To overcome this, some recommender systems employ some form of *diversity* to select a product set for presentation [14, 17]. This has two advantages, due to its coverage of a more diverse range of preferences. First, it increases the odds of offering a desirable option when a user is ready to terminate the navigation process; and second, it admits greater value of information—by distinguishing more diverse preference types—during critiquing.

While diversity-oriented recommender systems are generally motivated by trying to overcome imprecision in preference assessment, most such systems either fail to maintain an explicit utility model, or do not incorporate this in their judgement of diversity. In contrast, following [15, 24, 4] we argue that explicit consideration of the system's beliefs about user preferences is vital in determining the most appropriate (diverse) recommendations. Specifically, in this work we develop recommendation techniques that exploit *preference-based diversity*: the set of recommended products will be highly preferred choices for a *wide range* of user preferences (consistent with preferences revealed by the user so far). When a user must make a final choice from the set of presented products, this ensures that a good choice is available, no matter what the user's actual preferences are (again, consistent with what has been revealed). And when the options are presented in the form of a query, such sets tend to carry very high value of information.

In this paper, we adopt the *minimax regret* decision criterion [3, 20] for product recommendation under utility uncertainty and adapt it to the case of *set recommendations*. Minimax regret has proven to be an efficient, robust decision criterion and an effective means of preference elicitation. But, to date, it has only been applied to singleton recommendations. We extend the criterion to set recommendations, and develop computational procedures for its implementation. We also investigate its properties w.r.t. elicitation, proving that minimax optimal recommendation sets provide us with myopically optimal set-based comparison queries; i.e., if we view the user's choice as the answer to the query "Which of these products do you prefer?" the minimax optimal recommendation set is the *set-based comparison query* that refines utility function uncertainty most (i.e., reduces minimax regret the most).

Our regret-based approach is particularly suited to *critiquing* systems. User-controlled exploration in critiquing systems does not offer guarantees of either sufficient or efficient exploration of the

RecSys'09, October 23–25, 2009, New York, New York, USA.

product space—a user may cycle through a set of similar products or converge on a far-from-optimal product [12]. Our regret-based recommendation approach provides just such guarantees on decision quality. Furthermore, the fact that the same set of alternatives comprises the optimal (final) recommendation and the (myopically) optimal set-comparison query means that we can leave termination decisions in the hands of the user without changing the interface, or relying distinct decision/querying phases. Our simulations demonstrate that regret-based critiquing can lead to much more efficient exploration of the product space and better recommendations compared to existing critiquing systems.

2. BACKGROUND

We begin by describing our basic model and briefly reviewing approaches to preference assessment based on critiquing and explicit preference elicitation.

2.1 Underlying Decision Problem

We assume a recommendation system is charged with the task of recommending an option to a user in some multiattribute space, for instance, the space the possible product configurations from some domain (e.g., computers, cars, apartment rental, etc.). Products are characterized by a finite set of attributes $\mathcal{X} = \{X_1, ...X_n\}$, each with finite domains $Dom(X_i)$. Let $\mathbf{X} \subseteq Dom(\mathcal{X})$ denote the set of *feasible configurations*. For instance, attributes may correspond to the features of various apartments, such as size, neighborhood, distance from public transportation, etc., with $\mathbf{X}$ defined either by constraints on attribute combinations (e.g., constraints on computer components that can be put together), or by an explicit database of feasible configurations (e.g., a rental database).

The user has a *utility function* $u : Dom(\mathcal{X}) \rightarrow \mathbf{R}$. In what follows we will assume either a *linear* or *additive* utility function depending on the nature of the attributes [10]. In both additive and linear models, u can be decomposed as follows:

$$u(\mathbf{x}) = \sum_i f_i(x_i) = \sum_i \lambda_i v_i(x_i)$$

where each *local* utility function f_i assigns a value to each element of $Dom(X_i)$. In classical utility elicitation, these values can be determined by assessing local value functions v_i over $Dom(X_i)$ that are normalized on the interval $[0, 1]$, and importance weights λ_i ($\sum_i \lambda_i = 1$) for each attribute [9, 10]. This sets $f_i(x_i) = \lambda_i v_i(x_i)$ and ensures that global utility is normalized on the interval $[0, 1]$. A simple additive model in the rental domain might be:

$$u(Apt) = f_1(Size) + f_2(Distance) + f_3(Nbrhd)$$

When $Dom(X_i)$ is drawn from some real-valued set, we often assume that v_i (hence f_i) is linear in X_i.[1]

Since a user's utility function is not generally known, we write $u(\mathbf{x}; w)$ to emphasize the dependence of u on user-specific parameters. In the additive case, the values $f_i(x_i)$ over $\cup_i\{Dom(X_i)\}$ serve as a sufficient parameterization of u (for linear attributes, a more succinct representation is possible). The optimal product for the user with utility parameters w is that $\mathbf{x} \in \mathbf{X}$ that maximizes $u(\mathbf{x}; w)$. Our goal is to recommend, or help the user find, an optimal product, or one whose utility is near optimal.

[1]Our presentation relies heavily on the additive assumption, though our approach is easily generalized to more general models such as GAI [9, 5]. The assumption of linearity is simply a convenience; nothing critical depends on it.

2.2 Critiquing-based Recommender Systems

Example critiquing is a common and effective technique for product recommendation [6, 16, 17, 24]. In critiquing systems, a user is presented with one or more product options and invited to *critique* the displayed options, suggesting ways in which they could be improved. Interactions based on critiquing are especially helpful for users who are not familiar with the items available or their characteristics: the process of critiquing assists users in both exploring the space of possible options, and understanding (or even constructing [21]) their own preferences.

A great variety of critiquing methods have been proposed, varying in several important dimensions. In many systems the user proceeds by *tweaking* the current example product (e.g., "I like this product, but find me something cheaper;" or "This restaurant is OK, but find something with French cuisine") to make it conform more accurately to her preferences. Such systems often work by moving to a product that is *similar* to the current product, but reflects the new tweak (user preference). Examples of such systems include FindMe [6] and dynamic critiquing [16]; incremental critiquing [17] extends this framework with a preference model consisting of the critiques picked earlier in the process.

Unlike the preference elicitation techniques discussed below, most systems do not maintain an explicit user utility model. Among those systems that do (e.g., a linear utility model is used in [18]), the model is updated heuristically in response to critiques, without adopting an explicit semantics for the critique.

Other systems focus instead on user-generated critiques (the user is free to suggest alterations of the current product) with an *explicit* preference model [22, 24]. In FlatFinder [24] product suggestions are produced based on an analysis of users' current preference model and their potential hidden preferences (a form of preference-based diversity), stimulating the process of preference-awareness.

In this paper we choose to focus on system-generated critiques, motivated by the fact that the existence of an explicit utility model with a sound semantics gives the system the opportunity to suggest the most appropriate and informative critiques.

2.3 Regret-based Preference Elicitation

Much work in AI, decision analysis and operations research has been devoted to effective elicitation of preferences [19, 3, 7, 2, 23]. Adaptive preference elicitation generally differs from classical utility assessment in the recognition is that good, even optimal, decisions can often be recommended with very sparse knowledge of a user's utility function [3]; and that the value of information associated with specific elicitation actions (e.g., queries)—in terms of its impact on decision quality—is often not worth the cost of obtaining it [7, 2]. This means we must often take decisions in the face of an incompletely specified utility function.

In this work, we adopt the notion of *minimax regret* [20] as our decision criterion for robust decision making under utility function uncertainty. Minimax regret has been advocated as a means for robust optimization in the presence of data uncertainty [11], and has more recently been used for decision making with utility function (or objective function) uncertainty in optimization [19, 3]. Assume that through some interaction with a user, and possibly using some prior knowledge, we determine that her utility function w lies in some set W. (The form of W will become clearer when we discuss elicitation below). Following [3] we define:

Definition 1 *Given a set of feasible utility functions W, define the* pairwise max regret $MR(\mathbf{x}, \mathbf{y}; W)$ *of* $\mathbf{x}, \mathbf{y} \in \mathbf{X}$*; the* max regret $MR(\mathbf{x}; W)$ *of* $\mathbf{x} \in \mathbf{X}$*; the* minimax regret $MMR(W)$ *of* W*; and*

the minimax optimal configuration $\mathbf{x}^*_W$ *as follows:*

$$MR(\mathbf{x}, \mathbf{y}; W) = \max_{w \in W} u(\mathbf{y}; w) - u(\mathbf{x}; w) \quad (1)$$

$$MR(\mathbf{x}; W) = \max_{\mathbf{y} \in \mathbf{X}} MR(\mathbf{x}, \mathbf{y}; W) \quad (2)$$

$$MMR(W) = \min_{\mathbf{x} \in \mathbf{X}} MR(\mathbf{x}, W) \quad (3)$$

$$\mathbf{x}^*_W = \arg\min_{\mathbf{x} \in \mathbf{X}} MR(\mathbf{x}, W) \quad (4)$$

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

Minimax regret relies on relatively simple prior information in the form of bounds or constraints on user preferences (rather than probabilistic priors); and exact computation is much more tractable (in contrast with probabilistic models of utility that generally require reasoning with densities that have no closed form [2, 7]). In configuration problems, optimization over product space $\mathbf{X}$ is often formulated as a CSP or mixed integer program (MIP). In such domains, minimax regret computation can be formulated as a MIP, and solved practically for large problems using techniques such as Bender's decomposition and constraint generation [3, 5].

Minimax regret has proven to be an effective tool in utility elicitation in a variety of domains. A decision support or recommender system can query (or otherwise interact with) a user, determining additional constraints on the utility set W until minimax regret reaches some acceptable level (possibly optimality), elicitation costs become too high, or some other termination criterion is met.

A natural meta-heuristic for elicitation is the *current solution strategy (CSS)*, first described in [3]. The idea is as follows: a solution to the problem $MMR(W)$ generates a minimax-optimal product $\mathbf{x}^*_W$, an adversarial product $\mathbf{x}^a$ and utility function w that maximizes the regret of $\mathbf{x}^*_W$. CSS generates a query (depending on the space of queries allowed) that involves utility parameters in either $\mathbf{x}^*_W$ or $\mathbf{x}^a$ (or both). This is based on the insight that should a query provide information about no parameter that impacts the utility of either $\mathbf{x}^*_W$ or $\mathbf{x}^a$, minimax regret cannot change. As one example, consider *comparison queries*, in which a user is asked to compare one product $\mathbf{x}$ to another $\mathbf{y}$. A response that $\mathbf{x}$ is preferred imposes a linear constraint on W: $\sum_i f_i(x_i) > \sum_i f_i(y_i)$. In systems in which comparison queries are used, the CSS proposes a comparison query between $\mathbf{x}^*_W$ and $\mathbf{x}^a$.

3. OPTIMAL RECOMMENDATION SETS

Apart from allowing a users to have some control over the navigation process, critiquing systems provide the user the ability to explore their preferences effectively by simultaneously comparing multiple products at once. We want to blend this advantage with the benefits of explicit elicitation methodologies by constructing *optimal recommendation sets*. Unlike standard comparison queries (where a user is asked whether one option is better than another), our set queries will present a *set of alternatives*. In contrast to traditional critiquing systems, we associate a precise semantics with the user's choice from such a set: it is assumed the chosen product has utility at least as great as the unselected elements of the set. This is used to then refine our model of user utility.

We maintain a set W of feasible utility functions, and at each step, present a set of recommendations $\mathbf{Z}$ (these will be *jointly* optimal in a sense defined below). The user selects a single product as most preferred from $\mathbf{Z}$, and W is refined to reflect this. The process repeats until the user is satisfied with her choice, or minimax regret reaches some target. This is appealing for several reasons. First, the current set recommendation $\mathbf{Z}$ is always optimal; i.e., it minimizes setwise max regret given our beliefs about user utility. Second, max regret is a well-defined progress metric that lets the user know the value of further exploration of product space. Finally, the user selection from $\mathbf{Z}$ is maximally informative (as defined below). This last point is critical: it means that the best "query" set and the best "recommendation" set are identical.

Set recommendations have been addressed previously: in work that maximizes the joint diversity of a set [14]; or that optimizes set recommendations using expected maximum utility of the options in the set w.r.t. a probabilistic prior [15, 4]. These latter models focus implicitly on *preference-based diversity*: the set of recommendations attempts to span the (probabilistic) range of possible user utility functions to the greatest extent possible. Our setwise regret model does the same thing without requiring a density over utility functions, and the concomitant data and computational costs. Our approach can thus be viewed a generating dynamic, compound critiques using preference-based diversity. Set-based queries are studied extensively in conjoint analysis. Such queries are usually used to identify aggregate consumer preferences. But in some work adaptive elicitation of individual preferences is considered, using an explicit utility model, and volumetric methods for generating a *query (or choice) set* [23].

3.1 Setwise Minimax Regret

Suppose we have a slate of k options to present to the user and want to quantify the possible loss associated with restricting the user's decision to options in that slate. Intuitively, the user may select any of the k options as being "optimal." An adversary wanting to maximize regret should do so assuming the any such choice is possible—unlike max regret, we allow the user to select *any* from a set of k options. Formally, we choose the set of k options first, but delay the specific choice from the slate until *after* the adversary has chosen a utility function w. The regret of a set is the difference between the utility of the best configuration under w and the utility of the best option w.r.t. w in the slate. (To keep notation to a minimum, we assume $\mathbf{Z}$ is restricted to suitable subsets of $\mathbf{X}$ (e.g., of cardinality k) without making this explicit.)

Definition 2 *Let W be a feasible utility set, $\mathbf{Z} \subseteq \mathbf{X}$. Define:*

$$SMR(\mathbf{Z}, W) = \max_{\mathbf{x}' \in \mathbf{X}} \max_{w \in W} \min_{\mathbf{x} \in \mathbf{Z}} u(\mathbf{x}'; w) - u(\mathbf{x}; w)$$

$$SMR\text{-}Adv(\mathbf{Z}, W) = \arg\max_{\mathbf{x}' \in \mathbf{X}} \max_{w \in W} \min_{\mathbf{x} \in \mathbf{Z}} u(\mathbf{x}'; w) - u(\mathbf{x}; w)$$

$$SMMR(W) = \min_{\mathbf{Z} \subseteq \mathbf{X}} \max_{\mathbf{x}' \in \mathbf{X}} \max_{w \in W} \min_{\mathbf{x} \in \mathbf{Z}} u(\mathbf{x}'; w) - u(\mathbf{x}; w)$$

$$\mathbf{Z}^*_W = \min_{\mathbf{Z} \subseteq \mathbf{X}} \max_{\mathbf{x}' \in \mathbf{X}} \max_{w \in W} \min_{\mathbf{x} \in \mathbf{Z}} u(\mathbf{x}'; w) - u(\mathbf{x}; w)$$

The *setwise max regret (SMR)* of a set $\mathbf{Z}$ of k options reflects the intuitions above, and $SMR\text{-}Adv(\mathbf{Z}, W)$ is the adversarial alternative that maximizes regret. *Setwise minimax regret* is SMR of the minimax optimal set $\mathbf{Z}^*_W$, i.e., the set that minimizes $SMR(\mathbf{Z}, W)$.

Setwise max regret has some intuitive properties. First, adding new items to a recommendation set cannot increase SMR:

Observation 1 $SMR(\mathbf{A} \cup \mathbf{B}, W) \leq SMR(\mathbf{A}, W)$.

Incorporating options that are known to be dominated given W does not change setwise max regret:

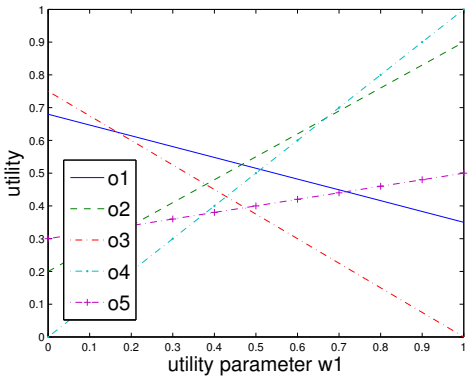

Figure 1: Utility as a function of weight w_1.

Observation 2 *If $u(\mathbf{a}, w) > u(\mathbf{b}, w)$ for some $\mathbf{a} \in \mathbf{Z}$ and all $w \in W$, then $SMR(\mathbf{Z} \cup \{\mathbf{b}\}, W) = SMR(\mathbf{Z}, W)$.*

Observation 3 *Setwise max regret can be rewritten:*

$$SMR(\mathbf{Z}, W) = \max_{\mathbf{y} \in \mathbf{X}} \max_{w \in W} [u(\mathbf{y}; w) - \max_{\mathbf{x} \in \mathbf{Z}} u(\mathbf{x}; w)] \quad (5)$$

This last observation captures the intuition that, given an utility function w, the option (among those in $\mathbf{Z}$) that determines setwise max regret is that with highest utility with respect to w. In fact, it is useful to explicitly partition utility space w.r.t. options in $\mathbf{Z}$; define:

$$W[\mathbf{Z} \rightarrow \mathbf{x}_i] = \{w \in W : u(\mathbf{x}_i; w) > u(\mathbf{x}_j; w)\ \forall j \neq i, 1 \leq j \leq k\}$$

(i.e., the region of w where $\mathbf{x}_i$ has greater utility than any other option in $\mathbf{Z}$). The regions $W[\mathbf{Z} \rightarrow \mathbf{x}_i]$, $\mathbf{x}_i \in \mathbf{Z}$, partition W (we ignore of ties over full-dimensional subsets of W, which are easily dealt with, but complicate the presentation). We call this the $\mathbf{Z}$*-partition of* W. An important observation we use later is that $SMR(\mathbf{Z}, W)$ can be determined using the maximum of the (individual) max-regrets over the $\mathbf{Z}$-partition:

Observation 4 *Let $\mathbf{Z} = \{\mathbf{x}_1, \ldots, \mathbf{x}_k\}$. Then*

$$SMR(\mathbf{Z}, W) = \max\{MR(\mathbf{x}_i, W[\mathbf{Z} \rightarrow \mathbf{x}_i]) : i \leq k\},$$

We illustrate this notion with a simple example, involving five options $o^1, \ldots, o^5$ with two features x_1 and x_2 as follows:

	o^1	o^2	o^3	o^4	o^5
x_1	0.35	0.9	0.	1	0.5
x_2	0.68	0.2	0.75	0	0.3

We assume linear user utility $u(\mathbf{x}; w_1) = w_1 x_1 + (1 - w_1)x_2$, with a single (unknown) parameter w_1 (we assume weight $w_2 = 1 - w_1$). Utility of each option, as a function of w_1 is shown in Fig. 1; we see each option except o^5 is optimal for some value of w_1. However, o^5 is minimax optimal: its max regret of 0.5—attained with adversarial choice o^4 at $w_1 = 1$—is less than that of any other option. This can be seen using a geometric interpretation of max regret: o^5's utility is a linear function of w_1 (in general, determining a hyperplane in $\mathbf{w}$-space); and its max regret is its maximum distance from the piecewise linear, convex function of w_1 determined by the upper surface in Fig. 1 (i.e., $\max_{o^i} u(o^i; w_1)$). This distance is less than that of any other option (e.g., the max regret of o_2 is 0.55, occurring at $w_1 = 0$ with adversarial choice o^3).

Sets of options have a similar graphical interpretation: instead of considering a single hyperplane, we consider the upper surface dictated by the recommendation set (i.e., the piecewise-linear, convex max) and its distance from the full upper surface. For instance, consider the recommendation of $k = 2$ options, $\{o_1, o_4\}$. By Obs. 4 we partition utility space where the two options have identical utility, $w_1 = 0.51$, and use o_1 for $w_1 < 0.51$ and o_4 for $w_1 > 0.51$. The setwise max regret is the max difference between the max of these two functions/options and the upper surface (i.e., 0.07 attained at $w_1 = 0$). This set is in fact setwise minimax optimal among pairs, though other pairs, e.g., $\{o_1, o_2\}$ have relatively low SMR. Note that any pair containing o_5, the best singleton recommendation, will have high (poor) SMR.

3.2 Optimal Myopic Elicitation

In critiquing systems, set recommendations are used not just to make a recommendations to the user, but also to give the user the chance to critique the proposed options and continue exploration of the product space. Our setwise minimax regret criterion can be used directly for this purpose, implementing a form of preference-based diversity. This stands in contrast to "product diversity" typically considered in critiquing systems. And unlike recent work in polyhedral conjoint analysis [23], which emphasizes volume reduction of the utility polytope W, our regret-based criterion is sensitive to the range of feasible products and does not reduce utility uncertainty for its own sake.

Any set $\mathbf{Z}$ can be interpreted as a query (or system-generated dynamic compound critique): we simply allow the user to state which of the k elements $\mathbf{x}_i \in \mathbf{Z}$ she prefers. We refer to $\mathbf{Z}$ interchangeably as a *query* or a *choice set*. The choice of some $\mathbf{x}_i \in \mathbf{Z}$ refines the set of feasible utility functions W by imposing the $k - 1$ linear constraints $u(\mathbf{x}_i; w) > u(\mathbf{x}_j; w), j \neq i$.

When treating $\mathbf{Z}$ as a choice set (as opposed to a recommendation set), we are not interested in its max regret, but rather in *how much a query response will reduce minimax regret*. In our distribution-free setting, the most appropriate measure is *myopic worst case regret (WR)*, a measure of the value of information of a query. Generalizing the pairwise measure of [3], we define:

Definition 3 *The* worst-case regret (WR) *of $\mathbf{Z} = \{\mathbf{x}_1, \ldots, \mathbf{x}_k\}$ is*

$$WR(\mathbf{Z}, W) = \max[MMR(W[\mathbf{Z} \rightarrow \mathbf{x}_1]), \ldots, MMR(W[\mathbf{Z} \rightarrow \mathbf{x}_k])]$$

An optimal choice set *$OptQuery(W)$ is any $\mathbf{Z}$ that minimizes worst case regret $MinWR(W)$:*

$$MinWR(W) = \min_{\mathbf{Z} \subseteq \mathbf{X}} WR(\mathbf{Z}, W)$$

Intuitively, each possible response $\mathbf{x}_i$ to the query $\mathbf{Z}$ gives rise to updated beliefs about the user's utility function. We use the worst-case response to measure the quality of the query (i.e., the response that leads to the updated W with greatest remaining minimax regret). The optimal query is that which minimizes this value.

It is not hard to show that the worst-case regret of choice set $\mathbf{Z}$ is never greater than its setwise max regret:

Observation 5 $WR(\mathbf{Z}, W) \leq SMR(\mathbf{Z}, W)$.

In the following we make use of a transformation that modifies a given recommendation set $\mathbf{Z}$ in such a way that SMR cannot increase, and usually decreases. This transformation will be used in two ways: to prove the optimality of SMR for choice set generation; and directly as a computationally viable heuristic strategy for choice set generation. Define *MMR-transformation* T to be a mapping that refines a recommendation set $\mathbf{Z}$ as follows: (a) we first

construct the $\mathbf{Z}$ partition of W; (b) we then compute the *single recommendation* that has minimax regret in each element (region of utility space) in the partition; and (c) let $T(\mathbf{Z})$ be the new recommendation set consisting of these new recommendations.

Definition 4 *Let* $\mathbf{Z} = \{\mathbf{x}_1, \ldots, \mathbf{x}_k\}$. *Define* MMR-transformation

$$T(\mathbf{Z}) = \{\mathbf{x}^*_{W[\mathbf{Z}\rightarrow\mathbf{x}_1]}, \ldots \mathbf{x}^*_{W[\mathbf{Z}\rightarrow\mathbf{x}_k]}\}$$

The optimality of minimax setwise recommendations, when used as queries, is based on the following lemma:

Lemma 1 $SMR(T(\mathbf{Z}), W) \leq WR(\mathbf{Z}, W)$

Proof Let $\mathbf{Z} = \{\mathbf{x}_1, \ldots, \mathbf{x}_k\}$ and $T(\mathbf{Z}) = \{\mathbf{x}'_1, \ldots, \mathbf{x}'_k\}$, where $\mathbf{x}'_i = \mathbf{x}^*_{W[\mathbf{Z}\rightarrow\mathbf{x}_i]}$. Define $W[\mathbf{Z} \rightarrow \mathbf{x}_i, T(\mathbf{Z}) \rightarrow \mathbf{x}'_j] = W[\mathbf{Z} \rightarrow \mathbf{x}_i] \cap W[T(\mathbf{Z}) \rightarrow \mathbf{x}_j]$ to be the subset of W where x_i is preferred in choice set $\mathbf{Z}$ and x_j in choice set $T(\mathbf{Z})$. The two expressions can be compactly represented as:

$$WR(\mathbf{Z}, W) = \max_{i,j}[MR(\mathbf{x}'_i, W[\mathbf{Z} \rightarrow \mathbf{x}_i, T(\mathbf{Z}) \rightarrow \mathbf{x}'_j])] \quad (6)$$

$$SMR(T(\mathbf{Z}), W) = \max_{i,j}[MR(\mathbf{x}'_j, W[\mathbf{Z} \rightarrow \mathbf{x}_i, T(\mathbf{Z}) \rightarrow \mathbf{x}'_j])] \quad (7)$$

We now compare the two expressions componentwise. Consider the utility space $W[\mathbf{Z} \rightarrow \mathbf{x}_i, T(\mathbf{Z}) \rightarrow \mathbf{x}'_j]$: if $i = j$ then the two MR components are the same. If $i \neq j$, consider any $w \in W[\mathbf{Z} \rightarrow \mathbf{x}_i, T(\mathbf{Z}) \rightarrow \mathbf{x}'_j]$. Since $w \in W[T(\mathbf{Z}) \rightarrow \mathbf{x}'_j]$, we must have $u(\mathbf{x}'_j; w) > u(\mathbf{x}'_i; w)$. Therefore $MR(\mathbf{x}'_j, W[\mathbf{Z} \rightarrow \mathbf{x}_i, T(\mathbf{Z}) \rightarrow \mathbf{x}'_j]) \leq MR(\mathbf{x}'_i, W[\mathbf{Z} \rightarrow \mathbf{x}_i, T(\mathbf{Z}) \rightarrow \mathbf{x}'_j])$. In the expression of $SMR(T(\mathbf{Z}))$ (Eq. 7), each element is no greater than its correspondent in the $WR(\mathbf{Z})$ expression (Eq. 6). Thus $SMR(T(\mathbf{Z}), W) \leq WR(\mathbf{Z}, W)$. ∎

From Lemma 1 and Observation 5, it follows that T cannot increase setwise max regret: $SMR(T(\mathbf{Z}), W) \leq SMR(\mathbf{Z}, W)$.

Theorem 1 *Let* $\mathbf{Z}^*_W$ *be a minimax optimal recommendation set. Then* $\mathbf{Z}^*_W$ *is an optimal choice set:* $WR(\mathbf{Z}^*_W, W) = MinWR(W)$.

Proof Suppose $\mathbf{Z}^*_W$ is not an optimal choice set, i.e., there is some $\mathbf{Z}'$ such that $WR(\mathbf{Z}', W) < WR(\mathbf{Z}^*_W, W)$. If we apply transformation T to $\mathbf{Z}'$ we obtain a set $T(\mathbf{Z}')$, and by the results above we have: $SMR(T(\mathbf{Z}', W)) \leq WR(\mathbf{Z}', W) < WR(\mathbf{Z}^*, W) \leq SMR(\mathbf{Z}^*_W, W)$. This contradicts the (setwise) minimax optimality of $\mathbf{Z}^*_W$. ∎

3.3 Alternative strategies

We now describe other set recommendation and query strategies that exploit minimax regret. Unlike the minimax optimal set, these strategies are heuristic and do not generally produce optimal sets. However, they are computationally less demanding than computing the SMR-optimal option. (We discuss algorithms for computing SMR in the next section.) We use the following notion: define an *adversarial choice* for W and $\mathbf{x}$ to be the witness product that maximizes the regret of $\mathbf{x}$: $Adv(\mathbf{x}, W) = \arg\max_{\mathbf{x}' \in \mathbf{X}} MR(\mathbf{x}, \mathbf{x}', W)$.

One simple strategy is the *current solution strategy (CSS)* for pairwise comparisons [3]. CSS asks a user to compare two products: the minimax optimal product $\mathbf{x}^*_W$ and its adversarial counterpart $Adv(\mathbf{x}^*_W, W)$. A pairwise comparison can be viewed as a choice set of size two (thus CSS is restricted to generating sets of size two). User selection of one of these products directly constrains the utility parameters that impact minimax regret.

The *chain of adversaries strategy (CAS)* generalizes CSS to produce a choice set with k options by repeatedly selecting an adversarial choice relative to the last option added to the set. More precisely, CAS produces choice set $\{\mathbf{x}^1, \ldots, \mathbf{x}^k\}$, where:

$$\begin{cases} \mathbf{x^1} = & \mathbf{x}^*_W \\ \mathbf{x^i} = & Adv(\mathbf{x^{i-1}}, W); \ \ 2 \leq i \leq k \end{cases}$$

CAS requires one minimax optimization to find $\mathbf{x}^1$ (which simultaneously determines witness $\mathbf{x}^2$). After that, only $k-2$ max regret computations are needed. If $k = 2$, CAS reduces to CSS.

The *setwise chain of adversaries strategy (SCAS)* selects k options by repeatedly maximizing setwise max regret given the current set. It can be viewed as a greedy, incremental approximation of the (setwise) minimax optimal set. As with setwise max regret, it explicitly maximizes diversity from the perspective of utility space. Formally, SCAS determines a set $\{\mathbf{x}^1, \ldots, \mathbf{x}^k\}$ where:

$$\begin{cases} \mathbf{x^1} = & \mathbf{x}^*_W \\ \mathbf{x^i} = & SMR\text{-}Adv(\{\mathbf{x^1}, .., \mathbf{x^{i-1}}\}, W) \ \ 2 \leq i \leq k \end{cases}$$

Recall that $SMR\text{-}Adv(\mathbf{Z}, W)$ is the adversarial option w.r.t. setwise max regret. SCAS can be seen as a generalization of CAS (and hence CSS) to sets of any size. Computing $SMR\text{-}Adv(\mathbf{Z}, W)$, which determines a single product at a time, is typically much faster than the joint optimization required by full computation of setwise minimax regret (see next section).

Finally, the MMR-transformation T introduced above gives rise to a natural heuristic search strategy for construction of good recommendation sets. Given an initial set $\mathbf{Z}$, we repeatedly apply T until a fixed point (w.r.t. setwise max regret) is found:

- **Repeat** $\mathbf{Z} := T(\mathbf{Z})$
- **Until** $SMR(T(\mathbf{Z}), W) = SMR(\mathbf{Z}, W)$

We initialize this algorithm *HCT (hillclimbing T)* with the slate $\mathbf{Z}$ given by CAS. Empirically this seems to produce the most promising sets.[2] The HCT algorithm can be terminated early.

4. COMPUTING SETWISE REGRET

We now discuss how to compute regret-based recommendations, distinguishing *configuration problems* and *database problems*.

Configuration problems In configuration problems, options are defined by a set of variables and configuration constraints (i.e., as solutions to a constraint satisfaction problem). In such domains, minimax regret computation for a *single* recommendation can be formulated as a MIP, and solved practically for large problems using techniques such as Bender's decomposition and constraint generation. We refer to [3, 5] for more details. Our MIP formulations for setwise minimax regret below draw heavily on these techniques, but require important modifications.

Setwise minimax regret for configuration problems can be formulated as the following MIP:

$$\begin{aligned} &\min \quad \delta \\ &\text{s.t. } M \geq \sum_{1 \leq j \leq k} R^j_w \quad \forall w \in W && (8) \\ &R^j_w \geq w(\mathbf{x}^*_w - \mathbf{X}^j) + (I^j_w - 1)M \ \ \forall j \leq k, w \in W && (9) \\ &\sum_{1 \leq j \leq k} I^j_w = 1 \quad \forall w \in W && (10) \\ &I^j_w \in \{0, 1\} \\ &V^j_w \geq 0 \quad \forall j \in [1, k], w \in W \end{aligned}$$

Intuitively, this MIP chooses k products $\mathbf{x}_j, j \leq k$, designated by variables $\mathbf{X}^j$ (where each $\mathbf{X}^j$ is a variable vector over the n product attributes). The objective minimizes δ subject to constraint (8),

[2] In the case no query can guarantee regret reduction for each response, i.e., $WR(\mathbf{Z}^*_W) = MMR(W)$, T degenerates to a singleton with only the minimax regret option. In such cases HCT returns the initial set produced by CAS.

ensuring δ is no less than the setwise regret of the selected options w.r.t. any $w \in W$ (i.e., no less than setwise max regret).[3]

Constraint (8) need not be expressed for each (of the continuously many) $w \in W$. Since setwise regret is maximized at some vertex of W, we can post constraints only for these vertices (i.e., $w \in Vert(W)$). (We also exploit the fact, in Constraint (9), that regret at any w is maximized by the adversary selecting the optimal product $\mathbf{x}^*_w$.) However, this MIP still requires (potentially) exponentially many constraints, one for each element of $Vert(W)$. We can make computation much more effective by applying constraint generation, observing that very few of these constraints will be active. Our procedure works as follows: we solve a relaxed version of the MIP above—the *master problem*—using only the constraints corresponding to a small subset $Gen \subset Vert(W)$ of the constraints in the MIP above. We then test whether any unexpressed constraints are violated at the current solution. This involves computing the true SMR of the recommendation set generated by the master problem. If SMR is of the set is greater than δ, we know that a constraint has been violated. Computation of SMR produces the element $w \in Vert(W)$ and optimal product $\mathbf{x}^*_w$ that corresponds to the maximally violated constraint in the current master solution. We add this maximally violated constraint to Gen, tightening the MIP relaxation, and repeat; if no violated constraint exists, we are assured that the current solution minimizes SMR.[4]

The setwise max regret subproblem used to generate constraints is itself easily encoded as a MIP (similar to [3]), which can be solved directly. Given a set $\mathbf{Z}$, it computes $SMR(\mathbf{Z}, W)$ as well as $SMR\text{-}Adv(\mathbf{Z}, W)$. Note that this procedure is also used to construct recommendation/choice sets in the SCAS heuristic above.

Database problems Database problems, where options are enumerated in a product database, do not lend themselves to a direct constraint optimization when forming (regret-based) set recommendations since no explicit variable constraints can be encoded in a MIP, CSP or other formalism. Instead, we adopt a computational model that repeatedly determines the pairwise regret between a candidate recommendation set and an adversarial option in order to identify the option with minimax regret. This can be seen as a game between the recommender and an adversary. The computation of single recommendations is greatly facilitated in practice by formulating the optimization as a minimax search and using standard pruning techniques. However, computing optimal recommendation sets has complexity $O(n^k)$ (for a DB of size n), thus becoming increasingly impractical the size of the desired set increases. The hill-climbing method HCT, SCAS, and CSS are best suited to this case.

5. EMPIRICAL EVALUATION

We evaluate the effectiveness of our regret-based approaches for generating recommendation sets in simulation. We also compare them to several state of the art critiquing algorithms.

[3]Here R^j_w is the actual regret of jth option $\mathbf{X}^j$ w.r.t. w when the indicator I^j_w is activated (indicating that the jth option is the one with maximal utility in the set). For any w, exactly one I^j_w is set to 1. Hence minimization of δ ensures $I^j_w = 1$ for the $\mathbf{x}^j$ with least actual regret. Constraint (9) ensures R^j_w has its intended meaning; here M is any upper bound on max regret.

[4]Note that the adding a new constraint requires the introduction of new variables to the master problem. Every time we add a new w to Gen, k new binary variables I and V are necessary.

5.1 Critiquing Algorithms

We first review three state-of-the-art approaches to compound critiques. The *dynamic critiquing (DC)* model [16] uses a similarity metric to retrieve the current product and the APriori datamining algorithm to propose alternatives. Each compound critique describes a set of products in terms of the features they have in common. For example, in the computer domain, one compound critique might be "Faster CPU and Larger Disk." When the current product is shown, such patterns are converted into a set of suggested compound critiques, each corresponding to a product that is, among products satisfying the pattern, most similar to the current one. In our experiments, we use Apriori [1] with a support threshold of 0.3, and select compound critiques using the *low-support* strategy [16].

Incremental critiquing (IC-wsim) [17] incorporates a user *preference model*. While the APriori algorithm (as above) is used to discover patterns, these patterns are converted to suggestions using a *quality metric* that values both the *score* given to the product by the preference model and its similarity to the current product. Among all products that satisfy the pattern, the system selects the $\mathbf{x}$ that maximizes $score(\mathbf{x}) \cdot Similarity(\mathbf{x}, \mathbf{y})$, where $\mathbf{y}$ is the previous suggestion, and $score$ is measures the fraction of previously stated user critiques that are satisfied by the product.[5]

Incremental Critiquing: MAUT (IC-maut) [18] is a version of incremental critiquing that uses a multiattribute utility (MAUT) model to make recommendations and generate suggestions. In this approach, a simple additive utility model u is generated, initially giving equal weight to all attributes; each time an attribute is critiqued, its weight is modified by a multiplicative constant α (we use $\alpha = 2$ in our experiments, following [18]). The algorithm assumes value functions of fixed form, parameterized by the current option. Suggestions are generated by optimizing products w.r.t. the estimated utility model, and the k best products are presented to the user. A limitation of this approach is its reliance on a single utility model (as opposed to reasoning with the space of possible user utilities); furthermore, these suggestions are unlikely to be diverse or informative enough to generate useful distinctions.

5.2 Empirical Results

We compare our regret-based strategies and the critiquing methods above by showing the reduction of max regret of the current recommendation as a function of the number of recommendation cycles (critiquing opportunities) presented to the user, Since our emphasis on recommendations that are also informative for further elicitation—since the interaction can be terminated by the user at any stage—we are interested in the "anytime" profile of suggestion quality as well as its ultimate convergence. While our methods are tuned to reducing max regret, one might argue that *true regret*, i.e., the difference between the utility of the recommended product and the optimal product under the user's *actual* utility function, is the most critical measure. This is informative since minimax regret provides only an upper bound on this loss. Of course, in practice, we do not have access to the true utility function, but we do in simulation. So we present results for true regret as well.

In our experiments, we use two different datasets. The first is a database of 187 rental accommodation options (drawn from a real student rental database). Each option is characterized by 10 attributes, such as price, size, distance to university, etc. Attribute domains are either numeric or discrete with domains of size 2–6. The second, larger dataset is a synthetic dataset of 5000 rental units generated by sampling a generative model (Bayesian network) over

[5]When a a suggestion is selected, the system implicitly assumes that the user is critiquing the attributes that differ between the suggestion and the current product.

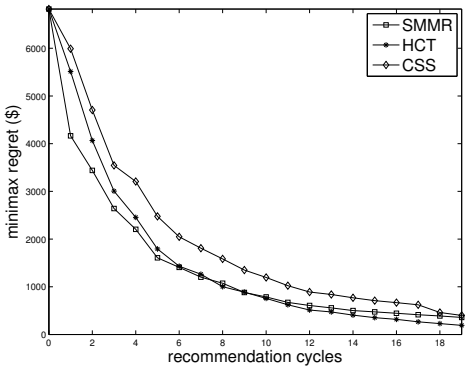

Figure 2: Max regret reduction ($k = 2$, db = 187, 50 runs).

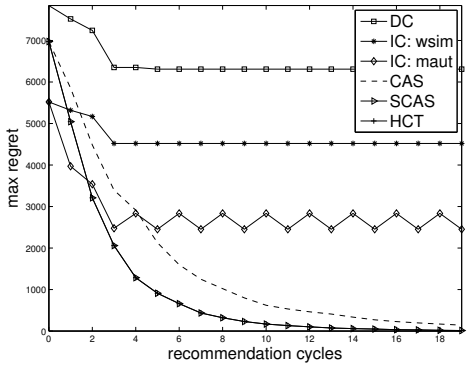

Figure 3: Max regret reduction ($k = 3$, db = 187, 50 runs).

the same 10 attributes.[6] The latter was used since it is much larger and has initial regret that is much higher.

We simulate the interaction of a single user by generating a random user utility function, and making user choices based on it.[7] We assume that the user initially orders the domain values of each attribute from best to worst (a simple and natural task). Computation time for SCAS and HCT in the 187 product domain averages 0.9s and 2.7s, respectively, and 7-15s in the 5000 product dataset. We are examining several techniques to speed computation, but even at these levels, they support real-time response.

We first compare our regret-based strategies—exact SMMR computation, HCT, and CSS—on the small, 187-option dataset, with 2 options in each recommendation set (k=2). Fig. 2 shows the max regret of the recommended product after each interaction (critique). While SMMR is optimal, both CAS and HCT offer very good approximations, and are much more computationally efficient (SMMR requires on average 285s per query). For this reason, we focus on CAS (and SCAS, which is identical to CAS when $k = 2$) and HCT in the remaining experiments.

We next compare the three regret-based approaches, HCT, SCAS, and CAS, and the three critiquing methods above, DC, IC, and IC-MAUT, with set size $k = 3$. Fig. 3 shows that traditional critiquing

[6]The model was generated from the smaller real database using the K2 Bayes net learning algorithm [8].

[7]Each attribute has a fixed independent probability p of being relevant to the user ($p \sim U[0.5, 1.0]$). Relevant attributes are randomly assigned an *importance*: *hi* (probability 0.2), *med* (0.3), or *lo* (0.5). Each importance has utility parameters drawn uniformly from $[0, 1]$ (*lo*), $[0, 5]$ (*med*), or $[0, 10]$ (*hi*). Numeric domains are characterized by a single weight, discrete domains, by a vector of local utilities where the least preferred value has zero marginal utility. The utility is then scaled with respect to the price range, so that the utility of an option is obtained by subtracting the price.

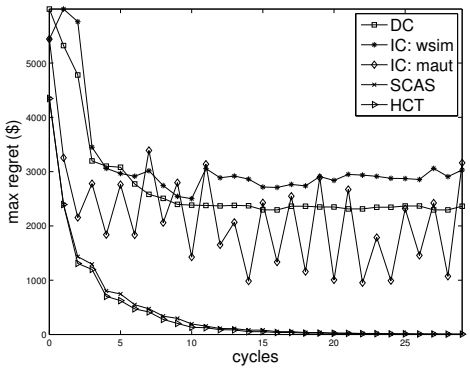

Figure 4: Regret reduction, alternating unit critiques and set suggestions ($k = 3$, db = 5000, 50 runs).

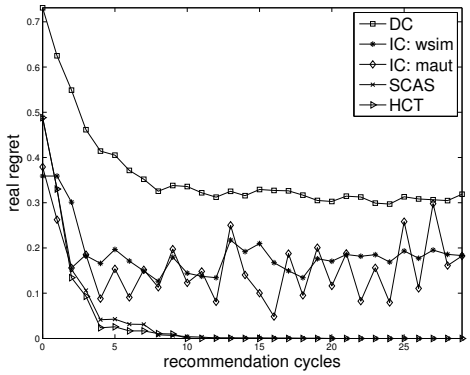

Figure 5: True regret (percentage of optimal value), alternate unit critiques/set suggestions ($k = 3$, db = 5000, 50 runs).

methods stall after only few cycles. We conjecture that the reliance of these approaches on a single ranking or utility model (rather than a space of possible user utilities) renders their suggestions insufficiently diverse to generate useful feedback. Regret-based recommendation sets, in contrast, reduce max regret very quickly, reaching zero regret in about 20 queries. HCT and SCAS are almost indistinguishable (the difference is not statistically significant) but both dominate CAS (with significant differences; we do not show error bars for legibility). This clearly demonstrates benefit of reasoning directly about the joint value of an option *set*.

We consider a third setting using the larger, 5000-apartment dataset, and allowing *unit critiques* in addition to our compound critiques (or set choices). In other words, apart from selecting an option from a set recommendation, the user can also critique a single attribute, asking for an improvement in its value. Unit critiques are modeled according to what we call the *ideal semantics*: the user is asked to choose the unit critique whose improvement for the current product $\mathbf{x}$ is best when the attribute is moved to its greatest level. We alternate between unit critiques and compound critiques (set suggestions) at each cycle. (More flexible user-controlled interactions are possible.) Fig. 4 shows the max regret of the recommended product at each cycle. Again, our regret-based recommendations provide better suggestions than the other methods by a significant margin. This is true when considering both the "anytime" profile of the method and its final convergence: our techniques not only discover the optimal product, but are able to "prove" its optimality (reach max regret is 0), in roughly 20 cycles. In contrast, the critiquing methods settle at relative max regret of about 40%. We notice that, compared to the first experiment, unit critiques seem to help regret reduction.

Regret-based critiquing is designed to attack *bounds* on regret (i.e., worst-case loss); generally, the recommended product will be closer to optimal than its max regret measure. Indeed, we may discover the optimal product long before being able to prove its optimality for the user. Since the other methods do not optimize regret bounds, perhaps they recommend good products despite being unable to "prove" they are good. Fig. 5 shows this not to be the case. It illustrates the *true regret* or the recommended product, that is, the difference between its true value to the user (given her utility function) and the value of the optimal recommendation. We express it as a percentage difference from the true optimal value. Regret-based critiquing offers better *actual recommendations*, as measured by the true regret of the recommended product. The other critiquing techniques recommend products that are better than their regret-bounds suggest; but regret-based critiquing consistently finds the optimal product (and finds a near-optimal product in as few as 5–6 interactions). By contrast, the other three methods are unable to identify the optimal option at convergence: incremental critiquing stabilizes at products that are nearly 20% worse than optimal; and dynamic critiquing never reaches products that are within 30% of optimal. The difference of true regret between our methods and dynamic/incremental critiquing is statistically significant.[8]

6. CONCLUSIONS

We have developed a novel, minimax-regret based formalization for recommending sets of alternatives in conversational recommender systems. Our setwise max regret criterion is a natural extension of max regret for single recommendations, not only providing robust recommendation sets, but also serving as a means of generating myopically optimal queries or suggestions for critique. We developed computational MIP methods for optimal recommendation sets, as well as tractable approximations. Even when approximations are used, our reliance on explicit utility modeling and minimax regret provides a powerful new means of generating good critiques and making good product recommendations. Our regret-based recommender often leads to optimal recommendations using very few compound critiquing interactions, and outperforms other dynamic critiquing techniques both in speed of convergence and the quality of the final recommendations.

Further verification of our regret-based approach requires user studies to determine the intuitive acceptability of regret-based recommendations;[9] we also plan a comparison with diversity-enhanced incremental critiquing [13]. Largely unaddressed in our critiquing model is the need for users to explore the product space (one of the main advantages of critiquing). We are developing hybrid models that distinguishes exploratory actions from improving actions. Finally, the development of models of cognitive costs using techniques from behavioral economics and decision theory remains an important avenue of future research.

7. ACKNOWLEDGMENTS

This work was partially supported by the Natural Sciences and Engineering Research Council (NSERC). The first author was partially supported by the Swiss National Science Foundation (grant PBEL2-120935).

[8]The standard deviation for both HCT and SCAS quickly decreases as recommendations converge to the optimal product; e.g., for HCT, deviation is less than 0.05 after the 10th cycle. All 50 runs recommend the optimal product at or before the 16th cycle with HCT, and at or before the 21st cycle for SCAS.

[9]These studies are planned; ongoing preliminary studies of regret-based elicitation in single item (not set-based) domains suggest that max regret works very well in practice.

The authors would like to thank Darius Braziunas for valuable discussion on minimax regret computation for database problems and the anonymous reviewers for their valuable suggestions.

8. REFERENCES

[1] F. Bodon. A fast apriori implementation. *IEEE ICDM Worksh. on Frequent Itemset Mining Implementations*, Melbourne, FL, 2003.

[2] C. Boutilier. A POMDP formulation of preference elicitation problems. *18th National Conf. on AI (AAAI-02)*, pp.239–246, Edmonton, 2002.

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

[4] C. Boutilier, R. Zemel, B. Marlin. Active collaborative filtering. *19th Conf. on Uncertainty in AI (UAI-07)*, pp.98–106. Acapulco, 2003.

[5] D. Braziunas and C. Boutilier. Minimax regret-based elicitation of generalized additive utilities. *23rd Conf. on Uncertainty in AI (UAI-07)*, pp.25–32, Vancouver, 2007.

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

[7] U. Chajewska, D. Koller, and R. Parr. Making rational decisions using adaptive utility elicitation. *17th National Conf. on AI (AAAI-00)*, pp.363–369, Austin, TX, 2000.

[8] G. Cooper, E. Herskovits. A Bayesian method for the induction of probabilistic networks from data. *Mach. Learn.*, 9:309–347, 1992.

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

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

[11] P. Kouvelis and G. Yu. *Robust Discrete Optimization and Its Applications*. Kluwer, Dordrecht, 1997.

[12] T. Hadzic and B. O'Sullivan. Critique graphs for catalogue navigation. *RecSys '08*, pp.115–122, Lausanne, 2008.

[13] K. McCarthy, J. Reilly, B. Smyth, L. McGinty Generating Diverse Compound Critiques. *Artif. Intell. Rev.* 24(3-4): 339-357 (2005)

[14] D. McSherry. Diversity-conscious retrieval. *6th Eur. Conf. on Advances in Case-Based Reasoning*, pp.219–233, London, 2002.

[15] R. Price and P. Messinger. Optimal recommendation sets: Covering uncertainty over user preferences. *20th National Conf. on AI (AAAI'05)*, pp.541–548, 2005.

[16] J. Reilly, K. McCarthy, L. McGinty, B. Smyth. Dynamic critiquing. In P. Funk, P. A. González-Calero, eds., *ECCBR*, *Lecture Notes in Computer Science* 3155, pp.763–777. Springer, 2004.

[17] J. Reilly, K. McCarthy, L. McGinty, and B. Smyth. Incremental critiquing. *Knowl.-Based Syst.*, 18(4-5):143–151, 2005.

[18] J. Reilly, J. Zhang, L. McGinty, P. Pu, and B. Smyth. Evaluating compound critiquing recommenders: a real-user study. *ACM Conf. on Electronic Commerce*, pp.114–123, 2007.

[19] A. Salo and R. 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.

[20] L. Savage. *The Foundations of Statistics*. Wiley, 1954.

[21] P. Slovic. The construction of preference. *American Psychologist*, 50(5):364–371, 1995.

[22] M. Torrens, B. Faltings, and P. Pu. Smartclients: Constraint satisfaction as a paradigm for scaleable intelligent information systems. *Constraints*, 7(1):49–69, 2002.

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

[24] P. Viappiani, B. Faltings, and P. Pu. Preference-based search using example-critiquing with suggestions. *J. Artif. Intell. Res.*, 27:465–503, 2006.