Optimal Set Recommendations based on Regret

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 do not offer guarantees on the quality of their recommendations, either because they do not maintain a model of a user's utility function, or do so in an *ad hoc* fashion. In this paper, we propose an approach to recommender systems that incorporates explicit utility models into the recommendation process in a decision-theoretically sound fashion. The system maintains explicit constraints on the user's utility based on the semantics of the preferences revealed by the user's actions. In particular, we propose and investigate a new decision criterion, *setwise maximum regret,* for constructing optimal recommendation sets. This new criterion extends the mathematical notion of *maximum regret* used in decision theory and preference elicitation to sets. We develop computational procedures for computing setwise max regret. We also show that the criterion suggests choice sets for queries that are myopically optimal: that is, it refines knowledge of a user's utility function in a way that reduces max regret more quickly than any other choice set. Thus setwise max regret acts both as guarantee on the quality of our recommendations and as a driver for further utility elicitation.

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.

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 decision process. User feedback can be entered in many forms, for instance, as direct answers to queries, or *critique* of the options displayed by the system.

Many recommender systems employ some form of *diversity* to show a set of products that might be appealing to the user. Intuitively, diversity overcomes a key problem with presentation of the *top-k* items based on some estimate of a user's score: the latter tends to produce results that are very similar one to each other, and thus not offer much actual "choice" for a user. This is especially true when we recognize that estimated scores or preferences are likely to be very crude. Diversity is also important in practice: we cannot generally predict how *patient* a user will be. They may terminate the exploration of product space at any time, hence the recommender system should be able to provide *anytime* recommendations, reflecting the best recommendations given the information provided by the user so far. This characteristic of conversational recommenders is similar to the exploration-exploitation dilemma in reinforcement learning. Since we do not know know how much time the user is willing to spend in order to improve the recommendation, we want to show products that are both: (a) expected to be rated highly given the current information about the user; and (b) are maximally informative should the user critique (or otherwise provide feedback on) them.

Many authors have considered the importance of diversity in the recommendations. For example, researchers in case-based reasoning have proposed techniques based on greedy maximization of diversity (McSherry 2002), defined as an aggregate of a distance metric, or as a weighted tradeoff between diversity and the recommendation score (Smyth and McClave 2001). However diversity and dissimilarity measures does not consider the information that we have about the user's preference. While they guarantee that the set contains alternatives that differ in their features, they do not use at all the information about a user's preferences available from previous user actions and feedback. It has been argued that diversity should be instead *tailored* to the system's belief about the user (Price and Messinger 2005).[1]

To maximize the information presented to the user in a recommendation set, and recommend a set of *optimal* recommendations, it is necessary to maintain an explicit representation of the uncertainty in the preference model and a sound decision-theoretic semantics of the interaction in the first place. In fact, most practical conversational recommender systems (especially those using critiquing) do not use an explicit model of a user's preferences, or only main-

[1]Indeed, the natural decision theoretic account of set recommendations immediately suggests diversity w.r.t. belief about a user's preferences (Boutilier *et al.* 2003).

tain such a model in an ad hoc, heuristic fashion. In this paper, we develop an approach to set-based recommendations with an explicit utility model. We represent the uncertainty w.r.t. the user model with constraints of her utility function induced by choices or critiques. To construct a suitable recommendation set, we develop a novel criterion, *setwise maximum regret*, that captures the idea of providing a set of *jointly* optimal recommendations. Our qualitative model of uncertainty has two key advantages over probabilistic models (Price and Messinger 2005): relatively simple prior information in the form of bounds or constraints on user preferences can be exploited (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 (Boutilier 2002; Chajewska and Koller 2000)).

To make this model effective, user actions should be associated with a precise, sound *semantics*. For instance, a user critique is assumed to reveal some aspect of the user's preferences and this is used to update an explicit utility model. More precisely, in our work, unit critiques and compound critiques places linear constraints on a user utility function. The advantage of this approach is that we can use decision-theoretically sound criteria to:

1. suggest or recommend a product;
2. bound the difference in the quality of a recommended product and the optimal option for the user;
3. determine which options and critiques carry the most information to help speed up the navigation process; and
4. suggest to the user when to terminate the process (i.e., when further interaction will offer only modest improvement in recommendation quality).

We adopt the notion of *minimax regret* (Boutilier *et al.* 2006a) to make product suggestions in the face of utility function uncertainty. This robust decision criterion allows us to bound the loss (difference from optimal) of any recommendation. We propose and investigate a new decision criterion, *setwise maximum regret*, for constructing optimal recommendation sets. This new criterion extends maximum regret to sets of products rather than a single product. We define set maximum regret, argue that minimizing setwise max regret is the best means for constructing a set of options for a user, and develop effective computational procedures for computing optimal recommendation sets for setwise regret.

We present *critiquing* as a possible application domain. While user-controlled exploration in traditional critiquing systems does not offer any guarantees (practical, empirical, or theoretical) of either sufficient or efficient exploration of the space (A user may cycle through a set of similar products or converge at a product far from optimal), our regret-based recommender allows us to provide guarantees on the quality (utility) of the recommended product vis-à-vis feasible alternatives. We also show with simulations that *regret-based critiquing* can lead to much more efficient exploration of the product space and lead to better decisions in practice.

In Sec. 2 we introduce our model of regret-based recommendation and describe our strategy for selection of a joint set of recommended alternatives using setwise minimax regret. In Sec. 3 we discuss computation of setwise max regret and minimax regret, both for configuration problems modeled as a constraint satisfaction problem (CSP) and for product databases, while in Sec. 4 we briefly discuss the performance of elicitation. Finally, in Sec. 5, we perform simulations of complete critiquing-based recommender systems, comparing our regret-based approach to state of the art critiquing algorithms such as dynamic critiquing and incremental critiquing.

Regret-based Recommendation Systems

We begin this section by presenting our formalization of the decision problem, reviewing minimax regret for robust recommendation and elicitation, and then defining our key concepts of setwise max regret and setwise minimax regret.

Underlying Decision Problem

We assume a recommendation system is charged with the task of recommending an option to a user in a multi attribute space (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* $u : Dom(\mathcal{X}) \to \mathbf{R}$. In what follows we will assume either a *linear* or *additive* utility function depending on the nature of the attributes (Keeney and Raiffa 1976). In both additive and linear models, we assume that 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 (Keeney and Raiffa 1976; Fishburn 1967). 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.[2]

Since a user's utility function is not generally known, we often write $u(\mathbf{x}; w)$ to emphasize the dependence of u on

[2]Our approach relies considerably on the additive assumption, though can easily be generalized to more general models such as GAI (Fishburn 1967; Bacchus and Grove 1995; Braziunas and Boutilier 2007a). The assumption of linearity is simply a convenience; nothing critical depends on it.

user-specific parameters. In the additive case, the values $f_i(x_i)$ over $\cup_i\{Dom(X_i)\}$ serve as a sufficient parametrization 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.

Regret-based Recommendation

In probabilistic approaches to recommendation, a distribution over preferences—typically in the form a density over utility function parameters—is maintained, and the option with highest expected utility is recommended (Chajewska *et al.* 2000; Boutilier 2002; Boutilier *et al.* 2003). When a set of alternatives need to be recommended, the expectimax or EMAX criterion can be used (Boutilier *et al.* 2003; Price and Messinger 2005). One difficulty with probabilistic models is that one requires probabilistic prior information over utility models, which can be difficult to formulate and represent. Another is that exact computation can often be computationally intense; this is especially true since (arguably) natural density models for utility functions are rarely closed under the type of evidence provided by user interaction (e.g., behavioral observation or answers to queries) (Boutilier 2002; Chajewska and Koller 2000)); as a result, computationally demanding fitting of (say) mixture models is required after every model update.

Instead, we propose the use of minimax regret to generate *recommendation sets*. As we will see, this obviates the need to complex probabilistic reasoning, yet can offer robust recommendations and provide very effective guidance for the user. In traditional regret-based approaches, a single recommendation is made using the minimax regret (Savage 1954) criterion. For multiple joint recommendations, we develop the notion of *setwise minimax regret* (defined below). We can summarize the correspondence between the Bayesian and the regret-based approach with the following table:

Probabilistic approach	Regret approach
Expected Utility	Minimax Regret
Expected Max (EMAX)	Minimax *Setwise* Regret

In this paper we propose a framework that maintains a set W of feasible utility models, and at each step, the system shows a set of recommendations that are *jointly* optimal with respect to minimax regret. At a very high level, our regret-based recommender works as follows:

1. The set W is initialized given some initial constraints;
2. The current recommendations are determined (using the setwise minimax regret);
3. After each user action, W is refined to reflect the new constraints imposed by the user's feedback;
4. The process repeats (steps 2 and 3) until the user is satisfied or minimax regret reaches some target threshold.

This process is appealing for two reasons. First, the current recommendation (i.e., set of options) is always optimal; in other words, it minimizes setwise max regret given the current information about the user's utility function, making it extremely robust in the presence of utility function uncertainty (in a way to be made precise below). Second, max regret is a well-defined progress metric that lets the user know the cost and benefit of further exploration of product space. Finally, the information contained in user selection of some choice from the recommended set is maximally informative (in a sense defined below).

Minimax Regret

Minimax regret has been advocated as a means for robust optimization (Kouvelis and Yu 1997), and has more recently been used for decision making with utility uncertainty (Boutilier *et al.* 2001; Salo and Hämäläinen 2001; Boutilier *et al.* 2006a).

Assume that through some interactions with a user, and possibly using some prior knowledge, we determine that her utility function w lies in some set W. Following (Boutilier *et al.* 2006a) we define:

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

Example Consider the following example, where the options o^i are defined using two features/coordinates x_1 and x_2:

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

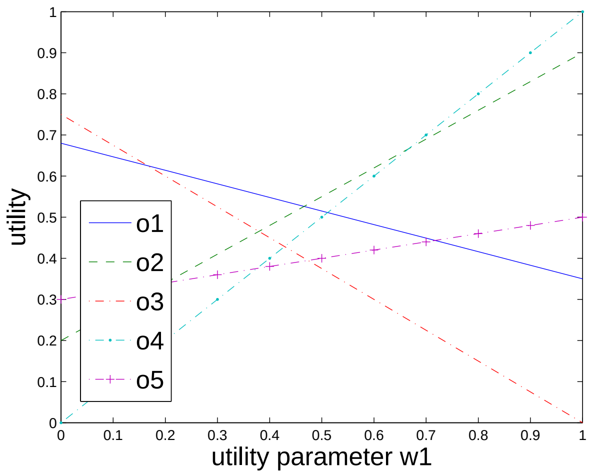

Figure 1: Each options is represented by a single line in the utility space.

We assume linear utility: $u(\mathbf{x};\mathbf{w}) = w_1x_1 + w_2x_2$ where $\mathbf{w}$ is vector of tradeoff weights, with $w_2 = 1-w_1, 0 \leq w_1 \leq 1$; and the local value functions for each coordinate are identity functions. (i.e., $v_i(x_i) = x_i$). Given these assumptions, utility is one-dimensional; it can be written as $u(\mathbf{x};\mathbf{w}) = (x_1 - x_2)w_1 + x_2$. So we deal only with the uncertainty on the single parameter w_1.

This simple example is convenient because it is easy to visualize option utilities as a 1D function of w_1 graphically. The utility of the different options are shown in Fig. 1 with respect to the parameter w_1. We notice that, for some values of w_1, each of the options o_1, o_2, o_3 and o_4 is optimal, but not so o_5. When considering a particular value of w_1 (a particular utility function) the *actual regret* (or real loss) is the difference between the utility of the best option given w_1 and the utility of our recommendation in that case. For instance, for $w_1 = 0.9$, the best option is o_4 with utility 0.9, while o_1 has utility 0.38, so the actual regret of o_1 would be $0.9 - 0.38 = 0.52$. Max regret accounts for the uncertainty over w_1 and it is the maximum of the possible actual regret values (for o_1 is 0.65 when $w_1 = 1$).

When the options are few as in this case, we can compute the max regret of each choice by explicitly enumerating the maximum the pairwise regret of that choice against any possible adversarial choice of option. The table below illustrates this, where each row corresponds to a recommendation, each column to an adversarial choice, and we display the pairwise max regret (allowing the adversary to choose utility) in the cells. The max regret of an option is shown in the last column, and corresponds to the maximum value in its row. In the unconstrained situation (where w_1 can take any value between 0 and 1), we have the following values:

$MR(o^i, o^j)$	o^1	o^2	o^3	o^4	o^5	$MR(o_i)$
o^1	0	0.55	0.07	0.65	0.15	0.65
o^2	0.48	0	0.55	0.1	0.1	0.55
o^3	0.35	0.9	0	1.00	0.5	1
o^4	0.68	0.2	0.75	0	0.3	0.75
o^5	0.38	0.4	0.45	0.5	0	0.5

Minimax regret is 0.5, and the minimax optimal recommendation is option o_5; its max regret occurs at adversarial choice of utility $w_1 = 1$, and choice of option o_4. It can be easily shown that regret is maximized at one of the vertexes of the feasible region W when W is a bounded, convex polytope (such a polytope induced by the interactions we discuss later).

Now imagine that, perhaps as the result of interactions with the user, we learn that $0.2 \leq w_1 \leq 0.6$. The new minimax regret value for this constrained case is 0.138. This value corresponds to recommendation o_1, and adversarial option o_2 and utility $w_1 = 0.6$. In this case, the constraints $0.2 \leq w_1$ and $w_1 \leq 0.6$ decrease minimax regret significantly. However usually constraints are not added directly as such, but result from the acquisition of knowledge acquired through a variety of interaction modalities, such as direct user preference queries, or passive observation of user behavior. For instance, comparison queries ask the user which of two proposed options is preferred. The impact of the information acquired depends greatly on the comparison, as different options can lead to different degrees of regret reduction.

A natural meta-heuristic for generating elicitation queries is the *current solution strategy (CSS)*, first described in (Boutilier *et al.* 2006a). This strategy would ask the user whether she prefers the minimax regret option x^*_W or the adversary option $x^w = MRAdv(x^*_W, W)$.

In our example, starting from the unconstrained space W, CSS would select $\{o_4, o_5\}$ (the minimax regret option and the adversarial option) and ask the user to compare them. Now, let's assume that the user asserts that she prefers o_4 over o_5; then a new constraint $u(o_4; w_1) \geq u(o_5; w_1)$, equivalent to $w_1 \geq 0.375$, is added to our model. In the space $W^{o_4 \succ o_5}$ resulting from the incorporation of this constraint, the minimax regret is 0.1, resulting from recommendation o_2 and adversarial option o_4. Option o_4, even if known to be better than o_5, has max regret of 0.18 (at $w_1 = 0.374$, with adversarial option o_1). Therefore, option o_2 will be recommended.

Minimax regret offers recommendations that are robust given the uncertainty of the preference model. In this example, o_5 is recommended (in the unconstrained setting) even though it cannot possibly be optimal for *any* user utility function; this is so because it prevents "disastrous" situations, such as would occur if options o_1 or o_2 are recommended when w_1 is very low (despite the fact that for a good part of utility space, these options are optimal. Note, that as knowledge of user utility increases, more accurate recommendations are made; for example, recommending o_2 when we learn that o_4 is preferred to o_5.

Optimal Recommendation Sets: Setwise Regret

In most cases the value of a set of recommendations is dependent on the elements of the set jointly, not on each individually. If the user is going to benefit from only one of the recommendations (example: recommending apartments) then the utility of the set is then the maximum utility among the individual options, i.e., the one the user will pick from the set.

The problem of set recommendations has been addressed using probabilistic expectation: Price and Messinger (Price and Messinger 2005) optimize set recommendations using the EMAX criterion, defined as the expectation of the maximum utility among the options in the set.

In order to retrieve optimal set recommendations, we define the notion of *setwise max regret*. The setwise max regret of a recommendations set can be seen as the equivalent of EMAX in our non-probabilistic framework. Suppose we have a slate of k options to present to the user and want to quantify the possible loss by 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 from among any of the set of k options. In this formalization, we choose the set of k options first, but delay the final choice from the slate only *after* the adversary has chosen a utility function w. The regret of a set is then the minimum difference between the utility of the best configuration under w and the utility of the options in the slate. Specifically, define the *setwise maximum regret* of option set $\mathbf{Z} = \{\mathbf{x}^1, \ldots, \mathbf{x}^j\}$ to be:

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

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

Setwise max regret has some intuitive properties. First, adding new items to a set cannot increase setwise max regret: $SMR(\mathbf{A} \cup \mathbf{B}, W) \leq SMR(\mathbf{A}, W)$. At the same time incorporating options that are known to be dominated given W does not change setwise max regret: in other words, if $u(\mathbf{a}, \mathbf{w}) > u(\mathbf{b}, \mathbf{w})$ for some $\mathbf{a} \in \mathbf{Z}$ and all $\mathbf{w} \in W$, then $SMR(\mathbf{Z} \cup \{\mathbf{b}\}, W) = SMR(\mathbf{Z}, W)$. Finally, the max regret associated with recommending the entire product set is zero: $SMR(\mathbf{X}, W) = 0$. This is the equivalent to asking the user to directly choose the best option from the space of available options—obviously, a task of with extreme cognitive cost, and one that runs counter to the spirit of recommendation assistance! But should the user be able to answer correctly, it guarantees optimality.

Setwise max regret can be equivalently written in as follows:

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

This captures the intuition that, given 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 can be useful to explicitly partition utility space with respect to which option in $\mathbf{Z}$ is maximal. We define the utility subset $W^{\mathbf{Z} \to \mathbf{x}_i}$ as the set of utilities such that $\mathbf{x}_i$ has greater utility than any option in $\mathbf{Z}$.

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

The set of all $W^{\mathbf{Z} \to \mathbf{x}_i}$ for any $\mathbf{x}_i \in \mathbf{Z}$ partitions W (we ignore the possibility of ties over full-dimensional subsets of W, which can easily be dealt with, but complicate the presentation marginally). An important observation (that will be used later) is that we can rewrite the setwise max-regret SMR as the aggregate maximum of the (individual) max-regret considering a partition of the utility space according to which option has higher utility.

Observation 1 *Given* $\mathbf{Z} = \{\mathbf{x}_1, \ldots, \mathbf{x}_k\}$ *and, for* $1 \leq i \leq k$,

$$SMR(\mathbf{Z}, W) = max[MR(\mathbf{x}_1, W^{Z \to \mathbf{x}_1}), \ldots, MR(\mathbf{x}_k, W^{Z \to \mathbf{x}_k})] \quad (6)$$

Example (continued) We now consider setwise max regret for the example introduced above. Let the number of options in a recommendation set be $k = 2$. The following combinations are ranked best according to the setwise regret criterion.

Set	SMR	Adversary	Adversary W
$\{o_1, o_4\}$	0.07	o_3	$w_1 = 0$
$\{o_1, o_2\}$	0.1	o_4	$w_1 = 1$
$\{o_3, o_2\}$	0.1	o_4	$w_1 = 1$
$\{o_3, o_4\}$	0.11	o_1	$w_1 = 0.42$

The set $\{o_1, o_4\}$ is the best choice for a joint recommendation of two options, corresponding to a value of regret of 0.07. Other combinations, such as $\{o_1, o_2\}$, $\{o_3, o_2\}$ and $\{o_3, o_4\}$, also have a relative low value of regret.

A set recommendation can often have dramatically lower regret than the minimax optimal *single* recommendation (in this case, o_5).

It is interesting the fact that the optimal recommendation set is composed of two options, o_1 and o_4 that, when considered alone, are associated with high regret. Any set including o_5, the single best recommendation, is ranked poorly with respect to setwise regret.

We now consider the case of larger sets. If we need to select a slate of three options ($k = 3$), the regret will be 0.04 and the recommendation would be $\{o_1, o_3, o_4\}$; in this case the adversary would pick o_2, and the value $w_1 = 0.51$ (intersection point of o_1 and o_4).

In the case of four options to be selected ($k = 4$), the set $\{o_1, o_2, o_3, o_4\}$ would be recommended and it would be associated to a setwise regret of 0: the slate includes all the options that can ever become optimal (considering Observation 1, it follows that for any $W^{\mathbf{Z} \to o_i}$ that partitions W, the max regret has to be 0).

Now we consider how setwise regret changes when new information is included. We consider a slate of two options to be selected ($k = 2$) and we suppose that the user asserts the preference of o_4 over o_5. The recommendation set is still

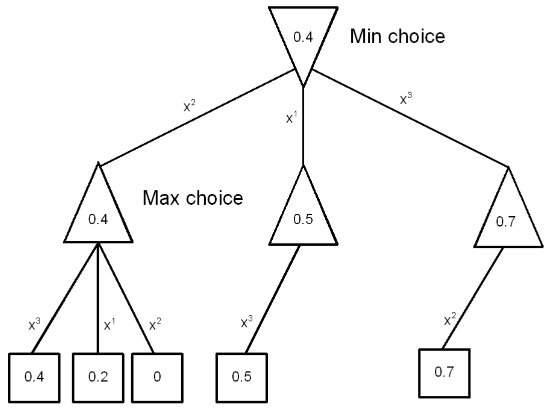

Figure 2: Alpha beta pruning can speed up the search, depending on the evaluation order. In this case, x^1 has regret 0.5 against x^3, that is worse than the value 0.4 (max regret of x^2), so we do not need to test x^1 against x^2.

$\{o_1, o_4\}$ but with a much lower value of (setwise) regret: only 0.04.

We conclude the discussion of the example with some remarks on the optimization process. The adversary's utility does not necessarily corresponds to one the vertex of the feasible region, as in the single recommendation case; it may also lie in any intersection of the hyperplanes associated with the options. For instance (in the unconstrained case) the setwise regret of $\{o_3, o_4\}$ is maximized for the value $w_1 = 0.42$ (the utility that makes o_3 and o_4 equally preferred).

Computation of Setwise Minimax Regret

In this section we discuss how to efficiently compute regret-based recommendations. We first discuss how to compute minimax regret for single recommendations and then describe how to modify these procedures to compute setwise minimax regret for recommendation sets. We distinguish two settings: *configuration problems*, where options are defined by variables and configuration constraints (i.e., as solutions to a constraint satisfaction problem (CSP)); and *database problems*, where options are enumerated in a product database.

Computing Minimax Optimal Single Recommendations

Configuration problems In configuration problems, optimization over product space $\mathbf{X}$ is formulated as a constraint optimization problem or 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. We refer to (Boutilier *et al.* 2006a; 2004; Braziunas and Boutilier 2007a) for more details. Our MIP formulations for setwise minimax regret below will draw heavily on these techniques, but necessitate important modifications.

Database problems When options are enumerated in a product database, minimax regret computation requires to repeated computation of the pairwise regret between a candidate recommendation and an adversarial option in order to identify the option with minimax regret. For ease of presentation, assume a linear utility function as above, defined by weights w_i over m attributes. Pairwise regret $MR(\mathbf{x}, \mathbf{y}, W)$ of recommendation $\mathbf{x}$ and adversarial option $\mathbf{y}$ is readily computed with the following LP:

$$\max_{\mathbf{w}: w_i \in [0,1]} \sum_{1 \leq i \leq m} w_i (y_i - x_i) \tag{7}$$

$$\text{s.t.} \sum_i w_i = 1 \tag{8}$$

$$\mathbf{w} \in W \tag{9}$$

Here we assume the feasible parameter set W is captured by linear constraints. A similar LP can be formulated for discrete-valued attributes, without assuming linearity, just additivity. Hybrid models with continuous and discrete attributes can easily be represented with a combination of these two representations. Generalized additive utility models models (Fishburn 1967; Braziunas and Boutilier 2006; 2007b) can also be easily represented in this framework. This means that pairwise regret can be computed extremely efficiently (e.g., in a few milliseconds using CPLEX on the types of problems discussed below).

Minimax regret computation is more complex because we need to maximize over all possible adversarial choices, and minimize over all possible recommendations. A naive approach would consider every pair of options, requiring $O(n^2)$ pairwise regret computations for a database of size n, where each of these computations requires the solution of an LP of size proportional to the number of utility parameters.

However, since minimax regret can be seen as a game between the recommender and an adversary, the computation can be greatly improved in practice by formulating the optimization as a minimax search and using standard pruning techniques. Unlike typical games, the search tree has very limited depth: only two ply, one choice of recommendation by the MIN player (attempting to minimize regret) and one choice of adversarial option by the MAX player (attempting to maximize the regret of the recommendation).[3] Note that the game has a large number of actions, once per product in the database. The MIN player (recommender) moves first, the MAX player (adversary) second. The leaves of the minimax tree are labeled with the pairwise max regret of the two choices on its path.

A full evaluation of the tree requires the solution of $n(n-1)$ pairwise regret LPs (noting that the MIN player's choice need not be explicitly evaluated or even represented as a possible MAX choice, since it must yield pairwise regret of 0). However, it is generally not necessary to evaluate every node of the tree as *Alpha-beta pruning* (see (Russell and Norvig 2003) for an introductory description) can be used to eliminate branches from evaluation.

[3]The choice of the utility function by the adversary is dictated by pair of options, so it need not be modeled as a move.

Alpha-beta pruning is simple in such a simple game tree: during the tree evaluation, we maintain an upper bound UB (initially $+$Inf) at the root, representing the max regret of the best solution found so far (from the perspective of MIN), and lower bounds $LB(n)$ at each MAX node, one for each possible MIN choice (or recommendation). Every time we evaluate a leaf node, we compute pairwise regret $MR(o_{\min}, o_{\max}, W)$ of MIN's choice $o_{\min}$ and MAX's choice $o_{\max}$ on the path. We update the lower bound at the corresponding MAX node, and prune (α cut [4]) whenever $LB(n) \geq UB$. This is because $MR(o_{\min}, o_{\max}, W) \leq MR(o_{\min}, W)$. At the same time, whenever we complete the evaluation of a MAX node n, we update the upper bound UB to $\min(UB, v(n))$ where $v(n)$, the value of the node, is the maximum value among the leafs.

The efficiency of this pruning depends on the order in which nodes are evaluated (Russell and Norvig 2003); this is especially true given the very shallow, broad nature of our tree. Pruning is most effective when, at each node, the best children (with respect to the relevant node evaluation, MIN or MAX) are evaluated first. Figure 2 shows, in our simple example, that in the best case only 5 nodes out of 9 need to be evaluated (i.e., 5 pairwise regret maximization). To speed up the search, we consider a heuristic that first evaluates choices at the MIN (recommender) node that are likely to be good candidates for minimizing max regret; and we first evaluate at at MAX (adversarial) nodes options that are likely to induce high regret against the given MIN choice. These heuristics give us an evaluation order for both MIN and MAX choices and can lead to considerable pruning. We discuss each in turn.

For the MIN node, we note that the regret of any option is maximized at one of the vertices of the feasible region W. Thus we sample t vertices (for instance, by considering extreme weights that maximize the importance of one of the attributes) and refer to the the $\mathbf{w}$ so-sampled as *reference utilities*. These are used to initialize the lower bounds $LB(n)$: we simply compute the actual regret with respect to these utilities for the option that leads to (MAX node) n. We then evaluate MIN's children n in increasing order of initial lower bound $LB(n)$.

To order the children of MAX node, for each MAX node n, we consider the feasible utility function $\mathbf{w}^-$ that minimizes the utility the MIN choice. (This requires a simple optimization.) The option that maximizes utility at $\mathbf{w}^-$ (i.e., the optimal choice under $\mathbf{w}^-$) is likely to give a high value of for pairwise regret and thus represents a potentially good adversary. Moreover, once we have generated $\mathbf{w}^-$, we can use it to update the lower bound by considering the actual regret for each option. MAX choices are evaluated in order of decreasing utility under $\mathbf{w}^-$.

In practice, these heuristics can significantly speed up the computation of minimax regret in product databases. Table 1 shows that number of pairwise regret checks (LPs) is almost linear in the number of options in the database; indeed, with these orderings, MAX nodes are often pruned immediately without even considering an adversarial choice. (These are experiments run on synthetic data for illustrative purposes.)

[4]beta cuts are not possible given the depth of the tree

size	attributes	constraints	num of pairwise checks
40	4	0	41
200	5	0	207
400	7	10	492
1000	10	0	1003
1000	10	60	1998
1000	15	30	999

Table 1: Number of pairwise regret checks to compute minmax regret on some sample datasets. We evaluate the search tree with our heuristics of reference utilities.

Set Recommendations: Setwise Minimax Regret

We now consider the modification of the techniques above for setwise minimax regret. Naturally, setwise max regret is more computationally demanding, requiring selection of a set of options. However, it is still possible to formulate the computation in a MIP for configuration problems. Database problems are more challenging: the adversarial search presented above for single-item recommendation can be applied directly, with replacement of a single move by the recommender (MIN player) by k moves, corresponding to the choice of k options for the slate. However, performance can take a dramatic hit as the size of the desired recommendation set increases. However, we develop a simple heuristic hill-climbing strategy that seems to provide very good recommendation sets in practice.

Configuration problems: MIP formulation For configuration problems we formulate the problem of setwise minimax regret following the general strategy for single-option minimax regret, formulating as a (MIP) minimization with exponentially many constraints. We use a constraint generation procedure to prevent enumeration of the entire constraint set (Boutilier *et al.* 2006a; 2004). However, there are some critical differences in the formulation, which we describe here.

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

$$\min_{M, I_w^j, \mathbf{X}^j, V_w^j} \quad M$$

$$\text{s.t. } M \geq \sum_{1 \leq j \leq k} V_{\mathbf{w}}^j \quad \forall \mathbf{w} \in \mathit{Vert} \tag{10}$$

$$V_w^j \geq \mathbf{w} \cdot (\mathbf{x}_{\mathbf{w}}^* - \mathbf{X}^j) + (I_w^j - 1) m_{big} \tag{11}$$

$$\forall j \in [1,k] \wedge \quad \forall \mathbf{w} \in \mathit{Vert} \tag{12}$$

$$\sum_{1 \leq j \leq k} I_{\mathbf{w}}^j = 1 \quad \forall \mathbf{w} \in \mathit{Vert} \tag{13}$$

$$I_{\mathbf{w}}^j \in \{0,1\} \tag{14}$$

$$V_{\mathbf{w}}^j \geq 0 \quad \forall j \in [1,k], \forall \mathbf{w} \in \mathit{Vert} \tag{15}$$

This MIP minimizes M by: (a) choosing k options (or configurations $\mathbf{x}_j$ designated by variables $\mathbf{X}^j$ (where each $\mathbf{X}^j$ is a vector of n attributes) for the recommendation set;

(b) selecting, for each adversary w, one of those options (the jth option) as the choice that has minimum max regret against an adversary, and ensuring that M is greater than the true regret of the jth option relative to every possible choice of adversary utility function and option.

Note however that this constraint need not be applied to (continuously many) utility functions or exponentially many adversarial choices. In the MIP, we post these constraints only for each vertex of W (i.e., $\mathbf{w} \in \mathit{Vert}(W)$) and for the optimal product choice $\mathbf{x}^*_{\mathbf{w}}$ for that vertex. This relies on the observation that regret maximized at vertices of W, and, for any adversarial choice of $\mathbf{w}$, the adversarial option that maximizes the pairwise regret for any user choice is the optimal option for $\mathbf{w}$.[5]

However, this MIP still requires (potentially) exponentially many constraints, one for each element of $\mathit{Vert}(W)$. We can make computation much more effective by applying constraint generation, observing that at the optimal solution, very few of these constraints are likely to 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 $\mathit{Gen} \subset \mathit{Vert}(W)$ 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 setwise max regret of the slate generated by the master problem. If the true setwise max regret is of the slate is greater than δ, we know that a constraint has been violated. Specifically, the computation of setwise max regret will produce the element $\mathbf{w} \in \mathit{Vert}(W)$ and optimal product $\mathbf{x}^*_{\mathbf{w}}$ that corresponds to the maximally violated constraint at the current master solution. So if a constraint is violated, we add this maximally violated constraint to Gen, tightening the MIP relaxation, and repeat; if not, we are assured that the current solution minimizes setwise max regret.[6]

In the formulation, m_{big} is an arbitrary big number, that we need to encode the fact that, for any given w, only the option with the highest utility (among those in the slate) with respect to w contributes to the actual setwise regret.

The SMR maximization subproblem can be also encoded with a MIP, similar to (Boutilier *et al.* 2006b). The optimization makes use of a decision variable to explicitly represent the setwise regret, M, to be maximized and we constrain M to be greater than the single max regret $MR(\mathbf{x}_j, W)$, for each option in the slate.

Database Problems: A Hill-climbing Strategy As discussed above, while minimax search can be applied directly to the problem of setwise minimax regret for database problems, scaling is sometimes a concern. We now present a heuristic hill-climbing strategy that scales much more effectively. We describe it in the context of database problems, but it can also be used directly for configuration problems.

The central idea is that is possible to modify a given recommendation set $\mathbf{Z}$ in such a way that setwise max regret cannot increase, and usually decreases until a high quality set is found. We define the *MMR-transformation* T to be a mapping that refines a recommendation set $\mathbf{Z}$ by partitioning the current feasible utility space W into $\{W^{\mathbf{Z}\to\mathbf{x}_i}\}, \forall \mathbf{x}_i \in \mathbf{Z}$, as discussed in Observation 1. In each partition we compute the *single* recommendation that has minimax regret in that region of utility space, and define the new set recommendation $T(\mathbf{Z})$ to be the collection of these (single) minimax-optimal recommendations.

Definition 2 *Define the* MMR-transformation $T : \mathbf{Z} \to \mathbf{Z}'$, *where* $\mathbf{Z} = \{\mathbf{x}_1, \dots, \mathbf{x}_k\}$, *to be* $T(\mathbf{Z}) = \{\mathbf{x}'_1, \dots, \mathbf{x}'_k\}$ *such that for all* $1 \le i \le k$*:*

$$\mathbf{x}'_i = \mathit{MMR\text{-}Opt}(W^{\mathbf{Z}\to\mathbf{x}_i})$$

We can show that T cannot increase setwise max regret.

Observation 2 *For any set recommendation* $\mathbf{Z}$ $\mathit{SMR}(T(\mathbf{Z})) \le \mathit{SMR}(\mathbf{Z})$

We use the MMR-transformation to define our heuristic search strategy to produce good recommendation sets; intuitively, we repeatedly T until a fixed point (with respect to setwise max regret, not the set itself) is found.

Alg 1 Hill-climbing-T algorithm (HCT)
The algorithm considers an initial set $\mathbf{Z}$*, and rewrites* $\mathbf{Z}$ *using* T *until a fixed-point is found.*

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

We initialize the slate $\mathbf{Z}$ using the current solution strategy (CSS), empirically, this seems to produce the most promising recommendation sets. For $k = 2$, this means that the initial set is $\mathbf{Z} = \{x^*_W, x^w\}$, where $x^* = \mathit{MMR\text{-}Opt}(W)$, and $x^w = \mathit{MRAdv}(W)$.

For larger sets ($k > 2$), there is not a standard definition of the CSS. We propose to use the following strategy, that we call *chain of adversaries*, to generate the initial slate. We start from $\{x^*_W, x^w\}$ and repeatedly maximize setwise max regret given the current set, in some sense maximizing the diversity of choices from perspective of utility space. This gives the set $\{\mathbf{x}^1, \dots, \mathbf{x}^k\}$ where:

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

The chain of adversaries requires to solve single minimax regret once, and then $k - 2$ setwise regret maximizations. The chain of adversaries can be seen as a generalization of CSS to sets of any size, and could also be considered as an alternative, faster strategy to select recommendations.

[5] $V^j_{\mathbf{w}}$ is the actual regret of option $\mathbf{X}^j$ of the slate with respect to the utility $\mathbf{w}$ when the corresponding $I^j_{\mathbf{w}}$ is activated. For any $\mathbf{w}$, one and only one $I^j_{\mathbf{w}}$ is set to 1. In order to minimize M, the optimization will activate the $I^j_{\mathbf{w}}$ corresponding to the $\mathbf{x}^j$ with lower actual regret. The first constraint (10) captures the idea that for a slate of options, given $\mathbf{w}$, the regret of the joint slate is the minimum among the individual values of regret (in the summation, all but one term are zeros).

[6] 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 variables I and V are necessary.

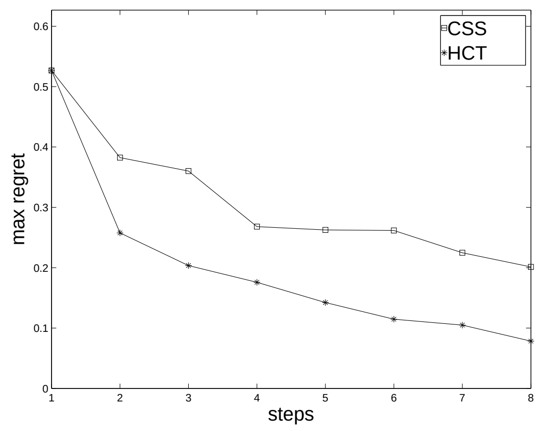

Figure 3: The hillclimbing strategy based on setwise regret outperforms the current solution strategy in this experiment (20 runs).

Myopic Elicitation

In addition to produce final recommendations, our criterion can also be used as a driver for further elicitation of the utility function of the user. In fact, whenever we consider a slate of recommendations, the user may give us some feedback, perhaps selecting the option that she prefers among those in the set. This information is very valuable, and as we have seen in the initial example, can be used to reduce regret. It is therefore interesting to assess the value of a recommendation set also with respect to the possible feedback.

An important observation is that in the case of comparison queries (the user selects the preferred option in a slate), the set of k optimal recommendations that minimize setwise regret is also the optimal choice set for a comparison query with respect to *myopic worst case regret* (WR), a measure of the value of information of a query.

The *Worst-case Regret* (WR) of a comparison query based on a choice set $\mathbf{Z} = \{\mathbf{x}_1, .., \mathbf{x}_k\}$ is defined as

$$WR(\{\mathbf{x}_1, .., \mathbf{x}_k\}) = max[MMR(W^{Z\to\mathbf{x}_1}), .., MMR(W^{Z\to\mathbf{x}_k})] \tag{16}$$

WR considers the "single" max regret in each possible scenario. It is possible to verify that $WR(\mathbf{Z}) \leq SMR(\mathbf{Z})$; the worst case regret is always lower (or equal) than the setwise max regret.

The optimality of minimax setwise recommendations (we omit the full proof for reasons of space) with respect to WR is based on the consideration (an extension of Observation 2) that the transformation T introduced in the previous section is also such that $SMR(T(\mathbf{Z})) \leq WR(\mathbf{Z})$ (the proof requires considering the different partitions imposed by Observations 1 and compare the two expression componentwise). We call $\mathbf{Z}^*$ the optimal recommendation set according to setwise regret. A set $\mathbf{Z}'$ such that $WR(\mathbf{Z}') < WR(\mathbf{Z}^*)$ but $SMR(\mathbf{Z}') > SMR(\mathbf{Z}^*)$ leads to contradiction.[7]

We performed some preliminary experiments in order to evaluate our recommendation strategy from the prospective of elicitation. We are interested in quantifying the reduction of regret in practice. In Figure 3 we compare the efficiency of an elicitation based on our SMR criterion and the current solution strategy (CSS), considering a syhnthetic dataset with 5000 options and 10 attributes. We plot max regret in function of the number of queries (steps). SMR is optimized using the hill-climbing strategy (HCT).

Example Critiquing

As an evaluation setting, we apply our regret-based recommender to example-critiquing. This domain is interesting because current systems usually rely on heuristics and we expect that an utility-based approach can be greatly beneficial.

Critiquing is a setting where the user expresses feedback on options that the system shows to her. In particular we consider a particular version of critiquing, often called the *dynamic critiquing* model (Reilly *et al.* 2005), where a *current* product or recommendation is displayed, and the user is invited to move to a different product by choosing particular actions (laid out in the interface) that change the product. They can include *unit critiques*, which request modification of a particular product attribute; e.g., "give me a laptop that is *lighter* than the current one."

Often alternative *suggestions* or *compound critiques* are used in which multiple attributes ("lighter and faster processor, but more expensive") are tweaked, or in which a selection is made from a system-suggested set of alternative products ("let me see laptop 3 instead of the current one").

The set of possible critiques is generated by the system, and the user chooses one of the possible actions. At each interaction, the user may choose to critique the current product if she is not completely satisfied with it, or simply because she wishes to explore the product space in more depth.

In general those systems use heuristics to generate the set of possible critiques. However, we expect that better performance can be obtained if critiquing suggestions are selected according to a decision-theoretically sound criterion as our setwise minimax regret.

In order to implement our approach, it is necessary to give a precise semantics to each of the critiquing actions. We identify two main reasons a user will critique an option. First, she may want to explore the product space in an effort to better understand either the space of feasible options or her own preferences. This latter desire makes sense especially when one adopts the view commonly held in behavioral economics that decision support systems should help people *construct* their preferences (not just articulate them) (Slovic 1995). Second, she may wish to improve the current product, making tradeoffs among her preferences for different attributes. It is this latter *exploitive* or *improvement mode* that critiquing systems fail to account for adequately when deciding on appropriate product suggestions. In this evalua-

[7]If we apply the transformation T to $\mathbf{Z}'$ we obtain a set $\bar{\mathbf{Z}}$ such that $SMR(\bar{\mathbf{Z}}) \leq WR(\mathbf{Z}') < WR(\mathbf{Z}^*) \leq SMR(\mathbf{Z}^*)$ but this means that $SMR(\bar{\mathbf{Z}}) < SMR(\mathbf{Z}^*)$, contradicting the optimality of $\mathbf{Z}^*$ with respect to SMR.

tion we use critiques of the latter type to constrain the set of possible user utility functions.

In the following, we describe our simulation setting and present our results.

Experiments

To validate our regret-based approach to critiquing we designed a framework that simulates a full interaction of a user with a user interface. As in a real system, each simulation comprises a number of cycles of interaction, each showing a current product which the user can critique using either unit or the selection of one of the suggested recommendations.

The simulated user continues the critiquing process until the perceived increase in utility is lower than some threshold. We assume that among all possible critiquing actions, the one with highest perceived improvement will be chosen by the user.

In our experiment, at each interaction the system displays:

- the *current product*,
- a choice of *unit critiques* of the current product (they request the modification of a particular product attribute),
- a set of *suggestions*, alternative options that can change focus for the search, presented as such or labeled as *compound* critiques ("lighter and faster processor, but more expensive")

At each step, the user can choose to either select the current product (and finish the interaction) or to tweak it in order to improve it and get better recommendations in the next cycle.

We compare our regret-based approach to three other approaches that use compound critiques. In our case, we use the generation of a set of recommendations (based on setwise regret) to display as alternatives. One is selected as *current product* and the others are displayed as *suggestions*.

We briefly review the different critiquing approaches and then we present the experimental results.

Dynamic Critiquing

The *dynamic critiquing* model (Reilly *et al.* 2004) makes use of a particular similarity metric to retrieve the current product and uses the APriori datamining algorithm to propose alternative compound critiques. The algorithm dynamically generates compound critiques by discovering common feature patterns among the set of products. Essentially, each compound critique describes a set of products in terms of the features they have in common. For example in the PC domain, a typical compound critique might be "Faster CPU and a Larger Hard Drive." Whenever a product is shown to the user as the current product, the APriori datamining algorithm is used to quickly discover these patterns and convert them into a set of suggested compound critiques. Each compound critique corresponds to a product that is, among all products satisfying the pattern most similar to the current one.

The generation of suggestions consists of two steps. First, each product is matched against the current product to produce lists of *critique patterns*, each comprising an attributes and a comparison operators from the set: $<$,$>$,$\neg$,$=$. An example pattern might be: $\{$[Price $>$], [ProcessorSpeed $>$]$\}$. Second, the algorithm uses APriori to find recurrent critiquing patterns; a compound critique based on a pattern is then presented to the user it has sufficient support in the product database. In our experiments, the support threshold is set to 0.3 and selection of compound critiques corresponds to the *low-support* strategy in (Reilly *et al.* 2004).

Incremental Critiquing

Incremental critiquing (Reilly *et al.* 2005) (IC) improves the basic dynamic critiquing model by incorporating a user model. While suggestions are still based on the APriori algorithm (as above), the retrieval of the next product associated with a critiquing action is based on a *quality metric* that values both the $score$ given to the product by the preference model and its similarity to the current product.

In the implementation we developed for our experiments, we take advantage of the fact that the preference ordering over attributes is known: the score is dictated by a linear utility function that gives equal weight to all attributes. The initial product is the option with maximum utility. When retrieving the next example from the set of products that satisfy the user-chosen critique, we select the product $\mathbf{x}$ that maximizes $score(\mathbf{x}) \cdot Similarity(\mathbf{x}, \mathbf{y})$, where $\mathbf{y}$ is the product recommended at the previous cycle, and $score$ is the heuristic utility function.

Incremental Critiquing: MAUT

Another implementation of incremental critiquing (Reilly *et al.* 2007) uses a simple multi attribute utility (MAUT) model to make recommendations and generate compound critiques (rather than similarity). 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 multiplied by a constant (and all weights renormalized). The original design of this algorithm makes use of parameterized value functions for each attribute, where the value taken by the current option is considered preferred. Since our experimental set up assumes that the local preference ordering over attribute values is known, we instead assume a linear utility model.

Suggestions are generated using optimization with respect to the estimated utility model, and the k best products are presented as alternative cases. A limitation of this approach is its reliance on a fixed utility model (as opposed to reasoning with the space of possible user utilities). Moreover, options that *all* have high value in a single utility sample are unlikely to be diverse or informative enough to generate useful distinctions.

Regret-based critiquing

Our version of dynamic critiquing exploits setwise minimax regret using the ideas above. Specifically, at any point in the interaction cycle, we generate the current optimal recommendation set (with respect to minimax setwise regret), and propose one of these options a current product. The remainder of the set is used to display suggestions.

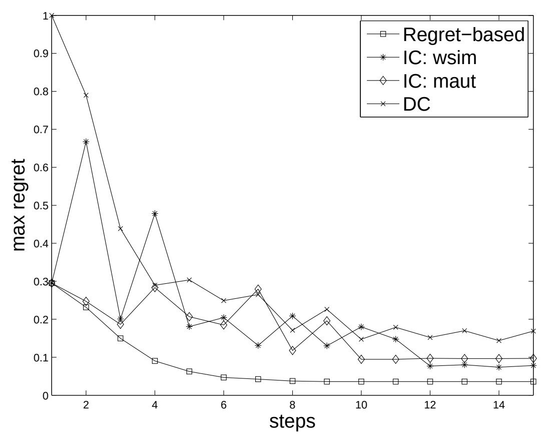

Figure 4: Maximum regret of the recommended option at each step for four algorithms: regret-based critiquing (regret), dynamic critiquing with compound critiques generated with APriori (DC), incremental critiquing with weighted similarity and APriori (IC) and incremental critiquing with a multiattribute utility model (IC maut).

Empirical Results

In the experiments we compare the four different versions of dynamic critiquing discussed above: the original dynamic critiquing algorithm (similarity plus APriori), incremental critiquing, incremental critiquing with MAUT, and regret-based critiquing. We used a C implementation of the APriori algorithm (Bodon 2003). All systems make available unit critiques of any attribute (and user's adopt an expected improvement semantics). We evaluate the performance of all algorithms with respect to recommendation efficiency and offer some speculative examination of regret-based critiquing in terms the tradeoff between cognitive cost and number of compound critique options presented at each interaction.

We evaluate the different critiquing methods by comparing the quality of the recommendations with respect to max regret. We tested the methods on a real database of 200 apartments, using randomly drawn utility functions (as described above), and $k = 3$ suggested products at each interaction cycle. All results are averaged over 20 simulated users. Fig. 4 shows the maximum regret of the recommended product at each stage of the interaction. We note that regret-based critiquing outperforms the other methods of generating compound critiques by a wide margin. This is true when considering both the "anytime" profile of the method (i.e., the degree to which minimax drops) and its final convergence: our technique converges on a product who max regret is about 3% on average, while the MAUT incremental critiquing settles at about 10% (and the others worse, with dynamic critiquing unable to reduce max regret to less than 18%).

More interesting is the fact that regret-based critiquing offers better "actual" recommendations, as measured by true regret (difference from the true optimal recommendation). Regret-based critiquing is designed to attack *bounds* on re-

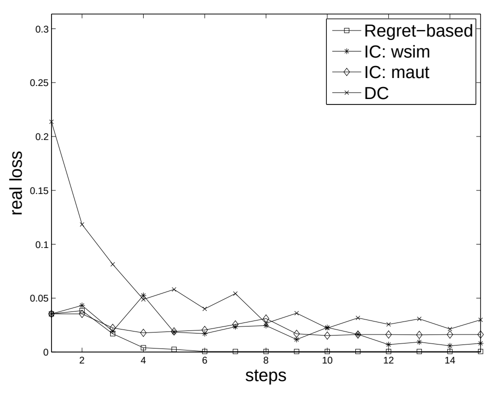

Figure 5: True regret (real loss) of the recommended option at each step for four algorithms: regret-based critiquing (regret), dynamic critiquing with compound critiques generated with APriori (DC), incremental critiquing with weighted similarity and APriori (IC) and incremental critiquing with a multiattribute utility model (IC maut).

gret (i.e., worst-case loss); so one might wonder whether other techniques find better products despite being unable to "prove" that they are good. Fig. 5 shows this not to be the case. While other critiquing techniques recommend products that are much better than their regret-bounds suggest, regret-based critiquing is able to consistently find the optimal product (and find a near-optimal product in as few as five or six interaction cycles). By contrast, the other three methods are unable to identify the optimal option at convergence.

Conclusions

In this paper we presented a novel formalization of recommendations of a joint set of alternatives based on the notion of regret. The criterion that we propose, setwise max regret, represents an intuitive extension of the traditional regret criterion for single recommendations.

We show how optimal recommendation sets (with respect to our criterion) can be computed with mixed integer programming (MIP) methods and the constraint generation technique when options are constructed from a set of configuration constraints. Alternatively, set recommendations can be obtained using a hill-climbing strategy interleaved with adversarial search in discrete settings.

We discuss the problem of utility elicitation, showing that our recommendation strategy reduces max regret more quickly than any other possible choice. Finally we present an application of these principles for critiquing systems.

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 critiquing recommender can often lead to optimal recommendations using very few, say, compound critiquing interactions, and outperforms other dynamic critiquing techniques both in speed of convergence and the quality of the final recommendations.

The incorporation of noisy feedback is an important next step; we are currently considering the possibility of a clarification dialogue. The idea is to verify information that is sensitive with respect to regret.

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 currently developing hybrid models in which the system and/or user explicitly distinguishes exploratory actions from improving actions. Even with such a distinction, there is still the interesting question of modeling user *search processes* in a way that would allow insight into preferences to be drawn during exploration as well. Finally, the development of models of cognitive costs using techniques from behavioral economics, decision theory and psychology remains an important avenue of future research.

References

Fahiem Bacchus and Adam 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.

Ferenc Bodon. A fast apriori implementation. In Bart Goethals and Mohammed J. Zaki, editors, *Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations (FIMI'03)*, volume 90 of *CEUR Workshop Proceedings*, Melbourne, Florida, USA, 19. November 2003.

Craig Boutilier, Fahiem Bacchus, and Ronen I. Brafman. UCP-Networks: A directed graphical representation of conditional utilities. In *Proceedings of the Seventeenth Conference on Uncertainty in Artificial Intelligence (UAI-01)*, pages 56–64, Seattle, 2001.

Craig Boutilier, Richard S. Zemel, and Benjamin Marlin. Active collaborative filtering. In Christopher Meek and Uffe Kjærulff, editors, *UAI*, pages 98–106. Morgan Kaufmann, 2003.

Craig Boutilier, Tuomas Sandholm, and Rob 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.

Craig Boutilier, Relu Patrascu, Pascal Poupart, and Dale Schuurmans. Constraint-based optimization and utility elicitation using the minimax decision criterion. *Artifical Intelligence*, 170(8–9):686–713, 2006.

Craig Boutilier, Relu Patrascu, Pascal Poupart, and Dale Schuurmans. Constraint-based optimization and utility elicitation using the minimax decision criterion. *Artif. Intell.*, 170(8-9):686–713, 2006.

Craig 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.

Darius Braziunas and Craig Boutilier. Preference elicitation and generalized additive utility. In *AAAI*, 2006.

Darius Braziunas and Craig 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.

Darius Braziunas and Craig 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.

Urszula Chajewska and Daphne Koller. Utilities as random variables: Density estimation and structure discovery. In *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence (UAI-00)*, pages 63–71, Stanford, 2000.

Urszula Chajewska, Daphne Koller, and Ronald 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.

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

Ralph L. Keeney and Howard Raiffa. *Decisions with Multiple Objectives: Preferences and Value Trade-offs*. Wiley, New York, 1976.

Panos Kouvelis and Gang Yu. *Robust Discrete Optimization and Its Applications*. Kluwer, Dordrecht, 1997.

David McSherry. Diversity-conscious retrieval. In *ECCBR '02: Proceedings of the 6th European Conference on Advances in Case-Based Reasoning*, pages 219–233, London, UK, 2002. Springer-Verlag.

Robert Price and Paul R. Messinger. Optimal recommendation sets: Covering uncertainty over user preferences. In *Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI'05)*, pages 541–548, 2005.

James Reilly, Kevin McCarthy, Lorraine McGinty, and Barry Smyth. Dynamic critiquing. In Peter Funk and Pedro A. González-Calero, editors, *ECCBR*, volume 3155 of *Lecture Notes in Computer Science*, pages 763–777. Springer, 2004.

James Reilly, Kevin McCarthy, Lorraine McGinty, and Barry Smyth. Incremental critiquing. *Knowl.-Based Syst.*, 18(4-5):143–151, 2005.

James Reilly, Jiyong Zhang, Lorraine McGinty, Pearl Pu, and Barry Smyth. Evaluating compound critiquing recommenders: a real-user study. In Jeffrey K. MacKie-Mason, David C. Parkes, and Paul Resnick, editors, *ACM Conference on Electronic Commerce*, pages 114–123. ACM, 2007.

Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, Englewood Cliffs, NJ, 2nd edition edition, 2003.

Ahti Salo and Raimo 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.

Leonard J. Savage. *The Foundations of Statistics*. Wiley, New York, 1954.

Paul Slovic. The construction of preference. *American Psychologist*, 50(5):364–371, 1995.

Barry Smyth and Paul McClave. Similarity vs. diversity. In David W. Aha and Ian Watson, editors, *ICCBR*, volume 2080 of *Lecture Notes in Computer Science*, pages 347–361. Springer, 2001.