On Following Pareto-Optimal Policies in Multi-Objective Planning and Reinforcement Learning

Diederik M. Roijers
Vrije Universiteit Brussel
Brussels, Belgium &
HU Univ. of Applied Sciences Utrecht
Utrecht, the Netherlands
diederik.roijers@vub.be

Willem Röpke
Vrije Universiteit Brussel
Brussels, Belgium
willem.ropke@vub.be

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

Roxana Rădulescu
Vrije Universiteit Brussel
Brussels, Belgium
roxana.radulescu@vub.be

ABSTRACT

Multi-objective sequential decision problems have been studied in the literature for a long time. White (1982) already proposed a value iteration algorithm for computing Pareto Coverage Sets (PCS), i.e., Pareto fronts, in infinite-horizon multi-objective Markov decision processes (MOMDPs). Likewise, reinforcement learning algorithms have been proposed to learn these PCSs. However, many papers stop after showing that the value vectors in the PCSs can be computed, and do not go into executing the policies that constitute these value vectors. In this paper, we show that when the transition function is stochastic, selecting a value vector and executing the corresponding policy (without explicitly representing it) leads to a combinatorial optimisation problem at each action selection step. We evaluate the consequences of this by performing a heuristic search algorithm to retrieve and execute the policy. Furthermore, we propose a method to train a policy in the form of a neural network at planning or learning time, and compare this to the heuristic optimisation approach for executing policies.

KEYWORDS

Policy execution, Multiple objectives, MOMDPs

1 INTRODUCTION

Many real-world decision problems have more than one objective [2]. In fact, multiple objectives is often a large component in why people perceive such decision problems as difficult [1]. In artificial intelligence we therefore need to be able incorporate such objectives in our models.

Much research in artificial intelligence (AI) focuses on simplified models in which all of a decision problem's objectives are forced into a single reward function. For some applications this may be fine, as such modelling may well lead to an acceptable agent behaviour. However, explicitly using multi-objective models, solutions and policies does make the decisions regarding trade-offs between the objectives explicit, and therefore can empower the human users of AI by enabling them to take well-informed decisions about these trade-offs [2]. This becomes especially important when the AI has a serious potential to impact human lives, as it is then essential to align the AI's policies with human preferences regarding the objectives in the decision problem [17].

In multi-objective planning and reinforcement learning (RL) [15], multiple objectives are incorporated by using vector-valued reward functions [10]. A key model in multi-objective planning and RL is the multi-objective Markov decision process (MOMDP). In planning this model has been studied as early as the late 1970s / early 1980's [4, 19]. In 1982, White [18] proposed a value iteration algorithm to compute Pareto coverage sets (PVI) and proved that this converges to the correct values. Much more recently, Van Moffaert and Nowé [6] proposed the Pareto Q-Learning algorithm (PQL) that does the same for RL settings.

PVI and PQL are able to plan or learn correct Pareto coverage sets (PCSs). This means that they are able to output the set of (close-approximately correct) attainable value vectors that are Pareto-undominated, thereby covering all possible monotonically increasing utility functions that a user might have to make trade-offs between value vectors. Such a PCS is thus a sufficient solution set for all multi-objective settings [13].

One might well think that PVI and PQL are therefore ready for real-world usage. However, as this paper will show, we may run into issues when the policies actually need to be executed. Specifically, when the transition function of the MOMDP is stochastic, executing the policy implies solving a combinatorial optimisation problem at each action selection step. This stands in contrast to deterministic-transition-function MOMDPs, for which Van Moffaert and Nowé [6] show that following the policy associated with a given value vector can be performed by keeping an estimate of the immediate expected reward function (separate from the long-term returns), and doing some simple calculations at each action selection step.

After showing that Pareto-optimal policy (POP) following leads to combinatorial optimisation problems at each timestep, we investigate two ways of performing this task. Firstly, we solve the POP following problem by calling a heuristic optimisation algorithm – iterated local search – at each action selection, and investigate how the quality of the policy executions changes as a function of the time invested at each action-selection step. Secondly, as we actually know how each value vector was generated from a combination of actions for possible states during planning or learning, we observe

Proc. of the 1st Multi-Objective Decision Making Workshop (MODeM 2021), Hayes, Mannion, Vamplew (eds.), July 14-16, 2021, Online, http://modem2021.cs.nuigalway.ie. 2021.

that this generates data that we can use to train a neural network for POP following. We investigate how such a POP network compares to the heuristic optimisation approach.

MODeM positioning

As the shape and size of optimal solution in multi-objective decision-making depends on the setting in which it is used and what is known about the utility function of the user(s) and allowed policies [2, 12], we would like to take a paragraph positioning this paper with respect to this field. We envision a multi-policy scenario (e.g., unknown weights or decision support) [12], where the policies are allowed to be non-stationary, but are forced to be deterministic (similar to [6] and [18]). We must note though that our non-stationary policies do not explicitly condition on time, but rather on the vector that is being followed, and the visited states. This results in policies that are, in practice, conditioned on the state-history.

2 THE MULTI-OBJECTIVE MDP MODEL

As a mathematical framework for modeling multi-objective decision making settings we use the Multi-Objective Markov Decision Process (MOMDP). A MOMDP is a tuple $M = (S, A, T, \gamma, \mathbf{R})$, with $d \geq 2$ objectives, where:

- S is the state space
- A is the set of actions
- $T: S \times A \times S \to [0, 1]$ is the probabilistic transition function
- γ is the discount factor
- $\mathbf{R}: S \times A \times S \to \mathbb{R}^d$ is the vectorial reward function for each of the d objectives

An agent behaves according to a policy $\pi : S \times A \to [0, 1]$, meaning that given a state, actions are selected according to a certain probability distribution. The user's utility is derived from the vector-valued returns, i.e.,

$$\sum_{t=0}^{\infty} \gamma^t \mathbf{r}_t,$$

the sum of discounted rewards accrued during policy execution, where $\mathbf{r}_t$ is the received vectorial reward obtained by the agent for performing action $a_t \in A$, at state $s_t \in S$ and transitioning to state $s_{t+1} \in S$. Please note that we also use $\mathbf{R}(s, a)$, which is the *expected* immediate reward for performing an action a in a state s.

The utility is typically modelled as a utility function that needs to be applied to the return vectors. For this, there are two choices [12]: to calculate the expected value of the return of a policy before applying the utility function leads to the scalarised expected returns (SER) optimisation criterion:

$$V_u^{\pi} = u\left(\mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \mathbf{r}_t \mid \pi, \mu_0\right]\right), \tag{1}$$

where μ_0 is the distribution over initial states $s_0 \in S$. SER is the most commonly used criterion in the multi-objective (single agent) planning and reinforcement learning literature [12]. The alternative, i.e., applying the utility function before computing the expectation, leads to the expected scalarised returns (ESR):

$$V_u^{\pi} = \mathbb{E}\left[u\left(\sum_{t=0}^{\infty} \gamma^t \mathbf{r}_t\right) \mid \pi, \mu_0\right]. \tag{2}$$

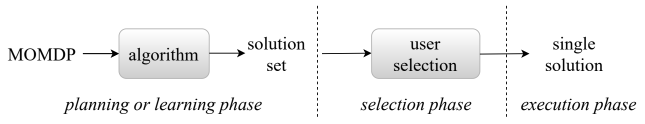

Figure 1: The unknown utility function scenario [12]

ESR is a more commonly used criterion in the game theory literature on multi-objective games [14].

In this paper, we focus on the SER optimality criterion; the more common criterion in the multi-objective reinforcement learning and multi-objective decision-theoretic planning literature. Under SER, optimising π is equivalent to maximising the expected discounted long-term reward with respect to the utility:

$$V_u^{\pi} = u(\mathbf{V}^{\pi}) = u\left(\mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \mathbf{r}_t \mid \pi, \mu_0\right]\right), \tag{3}$$

where μ_0 is the distribution over initial states s_0, γ is the discount factor and $\mathbf{r}_t$ the vectorial reward obtained by the agent for performing action $a_t \in A$, at state $s_t \in S$ and transitioning to state $s_{t+1} \in S$.

The policy value vectors, $\mathbf{V}^{\pi}$, lead to a partial ordering over policies. If the utility function is known, we may be able to optimise a single policy. However, if no further information about the *utility function*, u, of the user is available other than the fact that it is monotonically increasing in all objectives [12], and the policies are required to be deterministic, the optimal solution set becomes the set of all policy value vectors that are not Pareto dominated, i.e., the *Pareto Coverage Set*[1]:

$$PCS = \{\mathbf{V}^{\pi}, \pi \in \Pi \mid \nexists \pi' \in \Pi : \mathbf{V}^{\pi'} \succ_P \mathbf{V}^{\pi}\}, \tag{4}$$

In other words, for each policy in the Pareto Coverage Set, there exists no other policy with a value that is equal or better in *all* objectives and better in at least one objective. We note that in this work we restrict the set of all possible policies, Π, to deterministic, but possibly non-stationary policies, following the setting of [6] and [18].

Algorithms like Pareto value iteration (PVI) [18] and Pareto Q-Learning (PQL) [6] approximate the value vectors that constitute the PCS. However they do not explicitly learn the policy that constitute these vectors. Therefore the actual policies that belong to these value vectors need to be induced from the available information. This is not straightforward as in the single-objective case, as we will show in the next Section.

3 THE POP FOLLOWING PROBLEM

The Pareto-optimal Policy following problem arises from the following task: after a PCS inducing algorithm – like PVI or PQL – has outputted an approximate PCS in terms of value vectors (Equation 3), value vectors are shown to a user [23] to select which policy the user prefers, in the *selection phase* [2, 12]. Then, in the *execution phase* a policy that leads to this value vector, $\mathbf{V}^u$, needs to be executed. For an overview of this process see Figure 1.

After PVI or PQL terminates, the following information can and should be retained for policy execution:

[1]Following [12], we use the term Pareto Coverage Set rather than Pareto front.

- A local PCS for every state-action pair: $Q(s, a)$, containing all Pareto-undominated value vectors for that state-action pair. Please note that the original PVI by White [18] can easily be adapted to output this.
- An expected immediate reward function $\mathbf{R}(s, a)$ for each state-action pair. In PVI this is given (as it applies to the planning setting), in PQL this is learnt.
- A transition probability function $T(s'|s, a)$. In PVI this is given. The original PQL algorithm does not learn this, because it is applied to deterministic-transition MOMDPs only, but it can easily be adapted to learn $T(s'|s, a)$ alongside $\mathbf{R}(s, a)$.

Assuming the initial state is always the same, i.e., s_0, and the above-mentioned information is available, selecting the initial action is not a problem. Specifically, we select the action for which:

$$\arg\min_{a} \min_{\mathbf{q} \in Q(s_0, a)} ||\mathbf{q} - \mathbf{V}^u||. \tag{5}$$

This generalises to the following: given the value vector, $\mathbf{V}$, that we should follow in a given state, s, we should execute the action for which the local PCS $Q(s, a)$ contains the value vector that most closely resembles the target $\mathbf{V}$:

$$\hat{\pi}(s|\mathbf{V}) = \arg\min_{a} \min_{\mathbf{q} \in Q(s, a)} ||\mathbf{q} - \mathbf{V}||. \tag{6}$$

However, this does not constitute a full policy. This is due to the fact that, while we know which vector to follow at this timestep, this does not imply we automatically know which vector to follow in the next timestep. Specifically, to know which vector we need to follow in the next timestep, we need to analyse how the current vector $\mathbf{V}$ is built up from combinations of vectors in the next timestep. This is given by the (vectorial) Bellman equation:

$$\mathbf{V} = \mathbf{R}(s, a) + \gamma \sum_{s'} T(s'|s, a)\mathbf{v}_{s'}, \tag{7}$$

where $\mathbf{v}_{s'}$ is the appropriate vector that we should follow from state s', given that we started in s and were following $\mathbf{V}$. This cannot be solved by defining a single target vector to follow in every state. For example, imagine an MOMDP (with $\gamma = 1$) where in state s_0, we aim to follow $\mathbf{V^u} = (5, 5)$, for which we need to perform action a_0, that transitions to two possible subsequent states, $s_{1,1}$ and $s_{1,2}$, with equal probability (Figure 2). From $s_{1,1}$ the local PCS is $\{(10, 0), (4, 4)\}$. From $s_{1,2}$ the local PCS is $\{(0, 10), (4, 4)\}$. Now if we continue to follow $(5, 5)$ in both $s_{1,1}$ and $s_{1,2}$, we would end up with $(4, 4)$, but actually the original $(5, 5)$ vector is only attainable if we follow $(10, 0)$ from $s_{1,1}$ and $(0, 10)$ from $s_{1,2}$. In fact, $(4, 4)$ is a dominated vector from s_0, as we can attain $(5, 5)$.

To know what vector $\mathbf{v}_{s'}$ to follow in each possible subsequent state s' given that we were following $\mathbf{V}$ in the current state, let us first define the component of $\mathbf{V}$ that is constituted from value vectors for the next state, $\mathbf{N}$:

$$\mathbf{N} = \sum_{s'} T(s'|s, a)\mathbf{v}_{s'} = \frac{\mathbf{V} - \mathbf{R}(s, a)}{\gamma}. \tag{8}$$

To get to the appropriate vectors, $\mathbf{v}_{s'}$ to follow in the subsequent states, we should thus solve,

$$\arg\min_{\mathbf{v_1}, \cdots, \mathbf{v_n}} \left\| \mathbf{N} - \sum_{\{s_i \mid T(s_i|s,a) > 0\}} T(s_i|s, a)\ \mathbf{v}_i \right\|, \tag{9}$$

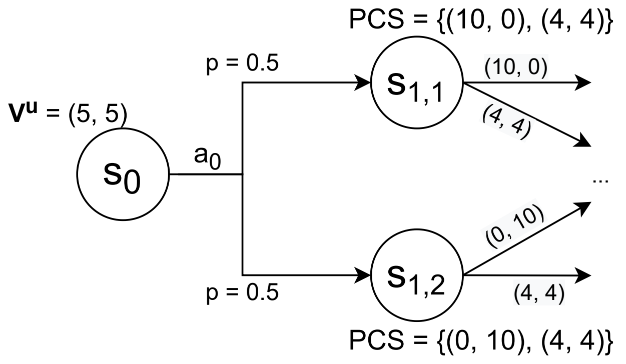

Figure 2: MOMDP demonstrating the POP-following problem. After taking one step in the environment, trying to follow the initial $\mathbf{V}^u = (5, 5)$ is no longer a straightforward process. Please note that after states $s_{1,1}$ and $s_{1,2}$ the episode ends after taking the either action (a_0 or a_1).

where $\mathbf{v}_i$ is selected from the corresponding $\mathcal{V}(s_i) = \bigcup_a Q(s_i, a)$, for every transition we encounter during policy execution.

This is a non-trivial problem. Firstly, because both PVI and PQL both approximate their results, the minimisation of Equation 9 will not be entirely 0, and because of the absolute value, the target is non-linear in its constituent components. Furthermore, because we need to select a vector for every subsequent state s_i for which $T(s_i|s, a) > 0$, the size of the search space of this optimisation problem grows exponentially in the number of subsequent states. Specifically, if the maximum size of a local PCS, $Q(s_i, a)$, is P, the size of the action space is A, and the maximum size of the set of possible subsequent states from any state s is X, the size of the search space is $O((AP)^X)$.

We thus come to the conclusion that following a Pareto-optimal policy – by selecting a value vector from the output of PVI or PQL in an MOMDP with stochastic state transitions – leads to a combinatorial optimisation problem *every timestep* during policy execution. To summarise, policy execution is presented in Algorithm 1. At each timestep, the algorithm starts from a value vector $\mathbf{V}$ and the current state s. First, the action to execute is selected using Equation 5, which is subsequently executed in the environment. This provides us with the next state, s', for which we have to select the next value vector to follow. Selecting this value vector is done by a function, `selectNextValueVector`$(\mathbf{N}, s, a, s')$, that either solves the aforementioned combinatorial optimisation problem heuristically (Section 4) or uses a pretrained solver for this problem in the form of a neural network (Section 5). Please note that to calculate $\mathbf{N}$, the expected reward $\mathbf{R}(s, a)$ is used, rather than the received rewards $\mathbf{r}$. This is because under SER, we need to calculate everything in expectation.

4 POPF LOCAL SEARCH

The most straightforward approach to solving the combinatorial optimisation problems that arise from POP following, is using heuristic search at every timestep. In this paper, we employ an iterative local search scheme. Firstly, we propose a local search algorithm

Algorithm 1 Pareto-Optimal Policy Execution

Input: The selected value-vector $\mathbf{V}$, the initial state s_0, the environment env, the local PCSs $Q(s,a)$

```
1: $s \leftarrow s_0$
2: while $s$ is not terminal do
3:     $a \leftarrow \arg\min_a \min_{\mathbf{q} \in Q(s,a)} ||\mathbf{q} - \mathbf{V}||$
4:     $\mathbf{r}, s' \leftarrow$ env.executeAction($a$)
5:     $\mathbf{N} \leftarrow \frac{\mathbf{V} - \mathbf{R}(s,a)}{\gamma}$
6:     $\mathbf{V} \leftarrow$ selectNextValueVector($\mathbf{N}, s, a, s'$)
7:     $s \leftarrow s'$
8: end while
```

Algorithm 2 Pareto-optimal Policy Following Local Search

Input: $\mathbf{N}$, s, a, a start solution vs $= [\mathbf{v}_1, ..., \mathbf{v}_n]$.
Output: a solution vs $= [\mathbf{v}_1, ..., \mathbf{v}_n]$ approximating Eq. 9.

```
1: $\mathcal{S}' \leftarrow$ queue of all possible $s'$ for which $T(s'|s,a) > 0$ ▷ in random order
2: optval $\leftarrow ||\mathbf{N} - \sum_i T(s_i|s,a)$vs$[i]||$
3: while $\mathcal{S}'$ is not empty do
4:     $s_i \leftarrow \mathcal{S}'.dequeue()$
5:     for all possible values $\mathbf{v}_i$ in $\mathcal{V}(s_i)$ do
6:         vs' $\leftarrow$ vs
7:         vs'$[i] \leftarrow \mathbf{v}_i$
8:         if $||\mathbf{N} - \sum_i T(s_i|s,a)$vs'$[i]|| <$ optval then
9:             optval $\leftarrow ||\mathbf{N} - \sum_i T(s_i|s,a)$vs'$[i]||$
10:            vs $\leftarrow$ vs'
11:            Reset $\mathcal{S}'$ with all possible subsequent states $s'$
12:        end if
13:    end for
14: end while
15: return vs, optval
```

that optimises for Equation 9, *Pareto-optimal Policy Following Local Search (POPF-LS)* (Algorithm 2), that given a state-action pair, s, a, and an $\mathbf{N}$-vector as defined in Equation 3, starts from a(n initially random) combination vs of vectors from the subsequent states, and optimises this combination in order to approach $\mathbf{N}$ as closely as possible. vs then contains a vector to follow in each possible subsequent state. Therefore, as soon as s' is observed, the corresponding vector can be selected, vs$[s']$.

As a local search scheme can quickly get stuck in local optima, we embed POPF-LS in an iterated local search scheme (Algorithm 3). The number of iterations in this iterated local search scheme (max_iter) will determine how much time is spent on heuristic search at each timestep. We hypothesise, due to the relatively small size of the problems in tabular settings, that it would be possible to find a qualitatively high solution to Equation 9, if given enough time. We test how much time is required in Section 6.

Please note that to use any of these optimisers in Algorithm 1, we should select the right value vector from vs. Further note that if we set $p = 1$ in Algorithm 3, we obtain what is known as a *multi-start local search* scheme rather than an iterated search scheme. Along with iterated POPF-LS we also test this multi-start POPF-LS approach in Section 6.

Algorithm 3 Iterated POPF-LS

Input: $\mathbf{N}$, s, a, a perturbation probability p, max_iter.
Output: a solution vs $= [\mathbf{v}_1, ..., \mathbf{v}_n]$ approximating Eq. 9.

```
1: vs $\leftarrow$ a random combination of vectors for each possible subsequent state.
2: val $\leftarrow -\infty$
3: $x \leftarrow 0$
4: while $x < max\_iter$ do
5:     vs' $\leftarrow$ vs
6:     for each subsequent state, $s_i$, for which $T(s'|s,a) > 0$ do
7:         if random number in [0,1] $< p$ then
8:             vs$[s_i] \leftarrow$ a random vector from $\mathcal{V}(s_i)$
9:         end if
10:    end for
11:    vs', val' $\leftarrow POPF-LS(\mathbf{N}, s, a, $vs'$)$        ▷ Algorithm 2
12:    if val' > val then
13:        val $\leftarrow$ val'
14:        vs $\leftarrow$ vs'
15:    end if
16:    $x$++
17: end while
18: return vs and optval
```

5 POP NETWORKS

An alternative to using heuristic search at policy execution time, is to train a POP following network during planning or learning.[2] In this paper, we will assume the PCSs are retrieved by planning via Pareto Value Iteration (PVI) [18].[3] This is a value iteration algorithm, that in each iteration bootstraps the PCSs in $Q(s',a')$ of the subsequent states, s', to build PCSs for a given state action pair $Q(s,a)$, using the following formula:

$$Q(s,a) \leftarrow \mathbf{R}(s,a) + \gamma \ \texttt{PPrune}\left(\bigoplus_{s'} T(s'|s,a) \bigcup_{a'} Q(s',a')\right), \quad (10)$$

where $\oplus$ denotes the cross-sum between sets of vectors, and PPrune is an algorithm that prunes away all Pareto-dominated vectors.[4]

Now let us recall the POP following problem. That is, given $\mathbf{N}$, as defined in Equation 3, s, a, and the subsequent state, s' as input, we need the right vector $\mathbf{v}_{s'}$ as output. We note that while we are executing Equation 10, we could in fact keep track of all the constituent vectors from $Q(s',a')$ that lead to a vector in $Q(s,a)$, noting that the $\mathbf{N}$ vector is merely the vector in $Q(s,a)$ minus the immediate expected reward $\mathbf{R}(s,a)$ divided by γ. In other words, we can create data of the following format:

$$\mathbf{N}, s, a, s' \rightarrow \mathbf{v}_{s'},$$

which can be used as input and targets for a supervised learning problem, while executing PVI. When we apply this data to train

[2]In practice, we generate the training data for the neural network during planning, and train the network parameters after planning, as this makes the code more modular, such that the planning does not need to be re-run while updating the neural network training procedure. Please note that, as this happens before selecting the value vector to execute, and before policy execution, we still consider it part of the planning/learning phase.
[3]Please note that in the original paper by White [18], the name PVI is not used. We adopted this name for the algorithm from [13].
[4]For an example of a simple implementation of PPrune see e.g., [10], Algorithm 2.

a neural network, we can obtain an approximate POP following solution specific to the MOMDP for which we performed PVI.

In the next section, we compare the neural network approach to the POPF Local Search approach on MOMDPs.

6 EXPERIMENTS

In this section we empirically compare the POPF local search, multi-start POPF local search, and iterated POPF local search approaches as described in Section 4 to the neural network approach described in Section 5. We employ randomly generated MOMDPs as used in [11]. For a random MOMDP, we run PVI, to learn the local PCSs. Subsequently, we use each POP-following method (one of the POPF local search approaches or POP networks) to do 200 roll-outs, following the same value vector, $\mathbf{V}^u$ from the starting state (selected at random). The average return obtained via these roll-outs is the estimate of the value of the followed policy $\hat{\mathbf{V}}^\pi$. We use the ε-metric [24] to evaluate the quality of $\hat{\mathbf{V}}^\pi$ with respect to $\mathbf{V}^u$. Specifically, we calculate:

$$\varepsilon = \inf_{\varepsilon \in \mathbb{R}_{\geq 0}} \left\{ \forall i = 1 \dots n \ : \ V_i^u \leq \hat{V}_i^\pi + \varepsilon \right\}, \tag{11}$$

where n is the number of objectives, as a measure of quality. Please note that this means that if $\hat{\mathbf{V}}^\pi$ Pareto-dominates $\mathbf{V}^u$, $\varepsilon = 0$. This can happen, as we run PVI [18] for a finite number of iterations, starting from a vector of zeroes, meaning that for the random MOMDPs (which have only positive rewards), the found value-vectors are a guaranteed underestimation. If this happens, we argue that the maximal utility loss incurred by the user is 0. The maximal utility loss (MUL) incurred for $\varepsilon > 0$ is $\epsilon\sqrt{n}L$, where L is the Lipschitz-constant describing how continuous the utility function of the user is [22].

For the POP networks, we use MLPs with 3 hidden layers with subsequently 16, 8, and 4 nodes, with ReLU activations. For training the network, we use the Adam optimiser, with a learning rate of 1×10^{-4} (0.0001) and the mean squared error loss. For the training procedure we use a $80-20$ split between the training and validation sets, with a batch size of 32, and save the best model obtained over 3000 epochs. We note that we did not optimise the neural network structure, or its hyperparameters, so there may well be performance gains achievable by performing such optimisations. For this paper however, we aim to show that this approach works out of the box with a fairly basic structure.

The code for our algorithms and experiments – including the exact MOMDPs and value vectors outputted by PVI – is available at https://github.com/rradules/POP-following.[5] All experiments were run on a AMD Ryzen 5 1600 Six-Core Processor (Clock speed 3.2 GHz), 80 GB RAM.

6.1 Random MOMDP 1

The first randomly generated MOMDP has 10 states, 2 actions, 2 objectives, and 4 subsequent states (s') that are reachable for each state-action pair, (s, a) with randomly generated transition probabilities $0 < T(s'|s, a) < 1$ (summing to one). The exact MOMDP definition will be included in the experimentation code upon publication. Running PVI lead to a total of 1563 vectors for local PCSs in the initial state, $Q(s = 0, a = 0) \cup Q(s = 0, a = 1)$. From this set, a (Pareto-undominated) vector, $\mathbf{V}_1^u$, to follow is chosen at random. In total there were 19208 value vectors in the PCSs across all state-action pairs.

In order to compare the local search approaches to the neural network approach, we run the algorithms on this MOMDP. We first compare the quality in terms of the ε-metric, of the value estimated by averaging over 200 roll-outs of using the POP Neural Network (POP-NN), versus POPF Local Search (LS), and Multi-Start and Iterated POPF-LS (MLS and ILS) using 5 iterations of POPF-LS each, with ILS having a perturbation probability $p = 0.3$.

Table 1: ε-metric results for MOMDP1. Number of iterations for ILS and MLS is 10.

	POP-NN	LS	MLS	ILS
ε-metric	0.05178	0.23108	**0.0**	0.09310

As the results in Table 1 indicate, POP-NN is able to get close to the selected value vector ($\varepsilon = 0.05178$) while one run of POPF LS is much further off ($\varepsilon = 0.23108$). This can be mitigated however, by running POPF LS multiple times each action selection from random initial solutions, i.e., MLS. Doing this 5 times resulted in an ε of 0. ILS with 5 iterations does not yet get to 0, but also takes less time than MLS.

When we raise the number of iterations for MLS and ILS to 10, they both obtain $\varepsilon = 0$ (Figure 3). Furthermore, raising the number of iterations further consistently keeps $\varepsilon = 0$. For an equal number

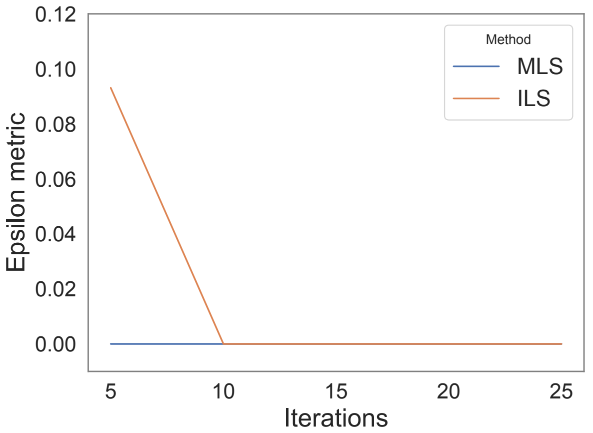

Figure 3: The ε (Equation 11) of the value vector estimate over 200 roll-outs, $\hat{\mathbf{V}}^\pi$, of MLS and ILS as a function of the number of iterations (i.e., `max_iter` in Algorithm 3) for MOMDP1.

of iterations, such as 10 as shown in Figure 4, ILS is faster than MLS. However, in terms of runtime, all local search methods are blown out of the water by POP-NN, which uses a factor 42 less runtime. We

[5]Please note that in the code, PVI – including the PCS output and the data set creation for the neural networks – is run separately from the training of the neural networks, and again separate from the policy execution code. We did so to keep the code modular, and to not have to re-plan and re-train when evaluating different policy execution approaches.

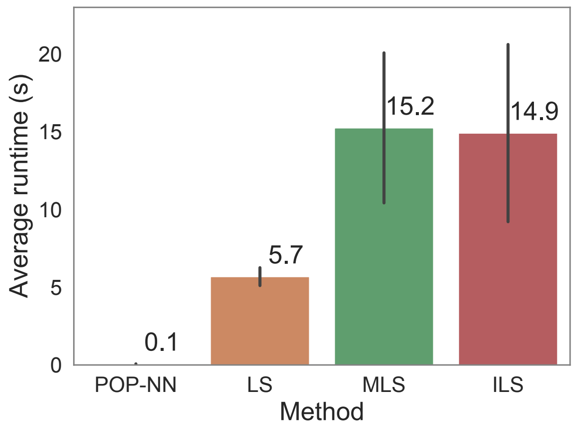

Figure 4: Average runtime per roll-out, estimated over 200 roll-outs for MOMDP1 (error bars represent the standard deviation).

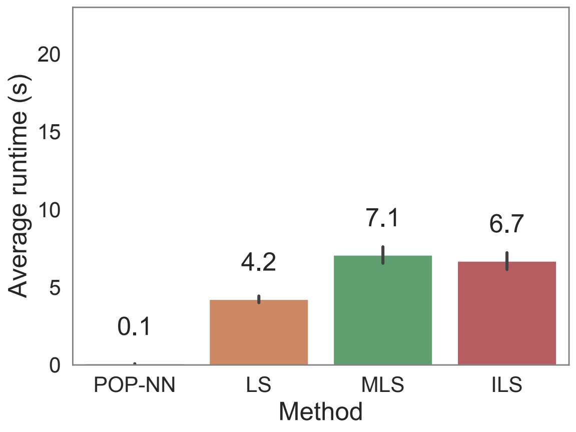

Figure 5: Average runtimes over 200 roll-outs for MOMDP2 (error bars represent the standard deviation).

Table 2: ε-metric results for MOMDP2. Number of iterations for ILS and MLS is 10.

	POP-NN	LS	MLS	ILS
ϵ-metric	**0.04131**	0.40360	0.31725	0.36711

further note that MLS uses less than double the amount of runtime as LS. This is because a lot of preprocessing is done (e.g., multiplying the vectors in the local PCSs by the transition probabilities), which does not need to be repeated the second time LS is called by MLS.

To summarise, we can conclude that POP-NN uses a fraction of the runtime of LS, and does achieve better results than LS. However, MLS and ILS are able to achieve better results than POP-NN, if given sufficient runtime, on this 10-state 2-action MOMDP.

6.2 Random MOMDP 2

The second randomly generated MOMDP has 20 states, 3 actions, 2 objectives, and 7 subsequent states (s') that are reachable for each state-action pair, (s, a) with randomly generated transition probabilities $0 < T(s'|s, a) < 1$ (summing to one). The exact MOMDP definition will be included in the experimentation code upon publication. Running PVI lead to a total of 284 vectors for local PCSs in the initial state, $Q(s = 0, a = 0) \cup Q(s = 0, a = 1) \cup Q(s = 0, a = 2)$. From this set, a (Pareto-undominated) vector, $\mathbf{V}_1^u$, to follow is chosen at random. In total there were 7971 value vectors in the PCSs across all state-action pairs. So while MOMDP2 has a larger state and action space, the PCSs are actually significantly smaller.

Again, we first compare the quality in terms of the ε-metric, of the value estimated by averaging over 200 roll-outs of using the POP Neural Network (POP-NN), versus POPF Local Search (LS), and Multi-Start and Iterated POPF-LS (MLS and ILS) using 10 iterations of POPF-LS each, with ILS having a perturbation probability $p = 0.3$. This results in Table 2. Furthermore, we determine the runtimes that correspond to the quality measurements of Table 2 resulting in Figure 5. A striking difference with MOMDP1 is that now POP-NN attains much better results than LS, MLS, and ILS. We hypothesise that this is because the number of value vectors in the PCSs of MOMDP1 is much higher than that of MOMDP2, leaving LS more options to gradually improve (at the expense of some runtime) and increasing the probability of multiple (near-)optimal solutions to the POP Following problems.

Another key observation in comparing the qualities and runtimes for MOMDP1 and MOMDP2, is that the POP neural networks perform consistently well on both problems. Furthermore, the runtime of the POP-NN approach is not significantly effected by differences in the number of value vectors in the PCS, nor the size of the size of the state and action spaces for these – admittedly small – MOMDPs. This is because doing a single forward pass through a neural network is highly efficient compared to running heuristic optimisation methods.

We note that the quality of the MLS and ILS approaches do seem to increase when providing them with more iterations, as shown in Figure 6. However, at 25 repetitions neither method comes close to the quality achieved by POP-NN.

We thus conclude that POP-NN consistently performs well, but not perfect, on both random MOMDP instances, at a fraction of the runtime of local-search-based approaches. Furthermore, when giving MLS and ILS more time to find a solution, it is possible that optimal solutions are found, as in MOMDP1. However, it is not clear a priori how many iterations are necessary for this to happen, as illustrated by the results of MOMDP2. Therefore, the ideal number of iterations – the number of iterations necessary to reach the best possible quality while minimising runtime – needs to be tuned per MOMDP instance, which is not ideal for use in practice.

Finally, we note that we report the runtimes of the algorithms at execution time. We believe that this is key, as a) it can be critical for a policy to be able to execute in real-time, and b) because we hope that the time the policy is used in practice is much larger than the time it takes to train a policy. Note though that this means

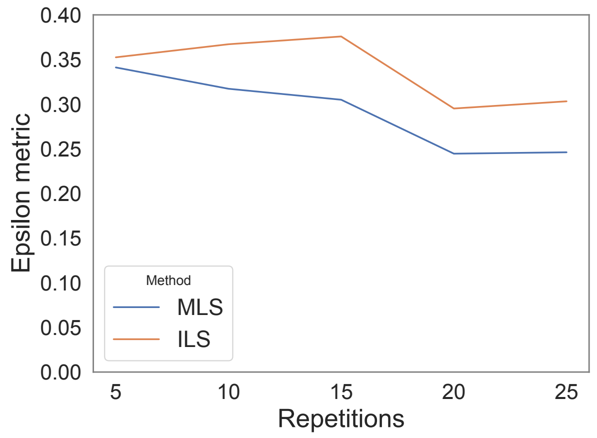

Figure 6: The ε (Equation 11) of the value vector estimate over 200 roll-outs, $\hat{\mathbf{V}}^{\pi}$, of MLS and ILS as a function of the number of iterations (i.e., `max_iter` in Algorithm 3) for MOMDP2.

that planning, i.e., the runtime of PVI, as well as the runtime of the training of the neural network policies (which also happens in advance), is not included in this analysis.

7 RELATED WORK

In this paper, we have examined policy execution for Pareto-optimal deterministic non-stationary policies in infinite-horizon MOMDPs with stochastic transitions. For this, we rely on PVI [18] to produce the Pareto coverage sets of policy value vectors for each s, a-pair. An alternative to PVI with finite precision is presented in [5], but as they rely on episodes of finite length and deterministic MOMDPs, we used standard PVI in this paper.

For related MOMDP settings, policy-following mechanisms do exist. Specifically, for MOMDPs with deterministic transitions, Van Moffaert and Nowé [6] show policies can be executed. For MOMDPs with stochastic transitions, but restricting policies to be deterministic and stationary, the excellent work by Wiering et al. [20, 21] shows how to do planning while simultaneously learning a deterministic stationary policy.

We further note that continuous PCS approximations exist, which also represent a manifold of possible policy parameters [8, 9]. However, these consider explicitly stochastic policies, which falls outside the scope of this paper. Furthermore, we note that for stochastic policies, the Pareto Coverage set (Pareto front) is always convex [12, 16], i.e., the PCS is also the Convex Coverage Set, which is a highly exploitable property for policy execution.

8 CONCLUSION

In this paper, we have identified the POP-following problem, i.e., the fact that in an MOMDP with stochastic transitions – even if the transition function, reward function, and the local PCSs for every state-action pair $Q(s, a)$ are known – following the value-vector from the PCS leads to a combinatorial optimisation problem to solve at every timestep during policy execution. These combinatorial optimisation problems have a search-space size of $O((AP)^X)$, where P is the maximum size of a local PCS, $Q(s_i, a)$, A is the size of the action space and X is the maximum size of the set of possible subsequent states from any state s.

To tackle the POP-following problem, we have proposed two types of heuristic methods, one based on local search, and one based on training a neural network during planning (or learning). When comparing these two approaches on random MOMDPs of different sizes, we have found that POP neural networks consistently perform well, though not perfect, at a fraction of the runtime of local-search-based approaches. Multi-start and iterated POPF local search do get better as the number of iteration increases, and it may well be possible to execute a policy which (almost) perfectly approximates the user-selected value vector. However, it is unclear a priori how many iterations this takes, as this number of iterations appears to be highly problem dependent.

We therefore conclude that it is possible to tackle the POP-following problem in practice to execute Pareto-optimal deterministic non-stationary policies in stochastic MOMDPs. Because of the favourable trade-off between runtime and quality, we would recommend using POP neural networks for Pareto-optimal policy execution.

For this paper, we have looked at a tabular setting, and used a planning algorithm, i.e., White's Pareto Value Iteration algorithm [18], to retrieve the Pareto coverage sets, while assuming that the transition and reward functions are given. We note that our approach also applies if the Pareto coverage sets result from reinforcement learning rather than planning [6], and the transition and reward functions are learned from interaction as well. However, we did not yet test this experimentally. We aim to investigate the reinforcement learning approach in future work. Furthermore we hope to extend our approach to settings in which the number of subsequent states is unknown, and with high-dimensional state signals, i.e., deep multi-objective RL [7].

Finally, we would like to note that we have studied POP-following for value vectors in coverage sets learnt for the scalarised expected returns (SER) optimality criterion [12]. Recently, coverage sets have also been proposed for the expected scalarised returns (ESR) optimality criterion [3], which offers an interesting line of future research for policy-following as well.

ACKNOWLEDGMENTS

We would like to express our sincere gratitude to Conor F. Hayes for his feedback on the paper. This research was supported by funding from the Flemish Government under the "Onderzoeksprogramma Artificiële Intelligentie (AI) Vlaanderen".

REFERENCES

[1] Robert T Clemen. 1996. *Making hard decisions: an introduction to decision analysis.* Brooks/Cole Publishing Company.

[2] Conor F Hayes, Roxana Rădulescu, Eugenio Bargiacchi, Johan Källström, Matthew Macfarlane, Mathieu Reymond, Timothy Verstraeten, Luisa M Zintgraf, Richard Dazeley, Fredrik Heintz, et al. 2021. A Practical Guide to Multi-Objective Reinforcement Learning and Planning. *arXiv preprint arXiv:2103.09568* (2021).

[3] Conor F Hayes, Timothy Verstraeten, Diederik M Roijers, Enda Howley, and Patrick Mannion. 2021. Dominance Criteria and Solution Sets for the Expected Scalarised Returns. In *Proceedings of the Adaptive and Learning Agents workshop at AAMAS*, Vol. 2021.

[4] MI Henig. 1980. Dynamic programming with returns in partially ordered sets. *Research Memorandum, University of British Columbia* (1980).

[5] Lawrence Mandow, JL de la Cruz, and N Pozas. 2020. Multi-objective dynamic programming with limited precision. *arXiv preprint arXiv:2009.08198* (2020).

[6] Kristof Van Moffaert and Ann Nowé. 2014. Multi-Objective Reinforcement Learning using Sets of Pareto Dominating Policies. *Journal of Machine Learning Research* 15, 107 (2014), 3663–3692. http://jmlr.org/papers/v15/vanmoffaert14a.html

[7] Hossam Mossalam, Yannis M Assael, Diederik M Roijers, and Shimon Whiteson. 2016. Multi-objective deep reinforcement learning. In *Proceedings of the Deep Reinforcement Learning workshop at NIPS'16*.

[8] Simone Parisi, Matteo Pirotta, and Jan Peters. 2017. Manifold-based multi-objective policy search with sample reuse. *Neurocomputing* 263 (2017), 3–14.

[9] Matteo Pirotta, Simone Parisi, and Marcello Restelli. 2015. Multi-objective reinforcement learning with continuous Pareto frontier approximation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 29.

[10] Diederik Marijn Roijers. 2016. *Multi-Objective Decision-Theoretic Planning.* Ph.D. Dissertation. University of Amsterdam.

[11] Diederik M Roijers, Denis Steckelmacher, and Ann Nowé. 2018. Multi-objective reinforcement learning for the expected utility of the return. In *Proceedings of the Adaptive and Learning Agents workshop at FAIM*, Vol. 2018.

[12] Diederik M Roijers, Peter Vamplew, Shimon Whiteson, and Richard Dazeley. 2013. A survey of multi-objective sequential decision-making. *Journal of Artificial Intelligence Research* 48 (2013), 67–113.

[13] Diederik M Roijers and Shimon Whiteson. 2017. Multi-objective decision making. *Synthesis Lectures on Artificial Intelligence and Machine Learning* 11, 1 (2017), 1–129.

[14] Roxana Rădulescu, Patrick Mannion, Diederik M. Roijers, and Ann Nowé. 2020. Multi-objective multi-agent decision making: a utility-based analysis and survey. *Autonomous Agents and Multi-Agent Systems* 34, 1 (April 2020), 10. https://doi.org/10.1007/s10458-019-09433-x

[15] R.S. Sutton and A.G. Barto. 1998. *Reinforcement Learning: An Introduction.* MIT Press, Cambridge, MA.

[16] Peter Vamplew, Richard Dazeley, Ewan Barker, and Andrei Kelarev. 2009. Constructing stochastic mixture policies for episodic multiobjective reinforcement learning tasks. In *Australasian joint conference on artificial intelligence*. Springer, 340–349.

[17] Peter Vamplew, Richard Dazeley, Cameron Foale, Sally Firmin, and Jane Mummery. 2018. Human-aligned artificial intelligence is a multiobjective problem. *Ethics and Information Technology* 20, 1 (2018), 27–40.

[18] DJ White. 1982. Multi-objective infinite-horizon discounted Markov decision processes. *Journal of mathematical analysis and applications* 89, 2 (1982), 639–647.

[19] D. J. White. 1978. *Finite Dynamic Programming: An Approach to Finite Markov Decision Processes*. John Wiley & Sons, Inc., USA.

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

[21] Marco A Wiering, Maikel Withagen, and Mădălina M Drugan. 2014. Model-based multi-objective reinforcement learning. In *2014 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)*. IEEE, 1–6.

[22] Luisa M Zintgraf, Timon V Kanters, Diederik M Roijers, Frans Oliehoek, and Philipp Beau. 2015. Quality assessment of MORL algorithms: A utility-based approach. In *Benelearn 2015: Proceedings of the 24th Annual Machine Learning Conference of Belgium and the Netherlands*.

[23] Luisa M Zintgraf, Diederik M Roijers, Sjoerd Linders, Catholijn M Jonker, and Ann Nowé. 2018. Ordered preference elicitation strategies for supporting multi-objective decision making. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*. International Foundation for Autonomous Agents and Multiagent Systems, 1477–1485.

[24] Eckart Zitzler, Joshua Knowles, and Lothar Thiele. 2008. Quality assessment of pareto set approximations. *Multiobjective optimization* (2008), 373–404.