Interactively Learning the User's Utility for Best-Arm Identification in Multi-Objective Multi-Armed Bandits

Mathieu Reymond
Vrije Universiteit Brussel
Brussels, Belgium
mreymond@ai.vub.ac.be

Eugenio Bargiacchi
Vrije Universiteit Brussel
Brussels, Belgium
ebargiac@ai.vub.ac.be

Diederik M. Roijers
Vrije Universiteit Brussel
Brussels, Belgium
diederik.roijers@vub.be

Ann Nowé
Vrije Universiteit Brussel
Brussels, Belgium
ann.nowe@ai.vub.ac.be

ABSTRACT

Many real-world problems have multiple, conflicting objectives. Without knowing the utility function of the decision maker, one must extensively learn all Pareto-efficient trade-offs to make sure that the true preferred policy is included in the learned set. Because such thorough exploration can be expensive (especially in high-dimensional multi-objective problems), a possible alternative is to allow some form of interaction with the decision maker as to gain some information about the utility function. In particular, in this work we assume that limited queries can be made to the policy maker to gather some information about the true utility function, *concurrently* to the search process being carried out. Improving our knowledge over the utility function narrows the search-space of the optimal policy. In turn, this results in more relevant trade-offs used to query the decision maker. Thus, correctly timing the queries is crucial to maximize information gain. We refer to this setting as fixed-budget best-arm identification for multi-objective multi-armed bandits, which adds to the traditional arm-pull actions a separate query-action that can be taken instead, where both actions have fixed but separate budgets. We propose Monte-Carlo Bayesian Utility Learning (MCBUL), a method based on Monte-Carlo planning that is able to optimize the timing of query-actions w.r.t. the arm-pull actions. We show that MCBUL significantly improves the chances of finding the optimal policy compared to baselines that interact with the decision maker at fixed intervals.

KEYWORDS

Multi-objective multi-armed bandits; Interactive learning; best-arm identification

ACM Reference Format:
Mathieu Reymond, Eugenio Bargiacchi, Diederik M. Roijers, and Ann Nowé. 2024. Interactively Learning the User's Utility for Best-Arm Identification in Multi-Objective Multi-Armed Bandits. In *Proc. of the 23rd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2024), Auckland, New Zealand, May 6 – 10, 2024*, IFAAMAS, 10 pages.

Proc. of the 23rd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2024), N. Alechina, V. Dignum, M. Dastani, J.S. Sichman (eds.), May 6 – 10, 2024, Auckland, New Zealand. © 2024 International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org).

1 INTRODUCTION

Decision makers faced with real-world problems often need to consider multiple objectives. Improving one objective often comes at the cost of another, i.e., objectives are conflicting, and one must find a compromise between them. For instance, using renewable energy sources involves economical and environmental factors [24]. In the medical field, radiotherapy should generally maximize the destruction of cancer cells while minimizing the damage to the healthy surrounding tissue [28]. The optimal trade-offs may vary on a case-by-case basis, as they depend on the preferences of the decision maker.

When the preferences – or utility – of the decision maker are known a priori, one may directly optimize on said utility and use single-objective optimization methods [33]. However, human decision makers find it challenging to express their preferences in absolute terms, as using numbers to express preferences can be unnatural and prone to errors [45]. In this case, they need to be informed with the values of actual trade-offs to be able to take well-informed decisions. Thus, explicitly multi-objective optimization techniques are required.

In contrast to single-objective optimization, that directly learns a single solution, multi-objective optimization methods learn the set of all possible optimal trade-offs, called the *Pareto front*. Once learned, the Pareto front stays fixed, since it does not depend on the preferences of the decision maker. The decision maker can then use the Pareto front to review all available policies, and use this knowledge to select their preferred one [23]. As a principal downside, searching for all these trade-offs requires extensive computational cost. This makes it unsuitable for settings with fixed or limited resources [3, 18].

Instead of letting the decision maker review the different trade-offs *a posteriori*, we aim to make use of the reviewing process *during* the learning phase, allowing is to steer the search based on the decision maker's feedback. This approach presents several advantages. First, we improve our knowledge over the utility function, which narrows the search-space of the optimal policy with respect to the user preferences. Thus, it becomes easier to refine and improve the solution over time. Moreover, this interactivity allows for our estimate of the utility function to be adapted to changing preferences or circumstances. As the decision maker's preferences evolve, the policy can be updated to reflect these changes. Finally, in case the decision maker is a person (or group of persons), their involvement

in the learning process means they gain a better understanding of how it works and how their preferences are being taken into account. This can increase their trust in the system and make them more willing to use it.

Given a fixed budget in computational resources, and a fixed number of interactions with the decision maker, we propose to optimize the timing of user-interactions to maximize our chances of learning the optimal policy. We do this in the context of multi-objective multi-armed bandits (MOMABs) using a a Bayesian approach, by learning a belief distribution over the preferences of the decision maker. In this work, we propose *Monte-Carlo Bayesian Utility Learning* (MCBUL), based on Monte-Carlo methods. At each timestep, we perform rollouts based on our belief distributions to recommend either an interaction or an arm-pull, and show that it significantly improves the chances of finding the optimal policy compared to interacting with the decision maker at fixed intervals.

2 BACKGROUND

2.1 Multi-objective multi-armed bandits

Formally, a multi-objective multi-armed bandit (MOMAB) [16] is a tuple $\mathbf{B} = \langle \boldsymbol{\mathcal{R}}, u \rangle$, where $\boldsymbol{\mathcal{R}} = \{\boldsymbol{\mathcal{P}}_{\theta_1}, \ldots, \boldsymbol{\mathcal{P}}_{\theta_A}\}$ is a set of parametric multivariate stochastic reward functions $\boldsymbol{\mathcal{P}}_{\theta_1}, \ldots, \boldsymbol{\mathcal{P}}_{\theta_A}$, and $u : \mathbb{R}^n \rightarrow \mathbb{R}$ is the utility function defining the preferences of the decision maker. Given a n-dimensional vector as input, where n represents the number of objectives, it returns a scalar preference score. For MOMABs, each $\boldsymbol{\mathcal{P}}_{\theta_a}$ is a multivariate distribution with the number of dimensions equal to the number of objectives.

The optimal policy, or optimal arm, is the arm resulting in the maximal utility with respect to its multivariate mean:

$$\pi^* = \arg\max_{a \in \mathcal{A}} u(\boldsymbol{\mu}_a), \tag{1}$$

where $\boldsymbol{\mu}_a$ is the mean of the multivariate distribution $\boldsymbol{\mathcal{P}}_{\theta_a}$. In this work, we focus on the best-arm identification setting [3] where, given a fixed budget, the goal is to recommend the optimal arm. This is a pure exploration setting, as there is no cost incurred when spending budget on sub-optimal arms, as long as the optimal arm is recommended after all the budget has been spent. Our budget is two-fold. First, we have a computational budget, represented as a fixed number of arm-pulls. Selecting an arm $a \in \mathcal{A}$ results in a sample $\boldsymbol{r} \sim \boldsymbol{\mathcal{P}}_{\theta_a}$. Second, there is an interaction budget, represented as the number of times we ask the decision maker for feedback (i.e., the number of times we call u).

Most body of work in the MORL literature assumes that the utility function is a linear scalarization over the objectives:

$$u(\boldsymbol{V}) = \boldsymbol{w}^\top \boldsymbol{V}, \{\boldsymbol{w} \in \mathbb{S}^{n-1}\}, \tag{2}$$

with $\mathbb{S}^{n-1}$ the $n-1$ dimensional simplex, resulting in a weighted sum over the objectives.

For this work, we assume the reward distributions are normally distributed, as this is often the case in the bandit literature [8, 34, 36]. We assume no correlation between the random variables of $\boldsymbol{\mathcal{P}}_\theta$, as this does not affect the multivariate mean, i.e., $\boldsymbol{\mathcal{P}}_{\theta_a} = \{\mathcal{N}(\mu_a^1, \sigma_a^1), \ldots, \mathcal{N}(\mu_a^n, \sigma_a^n)\}$. Thus, $\theta_a = \langle \boldsymbol{\mu}_a, \boldsymbol{\sigma}_a \rangle$, with $\boldsymbol{\mu}_a = [\mu_a^1, \ldots, \mu_a^n]$, $\boldsymbol{\sigma}_a = [\sigma_a^1, \ldots, \sigma_a^n]$ as the multivariate mean, standard deviation for arm a, respectively.

2.2 Top-two Thompson sampling

A well-known Bayesian algorithm for best arm identification in single-objective multi-armed bandits is called Top-two Thompson Sampling (TTTS) [37]. The primary goal of TTTS is to distinguish the best arm from the second-best arm. The stronger this distinction, the highest confidence it has that the estimated best arm is indeed the optimal arm. Since the other arms are worse than the second-best arm, their ordering does not matter, so we should avoid spending our budget on them.

To distinguish arms, TTTS maintains a belief distribution over each reward distribution $\boldsymbol{\mathcal{P}}_{\theta_a}, a \in \mathcal{A}$. That is, TTTS estimates the parameters θ_a based on the history of observed samples $\mathcal{H}_{a,t} = \{r_0, \ldots, r_{t-1}\}$ from $\boldsymbol{\mathcal{P}}_{\theta_a}$.

We would like to compute the probability distribution $\mathcal{P}(\theta_a \mid \mathcal{H}_{a,t})$ over the possible distribution parameters, given the history $\mathcal{H}_{a,t}$. This is called the *posterior distribution*. Initially, when our history $\mathcal{H}_{a,t}$ is empty, we are uncertain about θ_a and use default parameters ϕ_a. The distribution $\mathcal{P}(\theta_a \mid \phi_a)$ is called the *prior distribution*. We refer to Appendix A for further details on the posterior distribution of a Normal distribution.

At time t, TTTS samples from a Bernoulli distribution (typically with $p = 0.5$ of success) $b \sim \mathcal{B}(p)$ to decide if it should pull the best arm. The ordering of arms is decided by sampling from each of the belief distributions, and sorting the arms according to their associated sample. When b is a success, we pull the best arm. Otherwise, we aim to pull the second-best arm. The second-best arm is decided by saving the sampled best arm, and then resampling from each belief distribution until the resampled best arm is different from the saved best arm. That arm is then pulled.

At $t = 0$, when our beliefs have no information about the reward distributions, each arm is equally likely to be selected as best. However, as we pull arms, our belief distributions become increasingly informative, and the likelihood of the highest ranked arm corresponding to the optimal arm increases as well.

We repeat this process until the budget has been exhausted. At this point, the arm associated with the belief distribution with the highest mean is identified as the best arm.

3 BELIEF DISTRIBUTION OF THE UTILITY FUNCTION

An important aspect of MOMABs is the inclusion of the utility function u. This function is initially unknown. To find the best arm, we need to have an understanding of u. Similarly as for arms, we keep a belief distribution over u, which we improve over time by interacting with the decision maker. Using this belief distribution, we can sample utility function estimates $\hat{u}$ and rank the multivariate samples coming from the arm belief distributions. This allows us to pull arms following the same strategy as the TTTS algorithm.

While the utility function could be a formal process returning an absolute score, in many real-world problems, the decision maker is human. Humans find it challenging to express their preferences in absolute terms (e.g., "I like this movie 0.4 much"), as using numbers to express preferences can be unnatural and prone to errors [45]. Additionally, values may change [40] depending on the user's mood, which can be influenced by seemingly trivial factors like the weather [17, 42]. On the other hand, expressing preferences

in relative terms (e.g., "I prefer movie A over B") is easier for humans and tends to be more consistent over time [45, 56]. Therefore, we focus on learning the utility function using relative feedback. We can translate this process as a binary classification task where, given two propositions, the goal is to predict if the first proposition is preferred over the second one. Formally, given two propositions $\boldsymbol{r}^0, \boldsymbol{r}^1$, we define $\succ : \mathbb{R}^n \times \mathbb{R}^n \rightarrow \{1, 0\}$ as the binary preference operator, where $\boldsymbol{r}^0 \succ \boldsymbol{r}^1$ outputs 1 when the decision maker prefers $\boldsymbol{r}^0$ over $\boldsymbol{r}^1$, and 0 otherwise. Thus, we define each interaction with the decision maker as a pair, where the first element is the two propositions to compare, and the second element is the decision maker's answer:

$$\langle \langle \boldsymbol{r}^0, \boldsymbol{r}^1 \rangle, \boldsymbol{r}^0 \succ \boldsymbol{r}^1 \rangle. \tag{3}$$

We keep each interaction in an interaction history $\mathcal{H}^q$, which is used to update the belief distribution over u.

We propose to use particle filtering [15, 21] as a belief distribution over the utility function. Particle filtering uses a set of particles to represent a distribution, where each particle is weighted according their likelihood given the datapoints. Sampling from this belief distribution then amounts to sampling a particle according to their likelihood. Formally, given a set of particles $x_0, \ldots, x_P$, where P is the total number of particles, each x_i is associated with a particle weight $\boldsymbol{\omega}_i$, whose value is:

$$\boldsymbol{\omega}_i = \mathcal{P}(x_i \mid \mathcal{H}^q). \tag{4}$$

The mean of the particle distribution is then the weighted average of the particles:

$$\boldsymbol{\mu} = \sum_{i=0}^{P} \boldsymbol{\omega}_i x_i. \tag{5}$$

Since we can freely choose how we define our particles, one advantage of particle filtering is that it can approximate distributions of arbitrary shapes. Thus, we can represent u by its weights (see Equation 2). Since the weights from u belong to $\mathbb{S}^{n-1}$, we sample our particles from it. However, as a downside, the number of particles required to accurately represent the $n-1$ simplex increases exponentially with the number of objectives.

At the beginning of training, we assume that we do not possess any knowledge over u. Thus, initially all particles are weighted equally. However, with each new datapoint, the likelihood of each particle is updated. Intuitively, we give a high likelihood to all particles that match the decision maker's answers to past queries, and a low likelihood to the other particles. Given our history $\mathcal{H}^q$ of relative queries, the weight $\boldsymbol{\omega}$ of particle x is defined as:

$$\boldsymbol{\omega} = \prod_{h \in \mathcal{H}^q} |(\boldsymbol{r}_h^0 \succ \boldsymbol{r}_h^1) - \eta| (\boldsymbol{\omega}^\top \boldsymbol{r}_h^0 \geq \boldsymbol{\omega}^\top \boldsymbol{r}_h^1), \tag{6}$$

where η accounts for potential mistakes, or change of preference from the decision maker. Thus, when $\eta = 0$, only the particles of weights for which *all* answers of the decision maker correspond to the solution with the highest utility have a non-zero probability of being sampled. Moreover, these particles are equally likely.

3.1 Selecting queries for the decision maker

Although pairwise comparisons are more reliable in terms of human answers, they provide less information than absolute scores, and are thus less effective to estimate u. It is thus important that the pairwise comparisons are informative and realistic. For the comparisons to be realistic, we select the solution pairs based on our current belief distributions over arms. Moreover, since our goal is to distinguish the top-two arms, we aim to provide queries that further discriminate the top-two arms in terms of utility.

Inspired by Interactive Thompson Sampling (ITS) [36], an algorithm for regret minimization for MOMABs, we base our query-selection mechanism on Thompson sampling [46]. First, we sample a utility function estimate $\hat{u}$ from our belief distribution. Next, for each belief distribution over arms, we sample a vectorial reward $\hat{\boldsymbol{r}}_a \sim \boldsymbol{\mathcal{P}}_{\hat{\theta}_a}$ with $\hat{\theta}_a \sim \mathcal{P}(\theta_a \mid \mathcal{H}_{a,t})$. For each arm a, we compute its utility $\hat{u}(\hat{\boldsymbol{r}}_a)$ using the corresponding sampled rewards and the sampled weights. We can then rank the arms according to their computed utility. We give the samples corresponding to the top-two ranked arms as a query to the decision maker. This allows us to have varied queries (as they are based on samples), that focus on a narrow region of the utility-space (the region that distinguishes the first and the second arm). We compare the performance of our proposed query-selection mechanism with the one used in ITS in Appendix B, and show that this results in more pertinent queries.

4 MCBUL FOR QUERY OPTIMIZATION

Learning the utility function is tied with learning the optimal policy, as our query-selection mechanism depends on our belief distribution over arms. Thus, we argue that we can optimize the timing of querying the decision maker.

We propose, to the best of our knowledge, the first algorithm for best arm identification in the multi-objective setting. Our algorithm, Monte-Carlo Bayesian Utility Learning (MCBUL), takes inspiration from Partially Observable Monte-Carlo Planning (POMCP) [41], which effectuates Monte-Carlo sampling to break the curse of dimensionality of large search-spaces. Moreover, POMCP can cope with partial observability of MDPs, by keeping and updating beliefs over states. This makes POMCP compatible with our setting, as we keep belief distributions over the arms and utility function, which are updated with each pull, and query, respectively. Finally, since POMCP is a planning algorithm based on tree-search, it is made for episodic settings. This is the case for our best-arm identification setting, where the pulling budget and query budget define the number of timesteps that can be executed.

POMCP is an online algorithm for action-recommendation. In its essence, it is an extension of Monte-Carlo tree search (MCTS) [13] for partially observable MDPs (POMDPs). Although, contrary to MDPs and POMDPs, MOMABs do not have states, we will see in Section 4.1 that we can use the same concept behind these algorithms differently in MOMABs, by considering the whole process towards finding the best arm as a sequential decision process.

We first explain the main idea behind MCTS, as it is essentially the same as for POMCP. At each timestep t, MCTS performs a number of simulations, or *rollouts*, using a model of the environment. All rollouts start from the current state s_t and are assigned a score. Based on these simulated rollouts, it estimates the average score μ_a for each action a of s_t. It then recommends executing the action with the highest estimated μ_a in the environment. This leads to a new state s_{t+1}, at which point the process repeats: recommending an

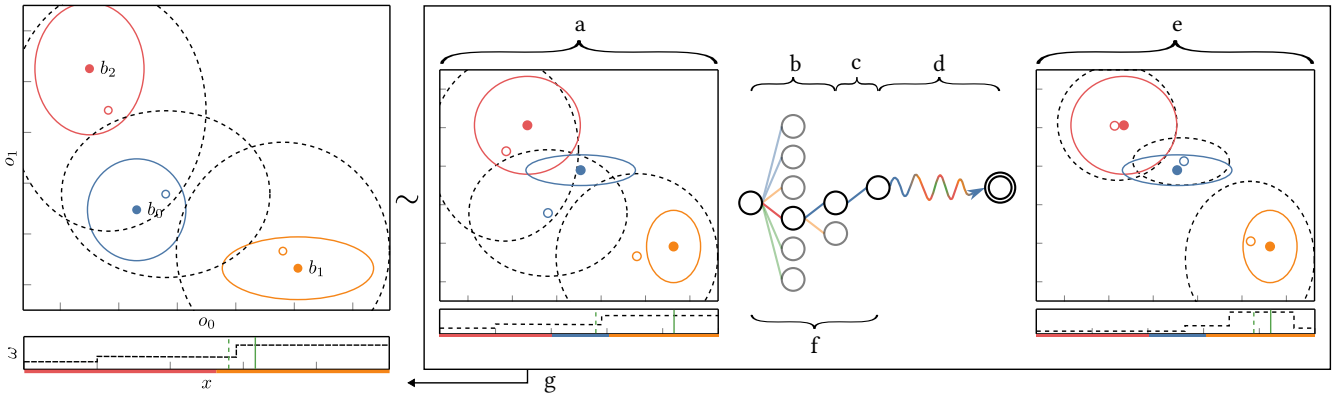

Figure 1: Illustration of MCBUL on a 3-arm, 2-objective MOMAB. Top-left depicts the original MOMAB, with its three arms $\mathcal{P}_{\theta_a}, a \in \mathcal{A}$ and their true means in solid blue, orange and red. The black dotted ellipses represent our current arm estimates $\mathcal{P}(\theta_a \mid \mathcal{H}_{a,t})$, with the hollow dots our mean estimates $\hat{\mu}_a$. Bottom-left shows the utility weight space $\mathbb{S}$. The solid colors represent the sections of $\mathbb{S}$ where a given arm is truly optimal (blue is always suboptimal), and the solid green line represents the decision maker's true preference. Thus, the optimal recommendation for this MOMAB is b_1. The dotted line shows our belief distribution over u as weighted particles with in dashed green the currently estimated mean. — a) Sampled MOMAB from the current belief distribution. Note that the solid lines represent the ground truth in this sampled new MOMAB, while our dashed estimates are unchanged — b-c-d) A single MCTS-like rollout is performed, using UCB within the tree and MOTTTS outside. — e) Our updated branch estimates are used to score the rollout ($\mathcal{R}_1 = 1$, as the recommended arm is also the best arm of the sampled MOMAB). — f) We update the values of the traversed nodes with the score. — g) After multiple executions of a-f, we pull the arm with the highest value at MCBUL's root-node on the real MOMAB, update our estimates, and repeat the process.

action to execute in s_{t+1}, based on simulated rollouts starting from s_{t+1}. The main particularity of POMCP, compared to MCTS, is that POMCP does not actually *know* the state s_t (or s_{t+1}) it is currently in. It has only a limited view on the state, and a belief distribution $\mathcal{P}$ on what this state could be. For each rollout, POMCP samples an estimated state $\hat{s}_t$, and starts the simulation from there. The recommended action is then based on the aggregated μ_a estimates over all $\hat{s}_t$ samples.

4.1 Transition model for simulated rollouts

POMCP requires a model of the environment to make simulations. We propose to create a such a model, entirely based on our belief distributions over arms and utility function.

Although the MOMAB's true parameters $\langle \boldsymbol{\mu}_a, \boldsymbol{\sigma}_a \rangle, a \in \mathcal{A}$ and the true utility function u are unknown, our belief distribution allows us to sample a virtual MOMAB by sampling an estimated mean, standard deviation $\hat{\boldsymbol{\mu}}_a, \hat{\boldsymbol{\sigma}}_a \sim \mathcal{P}(\cdot \mid \mathcal{H}^A_{a,t}), a \in \mathcal{A}$ for each arm, and sampling a utility function $\hat{u} \sim \mathcal{P}(\cdot \mid \mathcal{H}^q_t)$.

Moreover, we can interact with this virtual MOMAB as we would with the real MOMAB, by pulling its arms and observing rewards $\boldsymbol{r} \sim \mathcal{N}(\hat{\boldsymbol{\mu}}_a, \hat{\boldsymbol{\sigma}}_a)$, and asking queries that will be answered by $\hat{u}$. Thus, the total number of actions is $A + 1$: A actions to pull arms $a_1, \ldots, a_A$, and 1 action to query the decision maker, using the querying strategy explained in Section 3.1. The resulting observations can be added to $\mathcal{H}^A_t$ and $\mathcal{H}^q_t$, which respectively update the belief distribution over arms and utility function. The total number of interactions T with this virtual MOMAB (and thus the length of an episode and maximal depth of the search tree built by POMCP) is defined by the sum of the leftover query budget and pulling budget.

4.2 Simulating rollouts

Recommending an action is based on simulated rollouts. To identify with the highest confidence possible which action to recommend, the rollouts use a targeted exploration of the available model defined in Section 4.1. This is done by building a tree of the possible action-sequences, and following the most promising branches of this tree.

Initially, our tree consists of a root-node (Figure 1a), corresponding to $\mathcal{H}^A_t$ and $\mathcal{H}^q_t$.This *belief-node* has one child-node per possible action, called *action-node*. Executing the corresponding action on the available model results in an updated belief $\mathcal{H}^A_{t+1}, \mathcal{H}^q_{t+1}$. Thus, the tree alternates between belief-nodes and action-nodes.

Each node keeps track of the number of times it has been visited (the *visitation count*), as well as the average return of all rollouts passing through that node (the value of that node).

A rollout is split in 4 phases. In the first phase, we walk down our current tree (Figure 1b). At each belief-node, we select an action-node based on the Upper Confidence Bound [4], as is done in MCTS and POMCP. We execute the action in the virtual MOMAB, leading us to an updated belief distribution. If this belief does not correspond to a child-node of the selected action-node, we go to the next phase. Otherwise, we walk to that child-node, and repeat the process.

Thus, the second phase starts from an action-node that leads to a belief that has not been encountered before. We create a new belief-node, and add it as a child of the current action-node (Figure 1c). Since this occurs at every rollout, each rollout creates one belief-node. The tree-size is thus proportional to the number of rollouts.

Next, the third phase starts from the newly created node, and executes a fixed policy in the virtual MOMAB until all leftover budget has been used (Figure 1d). We then assess the performance

of the rollout using a scoring function (i.e., the rollout's return, Figure 1e), which we explain in more detail in Section 4.4.

Finally, the return is backpropagated, updating the visitation count and value of each visited node during this rollout (Figure 1f).

For each rollout, we sample a different virtual MOMAB. This ensures that the recommended action at the root-node is the best one across our whole belief distribution. However, this comes with some challenges. Mainly, the quality of the estimated values at the root-node are significantly impacted by the branching factor of the tree, as a larger branching factor requires exponentially more simulations. We note 2 different branching factors, one for the belief-nodes, and another for the action-nodes. Each belief-node has one child for each possible action. Thus, the branching factor increases with the number of possible arms. Hence, we expect a decrease of performance for MOMABs with a large number of arms. This is a known problem for tree-search approaches such as MCTS and POMCP, and multiple approaches have been proposed to cope with large action-spaces, typically by keeping a small subset of candidate-actions [9, 11, 12, 20].

Analogously, action-nodes have a branching factor that depends on the number of subsequent encountered beliefs. However, each $\mathcal{H}^A_{t+1}, \mathcal{H}^q_{t+1}$ is unique, as pulling an arm in our virtual MOMAB results in a continuous reward-vector. As such, the branching factor for belief-nodes is infinite, bounding the tree-depth to 3: the root-node, the actions executable from it, and finally an endlessly growing number of next-belief-nodes. To ensure a meaningful tree-search, we need to cope with this infinite branching factor.

4.3 Aggregating belief-nodes together

A straightforward way of dealing with continuous state-spaces is to discretize the states. For our setting, this amounts to splitting the range of possible reward values in a fixed number of bins. We can decide on the branching factor by choosing the number of bins. A higher number of bins increases the accuracy of the discretization, at the cost of a higher branching factor.

We use an adaptive binning mechanism, based on the belief distribution over the arm we are currently pulling. As the belief distribution becomes more accurate, due to the additional samples, then the bins become increasingly precise. Since, in our setting, the reward-distributions follow a Normal distribution, its conjugate prior is a Normal-gamma distribution (see Equation 12 in Appendix A). We use the estimated mean and standard deviation from the conjugate prior to produce the boundaries between each bin.

Concretely, we compute a range of possible values $[\mu - c\sigma, \mu + c\sigma]$, where c is a constant defining up to how many standard deviations we can be away from the estimated mean, that we partition equally between the chosen number of bins. We set, using prior hyperparameters $\alpha = \frac{1}{2}, \beta = 0.1$:

$$\mu = \hat{\mu}, \text{with } \hat{\mu} \text{ the sample mean,}$$

$$k = \alpha + \frac{N}{2}, \text{with } N \text{ the number of samples,}$$

$$\tau = (\beta + \frac{N}{2}\hat{\sigma}^2)^{-1}, \text{with } \hat{\sigma} \text{ the sample standard deviation,}$$

$$\text{assuming a zero-prior,}$$

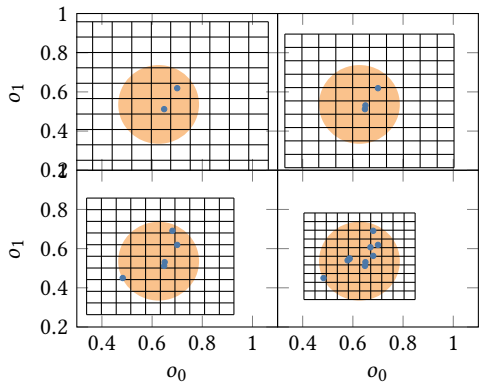

Figure 2: Example of produced bins, depending on the number of samples. As the number of samples increases, our belief distribution becomes more precise, which means the bins better match the reward distribution.

$$\sigma = \sqrt{\frac{1}{k\tau}}, \text{since } k\tau \text{ is the mean of a Gamma distribution.}$$

Figure 2 shows an illustration of our binning method depending on the number of samples, using 10 bins per objective-dimension. It displays, in orange, the bivariate normal distribution (up to 2 standard deviations) from which the samples (in blue) are drawn. The grid (in black) represent the different bins. As the number of samples increase, so does the confidence of the belief distribution in its estimation of the mean and standard deviation. The bins increasingly concentrate around the true distribution.

Our adapted binning allows for an automatic segmentation of the sampled rewards. However, the number of bins required to split the state-space increases exponentially with the number of objectives. Still, we argue that, when the number of objectives are limited, binning is a reasonable approach, as it is conceptually simple and adaptive to each individual belief distribution.

4.4 Evaluating rollouts

Since we can cope with the infinite branching of belief-nodes, we can execute simulated rollouts that each will increase the size and depth of the search tree. At the end of a rollout, we need to assess its quality. Since our aim is to provide the best arm within a fixed budget, we would like our algorithm to recommend the action that maximally increases our chances to find this best arm. As such, our scoring function should reflect this.

Since we can best assess the quality of a rollout after it has been executed, and the score is backpropagated through the tree, we only provide a score at the end of each simulation. This score is computed using the posterior belief over arms and over the utility function, since they define our estimate over the best arm.

To assess the importance of the scoring function compared to the number of rollouts, we have analyzed 2 alternative scoring mechanisms. As a first scoring function, we compute our estimated utility of each arm, by using the estimated mean of the posterior belief over the utility function and the estimated mean of each arm. If the arm with the highest estimated utility, i.e., the proposed best

arm, matches the best arm from the simulated bandit generated at the root node of our tree, the simulation is considered a success, and returns a score of 1. Otherwise, the simulation is considered a failure, and the returned score is 0:

$$\mathcal{R}_1 = \begin{cases} 1 & \text{if } \hat{a}^* = \tilde{a}^* \quad \hat{a}^* = \arg\max_{a \in A} \hat{\boldsymbol{\mu}}_u^\top \hat{\boldsymbol{\mu}}_a \\ 0 & \text{otherwise,} \end{cases} \tag{7}$$

where $\tilde{a}^*$ is the proposed best arm. This is the same way we would recommend the best arm to the real decision maker (and not the simulated utility function in our algorithm). Since each node keeps track of the average score from all simulations passing through that node, the scores of the action-nodes at the root represent the probability of recommending the best arm to the decision maker.

Not only is this scoring function aligned with the goal of our problem setting, it has also the advantage of being computationally inexpensive, which allows us to spend the computational budget on additional simulations, thus improving the accuracy of the estimated probabilities at the root's action-nodes. However, due to its binary nature, this scoring function is sparse. Moreover, since the success depends on our initial belief over the MOMAB, it can be noisy. Thus, many simulations are required for accurate best arm identification probabilities at the root node.

The first scoring function requires many simulations for accurate best arm identification probabilities at the root node. When the branching factor becomes too large, this can become an issue. Thus, we envisage a second, less sparse and more informative scoring function. We take inspiration from TTTS which, to identify the best arm, aims to discriminate the best and second-best arm as much as possible. Similarly, we estimate the confidence of our posterior belief at the end of the simulation in recommending the best arm.

To compute this confidence score, we sample weights and arm-rewards multiple times from our posterior belief over the MOMAB. For each arm, we count the number of times it is recommended as best. This gives us a recommendation percentage for each arm. The returned score is the highest recommendation percentage.

More formally, assume we sample N times, for each arm a, its estimated mean $\hat{\boldsymbol{\mu}}_a \sim \mathcal{P}(\cdot \mid \mathcal{H}_a^A)$, and a utility function $\hat{u} \sim \mathcal{P}(\cdot \mid \mathcal{H}^q)$. Let us call the n-th samples $\hat{\boldsymbol{\mu}}_a^n$ and $\hat{u}^n$ respectively. Then we have:

$$\mathcal{R}_2 = \max_a \sum_{n=0}^{N} \delta(a, n) \tag{8}$$

$$\delta(a, n) = \begin{cases} 1, & \text{if } \hat{u}^n(\hat{\boldsymbol{\mu}}_a^n) > \hat{u}^n(\hat{\boldsymbol{\mu}}_b^n) \quad \forall b \in A^{-a}, \\ 0, & \text{otherwise.} \end{cases} \tag{9}$$

Since the recommendation percentage depends on the number of times we sample from our posterior belief, it is more computationally expensive. However, it might be a beneficial trade-off depending on the properties of the bandit (e.g., number of arms, number of objectives).

5 EXPERIMENTS

With MCBUL, our aim is to optimize the timing of the query process. To assess the impact of this timing, we propose a baseline algorithm that uses fixed timings to query the decision maker. We use different variants of this baseline, that have different timings.

5.1 Multi-objective TTTS

For single-objective optimization, TTTS is an efficient algorithm for best-arm identification. We propose to extend it to the multi-objective setting, by learning multivariate belief distributions over the arms. We call this algorithm multi-objective top-two Thompson sampling (MOTTTS).

If learning the utility function and learning the optimal arm are separate, disjoint processes, we can do one process followed by the other one. Thus, our first variant asks all queries first, based on the initial belief distributions over arms, then learns the optimal policy using the learned utility function. This variant is MOTTTS-start.

Analogously, our second variant first learns the belief distributions over the arms, then asks all the queries based on these learned beliefs. We call this variant MOTTTS-end.

Finally, to assess the impact of combining both processes together, we propose a third variant, where the timing of each query is spread out equally over the arm-pulling budget. We call this variant MOTTTS-interleaved.

5.2 Experimental setup

All our experiments are performed on randomly generated MOMABs. So that learning the optimal policy is challenging, i.e., it is not possible to reliably find the optimal policy by randomly pulling arms, we enforce some properties on the generated bandits. For example, so there can be different optimal arms depending on the utility function, we ensure a percentage of arms are non-dominated (40% in our experiments). More details are provided in Appendix C.

To provide further insights on the generated bandits, we include additional baselines that serve as upper and lower bounds on the probability of identifying the optimal arm. As an upper bound, we assume knowledge of the utility function, and apply TTTS on the utility of the sampled multi-objective rewards, which is equivalent to MOTTTS without having to learn the utility function. We call this upper bound MOTTTS-cheat.

As a lower bound, we use a round-robin strategy, i.e., the arm-pulling budget is split equally for each arm. After pulling each arm the same number of times, all queries are asked to learn the utility function. This strategy avoids pulling arms in a smart way and does not take advantage of the learned belief distributions over arms.

We generate 30, 2-objective MOMABs, and perform 1000 experiments on each MOMAB. We use 100 particles equally-spaced over the $n-1$ simplex, initialized with uniform probability-weights. We set the UCB exploration factor β to 0.1. We set the noise η accounting for mistaken answers by the decision maker to 0.05.

5.3 Baseline results

Results for the baselines are displayed in Table 1. For each baseline, we report the best arm identification percentage (BAI%) averaged over all MOMABs. Since the generated MOMABs are different, it is more difficult to find the optimal arm for some than for others. This means the BAI% inherently varies across MOMABs. As such, we did not find any insight in including the standard deviation. Instead, we report the percentage of MOMABs for which each query-timing outperforms the others in terms of BAI% (i.e., WIN%[1] in the Table).

[1]These percentages do not exactly sum to 100, due to the rare occurrences where performance is equal.

	MORobin-end	MOTTTS-start	MOTTTS-end	MOTTTS-interleaved	MOTTTS-cheat
BAI%	66.96%	72.70%	67.80%	75.07%	83.29%
WIN%	N/A	13.53%	0.11%	86.38%	N/A

Table 1: Comparisons between our MOTTTS-based baselines. The round-robin strategy and the "cheat" version of MOTTTS which uses the true utility function are shown on the side as lower and upper bounds respectively. We show the best-arm-identification percentage (BAI%) for each strategy, based on 100000 experiments on 10000 MOMABs. We do not show the standard deviation, as the different MOMABs exhibit different properties. In its place, the WIN% value represents the proportion of MOMABs in which each variant has the highest BAI%. We see that MOTTTS-interleaved performs best in both metrics.

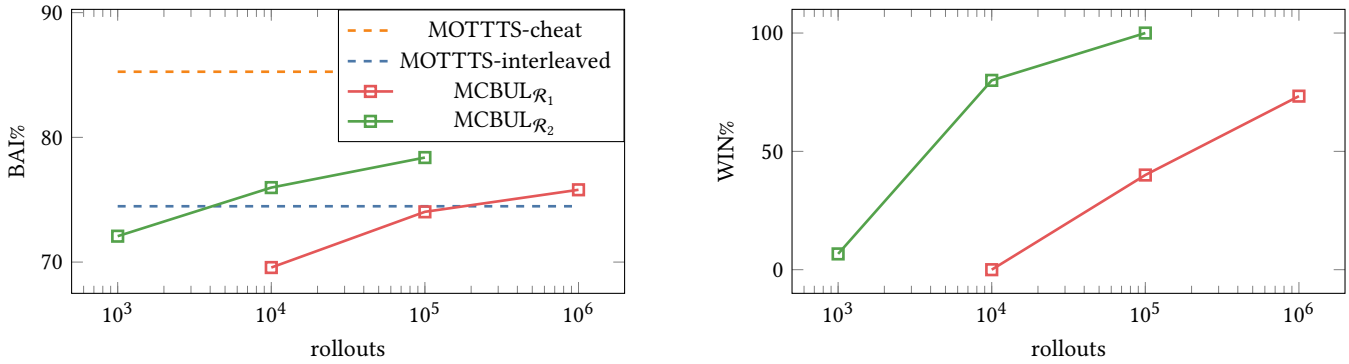

Figure 3: Best-arm-identification percentage (BAI%) of MCBUL depending on the number of rollouts. MCBUL$_{\mathcal{R}_1}$ uses the binary scoring function, while MCBUL$_{\mathcal{R}_2}$ estimates the posterior confidence with 1000 samples. We note that MCBUL$_{\mathcal{R}_1}$ needs one order of magnitude more rollouts than MCBUL$_{\mathcal{R}_2}$ to reach similar performance in BAI%. However, since the binary scoring function is less expensive to compute, the wall-time is similar. We observe that, given enough rollouts, the timing of the queries proposed by MCBUL results in a higher BAI% than the MOTTTS-interleaved baseline. Moreover, for similar wall-time, using the second scoring function results in a higher BAI% than using the binary scoring function.

First, we notice that, as expected the round-robin strategy performs worst, as it does not use targeted exploration. Second, we observe that knowing u does indeed result in better performances, as the MOTTTS-cheat upper-bound baseline performs best. Next, we observe that asking all queries before pulling arms results in a better performance than asking all queries after having used up the pulling budget. Since, at the start of training, we use uninformative belief distributions, our query-selection strategy (see Section 3.1) samples different utility functions, and random vectorial rewards, resulting in diverse queries. While not realistic, these queries allow to narrow down the range of possible utility functions, resulting in a more reliable ranking of arms during the arm-selection steps.

In contrast, having no information on the utility function means each non-dominated arm is potentially optimal. Thus, for all these arms, an accurate belief distribution is required, resulting in the pulling budget being split across more arms than necessary. Even if, afterwards, the queries are realistic, the uncertainty on the belief distributions over arms might be too high to accurately select the best arm. Indeed, we observe that the performance of MOTTTS-end is similar to the lower-bound, MORobin-end, as the exploration has been spread over too many arms.

Across all the query-selection timings, MOTTTS-interleaved performs best. This indicates that improving knowledge over the utility function and improving knowledge over the policy's search space are intertwined processes. As an additional analysis, we observe using the WIN% that on 86.38% of the MOMABs, MOTTTS-interleaved has a higher BAI% than both the MOTTTS-start baseline and the MOTTTS-end one. This supports our conclusion about the intertwined processes.

Finally, although MOTTTS-interleaved has the highest BAI% across the query-selection variants, there is still a large gap with the upper-bound performance, which reaches 83.29%. We believe that, by further optimizing the timing at which the queries are asked, we can close this gap.

5.4 MCBUL results

Results are shown in Figure 3. We show MCBUL using the 2 scoring functions. MCBUL$_{\mathcal{R}_1}$ uses the binary scoring function, while MCBUL$_{\mathcal{R}_2}$ estimates the posterior confidence with 1000 samples. We analyze the effect of the number of rollouts on MCBUL's performance, by repeating the experiments with an increasing number of rollouts. We note that, since MCBUL$_{\mathcal{R}_1}$ uses a less computationally expensive scoring function than MCBUL$_{\mathcal{R}_2}$, it can perform more rollouts for similar wall-times (in this case, one order of magnitude more). This is why we perform experiments with $10^4, 10^5, 10^6$ rollouts for MCBUL$_{\mathcal{R}_1}$, and $10^3, 10^4, 10^5$ rollouts for MCBUL$_{\mathcal{R}_2}$.

In general, the BAI% increases with the number of rollouts. Provided enough rollouts, MCBUL beats the MOTTTS-interleaved baseline, for both scoring functions. However, for similar wall-times,

$\text{MCBUL}_{\mathcal{R}_2}$ systematically outperforms $\text{MCBUL}_{\mathcal{R}_1}$. Thus, it seems that using a more informative scoring function has a higher impact on performance than increasing the number of rollouts. $\text{MCBUL}_{\mathcal{R}_2}$ with 10^5 rollouts reaches a BAI% of 78.38%, compared to 74.48% for MOTTTS-interleaved and 85.26% for the upper bound, MOTTTS-cheat. Since we can maximally improve the BAI% by 10.78% compared to the baseline, the 3.9% improvement by $\text{MCBUL}_{\mathcal{R}_2}$ represent a substantial increase in performance. Moreover, looking at the corresponding WIN% value on Figure 3, we see that $\text{MCBUL}_{\mathcal{R}_2}$ has a higher BAI% than MOTTTS-interleaved on 100% of the generated MOMABs, showing that it reliably optimizes the timing of queries, regardless of the MOMABs' properties. In contrast, $\text{MCBUL}_{\mathcal{R}_1}$ with 10^6 rollouts has a higher average BAI%, but is better than MOTTTS-interleaved on 73.33% of the generated MOMABs. Moreover, we expect the WIN% and BAI% to increase as we increase the number of rollouts. Since the best-arm-identification setting is not necessarily an online setting, we expect this to be possible depending on the problem at hand. Thus, our MCBUL algorithm learns different timings depending on the MOMAB, pulling arms more efficiently with respect to the estimated utility, resulting in a higher chance of finding the best arm than when using a fixed timing for queries.

6 RELATED WORK

While, to the best of our knowledge, we are the first to propose an algorithm for best arm identification in the multi-objective setting, this setting is related to different areas of work.

MOMABs [16] have been studied to learn the set of Pareto optimal policies [19, 30]. Yahyaa et al. [52, 53] assume Bernoulli distributions over the rewards, and additionally aim for a fair pulling of all Pareto-efficient arms. These methods minimize the regret using a multi-objective performance metric. For example, Daulton et al. [14] learn the set of Pareto-efficent arms using a fast approximation of the expected hypervolume improvement, while Belakaria et al. [5] select the action that maximizes information gained about the Pareto front. Also, the Pareto front can be learned by sampling different scalarization functions, based on the hypervolume [54].

In contrast to these methods that do not assume knowledge over u, MOMABs have been used for known utility functions in constrained optimization, either optimizing a single objective subject to predefined linear constraints [29], or non-linear constraints [43, 44]. For MABs, this has been used in the context of clinical trials, by identifying dosages that satisfy toxicity constraints [39].

Instead of constraints, another approach is to provide an order of preferences over the different objectives, and incorporate that into a multi-objective Bayesian optimisation framework [1].

More closely related to our setting, Paria et al. [32] minimize the regret of MOMABs by randomly sampling utility functions. The sampling strategy can incorporate prior knowledge over u, thus fine-tuning the set of optimal arms. While they do not incorporate a way to choose or update this prior, they argue their method is compatible with potential updates of information over preferences during the optimization process.

One other work that considers interactive preference learning for MOMABs is [2]. Like in our work, they incorporate pairwise relative queries, and they use parametric utility functions. Moreover, on top of linear utility functions, they consider quadratic and exponential utility functions. However, they focus on regret minimization, and ask queries at fixed intervals, like ITS [36].

Our query-strategy is based on Thompson sampling so we can quickly propose queries. Alternative strategies based on volume removal maximization [38], information gain [6, 7, 22, 48] or regret [49] have been proposed to efficiently learn u.

Finally, preference learning has been considered in contextual MABs [26] and RL [50, 51]. However, while they use relative queries, they use partial trajectories, or states, instead of multi-objective trade-offs [10, 27]. Interestingly, [55] consider both pairwise relative preferences and ranking preferences, by ordering 4 samples.

7 CONCLUSION AND FUTURE WORK

We have shown that learning both the utility function and the policy requires an intertwined approach, as demonstrated by MOTTTS-interleaved which performs the best out of all baselines. Because MOTTTS-interleaved suffers from a significant performance gap with respect to the (unachievable) multi-objective TTTS oracle upper bound, we argued that its strategy can still be significantly improved. For this reason, we propose MCBUL[2], a Bayesian approach towards query-timing optimization for best-arm identification. Given enough simulated rollouts, MCBUL can accurately estimate which action maximally improves its belief over the best arm in terms of utility. As an additional feature, MCBUL only requires information about the utility function in the form of relative queries, which are easier to gather from human decision makers. We showed how MCBUL achieves higher identification scores in multiple generated mo-bandits than the competing baselines.

While the concept of the MCBUL algorithm can be applied directly on many multi-objective problems, the implementation we demonstrated in this work has two main limitations. The first is that our work showcases MCBUL only on MOMABs with linear utility functions. While particle filtering should be adaptable to more complex settings, for example those using non-linear parametric scalarization functions [47], the number of particles required to cover the parametric space increases exponentially with the number of objectives. As an alternative, MCBUL could be adapted to use Gaussian Processes (GPs) to model non-linear utility functions [35].

The second limitation is that, due to the fact that computing the return of a single rollout is (in relative terms) a fairly expensive operation, MCBUL finds it difficults to deal with settings with large number of arms and/or objectives. In these cases, the top branches cannot be explored enough without an excessive computational cost, which limits MCBUL's applications. A possible way to deal with this issue is to apply adaptive particle filtering techniques that allow to only resample filters in more interesting regions of the weight space [25, 31]. We could similarly scale our experiments to MOMABs with more arms, by using, e.g., progressive widening [9, 12] on the actions in MCBUL's search tree.

ACKNOWLEDGMENTS

This research was supported by funding from the Fonds voor Wetenschappelijk Onderzoek (FWO) through the grant of E.B. (#1SA2820N) and from the Flemish Government under the "Onderzoeksprogramma Artificiële Intelligentie (AI) Vlaanderen" program.

[2] code available at https://github.com/mathieu-reymond/MCBUL

REFERENCES

[1] Majid Abdolshah, Alistair Shilton, Santu Rana, Sunil Gupta, and Svetha Venkatesh. 2019. Multi-objective Bayesian optimisation with preferences over objectives. In *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (Eds.), Vol. 32. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2019/file/a7b7e4b27722574c611fe91476a50238-Paper.pdf

[2] Raul Astudillo and Peter Frazier. 2020. Multi-attribute Bayesian optimization with interactive preference learning. In *International Conference on Artificial Intelligence and Statistics*. PMLR, 4496–4507.

[3] Jean-Yves Audibert, Sébastien Bubeck, and Rémi Munos. 2010. Best arm identification in multi-armed bandits.. In *COLT*. 41–53.

[4] Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. 2002. Finite-time analysis of the multiarmed bandit problem. *Machine learning* 47 (2002), 235–256.

[5] Syrine Belakaria, Aryan Deshwal, Nitthilan Kannappan Jayakodi, and Janardhan Rao Doppa. 2020. Uncertainty-aware search framework for multi-objective Bayesian optimization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34. 10044–10052.

[6] Erdem Biyik and Malayandi Palan. 2019. Asking Easy Questions: A User-Friendly Approach to Active Reward Learning. In *Proceedings of the 3rd Conference on Robot Learning*.

[7] Urszula Chajewska, Daphne Koller, and Ronald Parr. 2000. Making Rational Decisions Using Adaptive Utility Elicitation. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence*. AAAI Press, 363–369.

[8] Olivier Chapelle and Lihong Li. 2011. An empirical evaluation of thompson sampling. *Advances in neural information processing systems* 24 (2011).

[9] Guillaume M Jb Chaslot, Mark HM Winands, H Jaap van den Herik, Jos WHM Uiterwijk, and Bruno Bouzy. 2008. Progressive strategies for Monte-Carlo tree search. *New Mathematics and Natural Computation* 4, 03 (2008), 343–357.

[10] Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. 2017. Deep Reinforcement Learning from Human Preferences. In *Advances in Neural Information Processing Systems*, I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.), Vol. 30. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2017/file/d5e2c0adad503c91f91df240d0cd4e49-Paper.pdf

[11] Adrien Couëtoux, Jean-Baptiste Hoock, Nataliya Sokolovska, Olivier Teytaud, and Nicolas Bonnard. 2011. Continuous upper confidence trees. In *Learning and Intelligent Optimization: 5th International Conference, LION 5, Rome, Italy, January 17-21, 2011. Selected Papers 5*. Springer, 433–445.

[12] Rémi Coulom. 2007. Computing "elo ratings" of move patterns in the game of go. *ICGA journal* 30, 4 (2007), 198–208.

[13] Rémi Coulom. 2007. Efficient selectivity and backup operators in Monte-Carlo tree search. In *Computers and Games: 5th International Conference, CG 2006, Turin, Italy, May 29-31, 2006. Revised Papers 5*. Springer, 72–83.

[14] Samuel Daulton, Maximilian Balandat, and Eytan Bakshy. 2020. Differentiable expected hypervolume improvement for parallel multi-objective Bayesian optimization. *Advances in Neural Information Processing Systems* 33 (2020), 9851–9864.

[15] Arnaud Doucet, Simon Godsill, and Christophe Andrieu. 2000. On sequential Monte Carlo sampling methods for Bayesian filtering. *Statistics and computing* 10 (2000), 197–208.

[16] Madalina M Drugan and Ann Nowe. 2013. Designing multi-objective multi-armed bandits algorithms: A study. In *The 2013 international joint conference on neural networks (IJCNN)*. IEEE, 1–8.

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

[18] Victor Gabillon, Mohammad Ghavamzadeh, Alessandro Lazaric, and Sébastien Bubeck. 2011. Multi-bandit best arm identification. *Advances in Neural Information Processing Systems* 24 (2011).

[19] Eduardo C Garrido-Merchán and Daniel Hernández-Lobato. 2019. Predictive entropy search for multi-objective bayesian optimization with constraints. *Neurocomputing* 361 (2019), 50–68.

[20] Sylvain Gelly and David Silver. 2011. Monte-Carlo tree search and rapid action value estimation in computer Go. *Artificial Intelligence* 175, 11 (2011), 1856–1875.

[21] Neil J Gordon, David J Salmond, and Adrian FM Smith. 1993. Novel approach to nonlinear/non-Gaussian Bayesian state estimation. In *IEE proceedings F (radar and signal processing)*, Vol. 140. IET, 107–113.

[22] Shengbo Guo and Scott Sanner. 2010. Real-time Multiattribute Bayesian Preference Elicitation with Pairwise Comparison Queries. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics (Proceedings of Machine Learning Research, Vol. 9)*, Yee Whye Teh and Mike Titterington (Eds.). PMLR, Chia Laguna Resort, Sardinia, Italy, 289–296.

[23] Conor F Hayes, Roxana Rădulescu, Eugenio Bargiacchi, Johan Källström, Matthew Macfarlane, Mathieu Reymond, Timothy Verstraeten, Luisa M Zintgraf, Richard Dazeley, Fredrik Heintz, et al. 2022. A practical guide to multi-objective reinforcement learning and planning. *Autonomous Agents and Multi-Agent Systems* 36, 1 (2022), 26.

[24] Conor F Hayes, Mathieu Reymond, Diederik M Roijers, Enda Howley, and Patrick Mannion. 2023. Monte Carlo tree search algorithms for risk-aware and multi-objective reinforcement learning. *Autonomous Agents and Multi-Agent Systems* 37, 2 (2023), 26.

[25] Jeroen D Hol, Thomas B Schon, and Fredrik Gustafsson. 2006. On resampling algorithms for particle filters. In *2006 IEEE nonlinear statistical signal processing workshop*. IEEE, 79–82.

[26] Alihan Hüyük, Daniel Jarrett, and Mihaela van der Schaar. 2022. Inverse Contextual Bandits: Learning How Behavior Evolves over Time. In *Proceedings of the 39th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 162)*, Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato (Eds.). PMLR, 9506–9524.

[27] Borja Ibarz, Jan Leike, Tobias Pohlen, Geoffrey Irving, Shane Legg, and Dario Amodei. 2018. Reward learning from human preferences and demonstrations in atari. *Advances in neural information processing systems* 31 (2018).

[28] Ammar Jalalimanesh, Hamidreza Shahabi Haghighi, Abbas Ahmadi, Hossein Hejazian, and Madjid Soltani. 2017. Multi-objective optimization of radiotherapy: distributed Q-learning and agent-based simulation. *Journal of Experimental & Theoretical artificial intelligence* 29, 5 (2017), 1071–1086.

[29] Anmol Kagrecha, Jayakrishnan Nair, and Krishna Jagannathan. 2023. Constrained regret minimization for multi-criterion multi-armed bandits. *Machine Learning* (2023), 1–28.

[30] Marco Laumanns and Jiri Ocenasek. 2002. Bayesian optimization algorithms for multi-objective optimization. In *Parallel Problem Solving from Nature—PPSN VII: 7th International Conference Granada, Spain, September 7–11, 2002 Proceedings 7*. Springer, 298–307.

[31] Tiancheng Li, Miodrag Bolic, and Petar M Djuric. 2015. Resampling methods for particle filtering: classification, implementation, and strategies. *IEEE Signal processing magazine* 32, 3 (2015), 70–86.

[32] Biswajit Paria, Kirthevasan Kandasamy, and Barnabás Póczos. 2020. A flexible framework for multi-objective bayesian optimization using random scalarizations. In *Uncertainty in Artificial Intelligence*. PMLR, 766–776.

[33] Mathieu Reymond, Conor F. Hayes, Denis Steckelmacher, Diederik M. Roijers, and Ann Nowe. 2023. Actor-critic multi-objective reinforcement learning for non-linear utility functions. *Autonomous Agents and Multi-Agent Systems* 37, 2 (23 April 2023). https://doi.org/10.1007/s10458-023-09604-x

[34] Jason Rhuggenaath, Alp Akcay, Yingqian Zhang, and Uzay Kaymak. 2019. Optimizing reserve prices for publishers in online ad auctions. In *2019 IEEE Conference on Computational Intelligence for Financial Engineering & Economics (CIFEr)*. IEEE, 1–8.

[35] Diederik M Roijers, Luisa M Zintgraf, Pieter Libin, Mathieu Reymond, Eugenio Bargiacchi, and Ann Nowé. 2021. Interactive multi-objective reinforcement learning in multi-armed bandits with gaussian process utility models. In *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2020, Ghent, Belgium, September 14–18, 2020, Proceedings, Part III*. Springer, 463–478.

[36] Diederik M Roijers, Luisa M Zintgraf, and Ann Nowé. 2017. Interactive thompson sampling for multi-objective multi-armed bandits. In *Algorithmic Decision Theory: 5th International Conference, ADT 2017, Luxembourg, Luxembourg, October 25–27, 2017, Proceedings 5*. Springer, 18–34.

[37] Daniel Russo. 2016. Simple bayesian algorithms for best arm identification. In *Conference on Learning Theory*. PMLR, 1417–1418.

[38] Dorsa Sadigh, Anca D. Dragan, S. Shankar Sastry, and Sanjit A. Seshia. 2017. Active Preference-Based Learning of Reward Functions. In *Robotics: Science and Systems*.

[39] Cong Shen, Zhiyang Wang, Sofia Villar, and Mihaela Van Der Schaar. 2020. Learning for dose allocation in adaptive clinical trials with safety constraints. In *International Conference on Machine Learning*. PMLR, 8730–8740.

[40] Siegel Sidney. 1957. Nonparametric statistics for the behavioral sciences. *The Journal of Nervous and Mental Disease* 125, 3 (1957), 497.

[41] David Silver and Joel Veness. 2010. Monte-Carlo planning in large POMDPs. *Advances in neural information processing systems* 23 (2010).

[42] Ercan Sirakaya, James Petrick, and Hwan-Suk Choi. 2004. The role of mood on tourism product evaluations. *Annals of Tourism Research* 31, 3 (2004), 517–539.

[43] Yanan Sui, Alkis Gotovos, Joel Burdick, and Andreas Krause. 2015. Safe exploration for optimization with Gaussian processes. In *International conference on machine learning*. PMLR, 997–1005.

[44] Yanan Sui, Vincent Zhuang, Joel Burdick, and Yisong Yue. 2018. Stagewise safe bayesian optimization with gaussian processes. In *International conference on machine learning*. PMLR, 4781–4789.

[45] Gerald Tesauro. 1988. Connectionist learning of expert preferences by comparison training. *Advances in neural information processing systems* 1 (1988).

[46] William R Thompson. 1933. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika* 25, 3-4 (1933), 285–294.

[47] Kristof Van Moffaert, Madalina M Drugan, and Ann Nowé. 2013. Scalarized multi-objective reinforcement learning: Novel design techniques. In *2013 IEEE*

Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL). IEEE, 191–199.

[48] Paolo Viappiani and Craig Boutilier. 2020. On the equivalence of optimal recommendation sets and myopically optimal query sets. *Artificial Intelligence* 286 (2020), 103328. https://doi.org/10.1016/j.artint.2020.103328

[49] Nils Wilde, Dana Kulić, and Stephen L. Smith. 2020. Active Preference Learning using Maximum Regret. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 10952–10959. https://doi.org/10.1109/IROS45743.2020.9341530

[50] Christian Wirth, Riad Akrour, Gerhard Neumann, Johannes Fürnkranz, et al. 2017. A survey of preference-based reinforcement learning methods. *Journal of Machine Learning Research* 18, 136 (2017), 1–46.

[51] Christian Wirth, Johannes Fürnkranz, and Gerhard Neumann. 2016. Model-free preference-based reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 30.

[52] Saba Q Yahyaa, Madalina M Drugan, and Bernard Manderick. 2014. Annealing-pareto multi-objective multi-armed bandit algorithm. In *2014 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)*. IEEE, 1–8.

[53] Saba Q Yahyaa and Bernard Manderick. 2015. Thompson Sampling for Multi-Objective Multi-Armed Bandits Problem.. In *ESANN*.

[54] Richard Zhang and Daniel Golovin. 2020. Random hypervolume scalarizations for provable multi-objective black box optimization. In *International Conference on Machine Learning*. PMLR, 11096–11105.

[55] Daniel M Ziegler, Nisan Stiennon, Jeffrey Wu, Tom B Brown, Alec Radford, Dario Amodei, Paul Christiano, and Geoffrey Irving. 2019. Fine-tuning language models from human preferences. *arXiv preprint arXiv:1909.08593* (2019).

[56] Masrour Zoghi, Shimon Whiteson, Remi Munos, and Maarten Rijke. 2014. Relative upper confidence bound for the k-armed dueling bandit problem. In *International conference on machine learning*. PMLR, 10–18.