Ordered Preference Elicitation Strategies for Supporting Multi-Objective Decision Making

Luisa M Zintgraf
Vrije Universiteit Brussel
University of Oxford

Diederik M Roijers
Vrije Universiteit Brussel
Vrije Universiteit Amsterdam

Sjoerd Linders
City of Amsterdam,
Signal Control Design and Analysis

Catholijn M Jonker
Delft University of Technology

Ann Nowé
Vrije Universiteit Brussel

ABSTRACT

In multi-objective decision planning and learning, much attention is paid to producing optimal solution sets that contain an optimal policy for every possible user preference profile. We argue that the step that follows, i.e, determining which policy to execute by maximising the user's intrinsic utility function over this (possibly infinite) set, is under-studied. This paper aims to fill this gap. We build on previous work on Gaussian processes and pairwise comparisons for preference modelling, extend it to the multi-objective decision support scenario, and propose new ordered preference elicitation strategies based on ranking and clustering. Our main contribution is an in-depth evaluation of these strategies using computer and human-based experiments. We show that our proposed elicitation strategies outperform the currently used pairwise methods, and found that users prefer ranking most. Our experiments further show that utilising monotonicity information in GPs by using a linear prior mean at the start and virtual comparisons to the nadir and ideal points, increases performance. We demonstrate our decision support framework in a real-world study on traffic regulation, conducted with the city of Amsterdam.

KEYWORDS

Multi-Objective Decision Making; Decision Support; Preference Elicitation; Gaussian Processes; Active Learning

1 INTRODUCTION

Understanding what humans want is an integral part of artificial intelligence (AI), and of central importance when using AI to assists humans in making decisions. Consider tasks like picking which film to watch, or deciding on a route for a road-trip through Europe: the amount of options to choose from is often too large for a human to iterate through, making the search for the best option a possibly cumbersome process. In this case, AI can support – and accelerate – the user's decision-making. To this end, the AI system needs to
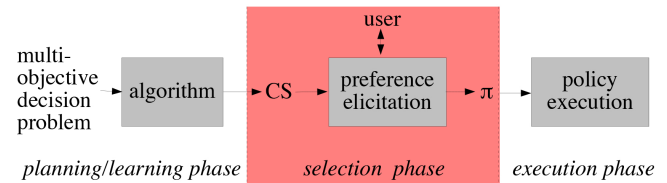

Figure 1: Decision support scenario. In the planning or learning phase a coverage set (CS) for a multi-objective decision problem is produced. In the selection phase, the user selects the policy that maximises his or her utility by interacting with an algorithm for preference elicitation. Finally, the selected policy is executed. In this paper, we focus on the selection phase.

learn about the user's preferences efficiently by *asking the right questions* and guide the search by *generalising to new options*.

AI-supported human decision making is used, e.g., in recommender systems [1, 11, 37] or negotiating agents [24, 27]. In this paper, we focus on decision support in the context of multi-objective decision making [20, 40], where each option has several attributes that influence the user's preferences. The AI subfield of multi-objective decision theoretic planning and learning [3, 31, 34, 35, 47, 49–51] studies complex multi-objective decision problems. Such studies typically focus on producing so-called coverage sets, i.e., a set that contains an *optimal policy for every possible user preference profile* with respect to the different objectives. It is commonly (explicitly or implicitly) assumed that user preferences can be modelled by a utility function that is monotonically increasing in all objectives, leading to a possibly infinitely sized Pareto front as coverage set. However, the selection problem that follows (fig 1), i.e., *selecting the policy the user likes best*, is typically left open. Since we do not have direct access to the user's *intrinsic utility function*, this step is far from trivial. We argue, in accordance with the utility-based approach to multi-objective decision making [39, 54], that the selection phase is an integral part of solving a multi-objective decision problem, and that doing this suboptimally can be detrimental to user utility. Therefore, we believe that algorithms for the selection phase should be seen as an essential part of multi-objective decision-theoretic planning and learning. These should learn about a specific user's utility with respect to the objectives, and find a single (approximately) optimal policy in terms of user utility which can be executed. In this paper we propose and analyse methods to do so, under two main considerations: how to elicit the user's preferences, and how to model these in the multi-objective

setting. To find the policy that the user likes best, we take an *active learning* approach, where we alternate between updating the model of the user's utility and querying the user for feedback by using *relative* feedback queries. Relative comparisons are a natural way for humans to express preferences (as discussed in sec 2.2). To model the user's preferences, we utilise Gaussian processes (GPs), since Bayesian methods like these work well with little data (and interactions with the user are a scarce resource, see sec 2.3). We build on a method that uses a novel likelihood for pairwise comparisons in GPs [13] (sec 2.3), and were further inspired by the active learning setting of Brochu et al. [8], who iterate between asking the user for feedback via a pairwise comparison, updating the GP with this information, and selecting the next query (sec 2.4).

In this paper we extend their approach to other query types, like asking the user to *rank* or *cluster* items (sec 4). Our main contribution is an in-depth evaluation of these in synthetic experiments (sec 5), and in a user study (sec 6). Our results show that ranking queries lead to better utility models, and are preferred by humans over the pairwise and clustering approach. We further propose to utilise the monotonicity assumption of multi-objective decision problems by using a linear prior mean for the GP, and virtual comparisons (sec 3). We show experimentally that this indeed improves performance (sec 5.2). We also found that while using a linear prior mean leads to a large performance increase at the beginning, it is essential to turn off the prior after some initial data has been collected, to not restrict the GP too much. Finally, we demonstrate the implementation of our method in a real-world example; a project in collaboration with the municipality for traffic regulations of Amsterdam (sec 7).

The source code for our methods and experiments can be found at https://github.com/lmzintgraf/gp_pref_elicit.

2 BACKGROUND

In this section we introduce the framework of multi-objective decision making and how user utilities are formalised within it. We then introduce Gaussian processes, and how to use them with pairwise comparative judgements and active learning.

2.1 Multi-Objective Decision Making

Much multi-objective decision making takes place in the context of the *decision support scenario* [39] (fig 1), which applies to multi-objective decision-theoretic planning and learning [39], but also to multi-objective (heuristic) optimisation with, e.g., evolutionary algorithms [15]. The ultimate goal in this scenario is to *maximise user utility* by executing the policy a specific user likes best. However, users typically cannot express their preferences directly, and thus *a priori* scalarisation of the objectives is not possible. In this case, we need an approach that delivers a *coverage set* (CS) of *possibly optimal* solutions. We refer to this as the planning/learning phase. The selection phase that follows, i.e, finding the policy from this set that maximises a specific user's utility, is the focus of this paper.

We call a single solution in a multi-objective decision problem a *policy* $\pi \in \Pi$ and denote its value as $\mathbf{v} \in \mathbb{R}^d$, where $d \geq 2$ is the number of objectives and Π the set of all possible policies. In the context of *multi-objective Markov decision problems* (MOMDPs) [50] for example, a policy's value is its expected cumulative reward in each objective. Following Roijers et al. [39], we assume that the

user has as an *intrinsic utility function*

$$u : \mathbf{v} \mapsto u_{\mathbf{v}} \qquad (1)$$

that maps a multi-dimensional policy value to a scalar $u_{\mathbf{v}} \in \mathbb{R}$ according to the user's preferences (a common assumption in general literature on decision making and preference elicitation [14, 19]). The higher this value, the higher the user's preference for the policy. Since all objectives are desirable, u is monotonically increasing in all objectives. Given this monotonicity property, the solution set is a *Pareto coverage set (PCS)*, i.e., Pareto front, that contains for any allowed policy $\pi' \in \Pi$ with value $\mathbf{v}'$ a policy that has a greater or equal value in all objectives [39]. Consequently it contains an optimal solution for any user preference profile,

$$\forall u \ \exists(\mathbf{v} \in \text{PCS}) \ : \ \forall \mathbf{v}' \in \Pi : u(\mathbf{v}) \leq u(\mathbf{v}') . \qquad (2)$$

When Π is an infinite set of policies (e.g., the set of all possible stochastic policies for an MOMDP), the PCS can contain infinitely many policies. However, due to a result by Vamplew et al. [46], it is typically possible to construct a PCS by mixing policies from a much smaller solution set, alleviating the necessity to compute the PCS explicitly. More precisely, we can use the *convex coverage set (CCS)* which contains non-dominated policies whose values satisfy

$$\forall \mathbf{w} \in \mathbb{R}^d \exists \mathbf{v} \in \text{CCS} : \forall \mathbf{v}' \in \Pi : \mathbf{w}^\top \mathbf{v}' \leq \mathbf{w}^\top \mathbf{v} , \qquad (3)$$

with non-negative weights $\mathbf{w}$ that sum to 1. This insight is important, because the CCS is typically much smaller than the PCS. E.g., in MOMDPs with a finite number of states and actions there is a CCS that is a finite set of deterministic stationary policies. The PCS for MOMDPs is then a piecewise linear surface comprised of (infinitely many) mixture policies that have policies from the CCS as constituent policies. A mixture policy is a stochastic mixture of two or more policies, and the value of this policy is the linear combination of the adjacent policies (see Vamplew et al. [46]). In the decision support scenario, we want to maximise the utility function u, *restricted to the mixtures of CCS policy values* as input. To this end, we need to elicit preference information from the user.

2.2 Preference Elicitation

We distinguish two ways of querying user preferences: scoring of items (absolute feedback), or comparisons between items (relative feedback). Expressing preferences in absolute terms is more difficult for humans and prone to errors, as numbers are an unnatural way to express preferences [45] (consider "I like this film 0.4 much") and the values might change over time [43] with the users mood [18, 44] which may depend on something seemingly trivial like the weather. Relative feedback [45, 55] is easier for humans to express (consider "I prefer film A over B"), and is typically more consistent over time [28]. We therefore focus on relative feedback.

When a user compares policies $\mathbf{v}_1$ and $\mathbf{v}_2$, we assume there are true utility values $u(\mathbf{v}_1)$ and $u(\mathbf{v}_2)$ which the user cannot directly access, but uses indirectly to compare items. Since personal evaluations by users are likely not always 100% accurate and vary depending on different factors, we model the outcome of a comparison based on *noisy* utility values $u(\mathbf{v}) + \varepsilon$, where $\varepsilon \sim \mathcal{N}(0, \sigma^2)$ is Gaussian noise with zero mean and unknown variance (following [8, 13]). The observations we get from the user thus stem from the

comparison of the true utility values, contaminated with noise,

$$u(\mathbf{v}_1) + \varepsilon_{\mathbf{v}_1} > u(\mathbf{v}_2) + \varepsilon_{\mathbf{v}_2}, \qquad (4)$$

denoted as $\mathbf{v}_1 > \mathbf{v}_2$ (meaning the user prefers $\mathbf{v}_1$). We want to use such relative preferences to find the policy π^* whose value $\mathbf{v}^*$ maximises the user's utility function u. In the next section we introduce an approximation method that is suitable for this task.

2.3 Gaussian Processes for Pairwise Feedback

A key objective when asking for feedback is to not bother the user with too many queries – otherwise, the user might quit using our system before its goals are achieved. We therefore need methods that work well on little data and take the uncertainty about the user's preferences into account. Bayesian optimisation techniques, like Gaussian processes, are particularly suited for tasks where there is little available data [17].

Gaussian processes (see [38] for a general introduction) are used to approximate functions, and can be seen as an infinite-dimensional extension of a multivariate Gaussian distribution. A GP approximates a function and captures its uncertainties by assigning to each point in its domain a normal distribution – the mean value reflecting the expected value, and the variance reflecting the uncertainty about the function value at that point. The Gaussian process is fully specified by a mean function m and a kernel k,

$$u(\mathbf{v}) \sim GP(m(\mathbf{v}), k(\mathbf{v}, \mathbf{v}')) . \qquad (5)$$

Before querying the user for information, the GP is initialised by defining a prior mean function and the kernel function. Common choices are the zero function $m(x) = 0$ for the prior mean and the squared exponential kernel which we also use here; an alternative prior mean function is discussed in section 3.2.

The prior belief $P(u)$ (defined by the GP) about the utility function u is updated in light of new data and its likelihood $P(D|u)$ using Bayes' rule $P(u|D) \propto P(u)P(D|u)$. The posterior $P(u|D)$ is the updated belief about the user's utility, and the current approximation of the utility function. The data $\mathcal{D}$ in our case is given in terms of pairwise comparisons (see equation (4)),

$$\mathcal{D} = \{\mathbf{v}_m > \mathbf{v}'_m\}_{m=1}^M . \qquad (6)$$

Chu and Ghahramani [13] introduce a probit likelihood for such noisy pairwise comparisons. Given this likelihood and a Gaussian process prior, the posterior is analytically not tractable and has to be approximated. Following [7], we use Laplace approximation.

To approximate the user's utility with a GP, we take an active learning approach, meaning we alternate between updating the GP (approximating the posterior) and querying the user for more information (based on our current approximation). In the next section, we describe how to select the next question for the user.

2.4 Active Learning

Chu and Ghahramani [13] show that GPs with the above defined pairwise likelihood work well on fixed datasets, where learning is done off-line and (during learning) we do not have influence on the information that is collected from the user. Here instead, we are interested in an active learning setting, where the queries for new information are based on the current information and belief about the user's preferences. Functions that select a next item given a

Gaussian process are called acquisition functions. These typically try to balance exploration (querying the user about items with high uncertainty in user utility) and exploitation (suggesting items to the user that have high expected utility). Following [7, 8], we use the *expected improvement* acquisition function [30] to, at each step, select an item about which the user is queried about next.

3 MULTI-OBJECTIVE DECISION SUPPORT WITH GAUSSIAN PROCESSES

To use GPs for optimising user utility in multi-objective decision support, we restrict the domain of the acquisition function (3.1) and utilise monotonicity information about the user's preferences (3.2).

3.1 Input Domain

The input domain of the user's utility function, as well as the GP we use to approximate it, is the d-dimensional hypercube defined by the (possibly unknown) minimum and maximum possible values of the d objectives. In the multi-objective planning or learning phase (fig 1), a *coverage set* is produced which contains an optimal solution for every possible user. This will typically be an infinite subset of the d-dimensional hypercube, since trade-offs between the objectives constrain the solution set. On this set, the user's utility function can have local maxima. Since we aim to maximise the user's utility on this solution set, we restrict the acquisition function to this set. This way, we only present achievable solutions to the user. Doing otherwise could bias the user, give rise to unrealistic expectations, and we might waste time asking the user about items that are not feasible.

3.2 Utilising Monotonicity Information

In the multi-objective decision support scenario, we assume the user's utility function to be monotonically increasing in each objective. This information can be used in the GP approximation.

3.2.1 Prior Mean. If no information about the function that is approximated is known a priori, the most commonly used GP prior mean is the zero function (sec 2.3). However, because we have a monotonic utility function, we want to use a better heuristic as a prior. We propose a *linear* prior mean function with equal weights for all objectives for this heuristic. Equal weights implies that the user would care equally about all objectives, and linearity means that the user's utility increases linearly in the value of each objective. Note that a linear prior mean function does not imply that the GP can only model linear functions; the shape of the functions is determined by the kernel.

In Section 5.2 we show that adding such a heuristic prior is highly useful when not many queries have been posed to the user, but can hinder the GP when more data becomes available. Hence, we propose to remove this heuristic after an initial number of queries.

3.2.2 Virtual Comparisons. We optimise the user's utility on the set of achievable and possibly optimal solutions, i.e., the PCS. Around the PCS however, we can add virtual comparisons (which are not shown to the user) that include items that lie *outside* the PCS, to enforce monotonicity. We propose to compare each item that is selected by the acquisition function to the *minimal* and *maximal* item (also called the nadir and ideal points). These are the vectors

that contain the minimal (resp. maximal) value for each objective, independent of the other objective values. We hypothesise that because this extra information enforces some monotonicity for the GP, it will lead to better approximations.

4 PREFERENCE ELICITATION STRATEGIES

This section introduces the query types we study in this paper; pairwise comparisons (from previous work), and new ranking/clustering strategies. We describe all query types in the following, and show orderings that may result in figure 2.

- **Pairwise Comparisons.** Brochu et al. [7] use an active learning approach with pairwise comparisons: in each step, the user compares two items; the winner of which will be compared to a new item in the next query. Thus after N queries, the user made $N-1$ comparisons, which are used to (sequentially) update the GP. This typically leads to a partial ordering of items (2a); the extreme cases being a total ordering and a degenerate case where the first item is better than all items from subsequent queries.
- **Ranking.** Here, the user is asked to make a full ranking of all inputs, leading to a total ordering over the items (2b). The user initially starts with ranking two items, and in each round one more item is added, which has to be sorted into the ranking. I.e., after N queries the user has evaluated $N-1$ items and ordered those as a sorted list. We use the $N-1$ pairs of *successive* items to update the GP. Note that this ranking approach leads to the same number of comparisons as pairwise comparisons, but we get a full ordering. In other words, the data quantity is the same (assuming the user does not re-arrange items), but the information quality is higher when doing ranking, compared to pairwise comparisons.
- **Clustering.** Instead of requesting a full ranking, we can ask the user to: pick a best item, and put the other items into clusters of decreasing utility (2c). Items whose utility values are close are put in the same cluster. The number of clusters can either be pre-determined or defined dynamically by the user. With two clusters, the user's preferences would then be expressed as

$$x_{best} > C_1 > C_2 \,, \tag{7}$$

where x_{best} is a single best item, and C_1 and C_2 are disjunct sets whose union contains the remaining items. $C_1 > C_2$ means that the items in cluster C_1 are better than the ones in C_2. We can then use this data in our Gaussian process with the pairwise comparison likelihood by comparing x_{best} to every item in C_1, and every item in C_1 to every item in C_2, giving us $|C_1|+|C_1|*|C_2|$ pairwise comparisons. In general, for any number of clusters, we have *at least* $N-1$ comparisons.
- **Top-Rank.** This query type is a mix of the above: the user is asked to rank the top k items (in figure 2d we chose k=3), and all remaining items are put in one cluster. This gives us $N-1$ pairwise comparisons. We test this approach since more information about the *upper region* of the user's utility function might help find the maximum faster.

Each experiment starts by asking the user to compare two items, independent of the query types. In each subsequent step, one new item is selected using the acquisition function, and the question asked to the user will depend on the query type. For all ranking/clustering queries (2b-2d), we allow the user to re-order the
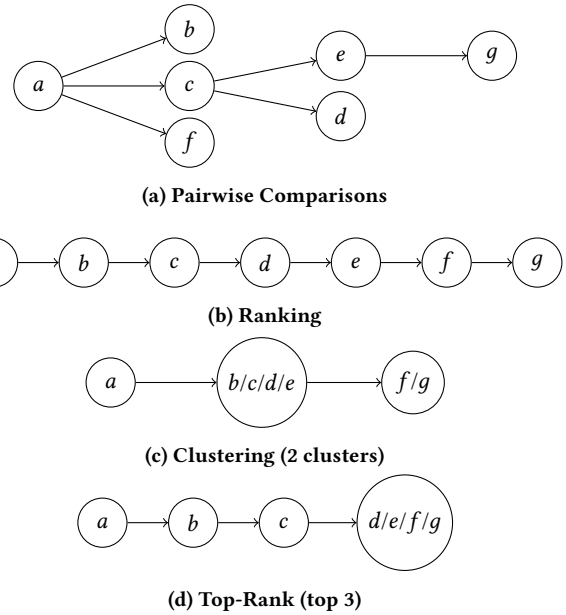

(a) Pairwise Comparisons

(b) Ranking

(c) Clustering (2 clusters)

(d) Top-Rank (top 3)

Figure 2: Possible outcomes of different query types for items a-g, with utilities $u(a) > \ldots > u(g)$. The arrows represent the preference information expressed by the user (preferred $\rightarrow$ unfavoured). Different elicitation strategies lead to different orderings: full ranking returns a total ordering (b); the other query types typically lead to partial orderings.

items in each time step, or merge items into the same cluster. The history of previous queries is kept in the dataset that is used to update the GP. Due to the noise in the utility values and since users are allowed to re-arrange items, it can happen that the relative order of two items is swapped between queries. We handle these apparent inconsistencies by keeping two contradicting pairwise comparisons in the dataset; experiments showed that there is no significant difference to removing them (results not shown).

In the following sections, we evaluate the different query types and study the effect of utilising monotonicity information. We first do this by simulating a user behaving according to a given utility function (sec 5) in order to have access to the true utility values. We then present results from a user study in which we let humans answer queries of different types in a multi-objective decision problem (sec 6). Finally, we combine our findings in a real-world application in collaboration with policy makers for traffic regulation in the city of Amsterdam (sec 7).

5 EXPERIMENTS I: OPTIMISATION QUALITY

In this section, we evaluate our proposed strategies in terms of how well we can optimise the user's utility, i.e., how *close* and how *fast* we get to the maximal achievable utility. For this we need access to the ground truth utility function of the user. We therefore use a virtual user whose utility function we know (read: define). We assess utilising monotonicity information in section 5.2, and the different query types in section 5.3. In the following, we first describe our experimental set-up and how we defined the virtual user's utility function and simulate the decision making process.

5.1 Virtual Utility Function

First, we define a virtual user by defining a monotonic utility function that represents its preferences, taking the following form

$$u(\mathbf{v}) = \sum_{i=1}^{d} w_i \cdot f_i(v_i) \,, \tag{8}$$

where $\mathbf{v}$ is the d-dimensional value of a policy, $\sum_{i=1}^{d} w_i = 1$, and $f_i(v_i)$ is a monotonically increasing function that maps the i-th entry of the vector $\mathbf{v}$ to a scalar value. The functions f_i can take two forms, either that of stacked sigmoids

$$f_i(x) = \sum_{j-1}^{n} \frac{1}{1 + \exp(-x(a - i) + (b + i))} \tag{9}$$

or a polynomial

$$f_i(x) = (cx - 1)^3 \,. \tag{10}$$

In our experiments, one run consists of a number of queries asked to the (virtual user). After each query, the GP is updated and a new item is chosen with the acquisition function. For each run, we first randomly (uniformly) choose the d utility functions f_i we use, as well as the values $a \in [10, 50]$, $b \in [1, 20]$, $n \in [1, 10]$ and $c \in [1, 5]$ (separately for each f_i). We further randomly generate solution sets (PCSs) which hold all possibly optimal policies, and on which we want to maximise user utility. The input space is always $x \in [0, 1]$, and we normalise the functions f_i individually to map to $[0, 1]$ (omitted here in favour of readability) to allow easy comparisons between the results. Figure 3 shows examples of stacked sigmoids, polynomials, and general utility functions.

Given this utility function, we simulate users expressing their preferences over items. We start by selecting two starting items x_1 and x_2 using the acquisition function (note: with zero prior information, these are selected uniformly at random) and then alternate between evaluating the query, updating the GP, and selecting a new item from the acquisition function again. How the queries are evaluated with our virtual utility function is described in the following for the different query types.

- **Pairwise**: At each step, the winner of the previous query x^* and a new item x_{new} are compared. We evaluate the utility function to obtain $u(x^*)$ and $u(x_{new})$, and add noise ε^* and ε_{new}, drawn i.i.d. from a normal distribution $\mathcal{N}(0, \sigma^2)$. The noisy utility values $u(x^*) + \varepsilon^*$ and $u(x_{new}) + \varepsilon^*$ are compared and the result is added to the dataset $\mathcal{D}$. The winner of this comparison is used in the next query (with different noise).
- **Ranking/Clustering**: At each step, we have N items from previous queries, and a new item x_{new}. We evaluate the utility function to obtain $u(x_1), \ldots, u(x_N), u(x_{new})$, and add noise $\varepsilon_1, \ldots, \varepsilon_N$, ε_{new}, drawn i.i.d. from a normal distribution $\mathcal{N}(0, \sigma^2)$. We then produce a ranking or clustering based on the noisy utility values. Note that in each query, we draw new noise values for all items.
- **Clustering**: To simulate the clustering, we use K-Means (see, e.g., [5]). There is no general agreement in the literature on how humans perform clustering, especially since this depends heavily on the task itself and prior knowledge. Just like the shape of the utility function itself, using K-Means is therefore a somewhat arbitrary choice based on the intuition of the authors.
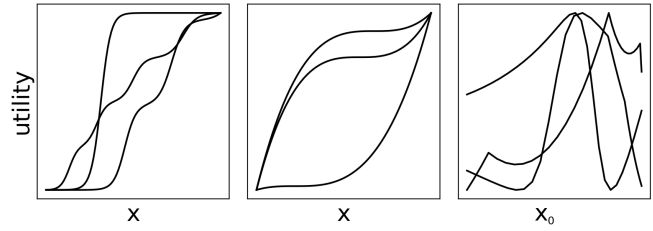

Figure 3: Examples of stacked sigmoids (9), polynomials (10), and virtual user utility functions (8). On the right, we used $d = 2$ and show a slice through the utility function: the mapping of the first objective (x_0) to the solution set (PCS).
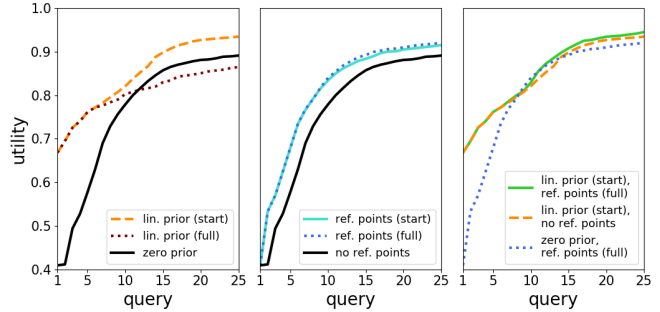

Figure 4: *Utilising Monotonicity*: **Performance when using different prior functions (left), reference points (middle), and a mix of both (right). We switch from a linear to a zero prior after 5 queries, and stop adding virtual comparisons after 5 queries as well. The results are averaged over all query types and 100 runs each. We used $d = 5$ objectives and low utility noise with $\sigma = 0.01$.**

5.2 Results: Utilising Monotonicity

Within the multi-decision making framework described in section 2.1, we assume that the user's utility function is *monotonic*, i.e., all objectives are desirable. In section 3, we proposed two strategies for utilising this information: adding a linear prior, and adding virtual comparison points. We compare these strategies (and combinations of them) by looking at the reached utility at each query step. Here, we are interested in both how fast good user utility is reached, and the final utility value.

Figure 4 shows the results for adding information via the GP prior mean function (left), adding virtual comparisons (middle) and a mix of these (right). Compared to not using any prior information, adding monotonicity information always gives a performance boost. Using a linear prior mean function for the GP at the beginning helps the acquisition function focus on the promising regions, and jump-starts the reached utility. However using a linear prior mean throughout all queries, the performance at some point drops lower than when using a zero prior mean from the start. The same does not hold for reference points. This is because while a linear prior mean function is a heuristic, that might hinder the search process later on, virtual comparisons only enforce monotonicity. The plot on the right shows that a combination of a linear prior at the beginning and virtual comparisons throughout all queries works best.
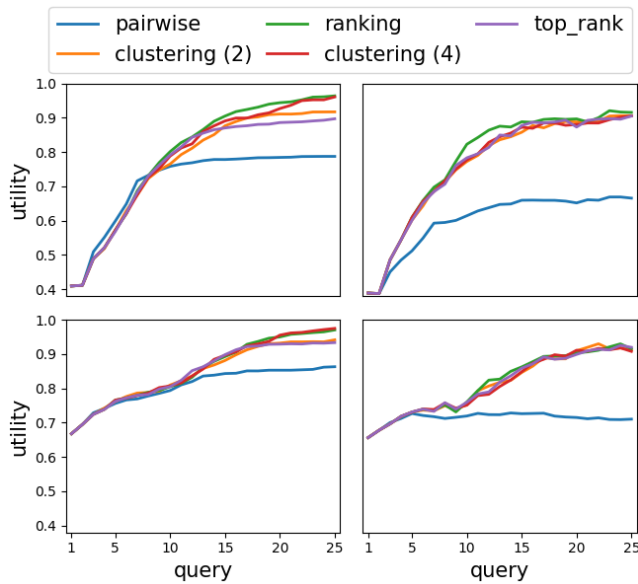
Figure 5: *Utility*: This graph shows the reached utility per query. We used $d = 5$ objectives and different noise levels $\sigma = 0.01$ (left) and $\sigma = 0.1$ (right). The results are averaged over 100 runs. The top row shows the performance without using prior information, and the bottom shows the performance when using a linear prior mean function at the beginning, and reference points throughout. Pairwise comparisons are outperformed by other query methods both regarding convergence speed and final reached utility. (Best viewed in colour.)
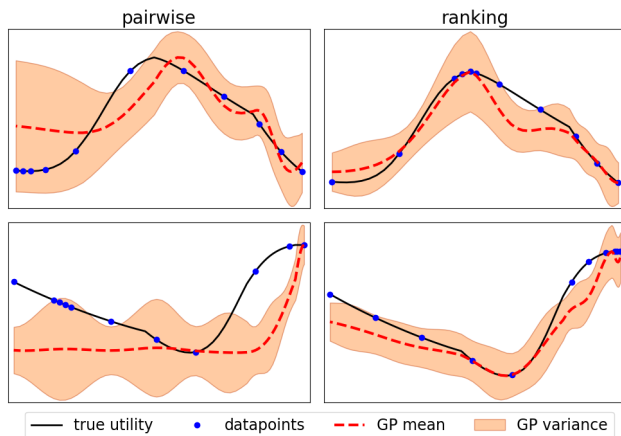

Figure 6: *GP Approximation*: This graph shows the approximation of the GP to the true utility function, for $d = 2$ objectives and when plotting the first objective against the utility on the PCS. We show this for two different random seeds (rows) and for the pairwise (left) vs. ranking (right) query, after 10 queries. As we can see, the ranking query is able to approximate the GP much better. This is due to the different types of orderings (partial vs full) available to the GP on the given datapoints. We hypothesise that a better approximation enables the acquisition function to suggest better points per query, which explains the superior performance of ranking queries.

5.3 Results: Query Types

In this section, we compare our proposed elicitation strategies to the pairwise approach of [8] in terms of approximating the utility function, and attaining a high utility. Figure 5 shows, per time step, the utility of the currently highest rated item, for $d = 5$ objectives and two different noise levels (low/high). The pairwise approach is outperformed by all other methods, especially when there is high noise in the utility function. The full ranking generally performs best, but the difference to clustering and top-rank is not significant when there is a lot of noise in the utility function.

We plot the number of queries against the reached utility, so that each method at each time step has the same datapoints available (since one item is added in each query, regardless of the query type). The difference is in the number of comparisons between those datapoints, which is lowest for the pairwise approach. We note though that the better performance cannot be solely because of the higher number of available comparisons: ranking and clustering methods not only reach higher utility faster, they also converge to a higher utility compared to the pairwise approach.

We believe that the superior performance of ranking and clustering queries stems from having a better approximation to the utility function. While we are only interested in finding the maximum, and not necessarily in closely approximating the user's utility function as a whole, a better approximation helps the acquisition function to make a more informed decision about which item to select next. In figure 6 we show the GP approximation to two different utility functions for pairwise queries and ranking after 10 queries. While both methods are close to the maximum, ranking queries give a better overall approximation. The pairwise method also found an item with near-optimal utility, but since it does not know anything about the relationship between the remaining datapoints, the approximation quality in these areas is poor, and the variance high.

6 EXPERIMENTS II: USER STUDY

In the previous section, we used a virtual user with a utility function we defined ourselves, and we found that ranking queries lead to the best results. In this section, we investigate how real users experience and interact with the system.

6.1 Description of User Study

To assess which query type humans prefer and how well they work in reality, we built a web-interface for multi-objective decision making, including three query types: pairwise, ranking, and clustering (with two clusters). We chose to not test the top-rank method, since this is a mix between ranking and clustering, and we think this might become more relevant for larger studies where many items are compared (and full ranking becomes infeasible).

We chose a decision making scenario over job offers (as previously used in, e.g., [36]). Each job offer has three attributes: the number of days per week on which the employee can work from home, the salary, and the probation time in months. We randomly generated 50 job offers with these attributes. We then defined a mathematical utility function over job offers, and selected three pairs of jobs with low utility to present in the initial query. We used a zero prior for the GP, and used the acquisition function as described in section 2.4 to select a new job offer at each step.
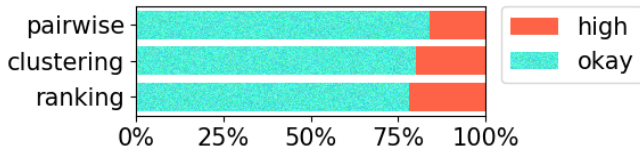
Figure 7: *Effort*: **For each query type, the participant was able to choose between the options 'effort was okay', and 'too much effort'. Users typically found the effort acceptable for all query types.**

Each participant was asked to answer queries about job offers for all query types, according to a *persona description*, in which we described the utility function over jobs in simple words. The participants were given one minute to answer queries of each type. At the end, the user was asked to complete a survey about his or her experience. We collected 50 responses for this user study via Amazon Mechanical Turk. When running the experiments, we randomised the order in which the user took the experiments (i.e., query types) as well as the pair of starting jobs.

6.2 Results

After participating in our experiment, we asked users about the perceived effort per query, whether they think the algorithm understood their preferences, and which query type they prefer for a decision-making tool. The results are shown in figures 7 - 9. We also logged the user's responses to the queries and show the average reached utility per step (according to the true utility function we described to the user in words) in figure 10.

The results show that the majority of participants found the effort to be acceptable across all queries (fig 7), and ranking queries were rated as high effort slightly more often. Most users felt that the algorithm understood their preferences and suggested better jobs over time (fig 8), and they felt most often so for the pairwise query. Despite these results, pairwise comparisons was not the query type that users prefer for a decision-making tool (fig 9). To our surprise, the ranking query was rated 1st most often, and 3rd least often. It further surprised us that clustering was consistently ranked lowest, and that users did not seem to like it. In terms of reached utility (fig 10), all query types show similar performance. The figure also shows the average number of queries answered in one minute per experiment type, and we see that the users were able to answer on average 15 pairwise queries, and 8-10 for clustering/ranking.

In conclusion, users prefer the ranking query type in this one-minute experiment, although it requires slightly more effort and the users felt less often understood by this method than the other query types. We hypothesise that this is because the ranking gives more control over the process, and items can be visually compared to previous items. We do not have a good intuition about why clustering was so unpopular. We expected this to be the easiest and most popular method, since jobs only had to be sorted into three categories (single best job/good jobs/bad jobs).

7 EXPERIMENTS III: APPLICATION

In order to test our decision making tool in a real-world application, we worked on a project together with the municipality of Amsterdam. Part of their daily work is traffic control in the city,
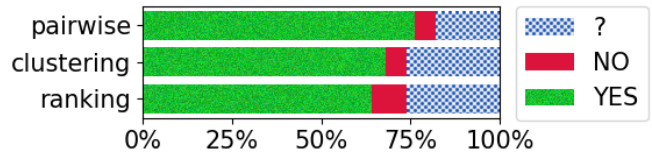

Figure 8: *Understanding*: **We asked the users whether they felt like the algorithm understood their preferences and suggested better jobs over time. Most people thought it did.**
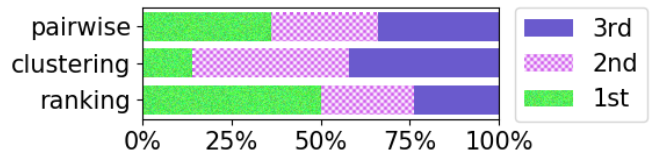

Figure 9: *Preference*: **We asked the users to rank the query types, according to what they would prefer as a tool to support them in a decision-making situation. Most users preferred ranking.**
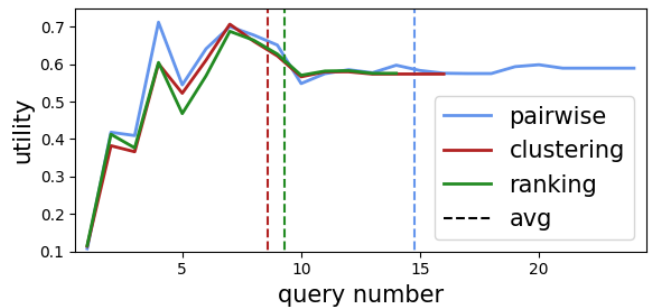

Figure 10: *Utility*: **This graph shows the reached utility over time, according to the utility function we defined and described to the participants in words. All query types have similar performance. We can also see that the average number of answered queries in the pairwise setting is about 15, and 8-10 in the clustering/ranking scenario within the given minute.**

which is a complex real-world task with many objectives. For critical intersections in the city, the municipality utilises specialised simulation software to test policies for traffic regulation. However, since there are many free parameters and each setting needs to be tested several times to account for variance in the simulation, it is near impossible for humans to manually test all settings. We thus hypothesise that policy makers can benefit from a decision-making support system like ours.

For our experiment, we gathered results from the simulation software (for 256 different parameters settings, averaged over 10 runs each), simulating two hours during rush hour for one of the city's busiest traffic areas. There are 11 objectives, which together reflect the delay duration and queue length in different directions and for different traffic participants. First, we removed the Pareto-dominated results, leaving us with 75 parameter settings with possibly optimal value. Two experts from the municipality then used our web-interface as a tool to find the best outcome, according to their intrinsic utilities. In this case, we have no access to the true utility

	Expert 1	Expert 2
Order	pairwise - ranking	ranking - pairwise
# Queries: pairwise	22	20
# Queries: ranking	8	12
Time (min): pairwise	4:06	2:37
Time (min): ranking	1:43	6:14
Preference	ranking	ranking

Table 1: Results in terms of how many items the experts looked at, how long each experiment took, and which query type they preferred.

values of the users. Our experiments therefore investigate whether the experts think the system is useful when used in a real-world problem, and which preference elicitation strategy they prefer.

We only used pairwise and ranking queries in this experiment, since clustering performed poorly in our user study (sec 6). Each participant started with one of the two query types, and then switched to the other. Instead of having a time limit, the participants now had the choice of either requesting a new item, or finishing the experiment by saying they are satisfied with the current outcome.

Table 1 shows the results from this experiment. Both experts answered about twice as many queries for pairwise comparisons than for ranking before they were satisfied with the outcome, and both spent more time on the first experiment than on the second, regardless of query type. Both experts preferred the ranking query types. They both found the same item in the ranking query, but different ones in the pairwise comparison. We received positive feedback from the experts, who found our multi-objective decision support system to be a useful tool. They preferred the ranking queries since it makes it easier to compare items.

We thus conclude that our multi-objective decision support system – using ranking, a temporary prior heuristic, and virtual comparisons – can be used by policy makers to aid them in their work.

8 RELATED WORK

In this paper we model the utility function using GPs, which has important advantages: it explicitly models the uncertainty over the function, enabling active learning, and in contrast to e.g., using POMDPs or predecessing methods [6, 10], it can handle a continuum of items. The main framework for relative preferences in Gaussian processes we use is taken from the work of Chu and Ghahramani [13]. They propose a likelihood for pairwise comparisons that can be used in GPs for preference learning. Their experiments are done on fixed datasets, i.e., they do off-line instead of active learning. The extension of these relative preference GPs to active learning by Brochu et al. [7, 8] asks the user to compare two items in each query; the winner from the last query and a new item suggested by the acquisition function. Chu and Ghahramani [12] also propose an active learning approach with relative preference GPs, but for the learning to rank task. They, at each step, select the *pair* of items with the highest expected entropy term. The objective in learning to rank is to approximate the unknown function, instead of finding the best item. Jensen et al. [26] extend the approach of Chu and Ghahramani [13] so that the user can in addition specify a degree of preference when comparing two items, for which they propose a

novel likelihood to use with the Gaussian process. While this idea is in principle orthogonal to ours, due to the intrinsically ordinal nature of human preferences, we consider the risk of misspecification by introducing a ratio scale for degrees of preferences too high to attempt integrating it with our method. Pairwise preferences with GPs have been applied to a variety of settings, such as sound quality reduction mechanisms in hearing aids [21], material design [8, 9] and learning to predict emotions expressed in music [32]. We believe that our strategies could be useful in many of these.

9 CONCLUSION

In this paper, we proposed new strategies to elicit user preferences in an active learning setting with Gaussian processes, by asking the user to rank or cluster items. We showed that compared to previous work which uses pairwise comparisons, ranking and clustering methods can both find a better approximation to the utility function, and select items with higher utility for the user in fewer queries. Furthermore, we showed that we can obtain better results by entering monotonicity information into the GP via virtual comparisons. We further showed that utilising a heuristic linear prior mean function during the first queries leads to increased performance, but that this heuristic should be turned off in later queries. A user study showed that the ranking method was most popular, although the perceived effort was slightly higher, and users felt less often understood by this method than for example the pairwise comparisons. Surprisingly, clustering methods were least popular. Finally, we successfully used our proposed decision-making tool in a real-world project on traffic regulation together with the municipality of Amsterdam. We showed how human expertise can successfully be combined with decision-making tools, and received positive feedback from our collaborators. We hope that this work inspires new real-world applications involving multiple objectives, and will be used in multi-objective optimisation [15, 16, 23, 29], planning [4, 41, 48, 53], and RL [2, 33, 42, 52] alike.

In future work, we aim to perform more extensive real-user experiments. We aim to extend our approach to multi-user settings, like has been done by Houlsby et al. [25] for pairwise feedback from multiple users or Guo et al. [22] who, for a new user, use an informed GP prior inferred from other. We also aim to study acquisition functions that pick more than one next item, based on look-ahead information gain [6]. Furthermore, we are interested in investigating a more principled approach to deciding when to disable the linear prior mean function.

ACKNOWLEDGMENTS

REFERENCES

[1] G. Adomavicius and A. Tuzhilin. 2015. Context-aware recommender systems. In *Recommender systems handbook*. Springer, 191–226.

[2] Peter Auer, Chao-Kai Chiang, Ronald Ortner, and Madalina Drugan. 2016. Pareto front identification from stochastic bandit feedback. In *Artificial Intelligence and Statistics*. 939–947.

[3] Leon Barrett and Srini Narayanan. 2008. Learning all optimal policies with multiple criteria. In *Proceedings of the 25th international conference on Machine learning*. ACM, 41–47.

[4] Nawal Benabbou and Patrice Perny. 2017. Adaptive elicitation of preferences under uncertainty in sequential decision making problems. In *The 26th International Joint Conference on Artificial Intelligence*.

[5] Christopher M Bishop. 2006. *Pattern recognition and machine learning*. springer.

[6] C. Boutilier. 2002. A POMDP formulation of preference elicitation problems. In *AAAI*. 239–246.

[7] E. Brochu, V. M. Cora, and N. De Freitas. 2010. A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *arXiv:1012.2599* (2010).

[8] E. Brochu, N. de Freitas, and A. Ghosh. 2008. Active preference learning with discrete choice data. In *NIPS*. 409–416.

[9] E. Brochu, A. Ghosh, and N. de Freitas. 2007. Preference galleries for material design.. In *SIGGRAPH Posters*. 105.

[10] U. Chajewska, D. Koller, and R. Parr. 2000. Making rational decisions using adaptive utility elicitation. In *AAAI*. 363–369.

[11] Konstantina Christakopoulou, Filip Radlinski, and Katja Hofmann. 2016. Towards Conversational Recommender Systems.. In *KDD*. 815–824.

[12] W. Chu and Z. Ghahramani. 2005. Extensions of gaussian processes for ranking: semisupervised and active learning. *Learning to Rank* (2005), 29.

[13] W. Chu and Z. Ghahramani. 2005. Preference learning with Gaussian processes. In *ICML*. 137–144.

[14] R. T. Clemen and T. Reilly. 2013. *Making hard decisions with DecisionTools*. Cengage Learning.

[15] Carlos A Coello Coello, Gary B Lamont, David A Van Veldhuizen, et al. 2007. *Evolutionary algorithms for solving multi-objective problems*. Vol. 5. Springer.

[16] Kalyanmoy Deb, Lothar Thiele, Marco Laumanns, and Eckart Zitzler. 2005. Scalable test problems for evolutionary multiobjective optimization. In *Evolutionary multiobjective optimization*. Springer, 105–145.

[17] M.P. Deisenroth, D. Fox, and C.E. Rasmussen. 2015. Gaussian processes for data-efficient learning in robotics and control. *IEEE Trans. on Pattern Analysis and Machine Intelligence* 37, 2 (2015), 408–423.

[18] J. P. Forgas. 1995. Mood and judgment: the affect infusion model (AIM). *Psychological bulletin* 117, 1 (1995), 39.

[19] J. Fürnkranz and E. Hüllermeier. 2010. *Preference learning: An introduction*. Springer.

[20] S. Greco, J. Figueira, and M. Ehrgott. 2005. Multiple criteria decision analysis. *Springer's Int. series* (2005).

[21] P. C. Groot, T. Heskes, T. M. H. Dijkstra, and J. M. Kates. 2011. Predicting preference judgments of individual normal and hearing-impaired listeners with Gaussian Processes. *IEEE Transactions on Audio, Speech, and Language Processing* 19, 4 (2011), 811–821.

[22] S. Guo, S. Sanner, and E. V. Bonilla. 2010. Gaussian process preference elicitation. In *NIPS*. 262–270.

[23] Pascal Halffmann, Stefan Ruzika, Clemens Thielen, and David Willems. 2017. A general approximation method for bicriteria minimization problems. *Theoretical Computer Science* 695 (2017), 1–15.

[24] T. Haynes, S. Sen, N. Arora, and R. Nadella. 1997. An automated meeting scheduling system that utilizes user preferences. In *AA*. 308–315.

[25] N. Houlsby, F. Huszar, Z. Ghahramani, and J. M. Hernández-Lobato. 2012. Collaborative gaussian processes for preference learning. In *NIPS*. 2096–2104.

[26] B. S. Jensen, J. B. Nielsen, and J. Larsen. 2011. Efficient preference learning with pairwise continuous observations and Gaussian processes. In *MLSP*. 1–6.

[27] C. M. Jonker, V. Robu, and J. Treur. 2007. An agent architecture for multi-attribute negotiation using incomplete preference information. *JAAMAS* 15, 2 (2007), 221–252.

[28] D. C. Kingsley. 2006. Preference uncertainty, preference refinement and paired comparison choice experiments. *University of Colorado* (2006).

[29] Arnaud Liefooghe, Bilel Derbel, Sébastien Verel, Hernán Aguirre, and Kiyoshi Tanaka. 2017. A fitness landscape analysis of Pareto local search on bi-objective permutation flowshop scheduling problems. In *International Conference on Evolutionary Multi-Criterion Optimization*. 422–437.

[30] D. J. Lizotte. 2008. *Practical Bayesian optimization*. U. of Alberta.

[31] Daniel J Lizotte, Michael Bowling, and Susan A Murphy. 2012. Linear fitted-Q iteration with multiple reward functions. *Journal of Machine Learning Research* 13, Nov (2012), 3253–3295.

[32] J. Madsen, B. S. Jensen, and J. Larsen. 2012. Predictive modeling of expressed emotions in music using pairwise comparisons. In *CMMR*. 253–277.

[33] P. Mannion, J. Duggan, and E. Howley. 2017. A Theoretical and Empirical Analysis of Reward Transformations in Multi-Objective Stochastic Games. In *AAMAS*. 1625–1627.

[34] Hossam Mossalam, Yannis M Assael, Diederik M Roijers, and Shimon Whiteson. 2016. Multi-Objective Deep Reinforcement Learning. *arXiv preprint arXiv:1610.02707* (2016).

[35] Simone Parisi, Matteo Pirotta, and Marcello Restelli. 2016. Multi-objective Reinforcement Learning through Continuous Pareto Manifold Approximation. *Journal of Artificial Intelligence Research* 57 (2016), 187–227.

[36] Alina Pommeranz, Joost Broekens, Pascal Wiggers, Willem-Paul Brinkman, and Catholijn M Jonker. 2012. Designing interfaces for explicit preference elicitation: a user-centered investigation of preference representation and elicitation process. *User Modeling and User-Adapted Interaction* 22, 4-5 (2012), 357–397.

[37] A. M. Rashid, I. Albert, D. Cosley, S. K. Lam, S. M. McNee, J. A. Konstan, and J. Riedl. 2002. Getting to know you: learning new user preferences in recommender systems. In *IUI*. 127–134.

[38] C.-E. Rasmussen. 2006. Gaussian processes for machine learning. (2006).

[39] D..M.. Roijers, P.. Vamplew, S.. Whiteson, and R.. Dazeley. 2013. A Survey of Multi-Objective Sequential Decision-Making. *JAIR* 48 (2013), 67–113.

[40] D.M. Roijers and S. Whiteson. 2017. *Multi-objective Decision Making*. Morgan & Claypool.

[41] D. M. Roijers. 2016. *Multi-Objective Decision-Theoretic Planning*. Ph.D. Dissertation. University of Amsterdam.

[42] D. M. Roijers, L. M. Zintgraf, and A. Nowé. 2017. Interactive Thompson Sampling for Multi-objective Multi-armed Bandits. In *ADT*. 18–34.

[43] S. Siegel. 1956. Nonparametric statistics for the behavioral sciences. (1956).

[44] E. Sirakaya, J. Petrick, and H.-S. Choi. 2004. The role of mood on tourism product evaluations. *Annals of Tourism Research* 31, 3 (2004), 517–539.

[45] G. Tesauro. 1988. Connectionist Learning of Expert Preferences by Comparison Training.. In *NIPS*, Vol. 1. 99–106.

[46] P. Vamplew, R. Dazeley, E. Barker, and A. Kelarev. 2009. Constructing stochastic mixture policies for episodic multiobjective reinforcement learning tasks. In *AI'09*. 340–349.

[47] Kristof Van Moffaert and Ann Nowé. 2014. Multi-objective reinforcement learning using sets of pareto dominating policies. *Journal of Machine Learning Research* 15, 1 (2014), 3483–3512.

[48] Weijia Wang and Michele Sebag. 2012. Multi-objective monte-carlo tree search. In *Asian conference on machine learning*, Vol. 25. 507–522.

[49] Weijia Wang and Michele Sebag. 2013. Hypervolume indicator and dominance reward based multi-objective Monte-Carlo Tree Search. *Machine learning* 92, 2-3 (2013), 403–429.

[50] C Ch White and Kwang W Kim. 1980. Solution procedures for vector criterion Markov decision processes. *Large Scale Systems* 1, 4 (1980), 129–140.

[51] Marco A Wiering and Edwin D De Jong. 2007. Computing optimal stationary policies for multi-objective markov decision processes. In *Approximate Dynamic Programming and Reinforcement Learning, 2007. ADPRL 2007. IEEE International Symposium on*. IEEE, 158–165.

[52] Marco A Wiering, Maikel Withagen, and Mădălina M Drugan. 2014. Model-based multi-objective reinforcement learning. In *ADPRL*. 1–6.

[53] Kyle Hollins Wray, Shlomo Zilberstein, and Abdel-Illah Mouaddib. 2015. Multi-Objective MDPs with Conditional Lexicographic Reward Preferences.. In *AAAI*. 3418–3424.

[54] L.M. Zintgraf, T.V. Kanters, D.M. Roijers, F.A. Oliehoek, and R. Beau. 2015. Quality assessment of MORL algorithms: A utility-based approach. In *Benelearn*.

[55] M. Zoghi, S. Whiteson, R. Munos, and M. De Rijke. 2014. Relative Upper Confidence Bound For The K-Armed Dueling Bandit Problem. In *ICML*. 10–18.