Robust Optimization of Recommendation Sets with the Maximin Utility Criterion

Paolo Viappiani, Christian Kroer

Robust Optimization of Recommendation Sets with the Maximin Utility Criterion

Paolo Viappiani[1] and Christian Kroer[2]

[1] CNRS-LIP6 and Univ. Pierre et Marie Curie, France
paolo.viappiani@lip6.fr
[2] Carnegie Mellon University, USA
ckroer@cs.cmu.edu

Abstract. We investigate robust decision-making under utility uncertainty, using the *maximin* criterion, which optimizes utility for the worst case setting. We show how it is possible to efficiently compute the maximin optimal recommendation in face of utility uncertainty, even in large configuration spaces. We then introduce a new decision criterion, *setwise maximin utility (SMMU)*, for constructing optimal recommendation sets: we develop algorithms for computing SMMU and present experimental results showing their performance. Finally, we discuss the problem of elicitation and prove (analogously to previous results related to regret-based and Bayesian elicitation) that SMMU leads to myopically optimal query sets.

1 Introduction

Reasoning about preferences [9] is an important component of many systems, including decision support and recommender systems, personal agents and cognitive assistants. Because acquiring user preferences is expensive (with respect to time and cognitive cost), it is essential to provide techniques that can reason with partial preference (utility) information, and that can effectively elicit the most relevant preference information. Adaptive utility elicitation [3] tackles the challenges posed by preference elicitation by representing the system knowledge about the user in the form of *beliefs*, that are updated following user responses. Elicitation queries can be chosen adaptively given the current belief. In this way, one can often make good (or even optimal) recommendations with sparse knowledge of the user's utility function.

Since utility is uncertain, there is often value in recommending a *set* of options from which the user can choose her preferred option. Retrieving a "diverse" set of recommended options increases the odds that at least one recommended item has high utility. Intuitively, such a set of "shortlisted" recommendations should include options of high utility relative to a wide range of "likely" user utility functions (relative to the current belief) [10]. This stands in contrast to some recommender systems that define diversity relative to product attributes. "Top k" options (those with highest expected utility) do not generally result in good recommendation sets.

Recommendation systems can be classified according to the way they represent the uncertainty about the user preferences (encoded by an utility function) and how such uncertainty is aggregated in order to produce recommendations that are believed to have

high utility. A common approach [4, 10, 3, 14] is to consider a distribution over possible user preferences, and make recommendations based on expected utility. Another line of work [1–3, 13] assumes no probabilistic prior is available (the *strict* uncertainty setting) and provides recommendations using the minimax regret criterion. The latter approach makes robust recommendations; regret measures the worst-case loss in utility the user might incur by accepting a given recommendation instead of the true optimal option.

In this paper, we take an approach similar to the second setting; we cast decision-making as a problem of optimization under strict uncertainty and we produce robust recommendations based on the *maximin* utility criterion. Maximin [15, 11] is the most pessimistic decision criterion; the recommended decision or option is the one that leads to the highest utility in the worst case. It is a well-known concept and we believe it is worth studying it from an utility elicitation perspective. While we recognize that maximin might not be the right decision criterion in many circumstances due to its intrinsic pessimism, we argue it is apt for high-stakes decisions requiring the strongest guarantees. As in works on regret-based utility elicitation, in our setting the uncertainty over possible utility functions is encoded by a set of constraints (usually obtained through some form of user feedback, such as responses to elicitation queries of the type: *"Which of these products do you prefer ?"*). Differently from regret, the recommendation ensures that a certain level of utility is attained.

In order to provide recommendation sets that efficiently cover the uncertainty over possible user preferences, we define a new *setwise maximin* utility criterion, formalizing the idea of providing a set of recommendations that optimize the utility of the user-selected option in the context of our framework. We show how linear and mixed integer programming techniques can be used to efficiently optimize both singleton recommendations and sets in large configuration spaces, and more computationally efficient heuristic techniques motivated by our theoretical framework. Finally, we discuss the problem of interactive elicitation (which can be viewed as active preference learning) and how to identify queries that are myopically optimal with respect to a non-probabilistic analogue of value of information.

2 Decision-making with Maximin Utility

Much work in AI, decision analysis and OR has been devoted to effective elicitation of preferences [1, 4, 12]. Adaptive preference elicitation recognizes that good, even optimal, decisions can often be recommended with limited knowledge of a user's utility function [1]; and that the value of information associated with elicitation queries is often not worth the cost of obtaining it [3]. This means we must often take decisions in the face of an incompletely specified utility function.

These approaches all represent the uncertainty about the user's utility function explicitly as "beliefs". In the case of strict uncertainty (no probabilistic prior is available), the belief takes the form of a set of possible utility functions, usually implicitly encoded by constraints [1, 13]. In this work, we adopt the notion of *maximin utility* as our decision criterion for robust decision making under utility function uncertainty. Maximin utility (like minimax regret [1, 2, 13]) relies on relatively simple prior information in the form of bounds or constraints on user preferences.

2.1 Basic Setting

The setting is that of [13]: we consider a multi-attribute space as, for instance, the space of possible product configurations from some domain (e.g., computers, cars, apartment rentals, 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*. 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). Let $\mathbf{x} \in \mathbf{X}$ be a feasible configuration, and x_i the value of the i-th attribute.

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 [8]. In both additive and linear models, u can be decomposed as follows[3]:

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

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 [6, 8]. 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: $u(Apt) = f_1(Size) + f_2(Distance) + f_3(Nbrhd)$. [4]

Since a user's utility function is not generally known, we write $u(\mathbf{x}; w)$ to emphasize the dependence of u on parameters that are specific to a particular user. 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 $argmax_{\mathbf{x} \in \mathbf{X}} u(\mathbf{x}; w)$. The goal of a decision aid system is to recommend, or help the user find, an optimal, or near optimal, product.

2.2 Singleton Recommendations

Assume that using some prior knowledge, we determine that the user's utility function w lies in some bounded set W.[5] Such prior knowledge might be obtained through some interaction with a user (the exact form of W will be defined in section 3.1). We define:

Definition 1. *Given a set of feasible utility functions W, the* minimum utility $MU(\mathbf{x}; W)$ *of $\mathbf{x} \in \mathbf{X}$ is defined as:*

$$MU(\mathbf{x}; W) = \min_{w \in W} u(\mathbf{x}; w)$$

[3] In our notation, we use bold lowercase for vectors.

[4] Our presentation relies heavily on the additive assumption, though our approach is easily generalized to more general models such as GAI [6, 2].

[5] We assume that W is topologically closed. Otherwise one should substitue $\min$ and $\max$ with $\inf$ and $\sup$ in the definitions below.

Definition 2. *The* maximin utility $MMU(W)$ *of* W *and a corresponding* maximin optimal configuration $\mathbf{x}^*_W$ *are defined as follows:*

$$MMU(W) = \max_{\mathbf{x}\in\mathbf{X}} MU(\mathbf{x};W) = \max_{\mathbf{x}\in\mathbf{X}} \min_{w\in W} u(\mathbf{x};w)$$
$$\mathbf{x}^*_W = \arg\max_{\mathbf{x}\in\mathbf{X}} MU(\mathbf{x};W) = \arg\max_{\mathbf{x}\in\mathbf{X}} \min_{w\in W} u(\mathbf{x};w)$$

Intuitively, $MU(\mathbf{x};W)$ is the worst-case utility associated with recommending configuration $\mathbf{x}$; i.e., by assuming an adversary will choose the user's utility function $\mathbf{w}$ from W to minimize the utility. The maximin optimal configuration $\mathbf{x}^*_W$ is the configuration that maximizes this minimum utility. Any choice that is not maximin optimal has strictly lower utility than $\mathbf{x}^*_W$ for some $\mathbf{w} \in W$.

In problems where the items or choices are explicitly listed in a database, we can in principle iterate over all candidate items, compute their minimum utility (this requires solving a linear program defined in Section 3.1), and pick the item with the highest value for recommendation. In configuration problems, the product space $\mathbf{X}$ is formulated as a constraint satisfaction problem (CSP) or mixed integer program (MIP). In Section 3.1 we show how computing maximin utility in configuration domains can be formulated as a mathematical programming problem and solved using techniques such as Bender's decomposition and constraint generation, adapting techniques developed for minimax regret optimization [1, 2].

2.3 Recommendation Sets: the Setwise Maximin Utility Criterion

Suppose we wish to pick a subset $\mathbf{Z} \subseteq \mathbf{X}$ of size k to present to the user and want to quantify the minimum utility obtained by restricting the user's decision to options in that set. In the maximin utility criterion, we choose the set of k options first, and then the adversary picks the utility function $\mathbf{w}$ such that it minimizes the utility of the best of the k options. We assume $\mathbf{Z}$ is restricted to subsets of $\mathbf{X}$ of cardinality k without making this explicit. In practical circumstances, constraints on the user interface design might lead to the choice of k.

Definition 3. *Let* W *be a feasible utility set,* $\mathbf{Z} \subseteq \mathbf{X}$. *Define:*

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

The *setwise minimum utility(SMU)* of a set $\mathbf{Z}$ of k options reflects the intuitions above. *Setwise maximin utility (SMMU)* is SMU of the minimax optimal set $\mathbf{Z}^*_W$, i.e., the set that maximizes $SMU(\mathbf{Z}, W)$.

Setwise maximin utility has some intuitive properties. SMU is monotone with respect to set inclusion: adding new items to a recommendation set cannot decrease SMU (Observation 1). Incorporating options that are known to be dominated given W does not change setwise maximin utility (Observation 2).

Observation 1. $SMU(\mathbf{A} \cup \mathbf{B}; W) \geq SMU(\mathbf{A}; W)$.

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

Observation 3. *MU and SMU can be explicitly expressed as the minimization over different utility spaces*

$$MU(\mathbf{A}; W_1 \cup W_2) = \min\{MU(\mathbf{A}; W_1), MU(\mathbf{A}; W_2)\}$$
$$SMU(\mathbf{A}; W_1 \cup W_2) = \min\{SMU(\mathbf{A}; W_1), SMU(\mathbf{A}; W_2)\}$$

The computation of SMU is made with respect to the item $\mathbf{x} \in \mathbf{Z}$ with highest utility, when the utility value is computed according to $\mathbf{w} \in W$. Due to this, the different choices of $\mathbf{x} \in \mathbf{Z}$ define a partition of the utility space, where a partition with respect to a given $\mathbf{x}$ is the region of W where the utility of $\mathbf{x}$ is highest among the options in $\mathbf{Z}$. More formally,

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

That is, $W[\mathbf{Z} \to \mathbf{x}_i]$ is the region of $\mathbf{w}$ where the utility of $\mathbf{x}_i$ is at least as high as any other option in $\mathbf{Z}$. (the regions $W[\mathbf{Z} \to \mathbf{x}_i]$, $\mathbf{x}_i \in \mathbf{Z}$, partition W if one ignores ties). We call this the $\mathbf{Z}$*-pseudo-partition*[6] *of* W. Using the $\mathbf{Z}$-pseudo-partition, we can rewrite SMU (this will be useful for optimization):

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

$$SMU(\mathbf{Z}, W) = \min_{\mathbf{x} \in \mathbf{Z}} \min_{w \in W[\mathbf{Z} \to \mathbf{x}]} u(\mathbf{x}, w) = \min_{i=1 \leq \ldots \leq k} MU(\mathbf{x}_i; W[\mathbf{Z} \to \mathbf{x}_i])$$

We use a similar notation to express the combination of two partitions: $W[\mathbf{Z}_1 \to \mathbf{x}_i, \mathbf{Z}_2 \to \mathbf{x}_j] = W[\mathbf{Z}_1 \to \mathbf{x}_i] \cap W[\mathbf{Z}_2 \to \mathbf{x}_j]$.

We introduce a transformation that modifies a given recommendation set $\mathbf{Z}$ in such a way that SMU cannot decrease and usually increases. This will be used as a heuristic for efficiently generating recommendation sets. It will also be useful when discussing elicitation. Define the transformation T to be a mapping that updates a given recommendation set $\mathbf{Z}$ in the following way: (a) First we construct the $\mathbf{Z}$-pseudo-partition of W; (b) we then compute the *single recommendation* that has maximin utility in each region of the pseudo-partition of W; (c) finally, we let $T(\mathbf{Z})$ be the new recommendation set consisting of these new recommendations. Note that $T(\mathbf{Z})$ may have cardinality less than $|Z| = k$.

Definition 4. *Let* $\mathbf{Z} = \{\mathbf{x}_1, \ldots, \mathbf{x}_k\}$. *We define* $T(\mathbf{Z}) = \{\mathbf{x}^*_{W[\mathbf{Z} \to \mathbf{x}_1]}, \ldots \mathbf{x}^*_{W[\mathbf{Z} \to \mathbf{x}_k]}\}$

We will discuss optimization of a recommendation set below in Section 3.2.

3 Maximin Utility Optimization

In this section we formalize the problem of generating recommendations (both single recommendations and setwise recommendations) using mathematical programming techniques (linear programming models and mixed integer programming models). These optimization techniques are adaptation of techniques previously proposed [1, 13] for minimax regret. We note that maximin is faster to compute than minimax regret.

[6] The definition of the $\mathbf{Z}$-partition first appeared in [13], in the context of recommendations based on the minimax regret crierion.

3.1 Optimization of Singleton Recommendations

In the following we assume the utility to be linear in w: $u(\mathbf{x}; w) = w \cdot \mathbf{x}$. In this case W is a convex polytope effectively represented by a set of constraints (whenever the user answers a query, new constraints are added) that we denote with *Constraints(W)*.

MU(x, W) Given configuration $\mathbf{x}$ and the space of possible utility functions W (encoded by linear constraints), the minimum utility of x can be found by minimizing the function $\mathbf{w} \cdot \mathbf{x} = \sum_{1 \leq i \leq n} x_i \cdot w_i$, subject to *Constraints(W)*, and $w_i^{\perp} \leq \mathbf{w}_i \leq w_i^{\top}$ for all $i \in \{1 \ldots n\}$, which is solvable by linear programming.

MMU(W) Given the possible utility functions W (encoded by linear constraints), the problem is to find the configuration $\mathbf{x}_W^*$ that is associated with maximin utility. In order to "break" the maximin optimization, we make use of Benders decomposition:

$$\begin{aligned} &\max_{\mathbf{x},\delta} \quad \delta \\ &\text{s.t. } \delta \leq \mathbf{w} \cdot \mathbf{x} \qquad\qquad \forall \mathbf{w} \in GEN \end{aligned} \tag{1}$$

In this model, δ corresponds to the *maximin utility* of the optimal recommendation $\mathbf{x}_W^*$. Constraint 1 ensures that δ is less than the utility of choice $\mathbf{x}$ for each $\mathbf{w}$. The optimization is exact when $GEN = W$ in constraint 1. However, all the constraints over W need not be expressed for each of the (continuously many) $\mathbf{w} \in W$. Since maximin utility is optimal at some vertex of W, we only need to add constraints for all vertices $Vert(W)$ of W, but they can still be exponential. We apply constraint generation in order to solve the MIP much more efficiently, as very few of the vertices are usually needed. This procedure works by solving a relaxed version of the problem above—the *master problem*— using only the constraints corresponding to a small subset $GEN \subset Vert(W)$. We then test whether any constraints are violated in the current solution. This is accomplished by computing the *minimum utility* of the returned solution (the *slave problem*). If MU is lower than what was found in the master problem, a constraint was violated. The constraint (corresponding to the choice w^a of the adversary) is added to the master problem, tightening the MIP relaxation. The master problem is recomputed, and this process is repeated until no violated constraint exists.

3.2 Optimization of Recommendation Sets

SMU(Z, W) Given a set Z and a space of possible utility functions W, by observation 4 the setwise minimum utility of Z can be found by solving k (k being the cardinality of Z) optimization problems. Using the Z-partition of W, we compute $MU(\mathbf{x}, W[\mathbf{Z} \rightarrow \mathbf{x}])$ for each $\mathbf{x} \in \mathbf{Z}$, using the LP model shown above. We then take the (arithmetic) minimum of the results: $\min_{x \in \mathbf{Z}} MU(\mathbf{x}, W[\mathbf{Z} \rightarrow \mathbf{x}])$.

SMMU(W) Given utility space W, we can compute the maximin optimal set (of cardinality k) using the following MIP.

$$\begin{aligned}
&\max \quad \delta \\
&\text{s.t. } \delta \leq \sum_{1 \leq j \leq k} v_w^j && \forall \mathbf{w} \in GEN && (2)\\
&v_w^j \leq \mathbf{w} \cdot \mathbf{x}^j && \forall j \leq k, w \in GEN && (3)\\
&v_w^j \leq w^\top I_w^j && \forall j \leq k, w \in GEN && (4)\\
&\sum_{1 \leq j \leq k} I_{\mathbf{w}}^j = 1 && \forall \mathbf{w} \in GEN && (5)\\
&I_{\mathbf{w}}^j \in \{0,1\} && \forall j \leq k, \mathbf{w} \in GEN
\end{aligned}$$

Decision variables: $\mathbf{x}^j$, δ, $\mathbf{I_w}$, $\mathbf{v}_w$

In this model, δ corresponds to the *setwise maximin utility* of the optimal set $\mathbf{Z}_W^*$. $w^\top$ is some upper bound on the values taken by the weight parameters. Constraints 2, 3 and 4 ensures that δ is less than the utility of the best option in $\{\mathbf{x}^1, ..., \mathbf{x}^k\}$ for each $\mathbf{w}$, by introducing a variable v (for each w and each element of the set) to represent the value of minimum utility for the item selected, and indicators $\mathbf{I}_w$ to represent the selection. Only one $\mathbf{v}_w$ will be different from zero for each w, and since the objective function is maximized, the optimization will set $v_w^j = \mathbf{w} \cdot \mathbf{x}^j$ for the j such that $I_w^j = 1$; constraint 4 enforces 0 in the other cases. Constraint 5 ensures that only one of the k items is selected for each utility function w.

We employ constraint generation in a way analogous to the single item case. At each step of the optimization, we compute the *setwise minimum utility*, solved using a series of LPs (as discussed above).

Alternative Heuristics Setwise optimization requires solving a large number of MIPs using constraint generation strategies. We present a number of heuristic strategies that are computationally less demanding.

- The *current solution strategy* (CSS) proceeds as follows. Consider $\mathbf{w}^a$, the adversary's utility parameters minimizing the utility of $\mathbf{x}_W^*$, the current maximin optimal recommendation; $u(\mathbf{x}_W^*; \mathbf{w}^a) = MU(\mathbf{x}_W^*; W)$. Let's further consider $\mathbf{x}^a = \arg\max_{\mathbf{x} \in \mathbf{X}} u(\mathbf{x}; w^a)$. CSS will return the set $\mathbf{Z}_{CSS} = \{\mathbf{x}_W^*, \mathbf{x}^a\}$. We extend this to sets with cardinality greater than two. Considering a set $\mathbf{Z}$, define $\mathbf{w}^a(\mathbf{Z}) = \arg\min_{\mathbf{w} \in W} \max_{\mathbf{x} \in \mathbf{Z}} u(\mathbf{x}; w)$ and $\mathbf{x}^a(\mathbf{Z}) = \arg\max_{\mathbf{x} \in \mathbf{X}} u(\mathbf{x}; \mathbf{w}^a(\mathbf{Z}))$. We start by initializing $\mathbf{Z}$ to be $\mathbf{Z}_{CSS}$, the set of size two returned by the current solution strategy, and then iteratively add one element ($k - 2$ times) by setting $\mathbf{Z} := \mathbf{Z} \cup \mathbf{x}^a(\mathbf{Z})$.
- The *query iteration strategy* (QIS) directly applies the T operator until a fixed point is reached. A fixed point is such that $SMU(T(\mathbf{Z}); W) = SMU(\mathbf{Z}; W)$. We start QIS with the solution found by CSS.

3.3 Evaluation of Optimization Strategies

Using randomly generated elicitation data we ran a number of experiments using the algorithms described above. For all experiments, we generated constraints on the pos-

		Computation time			Setwise utility		
n. of features	set size	exact SMMU	CSS	QIS	exact SMMU	CSS	QIS
5	1	0.1s	n.a.	n.a.	0.188	n.a.	n.a.
5	2	0.1s	0.1s	0.2s	0.349	0.321	0.347
5	3	0.1s	0.1s	0.3s	0.366	0.356	0.366
5	4	0.2s	0.2s	0.4s	0.366	0.363	0.366
5	5	0.2s	0.2s	0.5s	0.366	0.366	0.366
10	1	0.1s	n.a.	n.a.	0.218	n.a.	n.a.
10	2	0.2s	0.1s	0.3s	0.375	0.305	0.332
10	3	0.3s	0.1s	0.3s	0.389	0.369	0.376
10	4	0.5s	0.2s	0.5s	0.391	0.385	0.385
10	5	-	0.3s	0.6s	-	0.392	0.392
15	1	0.1s	n.a.	n.a.	0.213	n.a.	n.a.
15	2	0.5s	0.1s	0.4s	-	0.268	0.290
15	3	-	0.2s	0.5s	-	0.314	0.322
15	4	-	0.2s	0.7s	-	0.359	0.368
15	5	-	0.3s	0.8s	-	0.371	0.375

Table 1. Computation times and utility values for our strategies (averaged over 30 instances). On the rightmost columns, setwise minimum utility values of the sets retrieved by each strategy. A dash means that one or more instances for this set of parameters timed out. Note that for set size 1, all three approaches reduce to MMU in this case.

sible options using random binary constraints of the form $\neg f_1 \vee \neg f_2$ where f_1 and f_2 are features. We also assume some prior knowledge of user preferences, represented by random utility constraints of the form $\mathbf{w} \cdot \mathbf{x_k} \geq \mathbf{w} \cdot \mathbf{x_l}$, where $\mathbf{x_k}$ and $\mathbf{x_l}$ are random assignments $\in [0,1]^m$ (not necessarily feasible options) sampled with uniform probability over all possible assignments. The user's preference values $w_1 \ldots w_n$ are random and normalized such that $\sum_i w_i = 1$. All experiments were run on a laptop with a 2.5GHz Core 2 Duo processor and 2GB ram, with all mathematical programs solved with CPLEX, version 12.2.

First, we ran experiments to determine how the runtime of the algorithms are affected by increasing the problem size (number of features) and the size (cardinality) of the recommendation sets. This was done by running the algorithms on instances ranging from 5 to 15 features, with 30 experiments performed on each size. [7] As seen in Table 1, runtime of exact SMMU computation becomes rapidly higher, and we were unable to perform experiments with more than 15 features, as several of the 30 experiments per size would time out. In contrast to this we see that the runtime of the CSS and QIS algorithms increase much more gradually. In another experiment, we compared the average runtimes of our strategies when fixing the set size, and varying the number of features in the problem domain (Figure 1).

We then investigated whether the computational effort is worth it in terms of utility increase. In the rightmost column of Table 1, we show how our strategies perform on different problem settings. Showing a set of items, instead of a single top item, is very beneficial: the minimum utility roughly doubles with five items instead of a single one. Moreover, the set-wise utility values of our approximate strategies (CSS and QIS) are very close to optimal SMMU.

[7] For comparison, we include the "degenerate" case of set size equal 1, corresponding to retrieving the single best recommendation according to maximin.

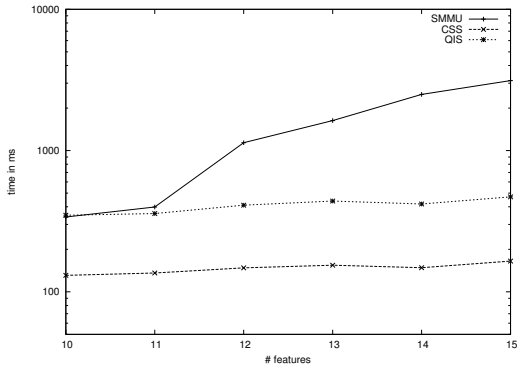

Fig. 1. Average runtime of maximin utility optimization for an increasing number of features. Averaged over 30 instances per size, with set size k = 3.

4 Utility Elicitation

Usually, utility information is not readily available, but must be acquired through an elicitation process. Since elicitation can be costly, it is important to ask queries eliciting the most valuable information. Our setwise 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 many recommender systems. And unlike recent work in polyhedral conjoint analysis [12], which emphasizes volume reduction of the utility polytope W, our maximin utility-based criterion is sensitive to the range of feasible products and does not reduce utility uncertainty for its own sake.

4.1 Optimal Myopic Elicitation

In general, there is a tension between recommending the best options to the user, and acquiring informative feedback from the user. While in the recommendation task the goal is to retrieve the best possible options to show to an user, in the elicitation task the objective is to identify candidate queries with high information value, so that better recommendations can be made when the user's response is incorporated in the model. The two tasks, recommendation and elicitation, considered separately in classic decision theory, are interleaved in decision aid tools such as conversational recommender systems where the user is in control.

Here, we consider *choice queries* requiring a user to indicate which choice/product is preferred from a set of k options. Hence, we can view any set of products as either a recommendation set or query (or choice) set. Given a set, one can evaluate the value of the set as a recommendation set and as a query set. Recently, Viappiani and Boutilier [14, 13] showed how these two problems are connected to each other, under both a Bayesian framework and when using *minimax regret*. In the following we show the same connection with maximin utility used as criterion.

Any set $\mathbf{Z}$ can be interpreted as a choice query: 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 $u(\mathbf{x}_i; \mathbf{w}) > u(\mathbf{x}_j; \mathbf{w}), j \neq i$.

When treating $\mathbf{Z}$ as a choice set (as opposed to a recommendation set), we are not interested in its maximin utility, but rather in *how much a query response will increase maximin utility*. In our distribution-free setting, the most appropriate measure is *posterior maximin utility*, a measure of the value of information of a query. We define:

Definition 5. *The* worst case posterior maximin utility (WP) *of* $\mathbf{Z} = \{\mathbf{x}_1, \ldots, \mathbf{x}_k\}$ *is*

$$WP(\mathbf{Z}, W) = \min[MMU(W[\mathbf{Z} \to \mathbf{x}_1]), \ldots, MMU(W[\mathbf{Z} \to \mathbf{x}_k])]$$

which can be rewritten as: $WP(\mathbf{Z}, W) = \min_{\mathbf{x} \in \mathbf{Z}} \max_{\mathbf{x}' \in \mathbf{X}} \min_{\mathbf{w} \in W[\mathbf{Z} \to \mathbf{x}]} u(\mathbf{x}', \mathbf{w})$. *An* optimal query set *is any* $\mathbf{Z}$ *that maximizes worst case posterior maximin utility* $MaxWP(W) = \max_{\mathbf{Z} \subseteq \mathbf{X}} WP(\mathbf{Z}, W)$

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 (the updated W with lowest maximin utility). The optimal query is the query that maximizes this value. We observe:

Observation 5. $WP(\mathbf{Z}; W) \geq SMU(\mathbf{Z}; W)$.

We now consider the transformation T introduced earlier (see Definition 4). Using Observation 3 and Observation 4, we prove the following.

Observation 6. *Let* $\mathbf{Z} = \{\mathbf{x}_1, \ldots, \mathbf{x}_k\}$. *Let* $W^1, ..., W^l$ *be any partition of* W.

$$\begin{aligned} WP(\mathbf{Z}, W) &= \min_i MMU(W[\mathbf{Z} \to \mathbf{x}_i]) = \min_i MU(\mathbf{x}^*_{W[\mathbf{Z} \to \mathbf{x}_i]}, W[\mathbf{Z} \to \mathbf{x}_i]) \\ &= \min_{i,j} \{MU(\mathbf{x}^*_{W[\mathbf{Z} \to \mathbf{x}_i]}, W[\mathbf{Z} \to \mathbf{x}_i] \cap W^j)\} \end{aligned}$$

In particular, if we consider $T(\mathbf{Z}) = \{\mathbf{x}'_1, \ldots, \mathbf{x}'_k\}$ *where* $\mathbf{x}'_i = \mathbf{x}^*_{W[\mathbf{Z} \to \mathbf{x}_i]}$ *and its induced partition on* W*, the expression above becomes the following.*

$$WP(\mathbf{Z}, W) = \min_{i,j} \{MU(\mathbf{x}^*_{W[\mathbf{Z} \to \mathbf{x}_i]}, W[\mathbf{Z} \to \mathbf{x}_i; T(\mathbf{Z}) \to \mathbf{x}'_j]\}$$

Using this, we can now prove the following lemma:

Lemma 1. $SMU(T(\mathbf{Z}), W) \geq WP(\mathbf{Z}, W)$

From observation 5 and lemma 1 it follows that $SMU(T(\mathbf{Z}), W) \geq SMU(\mathbf{Z}, W)$, supporting our use of T as a local search optimization strategy.

Theorem 7. *Let* $\mathbf{Z}^*_W$ *be a maximin optimal recommendation set. Then* $\mathbf{Z}^*_W$ *is an optimal choice set:* $WP(\mathbf{Z}^*_W, W) = MaxWP(W)$.

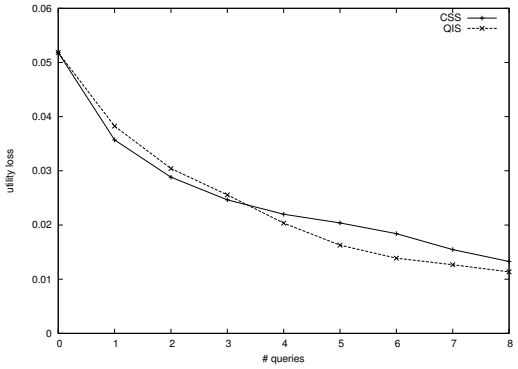

Fig. 2. Average utility loss with respect to the optimal recommendation

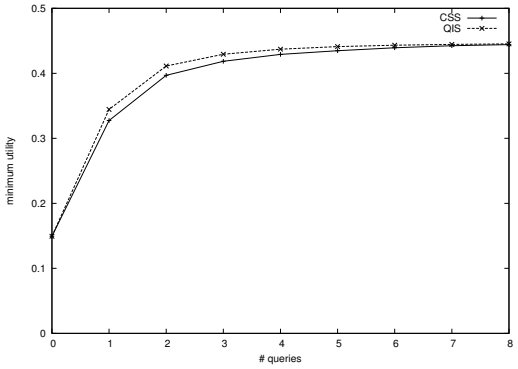

Fig. 3. Comparison of CSS and QIS. Results averaged over 30 instances, with set size $k = 3$.

4.2 Evaluation of the SMMU Criterion for Preference Elicitation

We simulated the interaction between a recommender system and a user; at each run a new utility function is randomly sampled. At each cycle the system asks a choice query presenting a set of items to the simulated user and the item with highest utility is selected; this is used by the system to produce a new set of items in the next cycle. Due to the high computation time of the exact method, we focused on CSS and QIS. [8]

In figure 2 we present the "true" utility loss by comparing, after each query answer, the utility of the optimal singleton recommendation according to MMU with utility of the true optimal recommendation (according to the utility function of the simulated user), as a function of the number of queries. The CSS and QIS algorithms have comparable performance, both improving utility loss when more queries are asked, but stalling approximately after 8 queries. In figure 3 we plot the minimum utility guarantee as a function of the number of queries. It quickly increases with the first 4-5 queries, but after that there is little improvement. While our theoretical results show that there is a connection between the problem of generating recommendations and queries, our experiments seem to indicate that maximin is generally unable to effectively elicit useful utility information beyond the first cycles. This is due to the extremely pessimistic nature of maximin and the "absorbing" nature of worst-case posterior maximin utility: in many situations there is no query leading to an improvement of maximin in the worst case (one needs to rely on ad-hoc tie-breaking strategies in these cases). It might be useful to adopt a non-myopic approach, or an alternative decision criterion. Further investigation is required to make long-term preference elicitation with maximin effective and avoiding stalling. [9]

[8] These were performed using larger instances, with 30 features per instance, 40 binary feature constraints and 40 utility constraints.

[9] We note that it is of course possible to use maximin as a decision criterion, while resorting to other strategies (perhaps based on regret or on probabilistic methods) for elicitation.

5 Discussion

In this paper we have developed a novel formalization for decision-making under utility uncertainty using the maximin utility criterion. This approach provides the highest degree of robustness, as the recommendation is guaranteed to yield the highest utility in the worst case. We addressed the problem of generating recommendation sets introducing a new decision criterion, setwise maximin utility, that formalizes the intuition that the best recommendation set is the one that is maximally "diverse" in a decision-theoretic way. Adapting ideas from [1, 13], we developed computational methods for optimizing sets according to this new criterion, as well as heuristics useful for large problem domains.

Following analogous models available for the minimax regret and Bayesian frameworks [13, 14], we showed the connection between the problem of generating optimal recommendation sets and myopically optimal elicitation queries. Our setwise maximin criterion (a natural extension of maximin to sets), in addition to providing robust recommendation sets, also serves as a means of generating myopically optimal choice queries (asking the user to pick his most preferred option in a set). In our experiments we evaluated the performance of our optimization methods on randomly generated data, showing that there is often value in recommending a set of options (instead of a single recommendation) to the user, and that recommendation sets can be efficiently optimized in practice. We also experimented with interactive utility elicitation, with elicitation driven by the (set-wise) maximin optimization; however in this setting the myopic elicitation with choice queries is not very effective (differently from [13, 14]).

In this work we consider the value of a set with respect to its capacity of "covering" the uncertainty associated with the partially known user utility function. The underlying assumption is that the user is looking for a single item to pick or purchase, and a set is shown to increase the chance that at least one item has high utility. We underline that there are works[5, 7] that consider recommendation sets with a different semantics (the problem of recommending a set of options accounting for positive or negative synergies between options). It would be interesting to consider their setting with a principled decision-theoretic view.

We conclude with a remark about the choice of the decision criterion. Maximin is very pessimistic; indeed expected utility may yield better recommendations in many cases. However, when a decision maker requires guarantees on the worst-case performance (e.g. in critical decisions with high stakes), she must be willing to sacrifice "average" utility. This is the price to pay for the (strong) worstcase guarantees of maximin.

6 Acknowledgments

The fist author would like to acknowledge Craig Boutilier for discussion about recommendation sets, minimax regret and utility elicitation. While he was not involved in this paper, his works on minimax regret (including joint works with the first author) laid down fundamental ideas that have been adapted to the maximin criterion here. We also thank the anonymous reviewers for valuable comments and suggestions.

References

1. 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.
2. Darius Braziunas and Craig Boutilier. Minimax regret-based elicitation of generalized additive utilities. In *Proc. of UAI-07*, pages 25–32, Vancouver, 2007.
3. Darius Braziunas and Craig Boutilier. Elicitation of factored utilities. *AI Magazine*, 29(4):79–92, 2008.
4. Urszula Chajewska, Daphne Koller, and Ronald Parr. Making rational decisions using adaptive utility elicitation. In *Proc. of AAAI-2000*, pages 363–369, Austin, TX, 2000.
5. Marie desJardins, Eric Eaton, and Kiri Wagstaff. Learning user preferences for sets of objects. In *International Conference on Machine Learning (ICML)*, pages 273–280, 2006.
6. Peter C. Fishburn. Interdependence and additivity in multivariate, unidimensional expected utility theory. *International Economic Review*, 8:335–342, 1967.
7. Yunsong Guo and Carla P. Gomes. Learning optimal subsets with implicit user preferences. In Craig Boutilier, editor, *IJCAI*, pages 1052–1057, 2009.
8. Ralph L. Keeney and Howard Raiffa. *Decisions with Multiple Objectives: Preferences and Value Trade-offs*. Wiley, New York, 1976.
9. Bart Peintner, Paolo Viappiani, and Neil Yorke-Smith. Preferences in interactive systems: Technical challenges and case studies. *AI Magazine*, 29(4):13–24, 2008.
10. Robert Price and Paul R. Messinger. Optimal recommendation sets: Covering uncertainty over user preferences. In *Proc. of AAAI-05*, pages 541–548, 2005.
11. Ahti Salo and Raimo P. Hämäläinen. Preference programming multicriteria weighting models under incomplete information. In *Handbook of Multicriteria Analysis*, volume 103 of *Applied Optimization*, pages 167–187. 2010.
12. Olivier Toubia, John Hauser, and Duncan Simester. Polyhedral methods for adaptive choice-based conjoint analysis. (4285-03), 2003.
13. Paolo Viappiani and Craig Boutilier. Regret-based optimal recommendation sets in conversational recommender systems. In *Proceedings of the 3rd ACM Conference on Recommender Systems (RecSys09)*, pages 101–108, New York, 2009.
14. Paolo Viappiani and Craig Boutilier. Optimal bayesian recommendation sets and myopically optimal choice query sets. In *Advances in Neural Information Processing Systems 23 (NIPS)*, pages 2352–2360, 2010.
15. A. Wald. *Statistical Decision Functions*. Wiley, New York, 1950.

7 Appendix

Proof of Observation 5 Considering the definition of $WP(\mathbf{Z}, W)$ and the equation for $SMU(\mathbf{Z}, W)$ in observation 4, we see that they are the same except that $WP(\mathbf{Z}, W)$ picks a maximizing $\mathbf{x}' \in \mathbf{X}$ after $\mathbf{x} \in \mathbf{Z}$ has been picked. Since $\mathbf{X}$ includes all options, $\mathbf{x}'$ can at worst be $\mathbf{x}$. ∎

Proof of Lemma 1 Let $T(\mathbf{Z}) = \{\mathbf{x}'_1, \ldots, \mathbf{x}'_k\}$ where $\mathbf{x}'_i = \mathbf{x}^*_{W[\mathbf{Z} \to \mathbf{x}_i]}$. The previous observations allow to write WP and SMU compactly

$$WP(\mathbf{Z}, W) = \min_{i,j}[MU(\mathbf{x}'_i, W[\mathbf{Z} \to \mathbf{x}_i, T(\mathbf{Z}) \to \mathbf{x}'_j])] \quad (6)$$

$$SMU(T(\mathbf{Z}), W) = \min_{i,j}[MU(\mathbf{x}'_j, W[\mathbf{Z} \to \mathbf{x}_i, T(\mathbf{Z}) \to \mathbf{x}'_j])] \quad (7)$$

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 MU 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 $MU(\mathbf{x}'_j, W[\mathbf{Z} \rightarrow \mathbf{x}_i, T(\mathbf{Z}) \rightarrow \mathbf{x}'_j]) \geq MU(\mathbf{x}'_i, W[\mathbf{Z} \rightarrow \mathbf{x}_i, T(\mathbf{Z}) \rightarrow \mathbf{x}'_j])$. In the expression of $SMU(T(\mathbf{Z}))$ (Eq. 7), each element is no less than its correspondent in the $WP(\mathbf{Z})$ expression (Eq. 6). Thus $SMU(T(\mathbf{Z}), W) \geq WP(\mathbf{Z}, W)$. ■

Proof of Theorem 7 Suppose $\mathbf{Z}^*_W$ is not an optimal query set, i.e., there is some $\mathbf{Z}'$ such that $WP(\mathbf{Z}', W) > WP(\mathbf{Z}^*_W, W)$. If we apply transformation T to $\mathbf{Z}'$ we obtain a set $T(\mathbf{Z}')$, and by the results above we have: $SMU(T(\mathbf{Z}'), W)) \geq WP(\mathbf{Z}', W) > WP(\mathbf{Z}^*, W) \geq SMU(\mathbf{Z}^*_W, W)$. This contradicts the (setwise) maximin optimality of $\mathbf{Z}^*_W$. If $T(\mathbf{Z}')$ has lower cardinality than the initial set, then a set of the original cardinality can be constructed in arbitrary way, since MMU is montone. ■