Efficient Exact Computation of Setwise Minimax Regret [⋆]

Federico Toffano[1], Paolo Viappiani[2], and Nic Wilson[1]

[1] Insight Centre for Data Analytics, Department of Computer Science, University College Cork, Cork, Ireland {federico.toffano,nic.wilson}@insight-centre.org
[2] UMR7606 CNRS, LIP6, Sorbonne Université, 4 pl. Jussieu, 75005 Paris, France paolo.viappiani@lip6.fr

Abstract. A key issue in incremental preference elicitation is choosing at each stage an appropriate query to the user in order to find a near-optimal solution as quickly as possible. A theoretically attractive method is to choose a query that minimises max setwise regret which is the worst case loss response in terms of value of information. We focus here on the situation in which the choices are represented explicitly in a database, and with a simple model of user utility as a weighted sum of the criteria; in this case when the user makes a choice we learn a linear constraint on the unknown vector of weights. We develop an algorithmic method for computing minimax setwise regret for this form of preference model, by making use of a SAT solver with cardinality constraints to prune the search space, and computing max setwise regret using an extreme points method. Our experimental results demonstrate the feasibility of the approach and the very substantial speed up over the state of the art.

Keywords: Preference elicitation · setwise regret · SAT with cardinality constraints.

1 Introduction

Preference elicitation is a key task in many artificial intelligence applications. Several authors [30, 8, 11, 12, 28, 7, 13, 26] have proposed interactive elicitation methods that revise a preference model based on a user's responses to questions asked with the intent of learning valuable preference information.

A principled approach is to represent the set of feasible utility functions that are consistent with the user's behavior, and to use minimax regret for selecting recommendations. Minimax regret selects as recommendation an alternative that minimises the worst case loss with respect to feasible parameters of the value function. The practical effectiveness of regret-based elicitation has been shown in numerous works (see, e.g., [30, 8, 10]) and in particular during a study carried out with real users [13].

[⋆] This material is based upon works supported by the Science Foundation Ireland under Grant No. 12/RC/2289-P2 which is co-funded under the European Regional Development Fund.

In [28] and [29] the authors generalized the concept of Minimax Regret defining the *Setwise max Regret* (*SMR*) which is used to evaluate a set of alternatives rather than a single alternative, and the setwise minimax regret (*SMMR*) which is used to select an optimal set w.r.t *SMR*. This provides a principled method for capturing the idea of recommendation sets. A remarkable property of setwise regret is that an optimal recommendation set with respect to *SMR* can be used for elicitation (asking to the decision maker which item is the most preferred) and constitutes a myopically optimal query, that is, the set maximizes an analogue of value of information [16] in a distribution-less sense. This makes it compelling to display an optimal set of items w.r.t. *SMR* with a combined elicitation and recommendation purpose: the system proposes a set of recommended items, the user picks the one he prefers, then the system updates the model and shows a new set of items; and this proceeds until a termination condition (max regret lower than a threshold; or simply when the user is satisfied) is met.

However, setwise regret is computationally very demanding to optimize; a straightforward approach requires the consideration of all subsets of a given cardinality, and the evaluation of their setwise max regret. Because of this high complexity, several heuristic methods are considered in [28, 29].

In this paper we address the problem of computing *SMMR* exactly for database problems (e.g., when a list of items with their features is readily available, as opposed to configuration problems where alternatives need to be constructed through constraint satisfaction). The main contribution of the paper is to propose an efficient algorithm for computing an optimal recommendation set, i.e., optimizing *SMMR*, exploiting some intrinsic properties of this criterion.

Our method relies on search; nodes in the search tree correspond to sets of alternative with cardinality up to k, and leaves to sets with cardinality exactly k. Pruning is done when we are sure that no extension of the current set can beat the previously found solution; to check this condition we use a SAT solver with cardinality constraints to prune the search space. This idea is combined with a fast subroutine to compute setwise max regret of a partial instantiation using extreme points (instead of using linear programming techniques as in previous works). The resulting algorithm is a method that is much faster than the state of the art, as is shown in our experimental tests.

The paper is organized as follows. In Section 2 we state our general assumptions, we recall the definitions of minimax regret and its setwise extensions, and provide some basic properties. In Section 3 we describe the main ideas behind our algorithm and its main components; while in Section 4 we provide a detailed description of the main algorithm to compute the *SMMR*. We provide some experimental results to validate our approach in Section 5, and conclude with a final discussion in Section 6.

2 Background

We now give some general background and notation, formally define minimax regret and its setwise variant, and introduce some basic properties.

We assume an underlying decision problem where the task is to choose one among a finite set A of alternatives (items, products, options). The user, also known as the Decision Maker (DM), is endowed with a utility or value function u_w, mapping from A to $\mathbb{R}$; w denote the parameters of the value function (a specific choice of w uniquely determines the value function). The goal is to pick $\arg\max_{x \in \mathrm{A}} u_w(x)$; however we assume that we (i.e., taking the point of view of a recommender system tasked to support decision-making) do not have access to the DM's true value function. The problem is to make recommendations under value function uncertainty (*strict uncertainty*); we suppose that our knowledge about the user's preferences is such that we can identify $\mathcal{W}$ as the set of scenarios representing all the consistent parametrisations w of a DM's value function u_w.

Minimax regret: The *Minimax Regret* (*MMR*) [23, 19] criterion is frequently used to solve decision problems under uncertainty. More recently, it has been used in artificial intelligence to to evaluate alternatives as potential recommendations, where the uncertainty refers to the parameters of the decision model [22, 11]. When considering a single recommendation, alternatives can be evaluated according to the *max regret*, quantifying the worst-case loss due to utility uncertainty:

$$MR_{\mathcal{W}}(\alpha, \mathrm{A}) = \max_{w \in \mathcal{W}} (\max_{\beta \in \mathrm{A}} u_w(\beta) - u_w(\alpha)) \tag{1}$$

$$= \max_{w \in \mathcal{W}} (\mathrm{Val}_{\mathrm{A}}(w) - u_w(\alpha)), \tag{2}$$

where we let $\mathrm{Val}_{\mathrm{A}}(w) = \max_{\alpha \in \mathrm{A}} u_w(\alpha)$ be the DM's value function for A defined as the maximum value that we can obtain from any alternative $\alpha \in \mathrm{A}$ with respect to the value function u_w. The *minimax regret* represents the minimum worst-case loss that can be attained by minimising max regret:

$$MMR_{\mathcal{W}}(\mathrm{A}) = \min_{\alpha \in \mathrm{A}} MR_{\mathcal{W}}(\alpha, \mathrm{A}) = \min_{\alpha \in \mathrm{A}} \max_{w \in \mathcal{W}} (\mathrm{Val}_{\mathrm{A}}(w) - u_w(\alpha)). \tag{3}$$

By recommending to the decision maker an alternative associated with minimax regret, i.e., alternative $\alpha^* \in \arg\max_{\alpha \in \mathrm{A}} MR_{\mathcal{W}}(\alpha, \mathrm{A})$, we provide robustness in face of uncertainty (due to not knowing the user's value function).

Regret-based elicitation has been applied to areas such as the elicitation of multi-attribute utilities (see, e.g., [30, 12, 5]), or the elicitation of preferences for ranking and voting problems (see, e.g., [20, 3, 7]).

Example 1. Consider the set of alternatives $\mathrm{A} = \{\alpha_1 = (4,4), \alpha_2 = (2,10), \alpha_3 = (10,2)\}$ whose value function $u_w(\alpha_i) = w \cdot \alpha_i$ with $w \in \mathcal{W} = \{w \in I\!R^2 : w_i \geq 0, \sum_{i=1}^{2} w_i = 1\}$ is shown in Figure 1. $MR_{\mathcal{W}}(\alpha_1, \mathrm{A}) = 6$ is maximised in $w_1 = 0$ and $w_1 = 1$, $MR_{\mathcal{W}}(\alpha_2, \mathrm{A}) = 8$ is maximised in $w_1 = 1$, and $MR_{\mathcal{W}}(\alpha_1, \mathrm{A}) = 8$ is maximised in $w_1 = 0$. Thus, $MMR_{\mathcal{W}}(\mathrm{A}) = 6$.

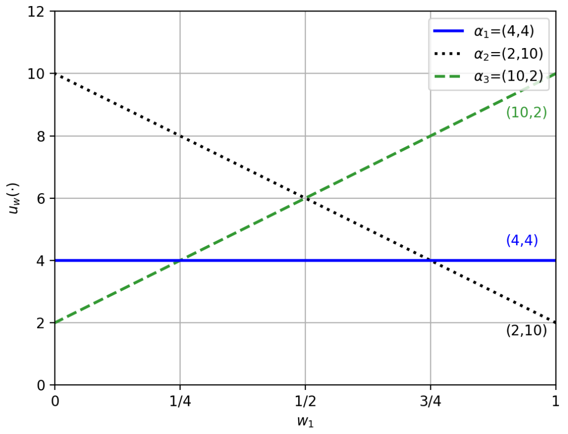

Fig. 1: Plot of the linear value functions $u_w(\alpha_i) = w \cdot \alpha_i$ for the alternatives $\alpha_1 = (4, 4)$ (blue solid), $\alpha_2 = (2, 10)$ (black dotted) and $\alpha_3 = (10, 2)$ (green dashed) with $w \in \mathcal{W} = \{w \in I\!R^2 : w_i \geq 0, \sum_{i=1}^{2} w_i = 1\}$.

Setwise regret: In many applications it is desirable to produce a *recommendation set*, and not just a single recommendation, giving the opportunity to the decision maker to pick the alternative (among those of the recommendation set) that provides most value to him/her. Intuitively, by providing several recommendations, it is more likely that at least one of them will have high utility value to the decision maker. As originally observed by Price and Messinger [21] it is therefore a good idea to show "diverse" recommendations that have high value for different parts of the parameter space $\mathcal{W}$.

Setwise regret constitutes a principled way to measure the quality of a recommendation set. Assume that, when we provide B as recommendation set, the decision maker is able to pick the most preferred item (the one with highest value) in B, thus perceiving value $\mathrm{Val_B}(w) = \max_{\alpha \in \mathrm{B}} u_w(\alpha)$ when the "true" value function is dictated by parameters w. The regret of a set B with respect to w is the difference between the utility of the best item under w in the whole dataset A and the utility of the best item w.r.t. w in the set B; that is, $\mathrm{Val_A}(w) - \mathrm{Val_B}(w)$. The *setwise max regret* (*SMR*) [28, 29] of a subset B of the finite set of alternatives A, with respect to the parameter space $\mathcal{W}$, is then defined as the maximum of this difference:

$$SMR_{\mathcal{W}}(\mathrm{B}, \mathrm{A}) = \max_{w \in \mathcal{W}} (\mathrm{Val_A}(w) - \mathrm{Val_B}(w)). \tag{4}$$

The value $SMR_{\mathcal{W}}(\mathrm{B}, \mathrm{A})$ is the worst case loss, due to value function uncertainty, of recommending the set B. Notice that, obviously, *SMR* reduces to *MR* when the set B is a singleton; at the other extreme, if $B = A$ (the whole dataset is recommended), then *SMR* is zero.

An optimal recommendation set of size k is a subset of A of cardinality k that minimizes setwise max regret with respect to $\mathcal{W}$. Thus, the *setwise minimax regret* (*SMMR*) [28, 29] of size k with respect to $\mathcal{W}$ is defined by:

$$SMMR^k_{\mathcal{W}}(\mathrm{A}) = \min_{\mathrm{B} \subseteq \mathrm{A} : |\mathrm{B}| = k} SMR_{\mathcal{W}}(\mathrm{B}, \mathrm{A}). \tag{5}$$

The value $SMMR^k_{\mathcal{W}}(\mathrm{A})$ is then the minimum setwise max regret we can obtain from all the possible subsets B of A with cardinality k with respect to any scenario $w \in \mathcal{W}$. Notice that k is usually a small number, usually identified by an application expert.

Recommendation sets can be used in elicitation, where they are treated as *choice queries* (i.e., questions of the kind *"Among a, b, and c, which one do you prefer?"*) with the goal of reducing uncertainty in order to improve the quality of future recommendations; that is, reducing minimax regret. It turns out [28, 29] that optimal recommendation sets w.r.t. *SMMR* are also myopically optimal in an elicitation sense, as they ensure the highest worst-case (with respect to the possible query's responses) reduction of minimax regret *a posteriori*.

Example 2. Consider the set of alternatives $\mathrm{A} = \{\alpha_1 = (4, 4), \alpha_2 = (2, 10), \alpha_3 = (10, 2)\}$ whose value function $u_w(\alpha_i) = w \cdot \alpha_i$ with $w \in \mathcal{W} = \{w \in I\!R^2 : w_i \geq 0, \sum_{i=1}^{2} w_i = 1\}$ is shown in Figure 1. $SMR_{\mathcal{W}}(\{\alpha_1, \alpha_2\}, \mathrm{A}) = 6$ is maximised in $w_1 = 1$, $SMR_{\mathcal{W}}(\{\alpha_1, \alpha_3\}, \mathrm{A}) = 6$ is maximised in $w_1 = 0$, and $SMR_{\mathcal{W}}(\{\alpha_2, \alpha_3\}, \mathrm{A}) = 0$ since $\mathrm{Val}_{\{\alpha_2, \alpha_3\}}(w) \geq u_w(\alpha_1)$ for any $w \in \mathcal{W}$. Thus, $SMMR^2_{\mathcal{W}}(\mathrm{A}) = 0$.

Optimization of setwise regret: Computation of *SMMR* differs with respect to the type of decision problem. When alternatives are constructed from a configuration problem, the decision space is encoded with variables and hard constraints express which combinations of variables are feasible [8, 12, 4]); for example, this is the case of configuring computer parts to obtain a customized laptop. In this type of problem setwise regret can be optimized with a mixed-integer program and solved by techniques such as Bender's decomposition and constraint generation [28, 29].

In this paper we focus on database problems, where the alternatives are enumerated and represented with an explicit list of multi-attribute outcomes (see, e.g., [12, 13, 7]); for example the problem of choosing one from a catalogue of already assembled laptops. In this case the straightforward approach to optimize *SMMR* is based on the generation of all the possible sets of a specific size k and choosing the one with lowest setwise maximum regret *SMR*. Given the high complexity of the computation of an optimal set using the *SMMR* criterion, in [28, 29] the authors have also proposed several heuristics methods that have been shown to have good performance in simulation.

Value functions: While the previously introduced concepts are quite general and apply to any kind value function, in this work we focus on multi-attribute problems with linear value functions. An alternative α is represented with a vector of p reals, with each component corresponding to a criterion, and $\alpha(i)$ being the evaluation of α with respect to the ith criterion. We define $u_w(\alpha) = \alpha{\cdot}w$ $(= \sum_{i=1}^{p} w_i \alpha(i))$ to be the value function parametrised with respect to w, where $\alpha \in \mathrm{A} \subset I\!\!R^p$ and $w \in \mathcal{U}$ and

$$\mathcal{U} = \{w \in I\!\!R^p : w_i \geq 0, \sum_{i=1}^{p} w_i = 1\},$$

and the weight w_i relates to the importance that the DM gives to criterion i.

In general, setwise (minimax) regret is defined for any closed (and thus compact) subset $\mathcal{W}$ of $\mathcal{U}$. Our algorithmic approach assumes that $\mathcal{W}$ is a compact convex polytope. For example, $\mathcal{W}$ could be a reduction of $\mathcal{U}$ given by a set Λ of DM's input preferences, with, for instance, a user preference of alternative α over β leading to the constraint $\alpha \cdot w \geq \beta \cdot w$ (see, e.g., [31]).

Basic properties of setwise regret: We now give some basic properties of setwise regret that we will use later. First of all we show how $\mathit{SMR}_{\mathcal{W}}(\mathrm{B}, \mathrm{A})$ is monotone, with respect to set inclusion, in both B and $\mathcal{W}$; and then show how $\mathit{SMMR}_{\mathcal{W}}^{k}$ is monotone in k.

Lemma 1. *$\mathit{SMR}_{\mathcal{W}}(B, A)$ is monotonically decreasing in B, and monotonically increasing in $\mathcal{W}$, i.e., if $B' \supseteq B$ and $\mathcal{W}' \subseteq \mathcal{W}$, then $\mathit{SMR}_{\mathcal{W}'}(B', A) \leq \mathit{SMR}_{\mathcal{W}}(B, A)$.*

An alternative $\alpha \in \mathrm{A}$ is said to be *$\mathcal{W}$-dominated* if there exists another alternative $\beta \in A$ such that the former has higher-or-equal value than the latter for any $w \in \mathcal{W}$, and the relation is strict for at least one value. For a given set A, let $\mathrm{UD}_{\mathcal{W}}(\mathrm{A}) \subseteq \mathrm{A}$ be the set of elements of A that are undominated in $\mathcal{W}$.

Alternatives that are known to be dominated given $\mathcal{W}$ can be removed as they do not impact the value of setwise max regret, as shown by the next Lemma.

Lemma 2. *$\mathit{SMR}_{\mathcal{W}}(B, A) = \mathit{SMR}_{\mathcal{W}}(\mathrm{UD}_{\mathcal{W}}(B), \mathrm{UD}_{\mathcal{W}}(A))$ and, for any $k \geq 1$, $\mathit{SMMR}_{\mathcal{W}}^{k}(A) = \mathit{SMMR}_{\mathcal{W}}^{k}(\mathrm{UD}_{\mathcal{W}}(A))$.*

3 An Efficient Algorithm to Compute Setwise Minimax Regret

The main idea behind our algorithm is to use a depth-first search over subsets of A, with setwise max regret computations at leaf nodes of the search tree, and with a method of pruning branches that reduces the number of setwise max regret computations. More precisely, for a given subset C of A with cardinality less than k, we use a method that determines, for a particular discrete subset $\mathcal{W}'$ of $\mathcal{W}$, if $\mathit{SMR}_{\mathcal{W}'}(\mathrm{B}, \mathrm{A}) \geq \overline{r}$ holds for all supersets B of C with cardinality k, where $\overline{r}$ is the current upper bound of $\mathit{SMMR}_{\mathcal{W}}^{k}(\mathrm{A})$. If this holds then, by Lemma 1, $\mathit{SMR}_{\mathcal{W}}(\mathrm{B}, \mathrm{A}) \geq \overline{r}$ for all such B, enabling us to backtrack at this point of the search.

In the next paragraph we define how we represent subsets of A; then we define how to evaluate the setwise max regret of a set of subsets of A simultaneously with a Boolean satisfiability (SAT) problem.

Search space: We consider the set of Boolean strings of length at most $n = |\mathrm{A}|$ as a representation of the search space over subsets of A with cardinality less or equal to k. For string x, let $\mathit{Len}(x)$ be the length of x. Let us label A as $\alpha_1, \ldots, \alpha_n$, where $n = |\mathrm{A}|$. We say that a string is *complete* if it is of length n, and otherwise it is *partial*. Each complete string x corresponds to a subset B_x of A, where B_x is the set of all $\alpha_i \in \mathrm{A}$ such that x has a one at its i-th position. We say that complete string *x is of cardinality k* if it contains *k ones*, i.e., if the corresponding subset B_x is of cardinality k. If x and y are Boolean strings then we say that *y extends x* if $\mathit{Len}(y) \geq \mathit{Len}(x)$ and the first $\mathit{Len}(x)$ places of y are the same as those of x. We say that y is a *complete extension of x* if y extends x and y is a complete string. Each partial string x represents a set $\mathcal{B}_x$ of subsets of A, i.e., all those subsets of cardinality k that correspond to extensions of x. $\mathcal{B}_x$ is thus the set of all sets B_y for complete extensions y of x of cardinality k. In Section 3.3 we define how to generate strings x in turn.

Example 3. Let $\mathrm{A} = \{\alpha_1, \ldots, \alpha_5\}$ be a set of $n = 5$ elements, and let $k = 3$. The complete string $z = 01101$ represents the subset $\mathrm{B}_z = \{\alpha_2, \alpha_3, \alpha_5\}$. The partial string $x = 011$ represents the subsets $\mathcal{B}_x = \{\{\alpha_2, \alpha_3, \alpha_4\}, \{\alpha_2, \alpha_3, \alpha_5\}\}$, where the complete extensions of x are $y = 01101$ and $y' = 01110$.

3.1 Pruning the Search Space using SAT

Evaluating subsets of A: Given a partial string x, if a set $\mathrm{B}_y \in \mathcal{B}_x$ is such that $\mathit{SMR}_{\mathcal{W}}(\mathrm{B}_y, \mathrm{A}) < \overline{r}$, then B_y has to contain at least one alternative with worst case regret lower than $\overline{r}$ for each $w \in \mathcal{W}'$. This concept is formally defined with the following lemma and it will be used to check if there could exists a set in $\mathcal{B}_x$ improving the current upper bound $\overline{r}$ of $\mathit{SMMR}^k_{\mathcal{W}}(\mathrm{A})$.

Lemma 3. *Let $\overline{r}$ be an upper bound of $\mathit{SMMR}^k_{\mathcal{W}}(A)$, $\mathcal{W}' \subseteq \mathcal{W}$ and $B \subseteq A$. For $w \in \mathcal{W}$, let Γ_w be the set of $\alpha \in A$ such that $\mathit{Val}_A(w) - u_w(\alpha) < \overline{r}$. Then $\mathit{SMR}_{\mathcal{W}'}(B, A) < \overline{r}$ if and only if for all $w \in \mathcal{W}'$, there exists $\alpha \in B$ such that $\alpha \in \Gamma_w$.*

Proof. From the definition of setwise max regret it follows that $\mathit{SMR}_{\mathcal{W}'}(\mathrm{B}, \mathrm{A}) < \overline{r}$ if and only if $\mathrm{Val}_{\mathrm{A}}(w) - \mathrm{Val}_{\mathrm{B}}(w) < \overline{r}$ for all $w \in \mathcal{W}'$, which is if and only if for all $w \in \mathcal{W}'$ there exists $\alpha \in \mathrm{B}$ such that $\mathrm{Val}_{\mathrm{A}}(w) - u_w(\alpha) < \overline{r}$, which is if and only if $\alpha \in \Gamma_w$ since $\mathrm{B} \subseteq \mathrm{A}$.

To check if there exists a set $\mathrm{B}_y \in \mathcal{B}_x$ such that $\mathit{SMR}_{\mathcal{W}'}(\mathrm{B}_y, \mathrm{A}) < \overline{r}$, we define a SAT problem with cardinality constraint c (see, e.g., [24]), where the cardinality constraint is used to define the size k of the sets in $\mathcal{B}_x$.

Example 4. Consider the following SAT formula: $X = (X_1 \vee X_2) \wedge (X_1 \vee X_3)$ with cardinality constraint $c = 1$, where X_i are $\{0, 1\}$-valued variable. X_i with $i = \{1, 2, 3\}$ are literals, and $(X_1 \vee X_2)$ and $(X_1 \vee X_3)$ are clauses. The cardinality constraint $c = 1$ means $\sum_{i=1}^{3} X_i = 1$. In this example, X is satisfiable since if $X_1 = 1$ then $X = 1$. But if for example we add the constraint $X_1 = 0$, then X is unsatisfiable since for any valid assignment of the cardinality constraint, i.e., $(X_1 = 0, X_2 = 1, X_3 = 0)$ or $(X_1 = 0, X_2 = 0, X_3 = 1)$, we get $X = 0$.

In our SAT problem, we use a $\{0,1\}$-valued variable X_i for each $\alpha_i \in \mathrm{A}$. These are used to reason about the unknown sets B_y in $\mathcal{B}_x$, which we want to be such that $SMR_{\mathcal{W}'}(\mathrm{B}_y, \mathrm{A}) < \overline{r}$. Then $X_i = 1$ means that $\mathrm{B}_y \ni \alpha_i$. Given a partial string x, we then define the corresponding SAT problem with cardinality constraint as follows:

(1) The cardinality constraint $|\mathrm{B}_y| = k$ is expressed as $\sum_{\alpha_i \in \mathrm{A}} X_i = k$.
(2) The constraint that y extends x is expressed as: for all $i \in \{1, \ldots, Len(x)\}$,
   - if $x(i) = 1$ then $X_i = 1$ (where $x(i)$ is the i-th value of x);
   - if $x(i) = 0$ then $X_i = 0$.
(3) For each $w \in \mathcal{W}'$ we define a clause $\bigvee_{\alpha_i \in \Gamma_w} X_i$, where Γ_w is the set of $\alpha \in \mathrm{A}$ such that $\mathrm{Val}_{\mathrm{A}}(w) - u_w(\alpha) < \overline{r}$.

This SAT problem is satisfiable if and only if there exists $\mathrm{B}_y \in \mathcal{B}_x$ such that for all $w \in \mathcal{W}'$ there exists $\alpha \in \mathrm{B}_y$ such that $\alpha \in \Gamma_w$, which is (by Lemma 3) if and only if there exists $\mathrm{B}_y \in \mathcal{B}_x$ such that $SMR_{\mathcal{W}'}(\mathrm{B}_y, \mathrm{A}) < \overline{r}$. Therefore, if the SAT problem is unsatisfiable, then for each $\mathrm{B}_y \in \mathcal{B}_x$, $SMR_{\mathcal{W}'}(\mathrm{B}_y, \mathrm{A}) \geq \overline{r}$, and thus (by Lemma 1) $SMR_{\mathcal{W}}(\mathrm{B}_y, \mathrm{A}) \geq \overline{r}$. This means that there is then no need to explore any string y extending x, so we can then backtrack from the current search node associated with x, saving us from computing $SMR_{\mathcal{W}}(\mathrm{B}_y, \mathrm{A})$ for $\mathrm{B}_y \in \mathcal{B}_x$.

Example 5. Consider the set of alternatives $\mathrm{A} = \{\alpha_1 = (4,4), \alpha_2 = (2,10), \alpha_3 = (10,2)\}$ whose value function $u_w(\alpha_i) = w \cdot \alpha_i$ with $w \in \mathcal{W} = \{w \in I\!R^2 : w_i \geq 0, \sum_{i=1}^{2} w_i = 1\}$ is shown in Figure 1. Let $k = 2$, $\mathcal{W}' = \{(0,1), (0.5), (1,0)\}$, $\overline{r} = 1$, and let x be the string 1. Thus, $\Gamma_{(0,1)} = \{\alpha_2\}$, $\Gamma_{(0.5,0.5)} = \{\alpha_2, \alpha_3\}$, $\Gamma_{(1,0)} = \{\alpha_3\}$ and $\mathcal{B}_x = \{\{\alpha_1, \alpha_2\}, \{\alpha_1, \alpha_3\}\}$ since the complete extensions of x with cardinality $k = 2$ are $y = 110$ and $y' = 101$. The corresponding SAT problem is then $X_2 \wedge (X_2 \vee X_3) \wedge X_3$ with cardinality constraints $c = 2$ and $X_1 = 1$. It is easy to see that in this case the SAT problem is unsatisfiable; therefore we can avoid the computation of $SMR_{\mathcal{W}}(\{\alpha_1, \alpha_2\}, \mathrm{A})$ and $SMR_{\mathcal{W}}(\{\alpha_1, \alpha_3\}, \mathrm{A})$. In fact, the subset B of A cardinality 2 that minimises $SMR_{\mathcal{W}}(\mathrm{B}, \mathrm{A})$ is $\mathrm{B} = \{\alpha_2, \alpha_3\}$.

3.2 Computation of max regret

With linear value functions $u_w(\cdot)$, a standard method to compute the setwise max regret $SMR_{\mathcal{W}}(\mathrm{B}, \mathrm{A})$ of a set $\mathrm{B} \in \mathrm{A}$ consists of the evaluation of a linear programming (LP) problem for each $\alpha_i \in \mathrm{A}$ (see [28]). Briefly, for $\alpha_i \in \mathrm{A}$ we have $SMR_{\mathcal{W}}(\mathrm{B}, \{\alpha_i\}) = \max_{w \in \mathcal{W}}(\alpha_i \cdot w - \mathrm{Val}_{\mathrm{B}}(w))$, which means that we can compute $SMR_{\mathcal{W}}(\mathrm{B}, \{\alpha_i\})$ as the maximum value δ_i subject to the constraints $w \in \mathcal{W}$, and $(\alpha_i - \beta) \cdot w \geq \delta_i$ for each $\beta \in \mathrm{B}$. We can then compute $SMR_{\mathcal{W}}(\mathrm{B}, \mathrm{A})$ as $\max_{\alpha_i \in \mathrm{A}} SMR_{\mathcal{W}}(\mathrm{B}, \{\alpha\})$.

However, here we also make use of a method, described briefly in the next paragraph, that we developed previously [2] for computing $SMR_{\mathcal{W}}(\mathrm{B}, \mathrm{A})$. In our experimental results in [2] we found this method to be between 4 to 60 times faster than the LP method for $p \leq 6$.

With linear value functions $u_w(\cdot)$, the max regret of an alternative β can be easily computed evaluating only the extreme points $Ext(\mathcal{W})$ of $\mathcal{W}$, i.e., $SMR_{\mathcal{W}}(\beta, \mathrm{A}) = SMR_{Ext(\mathcal{W})}(\beta, \mathrm{A})$. For the setwise max regret of a set B instead, we need to evaluate the extreme points points of $\mathcal{W}_\beta$ for each $\beta \in \mathrm{B}$, where $\mathcal{W}_\beta = \{w \in \mathcal{W} : \beta \cdot w \geq \beta_j \cdot w, \forall \beta_j \in \mathrm{B}\}$. Let $\gamma(\mathcal{W}, \mathrm{B}) = \{(w, r) : w \in \mathcal{W}, \quad \beta \cdot w \leq r \quad \forall \beta \in \mathrm{B}\}$; this can be seen to be the *epigraph* [9] of the value function Val_B on $\mathcal{W}$. Let $UE(\mathrm{B}) = \bigcup_{\beta \in \mathrm{B}} Ext(\mathcal{W}_\beta)$ be the union of the sets of extreme points $Ext(\mathcal{W}_\beta)$ for each $\beta \in \mathrm{B}$. In [2] it is shown that $UE(\mathrm{B})$ equals the projection in $\mathcal{W}$ of the set of extreme points $Ext(\gamma(\mathcal{W}, \mathrm{B}))$ of $\gamma(\mathcal{W}, \mathrm{B})$. Also, for each $(w, r) \in Ext(\gamma(\mathcal{W}, \mathrm{B}))$ we have that $r = \mathrm{Val}_\mathrm{B}(w)$. This implies that $SMR_{\mathcal{W}}(\mathrm{B}, \mathrm{A})$ can be computed as:

$$SMR_{\mathcal{W}}(\mathrm{B}, \mathrm{A}) = \max_{w \in UE(\mathrm{B})} (\mathrm{Val}_\mathrm{A}(w) - \mathrm{Val}_\mathrm{B}(w)). \tag{6}$$

Example 6. Consider the example in Figure 2 with $\mathrm{A} = \{(2,8), (8,2), (6,6)\}$ and let $\mathrm{B} = \{(2,8), (8,2)\}$. $\mathcal{W}_{(2,8)} = \{(0,1), (\frac{1}{2}, \frac{1}{2})\}$, $\mathcal{W}_{(8,2)} = \{(\frac{1}{2}, \frac{1}{2}), (1,0)\}$ and $Ext(\gamma(\mathcal{W}, \mathrm{B})) = \{((0,1), 8), ((\frac{1}{2}, \frac{1}{2}), 5)((1,0), 8)\}$, and $UE(\mathrm{B}) = \{(0,1), (\frac{1}{2}, \frac{1}{2}), (1,0)\}$. Then $SMR_{\mathcal{W}}(\mathrm{B}, \mathrm{A}) = \max((8-8), (6-5), (8-8)) = 1$ and it is maximised with $w = (\frac{1}{2}, \frac{1}{2})$, and $\mathrm{Val}_\mathrm{B}((\frac{1}{2}, \frac{1}{2})) = 5$.

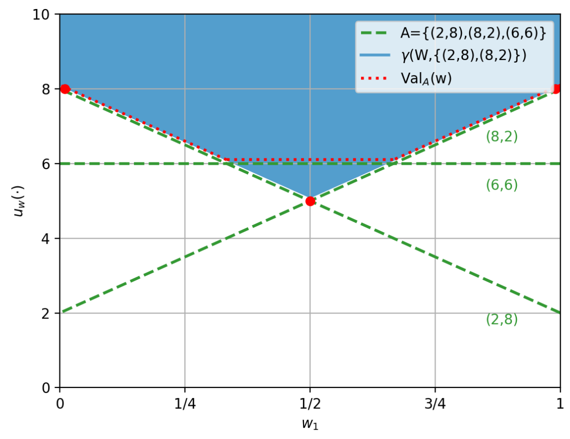

Fig. 2: Value function $u_w(\alpha_i) = w \cdot \alpha_i$ for each alternative in $\mathrm{A} = \{(2,8), (8,2), (6,6)\}$ (green solid) where $w \in \mathcal{W} = \{w \in I\!R^2 : w_i \geq 0, \sum_{i=1}^{2} w_i = 1\}$. Note that we show $u_w(\alpha_i)$ only with respect to w_1 since $w_2 = w_1 - 1$. The red dotted line represents $\mathrm{Val}_\mathrm{A}(w)$, the blue area is the epigraph $\gamma(\mathcal{W}, \mathrm{B}) = \{(w, r) : w \in \mathcal{W}, r \geq \mathrm{Val}_\mathrm{A}(w)\}$ for $\mathrm{B} = \{(2,8), (8,2)\}$, and the red points are the extreme points of the epigraph.

3.3 Generating subsets of A using depth-first search

We generate strings x representing subsets of A sequentially using a depth-first search with backtracking on a binary tree T, and with a fixed value and variable ordering (though the variable ordering depends on the value k: see Section 3.4). Note that we are not interested in all the possible binary strings of length n, but instead we want to generate complete strings x with k ones and the corresponding sub-strings since these will represents subsets B of A with $|\mathrm{B}| \leq k$. The order in which we reach complete strings (and their associated subsets) is based on the obvious lexicographic order, i.e., ascending numerical order if the strings are viewed as binary numbers. We then define T as follows: the root represents the empty string; internal nodes represent strings of length less than n; and leaves represent strings of length n with k ones. The out-edges of an internal node pointing to the corresponding left and right children have values 0 and 1 respectively. Thus, if an internal node represents the string x, then the left child represents the string $x0$ and the right child represents the string $x1$.

We generate strings sequentially starting from the left most leaf node representing the subset $(\alpha_{n-k+1}, \ldots, \alpha_n)$. Given a generic string x_j, we define two methods to generate the next string x_{j+1}, namely, the *backtracking case* and the *non-backtracking case.*

Backtracking case: Let $\mathit{NextBT}(x_j, n, k) = x_{j+1}$ be the backtracking case of the j-th string. With $\mathit{NextBT}(x_j, n, k)$ we move from the current node representing x_j toward the root until we find an edge e with value zero. Let v be the parent of e. We define $\mathit{NextBT}(x, n, k)$ as the string represented by the right child of v. We will use this method to generate the string x_{j+1} when x_j is a complete string, or when x_j is a partial string but $\mathit{SMR}_{\mathcal{W}}(\mathrm{B}, \mathrm{A}) \geq \overline{r}$ for all $\mathrm{B} \in \mathcal{B}_{x_j}$. Roughly speaking, we use $\mathit{NextBT}(x_j, n, k)$ when we want to evaluate a new set of subsets since $\mathcal{B}_{x_j} \cap \mathcal{B}_{x_{j+1}} = \emptyset$.

Non-backtracking case: Let $\mathit{Next}(x_j, n, k) = x_{j+1}$ be the non-backtracking case of the j-th string. With $\mathit{Next}(x_j, n, k)$ we compute the next string following the depth-first search logic. We will use this method to generate the string x_{j+1} for the cases not covered by the backtracking case, i.e., when x_j is not a complete string and we can't ensure that $\mathit{SMR}_{\mathcal{W}}(\mathrm{B}, \mathrm{A}) \geq \overline{r}$ for all $\mathrm{B} \in \mathcal{B}_{x_j}$. Roughly speaking, we use $\mathit{Next}(x_j, n, k)$ to reduce the sets to evaluate, in fact, $\mathcal{B}_{x_{j+1}} \subset \mathcal{B}_{x_j}$.

In both cases, when we visit the root, and the corresponding out-edges have already been visited, we stop the search. Note that if $\mathcal{B}_{x_{j+1}}$ is a singleton set with x_{j+1} not being a complete string, then we can speed up the computation by jumping to the leaf node corresponding to the unique set in $\mathcal{B}_{x_{j+1}}$. This can happen when x_{j+1} can be extended only with ones or only with zeros in order to satisfy the constraint that a complete string x must have k ones.

3.4 Further implementation details

Generating $\mathcal{W}'$: We start with $\mathcal{W}' = \emptyset$, then for each *SMR* computation of a subset B of A, if $\mathit{SMR}_{\mathcal{W}}(\mathrm{B}, \mathrm{A})$ is greater than the current upper bound $\overline{r}$ of $\mathit{SMMR}^{k}_{\mathcal{W}}(\mathrm{A})$, then we update $\mathcal{W}'$ as $\mathcal{W}' = \mathcal{W}' \cup \mathit{UE}(\mathrm{B})$ where $\mathit{UE}(\mathrm{B})$ is the projection of $\mathit{Ext}(\gamma(\mathcal{W}, \mathrm{B}))$ to $\mathcal{W}$. We use the set $\mathit{UE}(\mathrm{B})$ to update $\mathcal{W}'$ since these points have already been computed, during the evaluation of $\mathit{SMR}_{\mathcal{W}}(\mathrm{B}, \mathrm{A})$. One could update $\mathcal{W}'$ using only the point $w \in \mathcal{W}$ in which $\mathit{SMR}_{\mathcal{W}}(\mathrm{B}, \mathrm{A})$ is maximised; however, collecting more points in $\mathcal{W}'$ adds more clauses to the SAT problem, and thus increases the possibility of unsatisfiability, leading to pruning of the search tree.

SAT instances: For a given $\mathcal{W}'$, when the SAT problem associated with a string x is solvable, we can use the corresponding instantiation to define the SAT problem associated to a string y extending x. In fact, the SAT problem corresponding to y will be the same as that associated with x but with the additional constraints $X_i = y_i$ for all $i \in \{\mathit{Len}(x\} + 1, \ldots, \mathit{Len}(y))$. This is particularly useful when y is a substring of the solution X found for the SAT problem for x, since in this case X is a solution also to the SAT problem associated with y, and thus we do not need to call the SAT solver. For example, suppose $n = 5$, $k = 3$ and $x = 01$, and suppose that the solution of the SAT problem associated with x is $X = 01110$. Then, if $y = 011$ and $\mathcal{W}'$ has not changed, we don't need to define from scratch a new SAT problem since the SAT problem associated with y is the same as that associated with x but with the additional constraint $X_3 = 1$. Furthermore, in this case y is also a substring of X, thus we do not need to call the SAT solver since X is a solution also for the SAT problem associated with y.

4 Pseudocode

In this section we combine the concepts presented in Section 3, defining the whole procedure for the computation of $\mathit{SMMR}^{k}_{\mathcal{W}}(\mathrm{A})$. The inputs of our algorithm are:

1. A finite set A of alternatives α_i where each alternative is represented as a p-dimensional vector of reals.
2. The DM's preference state space $\mathcal{W}$ representing the possible parametrisations w of the value function $u_w(\cdot)$ expressed as a compact subset of $\{w \in \mathbb{R}^p : w_i \geq 0, \sum_{i=1}^{p} w_i = 1\}$.
3. An integer $k \leq |\mathrm{A}|$ representing the cardinality of the subsets of A that we want to evaluate.

We start with $\mathcal{W}' = \emptyset$, with x equal to $n - k$ zeros followed by n ones, and with $\overline{r} = \infty$. Then we proceed as follows:

1) If x is the empty string then we stop the algorithm and return $\overline{r}$.
2) If $\mathit{Len}(x) = n$ (i.e., x is a complete string) then

   a) we compute $\mathit{SMR}_{\mathcal{W}}(\mathrm{B}_x, \mathrm{A})$, where B_x is the set represented by x, also generating the set $\mathit{UE}(\mathrm{B}_x)$ (that is the projection on $\mathcal{W}$ of the set of extreme points of the epigraph $\gamma(\mathcal{W}, \mathrm{B}_x)$);
   b) we update the upper bound $\overline{r}$ by $\overline{r} = \min(\overline{r}, \mathit{SMR}_{\mathcal{W}}(\mathrm{B}_x, \mathrm{A}))$;
   c) we update $\mathcal{W}' = \mathcal{W}' \cup \mathit{Ext}(\mathcal{W}_{\mathrm{B}_x})$;
   d) we move to the next string with the backtracking case $\mathit{NextBT}(x, n, k)$.
3) Otherwise, $\mathit{Len}(x) < n$ (i.e., x is a partial string). We call Boolean function $\mathrm{SAT}(x, k, \mathrm{A}, \mathcal{W}', \overline{r})$, which returns TRUE if and only if the associated SAT problem (see Section 3.1) is satisfiable, i.e., there exists $\mathrm{B}_y \in \mathcal{B}_x$ such that $\mathit{SMR}_{\mathcal{W}'}(\mathrm{B}, \mathrm{A}) < \overline{r}$.
   a) If the SAT problem is satisfiable, we move to the next string with the non-backtracking case $\mathit{Next}(x, n, k)$;
   b) If the SAT problem is not satisfiable, we move to the next string with the backtracking case $\mathit{NextBT}(x, n, k)$.

When we have gone through all of the space of strings, i.e., when x is the empty string, the value of $\overline{r}$ will equal $\mathit{SMMR}^k_{\mathcal{W}}(\mathrm{A})$, i.e., the minimum value of $\mathit{SMR}_{\mathcal{W}}(\mathrm{B}, \mathrm{A})$ over all subsets B of A of cardinality k.

In Algorithm 1 we show the recursive procedure to compute $\mathit{SMMR}^k_{\mathcal{W}}(\mathrm{A})$.

Algorithm 1 Minimum setwise max regret

```
 1: procedure SMMR($k, \mathrm{A}, \mathcal{W}$)
 2:     $\overline{r} \leftarrow \infty$
 3:     $x \leftarrow \mathbf{0}[n-k]\mathbf{1}[k]$
 4:     BestSMR ← $n$ reals initialised to $\infty$
 5:     do
 6:         if $\mathit{Len}(x) = n$ then
 7:             $\mathit{SMR} \leftarrow \mathit{SMR}_{\mathcal{W}}(\mathrm{B}_x, \mathrm{A})$
 8:             $\overline{r} \leftarrow \min(\overline{r}, \mathit{SMR})$
 9:             $\mathcal{W}' \leftarrow \mathcal{W}' \cup \mathit{UE}(\mathrm{B}_x)$
10:             $x \leftarrow \mathit{NextBT}(x, n, k)$
11:         else if $\mathrm{SAT}(x, k, \mathrm{A}, \mathcal{W}', \overline{r})$ then
12:             $x \leftarrow \mathit{Next}(x, n, k)$
13:         else
14:             $x \leftarrow \mathit{NextBT}(x, n, k)$
15:     while $x \neq$ empty string
16:     return $\overline{r}$
```

5 Experimental Results

We used CPLEX 12.8 [17] as the linear programming solver, and we used the Python library pycddlib [27] for computing the extreme points of a the epigraph of the value function. As the SAT solver, we used Minicard implemented in the Python library Pysat [25] which have a native method to set a cardinality constraint. From Lemma 2 it follows that $\mathit{SMMR}^k_{\mathcal{W}}(\mathrm{A}) = \mathit{SMMR}^k_{\mathcal{W}}(\mathrm{UD}_{\mathcal{W}}(\mathrm{A}))$, where $\mathrm{UD}_{\mathcal{W}}(\mathrm{A})$ represents the set of undominated alternatives in $\mathcal{W}$. Thus, we generate sets of undominated random alternatives, where each alternative is defined as a vector of p rational numbers. In our experiments we have noticed that filtering out the dominated elements is a very worthwhile preliminary step. For example, generating 10 sets A with $|\mathrm{A}| = 25000$ and $p = 3$ using our random sets generator, we got an average of $|\mathrm{UD}_{\mathcal{W}}(\mathrm{A})| = 52.4$ alternatives.

In Table 1 we show the average computation time of $\mathit{SMMR}^k_{\mathcal{W}}(\mathrm{A})$ over 20 repetitions with $k \in \{2, 3\}$, $p \in \{3, 4\}$, and an input set of 50 undominated alternatives. Time SAT EPI and Time SAT LP indicate the average time in seconds to compute $\mathit{SMMR}^k_{\mathcal{W}}(\mathrm{A})$ using the SAT solver, where we compute $\mathit{SMR}_{\mathcal{W}}(\mathrm{B}, \mathrm{A})$ using the epigraph of the value function and a linear programming solver respectively. Time BF EPI and Time BF LP indicate the average time in seconds to compute $\mathit{SMMR}^k_{\mathcal{W}}(\mathrm{A})$ using the brute force algorithm, where also in this case we compute $\mathit{SMR}_{\mathcal{W}}(\mathrm{B}, \mathrm{A})$ using the epigraph of the value function and a linear programming solver respectively. Thus, the results in the first column relate to our best algorithm, and the results on the last column relate to the current state of the art. As we can see, with our algorithm we get a very significant improvement. Also, comparing EPI SAT with LP SAT, and EPI BF with LP BF, we can see that the computation of the setwise max regret using the epigraph of the value function seems to improve the performance with respect to the linear programming method, but given the experimental results in [2], the LP method is probably faster for $p \geq 7$.

Table 1: Average computation time of $\mathit{SMMR}^k_{\mathcal{W}}(\mathrm{A})$ over 20 repetitions varying k and p with an input set of 50 undominated alternatives and $\mathcal{W} = \mathcal{U}$.

k	p	Time[s] EPI SAT	Time[s] LP SAT	Time[s] EPI BF	Time[s] LP BF
2	3	2.113	31.144	26.984	413.718
2	4	3.821	32.258	46.557	424.904
3	3	5.144	53.537	667.38	6739.028
3	4	10.956	64.403	1191.759	6922.825

In Table 2 and Table 3 we show the average timing of our algorithm varying the number of user preferences and the size of the undominated input sets respectively. In both the tables we show also the average size of $\mathcal{W}'$ for the corresponding experiments. Note that with $k = 2$ we compute optimal queries represented by pairwise comparisons of alternatives that are often used in preference elicitation systems (see, e.g., [18, 1, 14, 12, 13, 15]). In Table 2, Λ represents the set of (consistent) user preferences (corresponding to linear constraints on the user preference space $\mathcal{U}$), each being of the form $aw_i + bw_j \geq cw_k$ for some random constants a, b and c. Each set of constraints Λ therefore defines a subset $\mathcal{W}_\Lambda$ of $\mathcal{U}$, which is in fact a compact convex polytope. For example, the set Λ could be elicited from a decision maker through a preference elicitation system; (in an iterative elicitation process, Λ can be the constraints generated from answers to earlier queries, and our algorithm generates an optimal next query to ask). As we can see in Table 2, setting for example $k = 2$ and $p = 4$ with $|\mathrm{UD}_{\mathcal{W}}(\mathrm{A})| = 50$, the computation time of $SMMR^k_{\mathcal{W}_\Lambda}(\mathrm{A})$ seems to be slightly decreasing with respect to the number of user preferences increases. In Table 3 we show the time performance of our main algorithm with respect to the size of the input set $\mathrm{UD}_{\mathcal{W}_\Lambda}(\mathrm{A})$ of undominated alternatives, growing approximately linearly.

Table 2: Average computation time of $SMMR^k_{\mathcal{W}}(\mathrm{A})$ over 20 repetitions varying the number of user preferences Λ with $|\mathrm{UD}_{\mathcal{W}}(\mathrm{A})| = 100$, $\mathcal{W} = \mathcal{U}$, $k = 2$ and $p = 4$.

$\lvert\mathrm{UD}_{\mathcal{W}_\Lambda}(\mathrm{A})\rvert$	$\lvert\Lambda\rvert$	Time EPI SAT	$\lvert\mathcal{W}'\rvert$
53.25	2	9.9	386.2
23.7	4	5.5	130.8
21.85	6	9.2	138.6
12.25	8	8.5	74.5
8.7	10	7.6	47.2

Table 3: Average computation time of $SMMR^k_{\mathcal{W}}(\mathrm{A})$ over 20 repetitions varying the size of the input set $\mathrm{UD}_{\mathcal{W}}(\mathrm{A})$ of undominated alternatives with $\mathcal{W} = \mathcal{U}$, $k = 2$, $p = 4$.

$\lvert\mathrm{UD}_{\mathcal{W}}(\mathrm{A})\rvert$	Time EPI SAT	$\lvert\mathcal{W}'\rvert$
100	6.8	397.2
150	12.1	710.3
200	17.8	906.6
250	24.1	1190.7
300	32.1	1435.1

In Figure 3 we show how our method scales with respect to k and p. The y-axis represents the average timing of our algorithm with a logarithmic scale. The x-axis represents the number of criteria $p \in \{2, \ldots, 6\}$. Each line represents the average time performance of 20 repetitions with an input set A of 50 undominated alternatives and varying $k \in \{2, \ldots, 5\}$. As we can see, the computational times increases exponentially with respect to k, reflecting the exponential growth of the number of subsets of A of cardinality k.

6 Conclusions

Interactive elicitation methods maintain a model of the user preferences that is revised incrementally by recording the answers to questions asked to the decision maker. In particular several works [30, 8, 10, 6, 7] have used (standard, single-item) minimax regret to provide a robust recommendation to decision maker.

The notion of setwise regret [28, 29] allows one to provide a sound and principled approach for generating, based on the current uncertainty about the decision maker's value function, a set of alternatives 1) to used as a recommendation set, and 2) to be used as a choice query to drive the elicitation forward.

Despite the attractiveness of setwise minimax regret, the high computational burden of this approach has limited its adoption in applications. To address this

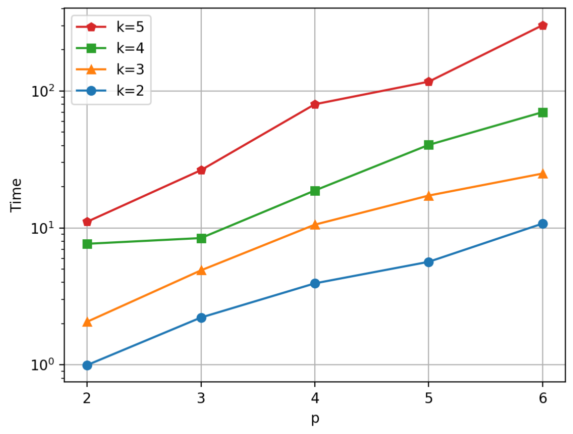

Fig. 3: Average computation time of $SMMR^k_{\mathcal{W}}(\mathrm{A})$ (y-axis) over 20 repetitions varying k and p with an input set of 50 undominated alternatives and $\mathcal{W} = \mathcal{U}$.

issue, in this paper we provided an efficient algorithm to compute exactly the setwise minimax regret for database problems, making use of a SAT solver to prune the search; this is valuable for generating queries and recommendation sets, to help a user find a most-preferred item in the database. Our algorithm may replace heuristic approaches when the query size is fairly small, since the complexity burden increases exponentially with respect to k. Note that in preference elicitation systems it is very common to ask binary queries, and with $k = 2$ we compute an optimal binary query with respect to the minimax regret criterion. Our approach could also be a starting point for new heuristic methods, based on exploring just the most promising parts of the search space.

We validated our approach in numerical experiments that showed a very substantial improvement with respect to the state of the art. Our implementation gives a proof of concept, using an algorithm of quite a simple structure. However, it can probably be speeded up a lot using various optimisations, and for example, a parallel evaluation of different branches whilst keeping track of a common upper bound.

Future works could involve testing our method in a preference elicitation context with the purpose of evaluating the quality of queries for the DM with respect to the setwise minimax regret criterion. Also, one could test the performances of our algorithm using an initial upper bound of the setwise minimax regret computed with an heuristic such as those in [28, 29]. It would be interesting also to explore a constraint programming approach for this problem, with a global constraint replacing the call to the SAT solver, potentially enabling propagation of literals to further reduce the search space.

References

1. Abbas, A.: Entropy methods for adaptive utility elicitation. IEEE Transactions on Systems, Science and Cybernetics **34**(2), 169–178 (2004)
2. Anonymous: Accepted paper, forthcoming (2020)
3. Benabbou, N., Di Diodoro, S.D.S., Perny, P., Viappiani, P.: Incremental preference elicitation in multi-attribute domains for choice and ranking with the borda count. In: International Conference on Scalable Uncertainty Management (SUM 2016). pp. 81–95. Springer (2016)
4. Benabbou, N., Lust, T.: An interactive polyhedral approach for multi-objective combinatorial optimization with incomplete preference information. In: International Conference on Scalable Uncertainty Management (SUM 2019). pp. 221–235. Springer (2019)
5. Benabbou, N., Perny, P.: Incremental weight elicitation for multiobjective state space search. In: Twenty-Ninth AAAI Conference on Artificial Intelligence (2015)
6. Benabbou, N., Perny, P., Viappiani, P.: Incremental elicitation of choquet capacities for multicriteria decision making. In: ECAI. pp. 87–92 (2014)
7. Benabbou, N., Perny, P., Viappiani, P.: Incremental elicitation of choquet capacities for multicriteria choice, ranking and sorting problems. Artificial Intelligence **246**, 152–180 (2017)
8. Boutilier, C., Patrascu, R., Poupart, P., Schuurmans, D.: Constraint-based optimization and utility elicitation using the minimax decision criterion. Artificial Intelligence **170**(8-9), 686–713 (2006)
9. Boyd, S., Vandenberghe, L.: Convex Optimization. Cambridge University Press, Cambridge, England (2004)
10. Braziunas, D.: Decision-theoretic elicitation of generalized additive utilities. Ph.D. thesis, University of Toronto (2012)
11. Braziunas, D., Boutilier, C.: Preference elicitation and generalized additive utility. In: Proceedings of AAAI. vol. 21 (2006)
12. Braziunas, D., Boutilier, C.: Minimax regret based elicitation of generalized additive utilities. In: Proceedings of UAI. pp. 25–32 (2007)
13. Braziunas, D., Boutilier, C.: Assessing regret-based preference elicitation with the utpref recommendation system. In: Proceedings of the 11th ACM conference on Electronic commerce. pp. 219–228 (2010)
14. Gajos, K., Weld, D.S.: Preference elicitation for interface optimization. In: Proceedings of the 18th Annual ACM Symposium on User Interface Software and Technology (UIST-05). pp. 173–182. Seattle, WA, USA (2005)
15. Guo, S., Sanner, S.: Real-time multiattribute Bayesian preference elicitation with pairwise comparison queries. In: Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, (AISTATS-10). pp. 289–296. Chia Laguna Resort, Sardinia, Italy (2010)
16. Howard, R.A.: Information value theory. IEEE Transactions on Systems Science and Cybernetics **2**(1), 22–26 (1966)
17. ILOG, I.: IBM ILOG CPLEX Optimization Studio, V12.8.0 (2017)
18. Iyengar, V.S., Lee, J., Campbell, M.: Q-Eval: Evaluating multiple attribute items using queries. pp. 144–153. Tampa, FL, USA (2001)
19. Kouvelis, P., Yu, G.: Robust discrete optimization and its applications, vol. 14. Springer Science & Business Media (2013)
20. Lu, T., Boutilier, C.: Robust approximation and incremental elicitation in voting protocols. In: Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence (IJCAI) (2011)

21. Price, R., Messinger, P.R.: Optimal recommendation sets: Covering uncertainty over user preferences. In: Proceedings, The Twentieth National Conference on Artificial Intelligence and the Seventeenth Innovative Applications of Artificial Intelligence Conference, July 9-13, 2005, Pittsburgh, Pennsylvania, USA. pp. 541–548 (2005)
22. Salo, A.A., Hamalainen, R.P.: Preference ratios in multiattribute evaluation (prime)-elicitation and decision procedures under incomplete information. IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans **31**(6), 533–545 (2001)
23. Savage, L.J.: The foundations of statistics. Courier Corporation (1972)
24. Sinz, C.: Towards an optimal cnf encoding of boolean cardinality constraints. In: International conference on principles and practice of constraint programming. pp. 827–831. Springer (2005)
25. Stoneback, R.: pysat 2.1.0 (2019), https://pypi.org/project/pysat/
26. Teso, S., Passerini, A., Viappiani, P.: Constructive preference elicitation by setwise max-margin learning. In: Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016. pp. 2067–2073 (2016)
27. Troffaes, M.C.M.: pycddlib python wrapper for komei fukuda's cddlib (2018), https://pycddlib.readthedocs.io/en/latest/
28. Viappiani, P., Boutilier, C.: Regret-based optimal recommendation sets in conversational recommender systems. In: Proceedings of the third ACM conference on Recommender systems (RecSys). pp. 101–108. ACM (2009)
29. Viappiani, P., Boutilier, C.: On the equivalence of optimal recommendation sets and myopically optimal query sets. Artificial Intelligence p. 103328 (2020)
30. Wang, T., Boutilier, C.: Incremental utility elicitation with the minimax regret decision criterion. In: Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI). vol. 3, pp. 309–316 (2003)
31. White, C.C., Sage, A.P., Dozono, S.: A model of multiattribute decisionmaking and trade-off weight determination under uncertainty. IEEE Transactions on Systems, Man, and Cybernetics (2), 223–229 (1984)