Efficient Exact Computation of Setwise Minimax Regret for Interactive Preference Elicitation

Federico Toffano
Insight Centre for Data Analytics,
School of Computer Science and IT
University College Cork, Ireland
federico.toffano@insight-centre.org

Paolo Viappiani
UMR7606 CNRS, LIP6
Sorbonne Université, 4 pl. Jussieu,
75005 Paris, France
paolo.viappiani@lip6.fr

Nic Wilson
Insight Centre for Data Analytics,
School of Computer Science and IT
University College Cork, Ireland
nic.wilson@insight-centre.org

ABSTRACT

A key issue in artificial intelligence methods for interactive 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 corresponds to 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 model of user utility as a weighted sum of the criteria; in this case when the user makes a choice, an agent learns 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

Interactive Preference Elicitation; Setwise Minimax Regret; Human-Agent Interaction

ACM Reference Format:
Federico Toffano, Paolo Viappiani, and Nic Wilson. 2021. Efficient Exact Computation of Setwise Minimax Regret for Interactive Preference Elicitation . In *Proc. of the 20th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2021), Online, May 3–7, 2021*, IFAAMAS, 9 pages.

1 INTRODUCTION

Multi-attribute utility theory (MAUT) [23] is a field of study involving decision-making problems where the available alternatives are evaluated w.r.t. (with respect to) a fixed number of conflicting criteria. The main purpose of MAUT is to define a value function for a specific decision-maker, which represents his/her preferences with respect to the criteria used to evaluate the alternatives of the decision problem. Of particular interest in this paper are interactive preference elicitation methods for MAUT models [7, 8, 11–13, 20, 29, 31, 33] in which an agent iteratively asks questions to a user to reduce the uncertainty of his value function and recommend an alternative.

A principled approach for selecting recommendations with uncertain value function is to use minimax regret. Minimax regret (*MMR*) selects as recommendation an alternative that minimises the worst case loss w.r.t. feasible parameters of the value function. The practical effectiveness of regret-based recommendations has been shown in numerous works (see, e.g., [8, 10, 33]) including a study carried out with real users [13].

In [31] and [32] the authors generalized the concept of Minimax Regret, defining the *Setwise Max Regret* (*SMR*) which is used to evaluate the maximum regret of a set of alternatives, and the setwise minimax regret (*SMMR*), which is used to select an optimal recommendation set of a given cardinality k w.r.t. *SMR*, i.e., a set of k alternatives that minimises the worst case loss w.r.t. feasible parameters of the value function. A remarkable property of *SMMR* is that an optimal recommendation set is also a myopically optimal query set, that is, the set maximizes an analogue of value of information [16] in a distribution-less sense. This gives a compelling reason to use this measure in an interactive user-agent system, with a combined preference elicitation and recommendation purpose: the agent proposes a set of recommended items, the user picks the one he prefers, then the agent updates the model and shows a new set of items; and this proceeds until a termination condition is met (max regret being at most a threshold value; or simply when the user is satisfied). However, optimising setwise regret is computationally very demanding; 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 [31, 32],

This paper addresses the problem of computing *SMMR* exactly for database problems (e.g., when a list of items with a fixed number of evaluation criteria is readily available, as opposed to configuration problems where alternatives need to be constructed through constraint satisfaction). The main contribution is an efficient algorithm for computing an optimal set of a given cardinality k w.r.t. the setwise max regret criterion, i.e., a set optimising *SMMR*, which can then be used for both elicitation and recommendation purposes.

Our method relies on search; nodes in the search tree correspond to sets of alternatives with cardinality up to k, and leaves correspond to sets with cardinality exactly k. Pruning is done when we are sure that no extension of the current set with cardinality less than k can beat the set with cardinality k with minimum *SMR* found so far; to check this condition we use a SAT solver with cardinality constraints. This idea is combined with a fast subroutine to compute the setwise max regret of a set of alternatives, using the extreme points of the epigraph of the value function (instead of using linear programming techniques as in previous works).

The paper is organized as follows. In Section 2 we state our general assumptions, we recall the definitions of minimax regret

Proc. of the 20th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2021), U. Endriss, A. Nowé, F. Dignum, A. Lomuscio (eds.), May 3–7, 2021, Online.

and its setwise extensions, and provide some basic properties. In Section 3 we describe the ideas behind our algorithm and its main components; in Section 4 we provide a detailed description of the algorithm to compute *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, we 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 assumed to be endowed with a utility or value function u_w, mapping from A to $\mathbb{R}$; w represents the parameters of the value function (a specific choice of w uniquely determines the value function). The goal is to pick an alternative x maximising $u_w(x)$; 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 parameters w of a DM's value function u_w.

Minimax regret. The *Minimax Regret* (*MMR*) [19, 25] criterion is frequently used to solve decision problems under uncertainty. More recently, it has been used in Artificial Intelligence to evaluate alternatives as potential recommendations, where the uncertainty refers to the parameters of the decision model [11, 24]. When considering a single recommendation, alternatives can be evaluated according to the *max regret*, quantifying the worst-case loss due to utility uncertainty:

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

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

where we let $\mathrm{Val}_{\mathrm{A}}(w) = \max_{\beta \in \mathrm{A}} u_w(\beta)$ be the DM's value function for A defined as the maximum value that can be obtained from any alternative $\beta \in \mathrm{A}$ w.r.t. the value function u_w. The *minimax regret* represents the minimum worst-case loss i.e., minimum max regret:

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

By recommending to the decision maker an alternative associated with minimax regret, i.e., alternative $\alpha^* \in \arg\min_{\alpha \in \mathrm{A}} MR_{\mathcal{W}}(\alpha, \mathrm{A})$, we provide robustness in the 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., [5, 12, 33]), or the elicitation of preferences for ranking and voting problems (see, e.g., [3, 7, 21]).

Example 2.1. Consider a bi-criteria problem with the set of alternatives $\mathrm{A} = \{\alpha_1 = (4,4), \alpha_2 = (2,10), \alpha_3 = (10,2)\}$. The two criteria evaluating each alternative could e.g., represent the cost and quality of a certain product, where the higher the value, the better. The associated value functions are shown in Figure 1: $u_w(\alpha_j) = w \cdot \alpha_j = \sum_{i=1}^{2} w_i \alpha_j(i)$ with $w \in \mathcal{W} = \{w \in R^2 : w_i \geq 0, \sum_{i=1}^{2} w_i = 1\}$ $MR_{\mathcal{W}}(\alpha_1, \mathrm{A}) = 6$ is maximised with $w_1 = 0$ and $w_1 = 1$, $MR_{\mathcal{W}}(\alpha_2, \mathrm{A}) = 8$ is maximised with $w_1 = 1$, and $MR_{\mathcal{W}}(\alpha_1, \mathrm{A}) = 8$ is maximised with $w_1 = 0$. Thus, $MMR_{\mathcal{W}}(\mathrm{A}) = 6$.

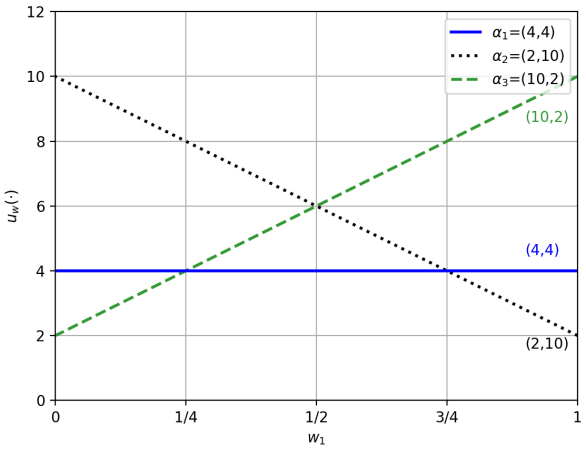

Figure 1: Plot of the linear value functions $u_w(\alpha_i) = w \cdot \alpha_i$ for the alternatives $\alpha_1 = (4,4)$ (blue solid), $\alpha_2 = (2,10)$ (black dotted) and $\alpha_3 = (10,2)$ (green dashed) with $w \in \mathcal{W} = \{w \in R^2 : w_i \geq 0, w_1 + w_2 = 1\}$.

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 [22] 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 $\mathrm{Val}_{\mathrm{B}}(w) = \max_{\alpha \in \mathrm{B}} u_w(\alpha)$ when the "true" value function is dictated by parameters w. The regret of a set B w.r.t. 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, $\mathrm{Val}_{\mathrm{A}}(w) - \mathrm{Val}_{\mathrm{B}}(w)$. The *setwise max regret* (*SMR*) [31, 32] of a subset B of the finite set of alternatives A, w.r.t. the parameter space $\mathcal{W}$, is then defined as the maximum of this difference:

$$SMR_{\mathcal{W}}(\mathrm{B}, \mathrm{A}) = \max_{w \in \mathcal{W}} (\mathrm{Val}_{\mathrm{A}}(w) - \mathrm{Val}_{\mathrm{B}}(w)). \quad (4)$$

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, by Equation 2, *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 w.r.t. $\mathcal{W}$. Thus, the *setwise minimax regret* (*SMMR*) [31, 32] of size k w.r.t. $\mathcal{W}$ 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. 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 [31, 32] that optimal recommendation sets w.r.t. *SMMR* are also myopically optimal in an elicitation sense, as they ensure the highest worst-case (w.r.t. the possible query's responses) reduction of minimax regret *a posteriori*.

Example 2.2. Consider the set of alternatives $\mathrm{A} = \{\alpha_1, \alpha_2, \alpha_3\}$, where $\alpha_1 = (4, 4), \alpha_2 = (2, 10), \alpha_3 = (10, 2)$, whose value function $u_w(\alpha_i) = w \cdot \alpha_i$ with $w \in \mathcal{W} = \{w \in R^2 : w_i \geq 0, w_1 + w_2 = 1\}$ 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, $SMMR^2_{\mathcal{W}}(\mathrm{A}) = 0$.

Optimisation of setwise regret. Computation of *SMMR* differs according 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 [4, 8, 12]; for example, this is the case when configuring computer parts to obtain a customized laptop. In this type of problem setwise regret can be optimised with a mixed-integer program and solved by techniques such as Bender's decomposition and constraint generation [31, 32].

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., [7, 12, 13]); for example the problem of choosing from a catalogue of already assembled laptops. In this case the straightforward approach to optimise *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 [31, 32] 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 of 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 w.r.t. 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 w.r.t. w, where $\alpha \in \mathrm{A} \subset R^p$ and $w \in \mathcal{U}$ and

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

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, $\mathcal{W}_\Lambda$ 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., [34]). Thus, $\mathcal{W}_\Lambda$ is the set of elements of $\mathcal{U}$ that satisfy the constraints induced by Λ.

Basic properties of setwise regret. We now give some basic properties of setwise regret that we will use later. The definitions easily imply that $SMR_{\mathcal{W}}(\mathrm{B},\mathrm{A})$ is monotone, w.r.t. set inclusion, in both B and $\mathcal{W}$, and that $SMMR^k_{\mathcal{W}}$ is monotone in k.

Lemma 2.3. *$SMR_{\mathcal{W}}(B, A)$ is monotonically decreasing in B, and monotonically increasing in $\mathcal{W}$, i.e., if $B' \supseteq B$ and $\mathcal{W}' \subseteq \mathcal{W}$, then $SMR_{\mathcal{W}'}(B', A) \leq SMR_{\mathcal{W}}(B, A)$. $SMMR^k_{\mathcal{W}}(A)$ is monotonically decreasing in k, i.e., for $1 \leq k \leq k'$, $SMMR^k_{\mathcal{W}}(A) \geq SMMR^{k'}_{\mathcal{W}}(A)$.*

An alternative $\alpha \in \mathrm{A}$ is said to be *$\mathcal{W}$-dominated in A* if there exists another alternative $\beta \in \mathrm{A}$ 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; otherwise, α is *$\mathcal{W}$-undominated in A*. Let $\mathrm{UD}_{\mathcal{W}}(\mathrm{A}) \subseteq \mathrm{A}$ be the set of elements of A that are $\mathcal{W}$-undominated in A. 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. This follows because for any $w \in \mathcal{W}$, and any set of alternatives A, we have $\mathrm{Val}_{\mathrm{UD}_{\mathcal{W}}(\mathrm{A})}(w) = \mathrm{Val}_{\mathrm{A}}(w)$.

Lemma 2.4. *$SMR_{\mathcal{W}}(B, A) = SMR_{\mathcal{W}}(\mathrm{UD}_{\mathcal{W}}(B), \mathrm{UD}_{\mathcal{W}}(A))$ and, for any $k \geq 1$, $SMMR^k_{\mathcal{W}}(A) = SMMR^k_{\mathcal{W}}(\mathrm{UD}_{\mathcal{W}}(A))$.*

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 $SMR_{\mathcal{W}'}(\mathrm{B},\mathrm{A}) \geq \overline{r}$ holds for all supersets B of C with cardinality k, where $\overline{r}$ is the current upper bound of $SMMR^k_{\mathcal{W}}(\mathrm{A})$. If this holds then, by Lemma 2.3, $SMR_{\mathcal{W}}(\mathrm{B},\mathrm{A}) \geq \overline{r}$ 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 bound the setwise max regret of a set of subsets of A simultaneously using 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 $Len(x)$ be the length of x. Let us label A as $\{\alpha_1, \ldots, \alpha_n\}$. 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 $Len(y) \geq Len(x)$ and the first $Len(x)$ 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 using a depth-first search on a binary tree.

Example 3.1. Let $A = \{\alpha_1, \ldots, \alpha_5\}$ be a set of $n = 5$ elements, and let $k = 3$. The complete string $z = 01101$ represents the subset $B_z = \{\alpha_2, \alpha_3, \alpha_5\}$. 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 $B_y \in \mathcal{B}_x$ is such that $SMR_{\mathcal{W}}(B_y, A) < \overline{r}$, 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 $SMMR^k_{\mathcal{W}}(A)$.

Lemma 3.2. *Let $\overline{r}$ be an upper bound of $SMMR^k_{\mathcal{W}}(A)$, $\mathcal{W}' \subseteq \mathcal{W}$ and $B \subseteq A$. For $w \in \mathcal{W}$, let Γ_w be the set of $\alpha \in A$ such that $Val_A(w) - u_w(\alpha) < \overline{r}$. Then $SMR_{\mathcal{W}'}(B, A) < \overline{r}$ 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 $SMR_{\mathcal{W}'}(B, A) < \overline{r}$ if and only if $Val_A(w) - Val_B(w) < \overline{r}$ for all $w \in \mathcal{W}'$, which is if and only if for all $w \in \mathcal{W}'$ there exists $\alpha \in B$ such that $Val_A(w) - u_w(\alpha) < \overline{r}$, with the latter condition being equivalent to $\alpha \in \Gamma_w$ since $B \subseteq A$. □

To check if there exists a set $B_y \in \mathcal{B}_x$ such that $SMR_{\mathcal{W}'}(B_y, A) < \overline{r}$, we define a SAT problem with cardinality constraint c (see, e.g., [27]), where the cardinality constraint is used to define the size k of the sets in $\mathcal{B}_x$.

Example 3.3. 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 with $i = \{1, 2, 3\}$ are $\{0, 1\}$-valued variables (with 1 meaning TRUE), 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 $\alpha_i \in A$. These are used to reason about the unknown sets B_y in $\mathcal{B}_x$, which we want to be such that $SMR_{\mathcal{W}'}(B_y, A) < \overline{r}$. Then $X_i = 1$ means that $B_y \ni \alpha_i$. Given a partial string x, we define the corresponding SAT problem with cardinality constraint as follows:

(1) The cardinality constraint $|B_y| = k$ is expressed as $\sum_{\alpha_i \in A} X_i = k$.
(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 $\alpha \in A$ such that $Val_A(w) - u_w(\alpha) < \overline{r}$.

This SAT problem is satisfiable if and only if there exists $B_y \in \mathcal{B}_x$ such that for all $w \in \mathcal{W}'$ there exists $\alpha \in B_y$ such that $\alpha \in \Gamma_w$, which (by Lemma 3.2) is if and only if there exists $B_y \in \mathcal{B}_x$ such that $SMR_{\mathcal{W}'}(B_y, A) < \overline{r}$. Therefore, if the SAT problem is unsatisfiable, then for each $B_y \in \mathcal{B}_x$, $SMR_{\mathcal{W}'}(B_y, A) \geq \overline{r}$, and thus (by Lemma 2.3) $SMR_{\mathcal{W}}(B_y, A) \geq \overline{r}$. 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 $SMR_{\mathcal{W}}(B_y, A)$ for $B_y \in \mathcal{B}_x$.

Example 3.4. Consider the set of alternatives $A = \{\alpha_1 = (4, 4), \alpha_2 = (2, 10), \alpha_3 = (10, 2)\}$ whose value function $u_w(\alpha_i) = w \cdot \alpha_i$ with $w \in \mathcal{W} = \{w \in R^2 : w_i \geq 0, \sum_{i=1}^{2} w_i = 1\}$ 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 constraints $c = 2$ (the cardinality constraint) and $X_1 = 1$. It is easy to see that in this case the SAT problem is unsatisfiable; hence, we can avoid the computation of $SMR_{\mathcal{W}}(\{\alpha_1, \alpha_2\}, A)$ and $SMR_{\mathcal{W}}(\{\alpha_1, \alpha_3\}, A)$. In fact, the subset B of A cardinality 2 that minimises $SMR_{\mathcal{W}}(B, A)$ is $B = \{\alpha_2, \alpha_3\}$.

3.2 Computation of max regret

With linear value functions u_w, a standard method to compute the setwise max regret $SMR_{\mathcal{W}}(B, A)$ of a set $B \subseteq A$ consists of the evaluation of a linear programming (LP) problem for each $\alpha_i \in A$ (see [31]). Briefly, for $\alpha_i \in A$ we have $SMR_{\mathcal{W}}(B, \{\alpha_i\}) = \max_{w \in \mathcal{W}}(\alpha_i \cdot w - Val_B(w))$, which means that we can compute $SMR_{\mathcal{W}}(B, \{\alpha_i\})$ 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 $\beta \in B$. We can then compute $SMR_{\mathcal{W}}(B, A)$ as $\max_{\alpha_i \in A} SMR_{\mathcal{W}}(B, \{\alpha\})$. However, here we also make use of a method, described briefly in the next paragraph, which was recently developed for computing $SMR_{\mathcal{W}}(B, A)$ [30].

With linear value functions u_w, the max regret of an alternative β can be easily computed evaluating only the extreme points $Ext(\mathcal{W})$ of $\mathcal{W}$, i.e., $SMR_{\mathcal{W}}(\beta, A) = SMR_{Ext(\mathcal{W})}(\beta, A)$. For the setwise max regret of a set B instead, we need to evaluate the extreme points points of $\mathcal{W}_\beta$ for each $\beta \in B$, where $\mathcal{W}_\beta = \{w \in \mathcal{W} : \beta \cdot w \geq \beta_j \cdot w, \forall \beta_j \in B\}$. Let $\gamma(\mathcal{W}, B) = \{(w, r) : w \in \mathcal{W},\ \beta \cdot w \leq r\ \forall \beta \in B\}$; this can be seen to be the *epigraph* [9] of the value function Val_B on $\mathcal{W}$. Let $UE(B) = \bigcup_{\beta \in B} Ext(\mathcal{W}_\beta)$ be the union of the sets of extreme points $Ext(\mathcal{W}_\beta)$ for each $\beta \in B$. In [30] it is shown that $UE(B)$ equals the projection in $\mathcal{W}$ of the set of extreme points $Ext(\gamma(\mathcal{W}, B))$ of $\gamma(\mathcal{W}, B)$. Also, for each $(w, r) \in Ext(\gamma(\mathcal{W}, B))$ we have that $r = Val_B(w)$. This implies that $SMR_{\mathcal{W}}(B, A)$ can be computed as:

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

Example 3.5. Consider the example in Figure 2 with $A = \{(2, 8), (8, 2), (6, 6)\}$ and let $B = \{(2, 8), (8, 2)\}$. The extreme points of the region of $\mathcal{W}$ in which $(2, 8)$ and $(8, 2)$ are optimal are $Ext(\mathcal{W}_{(2,8)}) =$

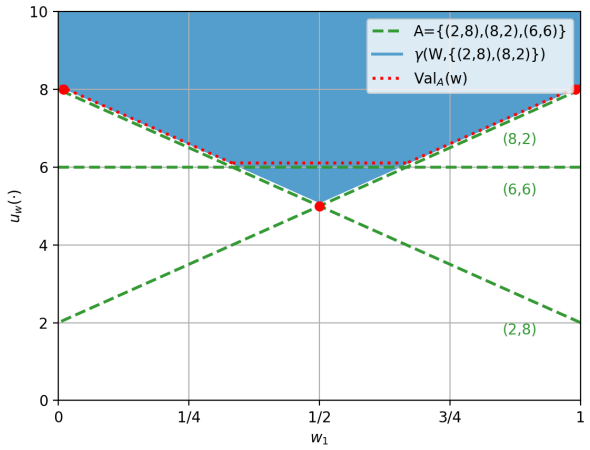

Figure 2: Value function $u_w(\alpha_i) = w \cdot \alpha_i$ for each alternative in $\mathbf{A} = \{(2,8),(8,2),(6,6)\}$ (green dashed) where $w \in \mathcal{W} = \{w \in R^2 : w_i \geq 0, \sum_{i=1}^{2} w_i = 1\}$. Note that we show $u_w(\alpha_i)$ only w.r.t. w_1 since $w_2 = w_1 - 1$. The red dotted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

$\{(0,1), (\frac{1}{2}, \frac{1}{2})\}$ and $Ext(\mathcal{W}_{(8,2)}) = \{(\frac{1}{2}, \frac{1}{2}), (1,0)\}$. The set of extreme points of the epigraph of B is $Ext(\gamma(\mathcal{W}, \mathrm{B})) = \{((0,1),8), ((\frac{1}{2}, \frac{1}{2}),5)((1,0),8)\}$, and the corresponding projection in $\mathcal{W}$ is $UE(\mathrm{B}) = \{(0,1), (\frac{1}{2}, \frac{1}{2}), (1,0)\}$. Then $SMR_{\mathcal{W}}(\mathrm{B}, \mathrm{A}) = \max((8-8),(6-5),(8-8)) = 1$ which is maximised with $w = (\frac{1}{2}, \frac{1}{2})$, and $\mathrm{Val_B}((\frac{1}{2}, \frac{1}{2})) = 5$.

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. Note that we are not interested in all the possible binary strings of length n, but instead we want to generate complete strings x (i.e., with k ones) and the corresponding sub-strings since these will represent 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 leftmost 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: Setting $NextBT(x_j, n, k) = x_{j+1}$ corresponds to the backtracking case for the j-th string. With $NextBT(x_j, n, k)$ we move from the current node representing x_j towards the root until we find an edge e with value zero. Let v be the parent of e. We define $NextBT(x, n, k)$ 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 $SMR_{\mathcal{W}}(\mathrm{B}, \mathrm{A}) \geq \overline{r}$ for all $\mathrm{B} \in \mathcal{B}_{x_j}$. Roughly speaking, we use $NextBT(x_j, n, k)$ 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: Setting $Next(x_j, n, k) = x_{j+1}$ corresponds to the non-backtracking case for the j-th string. With $Next(x_j, n, k)$ 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 $SMR_{\mathcal{W}}(\mathrm{B}, \mathrm{A}) \geq \overline{r}$ for all $\mathrm{B} \in \mathcal{B}_{x_j}$. Roughly speaking, we use $Next(x_j, n, k)$ 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 $\mathcal{W}' = \emptyset$ and $\overline{r} = \infty$. Then for each *SMR* computation of a subset B of A, if $SMR_{\mathcal{W}}(\mathrm{B}, \mathrm{A})$ is greater than the current upper bound $\overline{r}$ of $SMMR^{k}_{\mathcal{W}}(\mathrm{A})$, we update $\mathcal{W}'$. Depending on which method we use for the computation of *SMR* (see Section 3.2), we use one of the following method to update $\mathcal{W}'$:

(1) Epigraph of the value function: $\mathcal{W}' = \mathcal{W}' \cup UE(\mathrm{B})$ where $UE(\mathrm{B})$ is the projection of $Ext(\gamma(\mathcal{W}, \mathrm{B}))$ to $\mathcal{W}$.
(2) Linear programming: $\mathcal{W}' = \mathcal{W}' \cup \{w_1, \ldots, w_n\}$ where $w_i \in \mathcal{W}$ is the preference model in which $SMR_{\mathcal{W}}(\mathrm{B}, \{\alpha_i\})$ is maximised.

One could update $\mathcal{W}'$ using only the point $w \in \mathcal{W}$ in which $SMR_{\mathcal{W}}(\mathrm{B}, \mathrm{A})$ 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.

Incremental updating of 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 $i \in \{Len(x)+1, \ldots, Len(y)\}$. 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 that $n = 5$, $k = 3$ 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, so 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 Algorithm 1 we combine the concepts presented in Section 3, defining the an iterative procedure for computing $SMMR^k_{\mathcal{W}}(\mathrm{A})$. The inputs of our algorithm are:

(1) A finite set A of alternatives represented with p-dimensional vectors of reals.
(2) The DM's preference state space $\mathcal{W} \subseteq \{w \in R^p : w_i \geq 0, \sum_{i=1}^{p} w_i = 1\}$ representing the possible parameters w of the value function u_w.
(3) An integer $k \leq |\mathrm{A}|$ representing the cardinality of the subsets of A that we want to evaluate.

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 $SMMR^k_{\mathcal{W}}(\mathrm{A})$, i.e., the minimum value of $SMR_{\mathcal{W}}(\mathrm{B}, \mathrm{A})$ over all subsets B of A of cardinality k.

The notion $\mathrm{SAT}(x, k, \mathrm{A}, \mathcal{W}', \overline{r})$ refers to the SAT procedure described in Section 3.1 which returns *true* iff there exists $\mathrm{B}_y \in \mathcal{B}_x$ such that $SMR_{\mathcal{W}'}(\mathrm{B}_y, \mathrm{A}) < \overline{r}$. $UE(\mathrm{B}_x)$ is the projection of $Ext(\gamma(\mathcal{W}, \mathrm{B}_x))$ to $\mathcal{W}$.

Algorithm 1 EPI SAT

```
1:  procedure SMMR(k, A, W)
2:      W' ← ∅
3:      r̄ ← ∞
4:      x ← string of n − k zeros concatenated with k ones
5:      do
6:          if Len(x) = n then
7:              if W' = ∅ or SMR_{W'}(B_x, A) < r̄ then
8:                  SMR ← SMR_W(B_x, A)
9:                  if SMR < r̄ then
10:                     r̄ ← SMR
11:                     W' ← W' ∪ UE(B_x)
12:             x ← NextBT(x, n, k)
13:         else if SAT(x, k, A, W', r̄) then
14:             x ← Next(x, n, k)
15:         else
16:             x ← NextBT(x, n, k)
17:     while x ≠ empty string
18:     return r̄
```

5 EXPERIMENTAL RESULTS

In our experimental results we used CPLEX 12.8 [17] as the linear programming solver, and we used the Python library pycddlib [14] for computing the extreme points of the epigraph of the value function. As the SAT solver, we used Minicard implemented in the Python library Pysat [28] which has a native method to set a cardinality constraint. All experiments were performed on a computer facilitated by two Intel Xeon E5620 2.40GHz processors and 32 GB RAM. For the sake of reproducibility, we made the code available[1].

[1] https://github.com/federicotoffano/SMMR

From Lemma 2.4 we have $SMMR^k_{\mathcal{W}}(\mathrm{A}) = SMMR^k_{\mathcal{W}}(\mathrm{UD}_{\mathcal{W}}(\mathrm{A}))$, where $\mathrm{UD}_{\mathcal{W}}(\mathrm{A})$ represents the set of $\mathcal{W}$-undominated alternatives in A. In our experiments we noticed that filtering out dominated alternatives can be a very worthwhile preliminary step. For example, generating 10 random sets A with $|\mathrm{A}| = 25000$ and $p = 4$, we got an average of $|\mathrm{UD}_{\mathcal{W}}(\mathrm{A})| = 220$ alternatives computed in roughly 10 seconds.

In Table 1 we show the average computation time of $SMMR^k_{\mathcal{W}}(\mathrm{A})$ over 20 repetitions with $k \in \{2, 3\}$, $p \in \{3, 4\}$, and an input set of 50 random undominated alternatives. Time SAT EPI and Time SAT LP indicate the average time in seconds to compute $SMMR^k_{\mathcal{W}}(\mathrm{A})$ using the SAT solver, where we compute $SMR_{\mathcal{W}}(\mathrm{B}, \mathrm{A})$ 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 $SMMR^k_{\mathcal{W}}(\mathrm{A})$ using a straightforward algorithm evaluating all the subsets of size k, where also in this case we compute $SMR_{\mathcal{W}}(\mathrm{B}, \mathrm{A})$ using the epigraph of the value function, and a linear programming solver, respectively. Thus, the results on the first column relate to our best algorithm, and the results on the last column relate to a straightforward algorithm based on the definition of $SMMR^k_{\mathcal{W}}(\mathrm{A})$; 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 (rather than the LP method) seems to improve the performance.

In Tables 2 and 3 we show the average timing of our algorithm EPI SAT with $k = 2$ and $p = 4$, varying the number of user preferences and the size of the undominated input sets. The SMMR with $k = 2$ is of particular interest since the corresponding set with minimum setwise max regret is a myopically optimal binary query [32]. Binary queries have been often used in preference elicitation systems (see, e.g., [1, 4, 12, 13, 15, 18, 26]), and our algorithm may be used as a query selection strategy in these contexts when the set of alternatives is not too large. 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 $\alpha \cdot w \geq \beta \cdot w$. This constraint can be interpreted as a preference of alternative α to alternative β. Each set of constraints Λ therefore defines a subset $\mathcal{W}_\Lambda$ of $\mathcal{U}$ which is a convex polytope. The sets of constraints Λ used in our experiments are generated supposing a random user preference model w, and simulating an iterative elicitation process with binary queries. At each iteration, we simulate the user preference w.r.t. a myopically optimal binary query $Q = \{\alpha, \beta\}$ computed with our algorithm, and we use the simulated user preference model w to define the sign of the inequality associated with Q. The resulting constraint will then be added to Λ. As we can see in Table 2, setting for example $k = 2$ and $p = 4$ with $|\mathrm{UD}_{\mathcal{W}}(\mathrm{A})| = 500$ alternatives, the number of undominated alternatives rapidly decreases when increasing the number of constraints, with a consequent improving of the computation time of $SMMR^k_{\mathcal{W}_\Lambda}(\mathrm{A})$. In fact, as we can see in Table 3, the time performance of our main algorithm seems to grow exponentially w.r.t. the size of the input set $\mathrm{UD}_{\mathcal{W}_\Lambda}(\mathrm{A})$ of undominated alternatives. In Table 2 we reported also the computation time to filter out dominated alternatives, and, as we can see, it is a very fast operation in comparison with the computation of *SMMR*.

Table 1: Average computation time in seconds to compute $SMMR^k_{\mathcal{W}}(\mathbf{A})$ with EPI SAT, LP SAT, EPI BF and LP BF 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	0.17	8.5	4.15	413.71
2	4	0.19	6.69	5.88	424.90
3	3	0.43	17.46	76.43	6739.02
3	4	0.63	17.72	115.97	6922.82

Table 2: Average time in seconds to compute $SMMR^k_{\mathcal{W}}(\mathbf{A})$ with EPI SAT over 20 repetitions varying the number of user preferences Λ with $|\mathrm{UD}_{\mathcal{W}}(\mathbf{A})| = 500$, $\mathcal{W} = \mathcal{U}$, $k = 2$ and $p = 4$. $|\mathcal{W}'|$ represents the average number of user preference models used to evaluate subsets of A with SAT, and UD represents the algorithm to filter out dominated alternatives.

$\lvert\mathrm{UD}_{\mathcal{W}_\Lambda}(\mathrm{A})\rvert$	$\lvert\Lambda\rvert$	Time[s] EPI SAT	$\lvert\mathcal{W}'\rvert$	Time[s] UD
500	0	16.18	68.4	1.467
267.7	1	5.86	61.8	0.460
180.9	2	2.25	56.8	0.154
140.45	3	1.36	59	0.067
100.55	4	0.76	47.7	0.036
65.15	5	0.49	45.5	0.021
90.6	6	0.88	46.6	0.016
34.1	7	0.26	42.8	0.013
40.8	8	0.43	37.2	0.008
24.45	9	0.24	34.4	0.006

Table 3: Average computation time in seconds to compute $SMMR^k_{\mathcal{W}}(\mathbf{A})$ with EPI SAT over 20 repetitions varying the size of the input set $\mathrm{UD}_{\mathcal{W}}(\mathbf{A})$ of undominated alternatives with $|\Lambda| = 0$ (i.e., $\mathcal{W} = \mathcal{U}$), $k = 2$, $p = 4$. $|\mathcal{W}'|$ represents the average number of user preference models used to evaluate subsets of A with SAT.

$\lvert\mathrm{UD}_{\mathcal{W}}(\mathrm{A})\rvert$	Time[s] EPI SAT	$\lvert\mathcal{W}'\rvert$
100	0.37	46.9
200	1.45	54.3
300	5.29	58.6
400	9.69	62.2
500	28.27	56.9
600	35.95	72.5

In Figure 3 we show how EPI SAT scales w.r.t. 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 100 undominated alternatives and varying $k \in \{2, \ldots, 6\}$. As we can see, the computation times increases exponentially w.r.t. k, reflecting the exponential growth of the number of subsets of A of cardinality k. The computation time

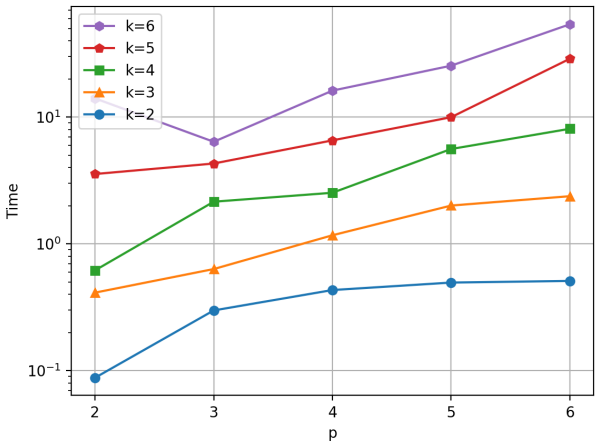

Figure 3: Average computation time in seconds to compute $SMMR^k_{\mathcal{W}}(\mathbf{A})$ with EPI SAT (y-axis) over 20 repetitions varying k and p with an input set of 100 undominated alternatives and $\mathcal{W} = \mathcal{U}$.

Table 4: Average computation time in seconds to compute $SMMR^k_{\mathcal{W}}(\mathbf{A})$ with EPI SAT over 20 repetitions varying p with $|\mathrm{UD}_{\mathcal{W}}(\mathbf{A})| = 100$, $|\Lambda| = 0$ (so $\mathcal{W} = \mathcal{U}$) and $k = 4$. $|\mathcal{W}'|$ represents the average number of user preference models used to evaluate subsets of A with SAT.

p	Time[s] EPI SAT	$\lvert\mathcal{W}'\rvert$
2	0.62	20.7
3	2.15	77.6
4	2.53	152.2
5	5.61	301.9
6	8.09	507.5

seems to grow exponentially also w.r.t. p, and this may be due to the exponential growth of the number of extreme points of $\mathcal{W}$ w.r.t. p (cf. Table 4 for $k = 4$).

We also tested our algorithm with the databases used in the experimental results of [32]:

(1) **Synthetic**: A synthetic database of 81 undominated alternatives evaluated with $p = 12$ criteria with binary domain

Table 5: Computation of $SMMR^k_{\mathcal{W}}(\mathrm{A})$ with the databases considered in the experimental results of [32]. The first four columns show information regarding the input databases. The fifth and the sixth columns show the performances of filtering out dominated elements. The last two columns show the time performance of our algorithm EPI SAT and the method used in [32] (whose results are shown in their Table 8.)

A	k	p	$\lvert\mathrm{A}\rvert$	$\lvert\mathrm{UD}_{\mathcal{W}}(\mathrm{A})\rvert$	$\mathrm{UD}_{\mathcal{W}}(\mathrm{A})$[s]	EPI SAT[s]	[32][s]
Synthetic	2	12	81	81	0.22	0.09	19.47
Synthetic	3	12	81	81	0.22	0.93	-
Synthetic	4	12	81	81	0.22	5.86	-
Synthetic	5	12	81	81	0.22	17.72	-
Rental	2	23	187	100	0.57	0.19	0.17
Rental	3	23	187	100	0.57	0.28	-
Rental	4	23	187	100	057	4.01	-
Rental	5	23	187	100	0.57	172.32	-
Boston	2	14	506	475	3.29	60.73	277.53
Boston	3	14	506	475	3.29	182.18	-

$\{0, 1\}$. This is a small dataset used in [32] to test the computation of *SMMR* with an exact algorithm.

(2) **Rental**: Real university student rental database with 187 alternatives evaluated with $p = 23$ criteria with four of them being real values in the interval $[0, 1]$ and the remaining being binary values $\{0, 1\}$.

(3) **Boston**: Boston housing database [2] with 506 alternatives evaluated with $p = 14$ real values. In this case, the alternatives are preprocessed to lie within a common scale.

We assume the higher the value of a criteria the better for all the alternatives of the above datasets.

In Table 5 we compare the performance of our algorithm EPI SAT w.r.t. the results reported in Table 8 of [32] related to the three datasets described above. The precise algorithm used in the experimental results of [32] to compute *SMMR* is not described, and the authors show the performance only for $k = 2$, with the computation of *SMMR* for $k > 2$ not being feasible with their algorithm. Regarding our algorithm EPI SAT, the computation of *SMMR* with $k = 4$ was unfeasible for the Boston database. However, we were able to compute it for $k \in \{2, 3\}$. With the datasets Synthetic and Rental we were able to compute *SMMR* even with $k = 5$. Furthermore, with $k = 2$, for example, our approach performed better than the heuristic approximate methods used in [32]. Only being able to manage up to $k = 3$ for the Boston database may be because of the large number of both alternatives and criteria (and the results from Table 3 and Figure 3 suggest that the time performance of our algorithm increases exponentially w.r.t. these parameters).

6 CONCLUSIONS

Interactive elicitation methods maintain a model of the user preferences which is incrementally revised by an agent asking questions to a decision maker. In particular, several works [6–8, 10, 33] have used (standard, single-item) minimax regret to provide a robust recommendation to the decision maker.

The notion of setwise regret [31, 32] 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 be 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 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 with an interactive user-agent system. Our algorithm may replace heuristic approaches when the query size is fairly small since the complexity burden increases exponentially w.r.t. k. However, in preference elicitation systems it is very common to ask binary queries, and with $k = 2$ we compute an optimal binary query w.r.t. the minimax regret criterion. Also, 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 w.r.t. 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 w.r.t. the setwise minimax regret criterion. 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.

ACKNOWLEDGMENTS

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, by the INDECOD project (International Emerging Action) of CNRS, and by the LOGISTAR project, which is funded by the European Commission under the Horizon 2020 programme.

REFERENCES

[1] Abbas Ali. 2004. Entropy methods for adaptive utility elicitation. *IEEE Transactions on Systems, Science and Cybernetics* 34, 2 (2004), 169–178.

[2] David E Bell. 1982. Regret in decision making under uncertainty. *Operations research* 30, 5 (1982), 961–981.

[3] Nawal Benabbou, Serena Di Sabatino Di Diodoro, Patrice Perny, and Paolo Viappiani. 2016. Incremental preference elicitation in multi-attribute domains for choice and ranking with the Borda count. In *Proceedings of International Conference on Scalable Uncertainty Management (SUM)*. Springer, 81–95.

[4] Nawal Benabbou and Thibaut Lust. 2019. An interactive polyhedral approach for multi-objective combinatorial optimization with incomplete preference information. In *Proceedings of International Conference on Scalable Uncertainty Management (SUM)*.

[5] Nawal Benabbou and Patrice Perny. 2015. Incremental weight elicitation for multiobjective state space search. In *Proceedings of Association for the Advancement of Artificial Intelligence (AAAI)*.

[6] Nawal Benabbou, Patrice Perny, and Paolo Viappiani. 2014. Incremental elicitation of Choquet capacities for multicriteria decision making.. In *Proceedings of European Conference on Artificial Intelligence (ECAI)*.

[7] Nawal Benabbou, Patrice Perny, and Paolo Viappiani. 2017. Incremental elicitation of Choquet capacities for multicriteria choice, ranking and sorting problems. *Artificial Intelligence* 246 (2017), 152–180.

[8] Craig Boutilier, Relu Patrascu, Pascal Poupart, and Dale Schuurmans. 2006. Constraint-based optimization and utility elicitation using the minimax decision criterion. *Artificial Intelligence* (2006).

[9] Stephen Boyd and Lieven Vandenberghe. 2004. *Convex optimization*. Cambridge university press.

[10] Darius Braziunas. 2012. *Decision-theoretic elicitation of generalized additive utilities*. Ph.D. Dissertation. University of Toronto.

[11] Darius Braziunas and Craig Boutilier. 2006. Preference elicitation and generalized additive utility. In *Proceedings of Association for the Advancement of Artificial Intelligence (AAAI)*, Vol. 21.

[12] Darius Braziunas and Craig Boutilier. 2007. Minimax regret based elicitation of generalized additive utilities.. In *Proceedings of Uncertainty in Artificial Intelligence (UAI)*.

[13] Darius Braziunas and Craig Boutilier. 2010. Assessing regret-based preference elicitation with the UTPREF recommendation system. In *Proceedings of ACM Conference on Electronic Commerce (EC)*. 219–228.

[14] Matthias C. M. Troffaes. 2018. *pycddlib Python wrapper for Komei Fukuda's cddlib*. https://pycddlib.readthedocs.io/en/latest/

[15] Krzysztof Gajos and Daniel S Weld. 2005. Preference elicitation for interface optimization. In *Proceedings of ACM Symposium on User Interface Software and Technology (UIST)*. 173–182.

[16] Ronald A Howard. 1966. Information value theory. *IEEE Transactions on systems science and cybernetics* 2, 1 (1966), 22–26.

[17] IBM ILOG. 2017. IBM ILOG CPLEX Optimization Studio, V12.8.0.

[18] Vijay S Iyengar, Jon Lee, and Murray Campbell. 2001. Evaluating multiple attribute items using queries. In *Proceedings of the ACM Conference on Electronic Commerce (EC)*. 144–153.

[19] Panos Kouvelis and Gang Yu. 2013. *Robust discrete optimization and its applications*. Vol. 14. Springer Science & Business Media.

[20] Tiep Le, Atena M Tabakhi, Long Tran-Thanh, William Yeoh, and Tran Cao Son. 2018. Preference elicitation with interdependency and user bother cost. In *Proceedings of International Conference On Autonomous Agents and Multi-Agent Systems (AAMAS)*. 1459–1467.

[21] Tyler Lu and Craig Boutilier. 2011. Robust approximation and incremental elicitation in voting protocols. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*.

[22] Robert Price and Paul R Messinger. 2005. Optimal recommendation sets: covering uncertainty over user preferences. In *Proceedings of Association for the Advancement of Artificial Intelligence (AAAI)*. 541–548.

[23] Howard Raiffa. 1968. *Decision analysis*. Addison-Wesley.

[24] Ahti A Salo and Raimo P Hamalainen. 2001. 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* (2001).

[25] Leonard J Savage. 1972. *The foundations of statistics*. Courier Corporation.

[26] Guo Shengbo and Sanner Scott. 2010. Real-time multiattribute Bayesian preference elicitation with pairwise comparison queries. In *Proceedings of the International Conference on Artificial Intelligence and Statistics, (AISTATS)*. 289–296.

[27] Carsten Sinz. 2005. Towards an optimal CNF encoding of boolean cardinality constraints. In *Proceedings of International Conference on Principles and Practice of Constraint Programming (CP)*. Springer, 827–831.

[28] Russell Stoneback. 2019. *pysat 2.1.0*. https://pypi.org/project/pysat/

[29] Stefano Teso, Andrea Passerini, and Paolo Viappiani. 2016. Constructive preference elicitation by setwise max-margin learning. In *Proceedings of the International Joint Conference on Artificial Intelligence, IJCAI*. 2067–2073.

[30] Federico Toffano and Nic Wilson. 2020. Minimality and comparison of sets of multi-attribute vectors. In *Proceedings of European Conference on Artificial Intelligence (ECAI)*. 913–920.

[31] Paolo Viappiani and Craig Boutilier. 2009. Regret-based optimal recommendation sets in conversational recommender systems. In *Proceedings of ACM Conference on Recommender Systems (RecSys)*.

[32] Paolo Viappiani and Craig Boutilier. 2020. On the equivalence of optimal recommendation sets and myopically optimal query sets. *Artificial Intelligence* (2020), 103328.

[33] Tianhan Wang and Craig Boutilier. 2003. Incremental utility elicitation with the minimax regret decision criterion. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, Vol. 3. 309–316.

[34] Chelsea C White, Andrew P Sage, and Shigeru Dozono. 1984. A model of multiattribute decisionmaking and trade-off weight determination under uncertainty. *IEEE Transactions on Systems, Man, and Cybernetics* 2 (1984), 223–229.