# A metaheuristic for inferring a ranking model based on multiple reference profiles

Arwa Khannoussi, Alexandru-Liviu Olteanu, Patrick Meyer, Bastien Pasdeloup

# A metaheuristic for inferring a ranking model based on multiple reference profiles

Arwa Khannoussi[1], Alexandru-Liviu Olteanu[3], Patrick Meyer[2] and Bastien Pasdeloup[2]

[1]IMT Atlantique, LS2N, UMR CNRS 6004, F-4430 Nantes, France.
[2]IMT Atlantique, Lab-STICC, UMR CNRS 6285, F-29238 Brest, France.
[3]Lab-STICC, UMR 6285, CNRS, Université Bretagne Sud, Lorient, France.

Contributing authors: arwa.khannoussi@imt-atlantique.fr; alexandru.olteanu@univ-ubs.fr; patrick.meyer@imt-atlantique.fr; bastien.pasdeloup@imt-atlantique.fr;

**Abstract**

In the context of Multiple Criteria Decision Aiding, decision makers often face problems with multiple conflicting criteria that justify the use of preference models to help advancing towards a decision. In order to determine the parameters of these preference models, preference elicitation makes use of preference learning algorithms, usually taking as input holistic judgments expressed by the decision maker. Tools to achieve this goal in the context of a ranking model based on multiple reference profiles are usually based on mixed-integer linear programming or constraint programming. However, they are usually unable to handle realistic problems involving many criteria and a large amount of input information. We propose here an evolutionary metaheuristic in order to address this issue. Extensive experiments illustrate its ability to handle problem instances that previous proposals cannot.

**Keywords:** multi-criteria decision aiding, preference elicitation and learning, ranking problem, reference profiles, meta-heuristic

# 1 Introduction

Difficult decisions usually involve multiple, often conflicting, perspectives over a set of alternatives. In *Multiple Criteria Decision Aiding* (MCDA), we distinguish three types of decision problems [15]: 1. *Choice* refers to selecting the best alternative or the best set of alternatives; 2. *Ranking* looks to order all alternatives from the best one to the worst; 3. *Sorting* seeks to assign alternatives to one or several predefined preferentially ordered categories. In this work, we focus on the the ranking problem, and, more specifically, the *Ranking based on Multiple reference Profiles* (RMP) [13, 4] preference model. In this model, pairs of alternatives are not compared directly, but through a set of predefined reference profiles. This process allows to construct a preference relation on the set of alternatives. The RMP model belongs to the class of MCDA methods that are based on outranking relations [14]. In this work, we restrict ourselves to a specific case of the RMP model, called *Simple RMP* (SRMP) [13], in which the importance of criteria is represented by additive weights.

The use of reference profiles is widely spread among MCDA methods, and is based on psychological evidence [16] which suggests that *decision makers* (DMs) often base their decisions on so-called references which correspond to their current expectations on the decision problem. For ranking problems we can mention the TOPSIS [7] method, which evaluates alternatives using distances to the ideal and anti-ideal points, to be minimized and maximized respectively. The MACBETH [2] method also uses two reference levels on each criterion, corresponding to "good" and "neutral" evaluations, in order to elicit a value based model. For sorting problems, we can mention the ELECTRE TRI [14] method and its multiple variants [10, 1] which compare alternatives to bounding or central categories profiles.

In order to be used in practice, the parameters of the SRMP ranking model need to be determined so that the model accurately reflects the perspective of the DM. The interaction between an analyst and a DM in order to set the parameters of a preference model is called the *preference elicitation* process. The direct elicitation approach requires the DM to give numerical values for the model parameters, while the indirect approach uses holistic information given by the DM in order to learn the model parameters [8] through a preference learning algorithm. The direct elicitation approach is usually difficult to apply in practice, as the DM has to perfectly understand how the preference model works [5] (model-driven approach), while the indirect approach does not require this expertise and simply asks the DM to express judgments on the desired output of the method for a few alternatives only (data-driven approach). In the case of the RMP model, this holistic information corresponds to pairwise comparisons of alternatives.

Previous work on inferring the parameters of an SRMP model [12] involves an exact resolution approach, more precisely a mixed integer linear optimization problem (MILP), that requires significant computational resources and time, even for a small number of input pairwise comparisons. Belahcène *et al.* [3] propose to use a boolean satisfiability approach (SAT) to learn the more

general RMP model. They reduce execution time compared to [12] and allow for larger sets of input pairwise comparisons to be used. A matheuristic algorithm, integrating linear programming and a heuristic in its design, is proposed by Liu *et al.* [11] to infer the parameters of an SRMP model. This approach is faster than the previous ones, allows to handle larger sets of pairwise comparisons of alternatives and more complex SRMP models (in terms of number of criteria and profiles), but this leads to a loss in the accuracy on the learning dataset. The topic of incrementally learning the parameters of the RMP model through repeated interactions with the DM has also been recently addressed by Khannoussi *et al* [9].

We extend these works by proposing an evolutionary metaheuristic, more specifically a genetic algorithm for learning the parameters of an SRMP model. We show that this proposal allows to deal with large instances in seconds for problems that were considered as difficult by the previous approaches. This allows the use of this learning algorithm in real-life problems. Furthermore, the time complexity of the genetic algorithm is linear with the size of the input data, hence we are able to get good results for very large problems in less than half an hour. Moreover, despite not always reaching 100% accuracy on the input data, the inferred models have a high generalization ability on unseen data, similar to those reported in the previous articles, but in far less time.

The paper is structured in the following way: Section 2 introduces the reader to the SRMP method. Section 3 presents the proposed preference learning algorithm, and Section 4 the tuning of its hyperparameters. Section 5 provides a numerical analysis of its performance on problem instances of different sizes and input information of varying quality. Finally, Section 6 concludes with several ending remarks and perspectives for future work.

# 2 SRMP: a Ranking model based on Multiple Profiles

We consider a finite set of $A$ distinct alternatives, and a fixed set of $C$ criteria. We note $\mathbf{A} \in [0,1]^{A \times C}$ the matrix of evaluations of all alternatives on all criteria, *i.e.*, for an alternative $a$ and a criterion $c$, $\mathbf{A}[a,c] \in [0,1]$. We use the ":" operator to span the entire dimension of an associated matrix. Hence $\mathbf{A}[:,c]$ is the $c^{\text{th}}$ column of $\mathbf{A}$, and $\mathbf{A}[a,:]$ its $a^{\text{th}}$ row.

Without loss of generality, we restrict all evaluations the to the interval $[0,1]$, and assume that higher evaluations of alternatives on criteria are preferred to lower ones.

## 2.1 Parameters of the SRMP model

The SRMP preference model [13] is characterized by the following preference parameters:

- $P$ reference profiles, each evaluated on all criteria, denoted by a matrix $\mathbf{P} \in [0,1]^{P \times C}$. Profiles satisfy the dominance rule defined by:

$$\forall c \in \{1,\ldots,C\}, \forall p \in \{1,\ldots,P-1\} : \mathbf{P}[p,c] \leq \mathbf{P}[p+1,c] \; ; \tag{1}$$

- A lexicographic order on the profiles, denoted by a vector $\mathbf{o} = \sigma(\{1,\ldots,P\})$, where $\sigma(\cdot)$ is a permutation function;
- A vector of weights $\mathbf{w} \in ]0,1[^C$ associated with the criteria, satisfying the following constraint:

$$\sum_c \mathbf{w}[c] = 1 \; . \tag{2}$$

An SRMP model $M$ is therefore described by the tuple $(\mathbf{P}, \mathbf{o}, \mathbf{w})$.

Figure 1 represents an example of an SRMP model, for $C = 4$ criteria and $P = 3$ reference profiles. Each criterion is represented by a vertical axis, on which the preferred performances are plotted towards the top (in the direction of the arrows). The dotted lines are the profiles, whose lexicographic order ($\mathbf{o}$) is shown on the left. The weights ($\mathbf{w}$) are shown below each axis.

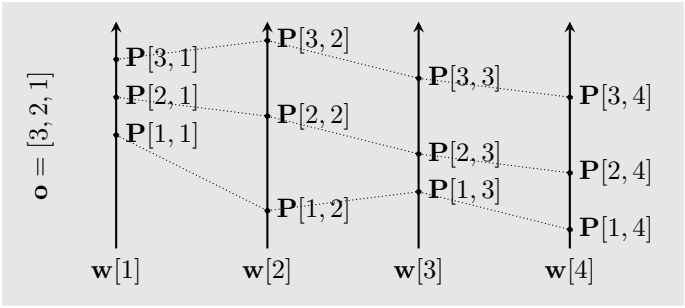



**Fig. 1**: Representation of an SRMP model for $C = 4$ criteria and $P = 3$ reference profiles.

## 2.2 Comparing alternatives, given an SRMP model

An SRMP model can be used to provide a ranking on the set of alternatives by comparing them sequentially, pairwise, to the reference profiles, in the order given by $\mathbf{o}$.

More precisely, two alternatives $a_1$ and $a_2$ are in a strict preference relation w.r.t. profile $p$, denoted $a_1 \succ_p a_2$, iff:

$$\sum_{\substack{c \\ \mathbf{A}[a_1,c] \geq \mathbf{P}[p,c]}} \mathbf{w}[c] > \sum_{\substack{c \\ \mathbf{A}[a_2,c] \geq \mathbf{P}[p,c]}} \mathbf{w}[c] \; . \tag{3}$$

Alternatively, they are in an indifference relation w.r.t. profile $p$, denoted $a_1 \sim_p a_2$, iff:

$$\sum_{\substack{c \\ \mathbf{A}[a_1,c] \geq \mathbf{P}[p,c]}} \mathbf{w}[c] = \sum_{\substack{c \\ \mathbf{A}[a_2,c] \geq \mathbf{P}[p,c]}} \mathbf{w}[c] \ . \tag{4}$$

An overall relation between two alternatives $a_1$ and $a_2$ is constructed by sequentially considering the profiles using the lexicographic order $\mathbf{o}$, *i.e.*, a strict preference relation $a_1 \succ a_2$ occurs when $a_1 \succ_{\mathbf{o}[p_1]} a_2$, and $a_1 \sim_{\mathbf{o}[p_2]} a_2$, $\forall p_2 < p_1$. If no such $p_1$ exists, then $a_1 \sim a_2$. It has been shown that $\succ$ and $\sim$ define a preorder on any set of alternatives [13].

# 3 The preference inference algorithm

The proposed inference algorithm follows the classical elements of a genetic algorithm [6]. A genetic algorithm is a metaheuristic from the class of evolutionary algorithms, which aims at generating good solutions to optimization problems. It is inspired by Charles Darwin's theory of natural evolution, and mimics the natural selection process, where the "fittest" individuals (the solutions) of a population are selected for reproduction in order to produce offsprings of the next generation.

Algorithm 1 sums up the elements introduced more precisely in the following sections. Implementation details and codes to reproduce our experiments are publicly available at https://github.com/BastienPasdeloup/learn_srmp.

## 3.1 Solution

In our context, a *solution* in the genetic algorithm consists of the parameters of an SRMP model $M$, *i.e.*, the reference profiles $\mathbf{P}$, criteria weights $\mathbf{w}$ and lexicographic order $\mathbf{o}$, as described in Section 2.

It is worth noting that an SRMP model requires the knowledge of the value of $P$, that determines the number of reference profiles in a solution. This value is unknown in practice and has thus to be given as an input argument of Algorithm 1.

## 3.2 Fitness

Let $\mathcal{D}_{\text{TRAIN}}$ be a set of pairwise comparisons of alternatives provided by the DM, *i.e.*, each element of $\mathcal{D}_{\text{TRAIN}}$ is a tuple $(a_1, a_2, \odot)$, where $a_1$ and $a_2$ are two distinct alternatives, and $\odot \in \{\succ, \prec, \sim\}$ is the preference relation $a_1 \odot a_2$ provided by the DM.

For a given solution $M$ in the genetic algorithm, the number of pairs of alternatives that $M$ is able to compare similarly to the DM therefore gives a quality measure of $M$. The higher this value, the more representative $M$ is of the DM. As a consequence, in our genetic algorithm, we use the following

*fitness function* to evaluate a solution:

$$\texttt{fitness}(M, \mathcal{D}_{\text{TRAIN}}) = \frac{1}{|\mathcal{D}_{\text{TRAIN}}|} \sum_{\{(a_1,a_2,\odot)\in\mathcal{D}_{\text{TRAIN}}\}} \begin{cases} 1 \text{ if } \odot_M = \odot \\ 0 \text{ otherwise} \end{cases}, \quad (5)$$

where $\odot_M \in \{\succ, \prec, \sim\}$ gives the ordering $a_1 \odot_M a_2$ obtained thanks to $M$.

**Algorithm 1** GA_SRMP ($\mathcal{T}$, $\mathcal{X}$, $\mathcal{M}$, $S$, $B$, $R$, $P$)

**Input**
▷ $\mathcal{T}$: Training set used to learn the SRMP model
▷ $\mathcal{X}$: Set of possible crossover operators with probabilities to be selected
▷ $\mathcal{M}$: Set of possible mutation operators with probabilities to be applied
▷ $S$: Population size of the genetic algorithm
▷ $B$: Ratio of best solutions in the population kept for next iteration
▷ $R$: Ratio of random solutions injected at next iteration
▷ $P$: Number of reference profiles to create in solution
**Output**
▷ $\widehat{M}$: Best solution found

$pop_{it} \leftarrow \texttt{initialize\_pop}(S)$ ▷ Initial population of random solutions
**while not** $\texttt{stopping\_condition}(pop_{it})$ **do** ▷ Check stopping condition
  $pop_{it+1} \leftarrow \texttt{prepare\_pop}(\mathcal{T}, pop_{it}, S, B, R)$ ▷ Initialize next pop
  **while** $\texttt{size}(\text{pop}_{it+1}) \neq S$ **do** ▷ Fill with generated solutions
    $M_1, M_2 \leftarrow \texttt{select}_2(pop_{it})$ ▷ Select 2 parents using roulette wheel
    $\texttt{crossover} = \texttt{select}_1(\mathcal{X})$ ▷ Draw 1 crossover with probabilities in $\mathcal{X}$
    $M \leftarrow \texttt{crossover}(M_1, M_2)$ ▷ Apply it (1 child for exposition)
    **for all** $\texttt{mutation} \in \mathcal{M}$ **do** ▷ Consider all possible mutations
      $M \leftarrow \texttt{mutation}(M)$ ▷ Apply with probability as given in $\mathcal{M}$
    **if** $\texttt{fitness}(M, \mathcal{T}) > \min(\texttt{fitness}(M_1, \mathcal{T}), \texttt{fitness}(M_2, \mathcal{T}))$ **then**
      $pop_{it+1} \leftarrow pop_{it+1} \cup \{M\}$ ▷ Keep if better than 1 parent
  $pop_{it} \leftarrow pop_{it+1}$ ▷ Go to next iteration
**return** $\texttt{solution\_with\_best\_fitness}(pop_{it})$ ▷ Return $\widehat{M}$

## 3.3 Initialization

The genetic algorithm *initializes* a population of solutions and then improves it through repetitive applications of *selection*, *mutation* and *crossover* operators. The *initial population* is filled by randomly generating valid solutions, *i.e.*, that respect Equations 1 and 2, as well as definition domains introduced in Section 2.

## 3.4 Main loop

The population at a given iteration is constructed with some of the best solutions from the population from the previous iteration, in addition to some new random valid solutions in order to maintain diversity. The rest of the population is obtained through a reproduction method, where some *parent solutions* are *selected* to create *child solutions*. Individual solutions are selected based on the value of their fitness using a *roulette wheel* procedure, which tends to prefer "good" solutions in the selection process, while allowing for "less good" solutions to be selected sometimes with a probability proportional to their fitness.

To generate new solutions, crossover and mutation operators are used. A *crossover* operator produces child solutions from the selected parent solutions, by keeping some characteristics of the parents. The children are then transformed through one or more *mutation* operators to diversify the population, and are finally inserted in the population if they improve the fitness of at least one of the parents.

After each operation – crossover or mutation –, solutions are regularized to ensure their validity. In more detail, the following corrections are applied:

1. For each criterion $c$, entries of $\mathbf{P}[:, c]$ are sorted. Entries of $\mathbf{P}$ are then clipped to belong to $[0, 1]$, *i.e.*, values outside the interval are updated to the closest interval edge;
2. Entries of $\mathbf{w}$ are first clipped to belong to $]0, 1[$. Then, for each criterion $c$, $\mathbf{w}[c] \leftarrow \frac{\mathbf{w}[c]}{\|\mathbf{w}\|_1}$, where $\|\cdot\|_1$ denotes the $\ell_1$ norm.

## 3.5 Proposed crossover operators

Some key elements of a genetic algorithm are the operators it uses. This section describes the crossover operators introduced in this work. These operators are all provided to Algorithm 1 as input variable $\mathcal{X}$, with associated probabilities depending on the experiments performed later in this article. They are illustrated in Figure 2.

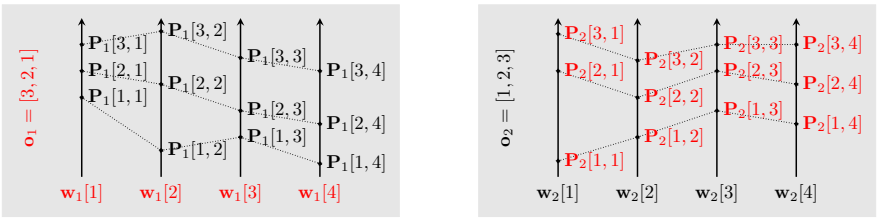
Let $M_1 = (\mathbf{P}_1, \mathbf{o}_1, \mathbf{w}_1)$ and $M_2 = (\mathbf{P}_2, \mathbf{o}_2, \mathbf{w}_2)$ be two selected parent solutions with $P$ reference profiles, and let $M$ be a child solution as generated by the presented crossover operators. In practice, the implementation also generates a second child complementary to $M$. We do not detail more for simplicity of exposition, and refer the interested reader to the provided Github repository:

- `swap_profiles`: Exchanges profiles between two solutions:

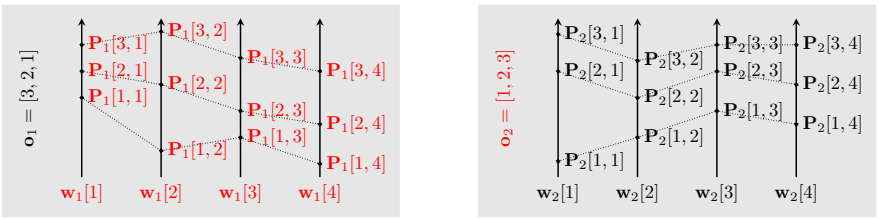
$$M \leftarrow (\mathbf{P}_2, \mathbf{o}_1, \mathbf{w}_1) \ ;$$

- `swap_orders`: Exchanges the lexicographical orders between two solutions:

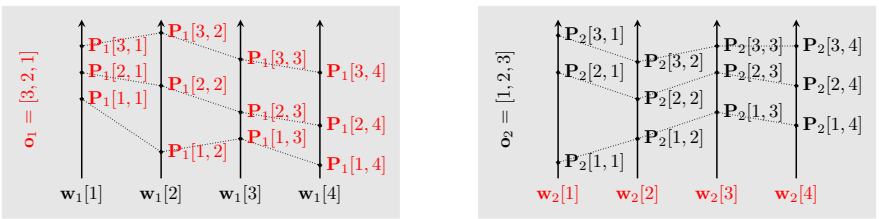
$$M \leftarrow (\mathbf{P}_1, \mathbf{o}_2, \mathbf{w}_1) \ ;$$



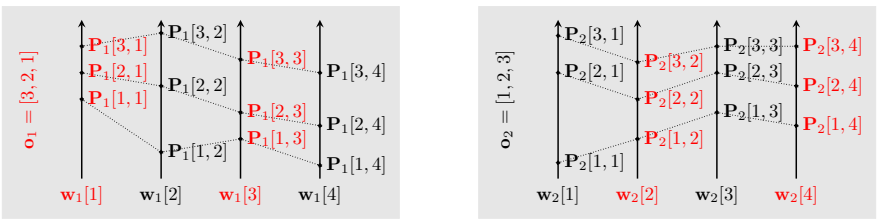

(a) `swap_profiles`




(b) `swap_orders`




(c) `swap_weights`




(d) `mix_criteria_and_weights`

**Fig. 2**: Proposed crossover operators. Elements in red are assembled to build a child. Note that elements in black can also be assembled to produce a second child, as mentioned in Section 3 and done in our implementation.

- `swap_weights`: Exchanges weights between two solutions:

$$M \leftarrow (\mathbf{P}_1, \mathbf{o}_1, \mathbf{w}_2) \; ;$$

- `mix_criteria_and_weights`: Exchanges the profile values and associated weights on a random subset of criteria between two solutions. Let $\mathcal{S} \subset$

$\{1, \ldots, C\}$. Let $\mathbf{P} \in \mathbb{R}^{P \times C}$ such that $\forall p, \forall c : \mathbf{P}[p, c] = \mathbf{P}_1[p, c]$ if $c \in \mathcal{S}$, and $\mathbf{P}[p, c] = \mathbf{P}_2[p, c]$ otherwise. Additionally, let $\mathbf{w} \in \mathbb{R}^C$ such that $\forall p, \forall c : \mathbf{w}[c] = \mathbf{w}_1[c]$ if $c \in \mathcal{S}$, and $\mathbf{w}[c] = \mathbf{w}_2[c]$ otherwise:

$$M \leftarrow (\mathbf{P}, \mathbf{o}_1, \mathbf{w}) \ .$$

### 3.6 Proposed mutation operators

Once created using a crossover operator, child solutions can be altered using one or more mutations. This section describes the mutation operators introduced in this work. These operators are all provided to Algorithm 1 as input variable $\mathcal{M}$, with associated probabilities depending on the experiments performed later in this article. They are illustrated in Figure 3.

Let $M = (\mathbf{P}, \mathbf{o}, \mathbf{w})$ be the solution to mutate:

- `random_profile_perturbation`: Applies a (i.i.d.) random perturbation of amplitude $\lambda_{\text{RPP}}$ to all the profile values. Let $\mathbf{X} \in \mathbb{R}^{P \times C}$ such that $\forall p, \forall c : \mathbf{X}[p, c] \sim \mathcal{U}(-1, 1)$:

$$\forall p, \forall c : \mathbf{P}[p, c] \leftarrow \mathbf{P}[p, c] + \lambda_{\text{RPP}} \mathbf{X}[p, c] \ ;$$

- `random_weights_perturbation`: Applies a (i.i.d.) random perturbation of amplitude $\lambda_{\text{RWP}}$ to the weights of the criteria. Let $\mathbf{x} \in \mathbb{R}^C$ such that $\forall c : \mathbf{x}[c] \sim \mathcal{U}(-1, 1)$:

$$\forall c : \mathbf{w}[c] \leftarrow \mathbf{w}[c] + \lambda_{\text{RWP}} \mathbf{x}[c] \ ;$$

- `shrink_profiles`: Shrinks the space between profiles by a factor $\lambda_{\text{SP}}$ to bring them closer to one another, while preserving the original mean of all profiles per criterion. Let $\mathbf{m}_1 \in \mathbb{R}^C$ such that $\forall c : \mathbf{m}_1[c] = \frac{1}{P} \|\mathbf{P}[:, c]\|_1$. We proceed in two steps:

$$\forall p, \forall c : \mathbf{P}[p, c] \leftarrow \frac{1}{1 + \lambda_{\text{SP}}} \mathbf{P}[p, c] \ .$$

  Then, let $\mathbf{m}_2 \in \mathbb{R}^C$ such that $\forall c : \mathbf{m}_2[c] = \frac{1}{P} \|\mathbf{P}[:, c]\|_1$ computed after this operation. We then update profile values as follows:

$$\forall p, \forall c : \mathbf{P}[p, c] \leftarrow \mathbf{P}[p, c] - \mathbf{m}_2[c] + \mathbf{m}_1[c] \ ;$$

- `expand_profiles`: Conversely, expands the space between profiles by a factor $\lambda_{\text{EP}}$ to push them apart, while preserving the original mean of all profiles per criterion. Let $\mathbf{m}_1 \in \mathbb{R}^C$ such that $\forall c : \mathbf{m}_1[c] = \frac{1}{P} \|\mathbf{P}[:, c]\|_1$. We proceed in two steps:

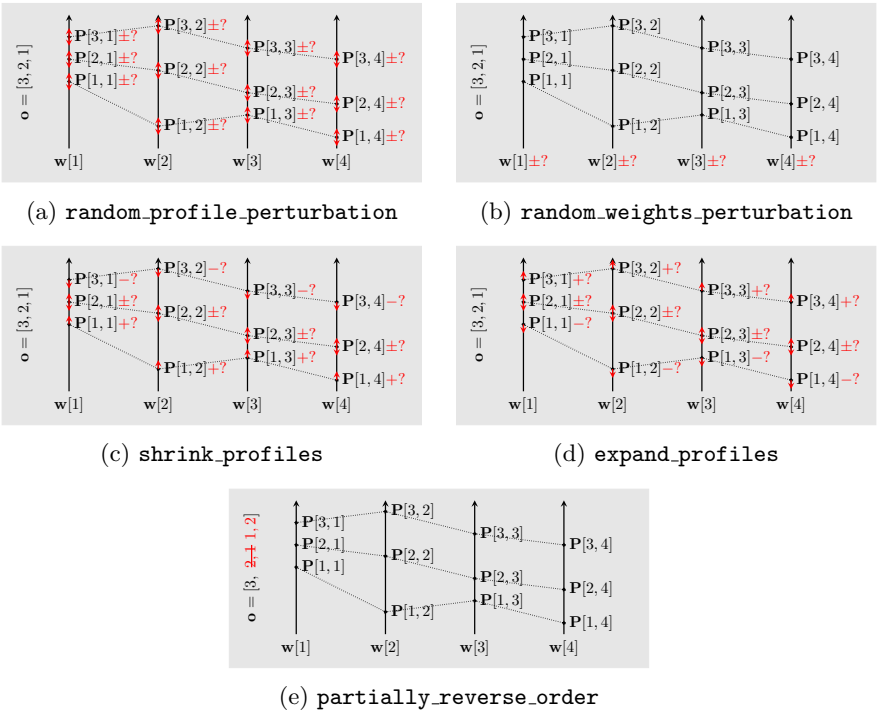
$$\forall p, \forall c : \mathbf{P}[p, c] \leftarrow (1 + \lambda_{\text{EP}}) \mathbf{P}[p, c] \ .$$

Then, let $\mathbf{m}_2 \in \mathbb{R}^C$ such that $\forall c : \mathbf{m}_2[c] = \frac{1}{P}\|\mathbf{P}[:, c]\|_1$ computed after this operation. We then update profile values as follows:

$$\forall p, \forall c : \mathbf{P}[p, c] \leftarrow \mathbf{P}[p, c] - \mathbf{m}_2[c] + \mathbf{m}_1[c] \ ;$$

- `partially_reverse_order`: Reverses a random part of the lexicographical order. Let $i$ and $j$ be random integers in $\{1, \ldots, P\}$ with $i < j$ :

$$\forall k \in [i, j] : \mathbf{o}[k] \leftarrow \mathbf{o}[j - k + i] \ .$$




(a) `random_profile_perturbation` (b) `random_weights_perturbation`

(c) `shrink_profiles` (d) `expand_profiles`

(e) `partially_reverse_order`

**Fig. 3**: Proposed mutation operators. Elements in red highlight the alterations operated on the child. Symbols ? indicate a random evolution of the parameter, with magnitude controlled by the corresponding $\lambda_\cdot$ parameter.

## 3.7 Stopping criterion

Multiple stopping criteria can be considered depending on the target application, such as maximum execution time, or no evolution of the $K$ best (*i.e.*, with highest fitness) solutions for a given number of consecutive iterations.

In all of our experiments below, we have chosen the latter, to favor solutions that maximize accuracy. Value of $K$ is chosen as $K = 0.1BS$, corresponding to no changes among the top 10% within the $B/S$ best solutions in the population, for 50 consecutive iterations.

# 4 Preliminary experiments

## 4.1 Considered settings

A first step before performing experiments to evaluate the performance of the proposed genetic algorithm consists in determining the values of its numerous hyperparameters. Additionally, we are interested in evaluating which of the crossover and mutation operators introduced in Section 3 are indeed useful, in the sense that using them helps reaching models with high accuracy.

In the experiments described in this section, we have chosen to focus on problems with the following characteristics:

- Number of reference profiles (ground truth value of $P$): 3;
- Number of criteria ($C$): 11;
- Number of alternatives ($A$): 50;
- Size of the training set ($|\mathcal{D}_{\text{TRAIN}}|$), whose distinct elements are chosen uniform randomly and independently among all possible pairs of distinct alternatives: 100.

We consider in this section 10 distinct random ground truth DMs, that we name $\{M_1^*, \ldots, M_{10}^*\}$. We note $\mathcal{D}^i_{\text{TRAIN}}$ the training set $\mathcal{D}_{\text{TRAIN}}$, where the comparisons $\odot$ between alternatives are obtained using the $i^{\text{th}}$ ground truth model $M_i^*$.

Also, we introduce a new matrix $\mathbf{A}' \in [0,1]^{A' \times C}$ of $A' = 300$ unseen alternatives, *i.e.*, distinct from those in $\mathbf{A}$. Similarly to $\mathcal{D}^i_{\text{TRAIN}}$, we note $\mathcal{D}^i_{\text{TEST}}$ the set of all possible comparisons of distinct alternatives in $\mathbf{A}'$, compared using the associated ground truth DM $M_i^*$.

## 4.2 Grid search

For each pair of a mutation and a crossover operators, we perform a grid search – *i.e.*, exhaustive combination – of the following values for the corresponding operators:

- Population size ($S$ in Algorithm 1): $\{100, 150, 200, 250, 300\}$;
- Ratio of best solutions transmitted to next iteration ($B$ in Algorithm 1): $\{0.1, 0.2, 0.3, 0.4, 0.5\}$;
- Ratio of new random solutions per iteration ($R$ in Algorithm 1): $\{0.1, 0.2, 0.3, 0.4, 0.5\}$;
- Number of reference profiles to learn ($P$ in Algorithm 1): 3.
- $\lambda_{\text{RPP}}$ (if applicable): $\{0.1, 0.2, 0.3, 0.4, 0.5\}$;
- $\lambda_{\text{RWP}}$ (if applicable): $\{0.1, 0.2, 0.3, 0.4, 0.5\}$;
- $\lambda_{\text{SP}}$ (if applicable): $\{0.2, 0.4, 0.6, 0.8, 1\}$;
- $\lambda_{\text{EP}}$ (if applicable): $\{0.2, 0.4, 0.6, 0.8, 1\}$.

Note that $P$ in Algorithm 1 is chosen to match the actual number of reference profiles of the ground truth DM to learn. In practical cases, we want to model real-life DMs, and no such *true* $P$ exists. Therefore, a choice of $P$ is needed. This point is discussed in more details in Section 5.

These experiments are repeated for each pair of a crossover and a mutation operator, in isolation of all other operators. This is done by setting *e.g.*, $\mathcal{X} = \{(\texttt{swap\_weights}, 1)\}$ and $\mathcal{M} = \{(\texttt{shrink\_profiles}, 1)\}$ in Algorithm 1. Each configuration is evaluated on all the ground truth DMs to provide average results. Therefore, for each pair of a mutation and a crossover operator, we have run $5 * 5 * 5 * 5 * 10 = 6250$ experiments (except when $\mathcal{M} = \{(\texttt{partially\_reverse\_order}, 1)\}$ for which we have run $5 * 5 * 5 * 10 = 1250$ experiments, since there is no associated $\lambda$). All together, we have performed $16 * 6250 + 4 * 1250 = 105,000$ experiments, corresponding to $10,500$ different configurations. Let $\Omega = \{\omega_1, \ldots, \omega_{10,500}\}$ be the set of all these configurations.

For each configuration $\omega \in \Omega$, we therefore obtain 10 estimated DMs $\{\widehat{M}_1^\omega, \ldots, \widehat{M}_{10}^\omega\}$ of the ground truth DMs $\{M_1^*, \ldots, M_{10}^*\}$. We measure the following quantities:

- The *train accuracy* of $\widehat{M}_i^\omega$, given using Equation 5 by:

$$\texttt{train\_accuracy}(\widehat{M}_i^\omega) = \texttt{fitness}(\widehat{M}_i^\omega, \mathcal{D}^i_{\text{TRAIN}}) \, .$$

  From this, we derive the *average train accuracy* of $\omega$:

$$\texttt{average\_train\_accuracy}(\omega) = \frac{1}{10} \sum_{i=1}^{10} \texttt{train\_accuracy}(\widehat{M}_i^\omega) \, ; \qquad (6)$$

- The *test accuracy* of $\widehat{M}_i^\omega$, given using Equation 5 by:

$$\texttt{test\_accuracy}(\widehat{M}_i^\omega) = \texttt{fitness}(\widehat{M}_i^\omega, \mathcal{D}^i_{\text{TEST}}) \, .$$

  Similarly, we derive the *average test accuracy* of $\omega$:

$$\texttt{average\_test\_accuracy}(\omega) = \frac{1}{10} \sum_{i=1}^{10} \texttt{test\_accuracy}(\widehat{M}_i^\omega) \, . \qquad (7)$$

## 4.3 Hyperparameters tuning

Given the outputs of the experiments described above, we select adequate values for the hyperparameters, and pertinent pairs of crossover and mutation operators to use. To do so, we proceed as follows:

1. Sort configurations in $\Omega$ by decreasing average train accuracy;
2. Initialize another set $\Omega^*$ with the first configuration from $\Omega$. Elements in $\Omega^*$ are considered to be significantly better than the remaining configurations from $\Omega$;

3. For any remaining configuration $\omega \in \Omega$, compare it to all configurations $\omega^* \in \Omega^*$ using a Welch's t-test [17] (with a standard significance level of 0.05) on the associated sets $\{M_1^{\omega}, \ldots, M_{10}^{\omega}\}$ and $\{M_1^{\omega^*}, \ldots, M_{10}^{\omega^*}\}$;
4. Add $\omega$ to $\Omega^*$ if it is not worse (according to the test) than any configuration in $\Omega^*$;
5. Remove any configuration from $\Omega^*$ that is worse than $\omega$;
6. Stop when $\Omega^*$ does no change for a given number of iterations (20 in this case).

After this procedure, we are left with $|\Omega^*| = 21$ configurations that cannot be considered significantly better than each other. For these solutions, we obtain an average train accuracy of 0.991 and an average test accuracy of 0.975. Regarding execution time, we observe an average of 32.44 seconds per experiment on a machine with Intel Xeon Gold CPU @ 2.7GHz $\times$ 56, with 256GB RAM, with the provided Python 3.6 implementation of Algorithm 1.

Among solutions in $\Omega^*$, we count the number of occurrences of each attributions of a value to a parameter in the grid search. Results are presented in Table 1.

Table 1 reveals some clear conclusions on the hyperparameters:

- $S$: The higher the better. We retain a value of $S = 300$ for subsequent experiments. Higher values may be used, but this has a consequence on execution time, and train accuracy is already close to 1 in all simulations;
- $B$: Again, high values are preferred. We retain a value of $B = 0.5$ to leave some space for new solutions;
- $R$: It appears that low values of $R$ lead to better performance, hence we keep $R = 0.1$. This may imply that the reproduction process does create good enough solutions out of the initial population without the need to introduce a lot of diversity across the process;
- Crossover and mutation operators: The main conclusion of this process is that some operators clearly outperform the others. Indeed, `mix_criteria_and_weights` and `partially_reverse_order` appear in 100% of retained configurations. Consequently, there is no information on the $\lambda_.$ parameters.

We remind that all these experiments were performed with pairs of operators taken in isolation of the others. Reported accuracies on $\Omega^*$ above show that single-operator settings are sufficient to solve problems already considered complicated in previous literature with high accuracy, in very reasonable time.

## 4.4 Multi-operator settings

### 4.4.1 Missing hyperparameters

Due to already high performance observed in single-operator settings, we have chosen to focus on multi-operator settings in further experiments. Indeed, one may want to alternate between applied operators across the search process to induce diversity. First, to give values to the $\lambda_.$ hyperparameters, we report in

| Hyperparameter | Value | Occurrences |
|---|---|---|
| Crossover operator | `swap_profiles` | 0 |
| | `swap_orders` | 0 |
| | `swap_weights` | 0 |
| | `mix_criteria_and_weights` | 21 |
| Mutation operator | `random_profile_perturbation` | 0 |
| | `random_weights_perturbation` | 0 |
| | `shrink_profiles` | 0 |
| | `expand_profiles` | 0 |
| | `partially_reverse_order` | 21 |
| $S$ | 100 | 0 |
| | 150 | 2 |
| | 200 | 2 |
| | 250 | 8 |
| | 300 | 9 |
| $B$ | 0.1 | 1 |
| | 0.2 | 3 |
| | 0.3 | 5 |
| | 0.4 | 5 |
| | 0.5 | 7 |
| $R$ | 0.1 | 10 |
| | 0.2 | 8 |
| | 0.3 | 3 |
| | 0.4 | 0 |
| | 0.5 | 0 |

**Table 1**: Number of occurrences of each hyperparameter value among the best configurations in $\Omega^*$. Crossover and mutation operator rows indicate the use of that operator in isolation of the others. The various $\lambda_\cdot$ have been grouped, as only one can appear at a time in a configuration. Best choices for each hyperparameter, as well as relevant choices for the operators – according to the observed counts – are highlighted in yellow.

Table 2 the occurrences of hyperparameter values for them among the 10% best configurations in $\Omega$ where the associated mutation operator is used.

Although all other mutation operators are outperformed by `random_profile_perturbation`, Table 2 shows some monotonous behaviors for the various $\lambda_\cdot$ parameters. A notable exception is $\lambda_{\text{RPP}}$ that requires intermediate values. This is probably due to no changes being observed in fitness for too low perturbations, and convergence to extreme profile values for too high ones.

### 4.4.2 Are all operators useful?

We are now interested in analyzing if all operators are good candidates for multi-operator settings. Indeed, an operator with individual low performance will degrade overall performance of other operators. Table 3 reports training

| Hyperparameter | | | | Value | | | | Occurrences | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $\lambda_{\text{RPP}}$ | $\lambda_{\text{RWP}}$ | $\lambda_{\text{SP}}$ | $\lambda_{\text{EP}}$ | 0.1 | 0.1 | 0.2 | 0.2 | 0 | 82 | 36 | 130 |
| | | | | 0.2 | 0.2 | 0.4 | 0.4 | 227 | 60 | 44 | 80 |
| | | | | 0.3 | 0.3 | 0.6 | 0.6 | 23 | 45 | 54 | 31 |
| | | | | 0.4 | 0.4 | 0.8 | 0.8 | 0 | 34 | 56 | 8 |
| | | | | 0.5 | 0.5 | 1 | 1 | 0 | 29 | 60 | 1 |

**Table 2**: Number of occurrences of $\lambda$. hyperparameter values among the 10% best configurations in $\Omega$ that use the corresponding mutation operator. Best choices for each hyperparameter – according to the observed counts – are highlighted in yellow.

| Operator | Avg. train accuracy | Avg. test accuracy | Avg. time (s) |
|---|---|---|---|
| `swap_profiles` | 0.937 | 0.896 | 58.45 |
| `swap_orders` | 0.935 | 0.892 | 57.012 |
| `swap_weights` | 0.937 | 0.895 | 55.731 |
| `mix_criteria_and_weights` | 0.979 | 0.955 | 49.286 |
| `random_profile_perturbation` | 0.955 | 0.922 | 63.472 |
| `random_weights_perturbation` | 0.967 | 0.95 | 63.797 |
| `shrink_profiles` | 0.972 | 0.934 | 41.515 |
| `expand_profiles` | 0.915 | 0.854 | 60.379 |
| `partially_reverse_order` | 0.983 | 0.972 | 33.259 |

**Table 3**: Training and testing accuracies of the various operators, among the 10% best configurations in $\Omega$ that use them. Execution time is reported for information. Operators that reach at least 95% training accuracy are highlighted in yellow.

and testing accuracies of operators in isolation, for the 10% best configurations in $\Omega$ that use the corresponding operator.

Table 3 reveals that, among crossover operators, `mix_criteria_and_weights` largely outperforms the others. Regarding mutation operators however, conclusions are more mitigated. If we find again that `partially_reverse_order` outperforms the others, we can still observe that `random_profile_perturbation`, `random_weights_perturbation` and `shrink_profiles` reach a training accuracy of at least 95% on average. Operator `expand_profiles` however degrades performance when it is used. Interestingly, the same observations can be made on the testing accuracy.

In multi-operator settings, one crossover is selected at random each time, and each operator has an individual probability of being applied, to give their chance to all operators. In Algorithm 1, this translates as the following parameters, where probabilities have been chosen arbitrarily:

- $\mathcal{X} = \{(\texttt{mix\_criteria\_and\_weights}, 1)\}$;
- $\mathcal{M} = \{(\texttt{random\_profile\_perturbation}, 0.2),$
  $(\texttt{random\_weights\_perturbation}, 0.2),$
  $(\texttt{shrink\_profiles}, 0.2),$
  $(\texttt{partially\_reverse\_order}, 0.2)\}$.

# 5 Numerical analysis

We study in this section the performance of our approach for inferring an SRMP model in multi-operator settings. We begin by describing the experimental protocol followed by results pertaining to 1. the ability to reproduce comparisons of the ground truth model when $P$ is supposed known; 2. the same ability when no assumption is made on $P$; 3. the computation time; 4. the ability to handle noisy data.

Similarly to preliminary experiments in Section 4, we consider here ground truth SRMP models containing $C \in \{7, 11, 15\}$ criteria and $P \in \{1, 2, 3, 4\}$ reference profiles, respecting the constraints introduced in Section 2. We consider a set $\mathbf{A}$ of $A = 500$ random alternatives, and we sample $|\mathcal{D}_{\text{TRAIN}}| \in \{100, 300, 500, 1000, 2000\}$ comparisons. The second set of alternatives $\mathbf{A}'$ consists of $A' = 5000$ new random alternatives, and we build $\mathcal{D}_{\text{TEST}}$ as the set of all comparisons between distinct alternatives in $\mathbf{A}'$. Finally, since we also want to study the impact on performance of the number of profiles in the inferred solutions compared to the ground truth ones, we denote $\hat{P} \in \{1, 2, 3, 4\}$ the argument given to Algorithm 1, which assumes the real $P$ to be unknown.

## 5.1 Reproduction of the behavior of a ground truth model

We begin by looking at the ability of our approach to model accurately ground truth SRMP models from a training set. Figure 4 reports the average train and test accuracies when inferring 50 SRMP models with a given number of criteria and profiles.

We observe that in all cases, our approach is able reach very high values of train accuracy regardless the number of comparisons that were provided. A slight decrease in performance is observed as the amount of input information increases. At the same time, as the number of training comparisons increases, so does the test accuracy, showing a close proximity of the inferred models to the original ones. As the size of the problem ($C$, number of criteria) increases, the performance slightly degrades. However, the effect of increasing the number of profiles for a given problem size is minimal.

We consider in Figure 5 the average execution time of our algorithm for 50 problem instance sizes and when the SRMP model that needs to be inferred contains between 1 and 4 profiles.

The results show that the execution time increases linearly with the number of provided binary comparisons. A similar but less pronounced trend can also be observed when considering problem instances with more criteria. The impact on the execution time for increasing the number of profiles of the SRMP model is present to an even lesser extent.

## 5.2 Case when $P$ is not known

We continue by looking at the ability of our approach to construct a model that restores the perspective of the DM when the ground truth number of profiles $P$ is unknown. Figure 6 illustrates the average train accuracy when

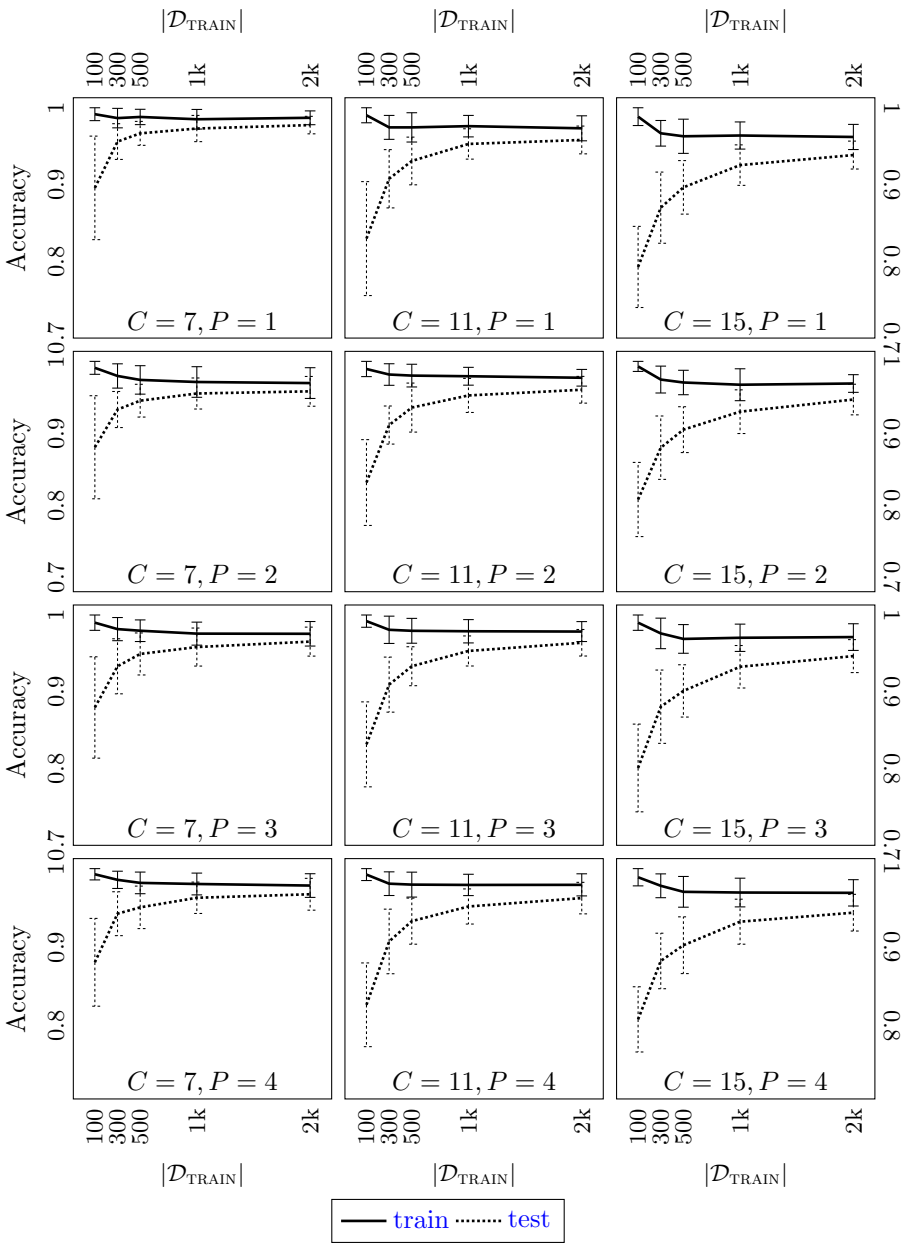



**Fig. 4**: Average train and test accuracy as a function of the number of training comparisons ($|\mathcal{D}_{\text{TRAIN}}|$) for all combinations of the number of criteria ($C$) and the number of profiles ($P = \hat{P}$). Bars represent the limits of the 95% confidence intervals.

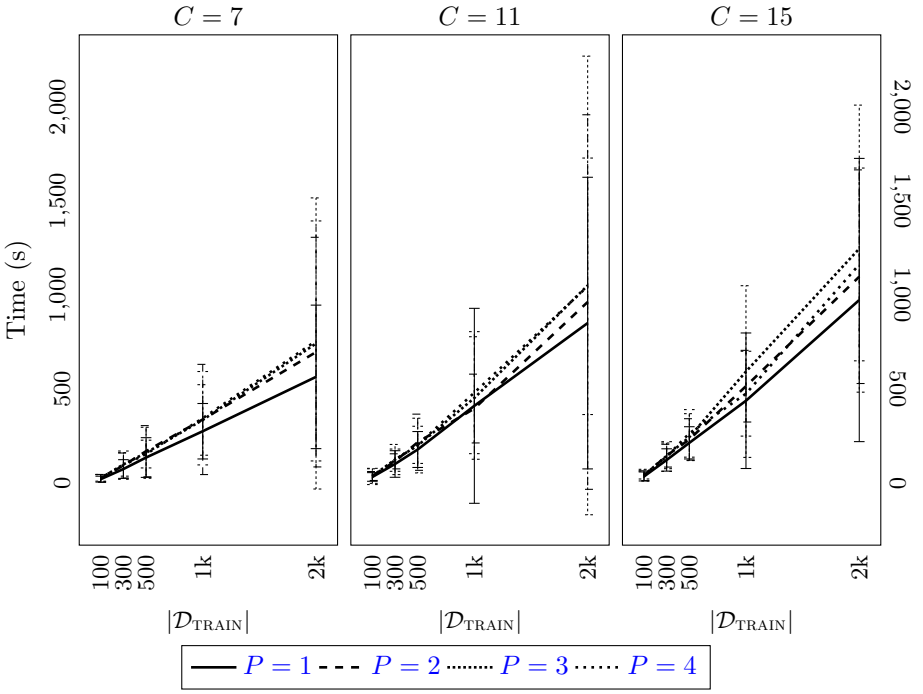



**Fig. 5**: Average execution time as a function of the number of training comparisons ($|\mathcal{D}_{\text{TRAIN}}|$) for all combinations of the number of criteria ($C$) and the number of profiles ($P = \hat{P}$). Bars represent the limits of the 95% confidence intervals.

inferring SRMP models with a given number of criteria and a fixed number of profiles. Each point corresponds to the results on 200 problem instances with an equal repartition of the values of $P$ described above.

We observe that inferring an SRMP model with a single profile leads to the lowest overall train accuracy as it does not appear to be able to reproduce all of the comparisons produced by underlying models with more profiles. Nevertheless, the train accuracy remains very high, above 0.95. The highest average performance is found when inferring models with two profiles, while, surprisingly, models with more than two profiles perform worse on average despite their potential of modeling more difficult binary comparisons. We hypothesize that the added degrees of liberty encumber the learning process.

Figure 7 illustrates the corresponding test accuracy for the previously presented results. We observe the same trend where SRMP models with a single profile perform the worst when approximating the perspective of the DM, while models with two profiles perform the best. The difference in performance is somewhat less pronounced, reaching in all cases very high test accuracy when large sets of binary comparisons are provided.

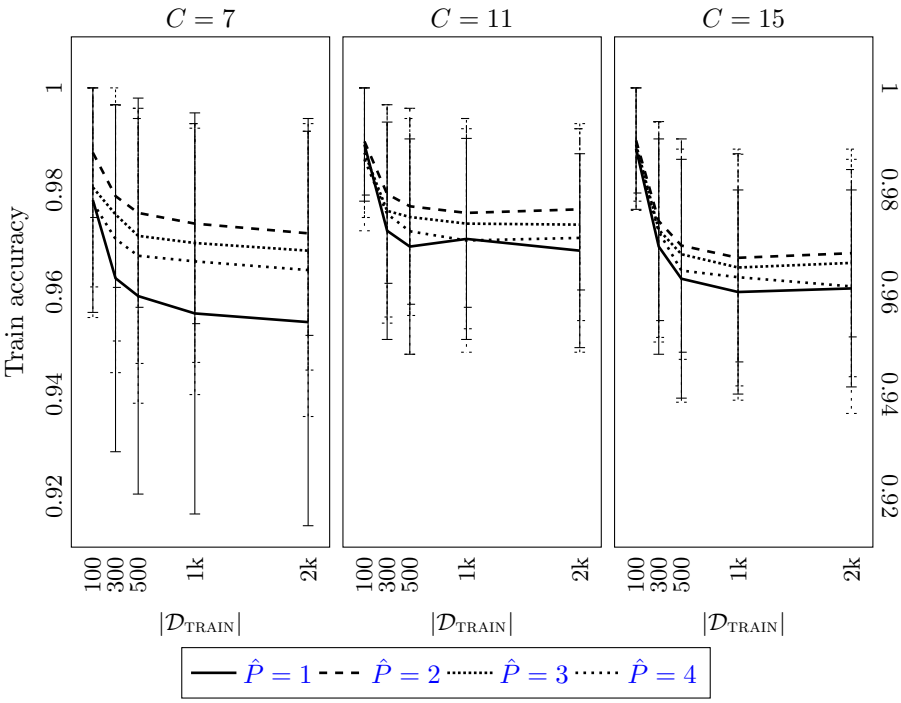



**Fig. 6**: Average train accuracy as a function of the number of training comparisons ($|\mathcal{D}_{\text{TRAIN}}|$) for any number of criteria ($C$) when inferring SRMP models with different numbers of profiles ($\hat{P}$). Bars represent the limits of the 95% confidence intervals.

Looking at the average execution times (Fig 8), we do not observe any significant difference with relation to the results from the previous subsection. The execution times remain linear with relation to the number of provided binary comparisons and increase slightly when more profiles are considered.

## 5.3 Handling noisy data

Finally, we look at the effect that errors in the learning comparisons may have on the performance of the genetic algorithm as well as the quality of the inferred model. We only illustrate the results when the inferred model matches the number of profiles of the original model, however the same conclusions can be drawn for the other situations.

Figure 9 illustrates the training accuracy over training sets of different sizes, for problem instances containing different numbers of criteria and for SRMP with fewer or more profiles. The lines indicate the training accuracy when different percentages of training comparisons are changed to indicate a different assertion from the one obtained thanks to the original SRMP model. This could either be a preference in the opposite direction or an indifference. Since

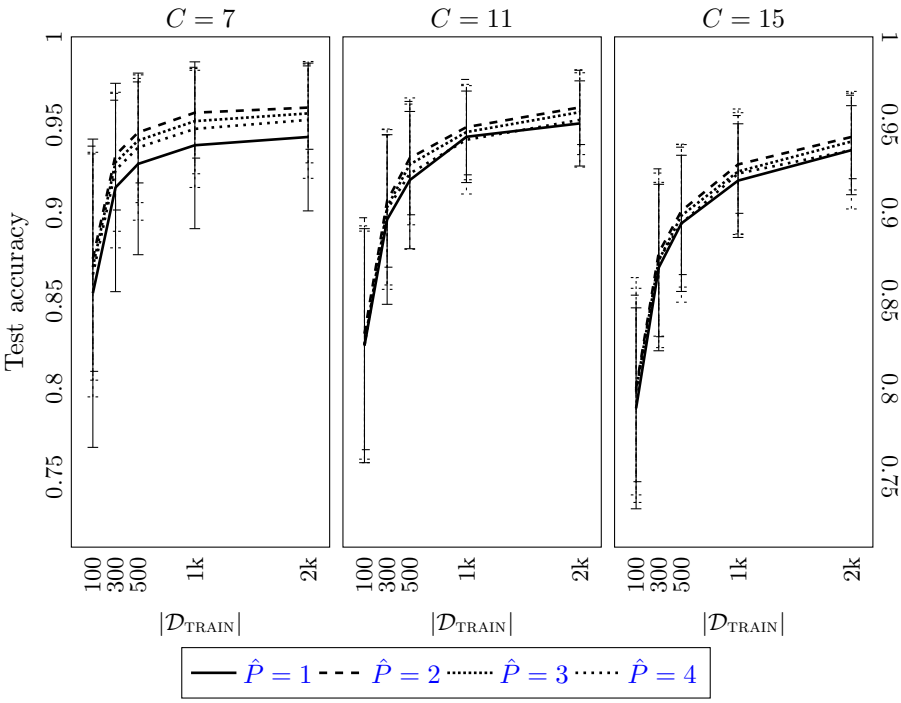



**Fig. 7**: Average test accuracy as a function of the number of training comparisons ($|\mathcal{D}_{\text{TRAIN}}|$) for any number of criteria ($C$) when inferring SRMP models with different numbers of profiles ($\hat{P}$). Bars represent the limits of the 95% confidence intervals.

indifferences between two alternatives have a significantly lower probability of appearing when compared to choices – especially when these alternatives are generated randomly – we make sure to keep the original proportions of indifferences and preferences when inserting errors in the training data.

We find that, as more and more errors are added to the training data, the training accuracy is significantly impacted. This impact seems to be proportional the the amount of errors, reaching 70% accuracy when up to 30% of the training data is erroneous. For problem instances with more criteria, the effect is less prominent when fewer binary comparisons are considered, however, as the training data increases, so does the penalty on the training accuracy.

Figure 10 illustrates the test accuracy for the same scenarios as above. Despite the rather significant effect on training accuracy, the test accuracy does not seem to suffer the same penalty. This is especially visible for problem instances with fewer criteria. As the number of comparisons in the training data increases, so does the test accuracy. It seems that the genetic algorithm is able to satisfy more of the comparisons that are coherent with the original model and a significant proportion of erroneous comparisons are not fulfilled. Nevertheless, as the number of criteria and the number of reference profiles

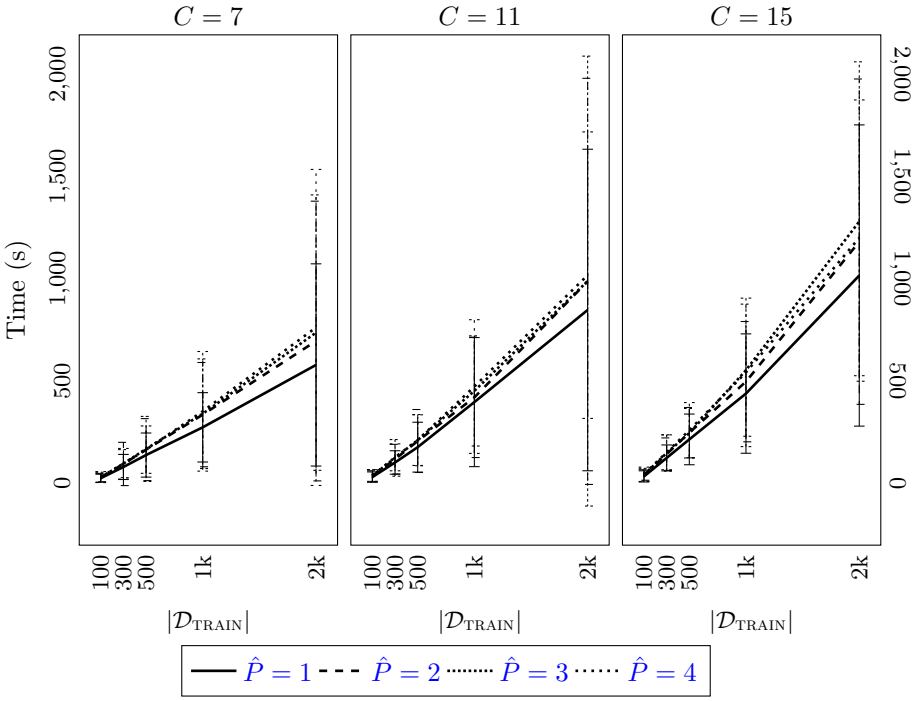



**Fig. 8**: Average execution time as a function of the number of training comparisons ($|\mathcal{D}_{\text{TRAIN}}|$) for any number of criteria ($C$) when inferring SRMP models with different numbers of profiles ($\hat{P}$). Bars represent the limits of the 95% confidence intervals.

increase, the penalty incurred by errors in the training data becomes more noticeable.

We do not illustrate the execution time for the previous tests as no difference was observed when introducing errors in the learning data. The algorithm converges in the same amount of time, on average, which is mainly dictated by the number of binary comparisons and the number of criteria.

# 6 Discussion and conclusion

This article proposes a novel metaheuristic approach for inferring a simple ranking model based on multiple reference profiles. The proposed algorithm is able to handle larger amounts of input information and is more effective in terms of computation times than previous proposals based on mixed-integer linear programming or constraint programming. The extensive experiments conducted in this study demonstrate a generalization power of the learned models similar to the ones reported in previous works, but in significantly lower computation times, even in presence of noisy training comparisons. This

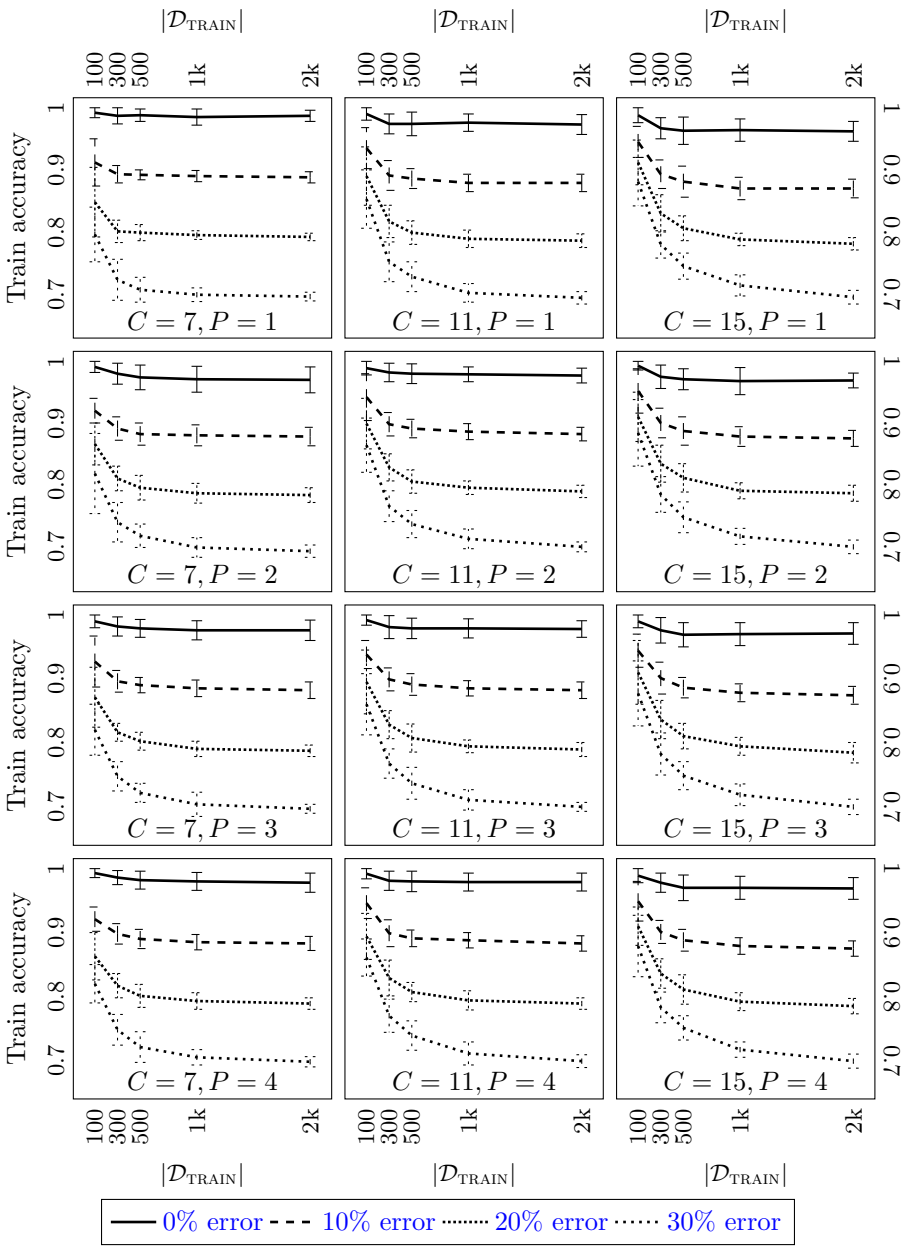



**Fig. 9**: Average train accuracy as a function of the number of training comparisons ($|\mathcal{D}_{\text{TRAIN}}|$) with relation to the error rate among these comparisons, for all combinations of the number of criteria ($C$) and the number of profiles in the original and inferred model ($P = \hat{P}$). Bars represent the limits of the 95% confidence intervals.

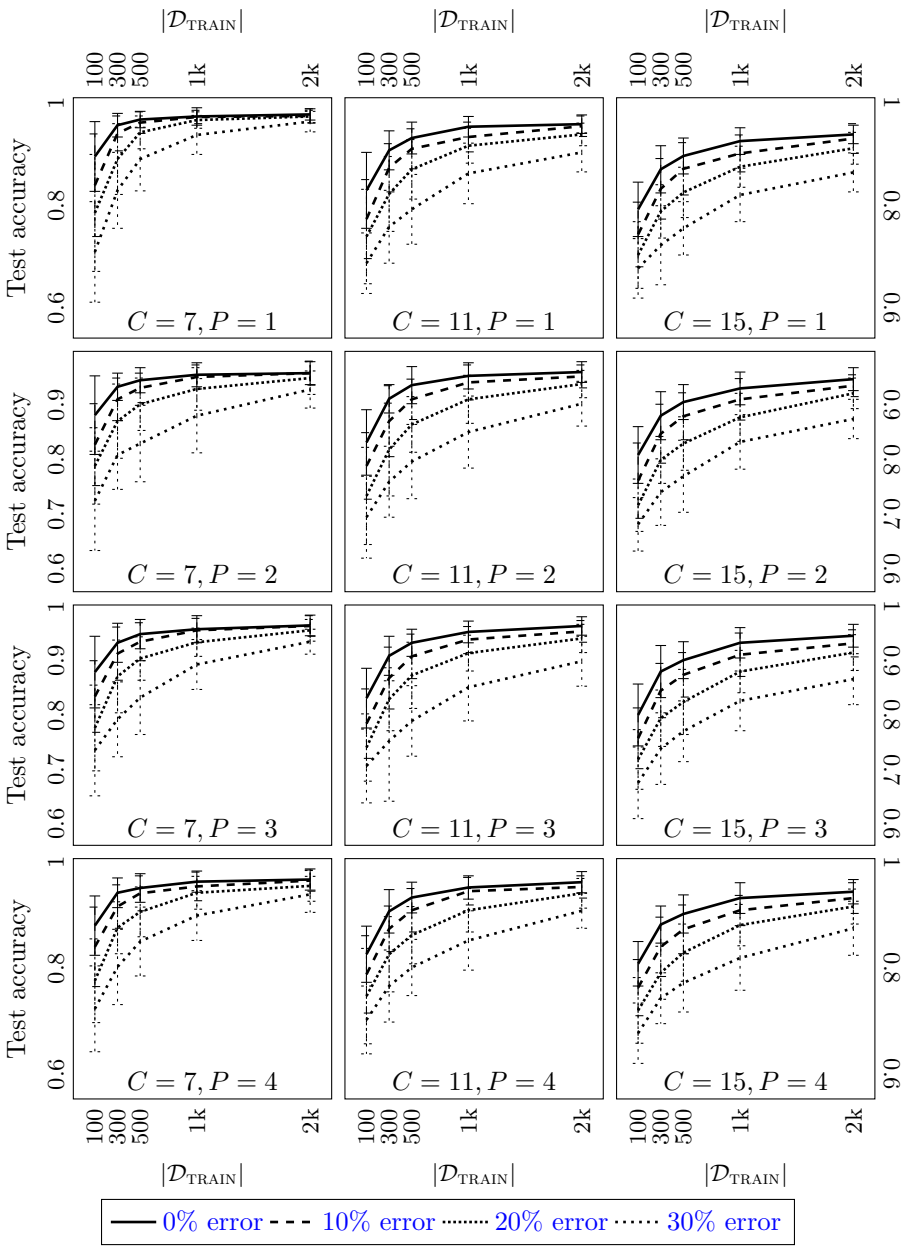



**Fig. 10**: Average test accuracy as a function of the number of training comparisons ($|\mathcal{D}_{\text{TRAIN}}|$) with relation to the error rate in these comparisons for all combinations of the number of criteria ($C$) and the number of profiles in the original and inferred model ($P = \hat{P}$). Bars represent the limits of the 95% confidence intervals.

allows the proposed approach to be used in real-life problems involving many criteria.

Table 4 illustrates the largest problem instances handled by previous works, in terms of number of criteria ($C$), number of reference profiles ($P$), number of input binary comparisons ($|\mathcal{D}_{\text{TRAIN}}|$) as well as the reported computation time required to solve them. For each of these results, we have reproduced the same problem size, averaged for 10 random DMs, for both the single-operator settings and multi-operator settings introduced in this work. Additionally, we have compared results for multiple choices of numbers of alternatives available, to increase the difficulty on our side.

| | | | [12] | [12] | [3] | [3] | [11] |
|---|---|---|---|---|---|---|---|
| | | $C =$ | 5 | 7 | 5 | 6 | 10 |
| | | $P =$ | 2 | 2 | 3 | 2 | 3 |
| | | $\lvert\mathcal{D}_{\text{TRAIN}}\rvert =$ | 100 | 100 | 1000 | 600 | 500 |
| Ref | | Training acc. | **1** | **1** | **1** | **1** | 0.97 |
| | | Testing acc. | 0.89 | 0.86 | **0.98** | **0.97** | n/a |
| | | Time | $6m16s$ | $33m50s$ | $8m45s$ | $18m20s$ | $3m2s$ |
| Ours, multi | $A = 50$ | Training acc. | 0.996 | 0.993 | 0.9881 | 0.992 | 0.984 |
| | | Testing acc. | 0.88 | 0.826 | 0.939 | 0.942 | 0.893 |
| | | Time | **16.229s** | $21.415s$ | $4m44s$ | $3m2s$ | $3m38s$ |
| | $A = 1000$ | Training acc. | 0.997 | 0.993 | 0.976 | 0.981 | **0.988** |
| | | Testing acc. | **0.924** | **0.891** | 0.961 | 0.957 | **0.952** |
| | | Time | $20.699s$ | $27.002s$ | $5m16s$ | $3m19s$ | $3m31s$ |
| Ours, single | $A = 50$ | Training acc. | 0.988 | 0.987 | 0.982 | 0.977 | 0.975 |
| | | Testing acc. | 0.865 | 0.832 | 0.937 | 0.913 | 0.875 |
| | | Time | $16.852s$ | **17.86s** | **4m39s** | $2m37$ | $2m27s$ |
| | $A = 1000$ | Training acc. | 0.981 | 0.988 | 0.973 | 0.97 | 0.971 |
| | | Testing acc. | 0.899 | 0.885 | 0.963 | 0.941 | 0.829 |
| | | Time | $18.742s$ | $18.717s$ | $5m24s$ | **2m2s** | **2m14s** |

**Table 4**: Comparison of the largest problem instances handled by previously published works with ours, averaged for 10 random DMs and choices of $A$ of two different orders, using either the single-operator or the multi-operator settings described in Section 4.

When time budget is not a limitation and problems are small, approaches based on exact problem solving obviously provide better training accuracy than our metaheuristic approach. However, training accuracy of our algorithm remains very high in all considered settings, and required time is generally a lot smaller. This allows us to tackle larger problems than previous work. This is especially true given that our implementation is done in Python without any optimization, while [12] and [11] make use of Cplex solver. [3] makes use of solvers CryptoMiniSAT and Maxino. A C/C++ implementation of our approach would therefore lead to higher gains for those interested in this aspect.

Complementing Table 4, we have observed that our approach in multi-operator settings, can solve problems with $C = 15, P = 4, A = 50$ and $|\mathcal{D}_{\text{TRAIN}}| = 2000$ with high accuracy in about $19m$. Therefore, for more or less the same executing time budget, our approach is able to handle problem instances three times larger in terms of the number of criteria and 20 times larger in terms of the number of input binary comparisons as opposed to a mixed-integer linear programming approach.

Still, a certain amount of questions and perspectives arise from our work. The performance of the proposed approach could probably be further improved by incorporating various problem-specific features, such as heuristics, local search, or hybridization with other optimization techniques. An adaptation for inferring the number of reference profiles without requiring this information as an input parameter could also be done.

Also, the proposed genetic algorithm handles a population of solutions, in which currently only one solution (the best in terms of training accuracy) is output. It could be worth looking in further details at the other "good" solutions of the population, to determine if some of their characteristics could be used to improve the best element of the population.

Finally, a limitation of all approaches in the literature – including ours – is the absence of focus on generalization. Indeed, one may be interested not particularly in the ability of the inferred model to reproduce existing decisions, but more to mimic the behavior a DM would have in new unseen situations. Drawing inspiration from the machine learning field, we believe that introducing a validation set, extracted from the training comparisons, to serve as a proxy for future testing data, would be worth exploring to maximize the generalization ability of inferred SRMP models. Experiments in this work still hint that maximizing training accuracy implies an increase in testing accuracy, and it seems that there is no overfitting problem here (as we are learning a model and not a classification/regression function). Still, taking generalization ability into account during model inference may help convergence. This is in our opinion a great direction for future work.

**Conflict of Interest:** The authors declare that they have no conflict of interest.

**Data availability:** The datasets generated during and/or analyzed during the current study are available in the GitHub repository, https://github.com/BastienPasdeloup/learn_srmp.

# References

[1] Juscelino Almeida Dias, José Figueira, and Bernard Roy. ELECTRE TRI-C: A multiple criteria sorting method based on characteristic reference actions. *European Journal of Operational Research*, 204(3):565–580, 2010.

[2] C.A. Bana e Costa and J-C. Vansnick. MACBETH—An interactive path towards the construction of cardinal value functions. *International*

*transactions in operational Research*, 1(4):489–500, 1994.

[3] Khaled Belahcene, Vincent Mousseau, Wassila Ouerdane, Marc Pirlot, and Olivier Sobrie. Ranking with multiple reference points: Efficient sat-based learning procedures. *Computers & Operations Research*, 150:106054, 11 2022.

[4] D. Bouyssou and T. Marchant. Multiattribute preference models with reference points. *European Journal of Operational Research*, 229(2):470–481, 2013.

[5] D. Bouyssou, T. Marchant, M. Pirlot, A. Tsoukiàs, and P. Vincke. *Evaluation and decision models with multiple criteria: Stepping stones for the analyst*. International Series in Operations Research and Management Science, Volume 86. Boston, 1st edition, 2006.

[6] John H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, MI, 1975. second edition, 1992.

[7] C-L. Hwang, Y-J. Lai, and T-Y. Liu. A new approach for multiple objective decision making. *Computers & Operations Research*, 20(8):889–899, 1993.

[8] E. Jacquet-Lagrèze and Y. Siskos. Preference disaggregation: 20 years of mcda experience. *European Journal of Operational Research*, 130(2):233–245, 2001.

[9] A. Khannoussi, A-L. Olteanu, C. Labreuche, and P. Meyer. Simple ranking method using reference profiles: incremental elicitation of the preference parameters. *4OR: A Quarterly Journal of Operations Research*, 20(3):499–530, 2022.

[10] A. Leroy, V. Mousseau, and M. Pirlot. Learning the parameters of a multiple criteria sorting method. In R. Brafman, F. Roberts, and A. Tsoukiàs, editors, *Algorithmic Decision Theory*, volume 6992, pages 219–233. Springer, 2011.

[11] J. Liu, Wassila Ouerdane, and Vincent Mousseau. A metaheuristic approach for preference learning in multicriteria ranking based on reference points. in Proceeding of the 2nd wokshop from multiple criteria Decision aid to Preference Learning (DA2PL), (2014).

[12] A-L. Olteanu, K. Belahcène, V. Mousseau, W. Ouerdane, A. Rolland, and J. Zheng. Preference elicitation for a ranking method based on multiple reference profiles. *4OR: A Quarterly Journal of Operations Research*, 20(1):63–84, 2022.

[13] A. Rolland. Reference-based preferences aggregation procedures in multicriteria decision making. *European Journal of Operational Research*, 225(3):479–486, 2013.

[14] B. Roy. The outranking approach and the foundations of ELECTRE methods. *Theory and Decision*, 31:49–73, 1991.

[15] B. Roy. *Multicriteria Methodology for Decision Aiding*. Kluwer Academic, Dordrecht, 1996.

[16] A. Tversky and D. Kahneman. Loss aversion in riskless choice: A reference-dependent model. *The Quarterly Journal of Economics*,

106(4):1039–1061, 11 1991.

[17] Bernard L Welch. The generalization of 'student's'problem when several different population variances are involved. *Biometrika*, 34(1-2):28–35, 1947.