Evaluating Tunable Agents with Non-Linear Utility Functions under Expected Scalarised Returns

Federico Malerba
Faculty of Science
University of Potsdam
malerbafede@gmail.com

Patrick Mannion
School of Computer Science
National University of Ireland Galway
patrick.mannion@nuigalway.ie

ABSTRACT

In many multi-objective decision making problems, the preferences of the user over outcomes are most naturally expressed using non-linear utility functions. However, most research to date has focused on simple linear utility functions. In order to enable future real-world applications of MODeM algorithms such as multi-objective reinforcement learning or planning, it is crucial that the behaviour of these algorithms when combined with non-linear utility functions is better understood. To address this gap in the literature, in this paper we evaluate the behaviour of a tunable Deep Q-Network agent with a wide range of non-linear utility functions under the expected scalarised returns criterion in a benchmark item gathering environment.

KEYWORDS

Multi-objective, reinforcement learning, non-linear utility

1 INTRODUCTION

The defining feature of tunable agents [6] is their ability to display a variety of different behaviours at runtime. One key possible application of this is for a real user to tell an agent what he/she values or is interested in; the agent would then solve whatever problem it has been trained for, while trying to maximize the user's utility. Non-linear utility functions can better approximate (in the general case) the user's utility function, and could even just be easier for a user to understand (e.g. assigning a weight of -1.5 to a car crash, versus setting a threshold of zero for it). While the problem of correctly estimating the user's utility is beyond the scope of this paper, the topic has received some attention to date. For example, in [17] an approach that uses Gaussian processes and pairwise comparisons is studied; techniques like these are almost certain to yield non-linear estimates of the user's utility function, so agents should therefore be trained to tackle such utilities.

Training and evaluating agents on non-linear utilities is a task that comes with several challenges. For example, establishing metrics to compare agents trained on different sets of utility functions is non-trivial; this is due to the fact that an agent's utility defines its own objective for which, in general, the optimal policy is not known. Different utility functions could also have the same common optimal policy, but it is difficult to determine when this is the case. In this work we describe the first experimental study that comprehensively compares the effect of several families of non-linear utility functions on the sets of policies learned by multi-objective reinforcement learning (MORL) [4] agents.

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.

2 BACKGROUND

2.1 Multi-Objective Reinforcement Learning

In traditional RL [5, 13], the environment provides the agent with a scalar reward at every time-step and the agent's learning process seeks to maximize the discounted expected reward over the entire episode. There are many real-world scenarios, however, where multiple different objectives need to be optimized at the same time. As an example, imagine a logistics problem where one might want to optimise for different quantities of interest such as cost, speed, reliability, carbon footprint, etc. These cases can be modelled under the framework of multi-objective reinforcement learning.

In MORL the environment that the agent interacts with provides a vector of rewards $\mathbf{r} = [r_1, \dots, r_n] \in \mathbb{R}^n$. Each entry refers to a specific objective and just like in traditional RL we can define the state value function $\mathbf{V}^\pi(s)$. Note that in this case $\mathbf{r}$ and $\mathbf{V}^\pi(s)$ are in bold as they denote n-dimensional vectors. This setting is extremely interesting from a game-theoretic standpoint and it introduces several new concepts such as Pareto optimality and various new types of equilibria, which will not be explored here[2][1][8][15]. The only important concept that will be useful within the scope of this work is that of *dominance*.

Definition 1. Let $\mathbf{r}_1, \mathbf{r}_2 \in \mathbb{R}^n$ be two reward vectors. $\mathbf{r}_1$ is said to *dominate* $\mathbf{r}_2$ iff

$$r_{1,i} \leq r_{2,i} \quad \forall\ 0 < i \leq n \tag{1}$$

Just like in traditional RL, the agent has the objective of maximizing the expected reward; however, the multi-dimensional nature of this reward makes the task no longer as "trivial" as before. The key problem is that the rewards are now in $\mathbb{R}^n$ which does not have a canonic order like $\mathbb{R}$, and the concept of *dominance* introduced can only serve as a partial ordering. To deal with this issue, a utility function can be used to select the preferred non-dominated value vector. A utility function u maps a value vector to a single scalar

$$\begin{aligned} u\colon \mathbb{R}^n &\to \mathbb{R} \\ \mathbf{V} &\mapsto u(\mathbf{V}) \end{aligned} \tag{2}$$

The problem can now be framed once again as one of optimization, where the agent seeks to find the optimal policy as determined by the combination of vectorized rewards with the specific utility function selected [10].

The utility function u can assume a multitude of different forms; typically it is assumed to be monotonically increasing [4][16], i.e.,

$$\mathbf{V} \leq \mathbf{V}' \implies u(\mathbf{V}) \leq (\mathbf{V}') \tag{3}$$

Where the partial order over the set $\mathbb{R}^n \ni \mathbf{V}$ is defined following Definition 1 by

$$\mathbf{V} \leq \mathbf{V}' \iff V_i \leq V_i' \quad \forall\, 0 < i \leq n \tag{4}$$

Linear utility functions, for example, are intuitively defined with a particular choice of weights $\mathbf{w} \in \mathbb{R}^n$ as

$$u(\mathbf{V}) = \sum_{i=1}^{n} w_i V_i \tag{5}$$

Other non-linear choices of utility functions are also possible, and they allow the exploration of many Pareto-optimal solutions which linear utility functions would never be able to optimize for. For example, it is well-known in the MORL literature that linear utility functions cannot be used to learn policies in concave regions of the Pareto front [14].

2.2 Learning in the MORL setting

Until now, the problem has been framed in terms of the state value function and the utility value obtained by mapping it via a utility function u. As will be now shown, this still does not address how to concretely update a DQN [7] model to improve its performance. The optimization problem in the traditional RL setting is framed in terms of minimizing the temporal difference error. Translating this to the MORL setting - where utility functions are also involved - is non-trivial and there are several factors to consider.

A first important problem is deciding whether to optimize for expected scalarized returns (ESR) or scalarized expected retuns (SER) [4]. Mathematically the two procedures are distinguished by where the utility function is applied with respect to the expectation of returns, i.e.

$$ESR = \mathbb{E}\left[u\left(\sum_{t=0}^{\infty} \gamma^t \mathbf{r}_t \right) \mid \pi, \mu_0 \right] \tag{6}$$

$$SER = u\left(\mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \mathbf{r}_t \mid \pi, \mu_0 \right] \right) \tag{7}$$

where π is the policy that is being followed by the agent, γ is the discount factor and μ_0 is the distribution over the initial states. The impact of this distinction and the considerations to be made on the usage of one over the other are beyond the scope of this work. For an exploration of the topic see e.g. [9, 11, 12]. One key observation to be made, however, is that in the case of linear utility functions the two expressions are equal. The experiments conducted in this paper follow the ESR approach.

Because non-linear utility functions in MOMDPs do not distribute across the sum of immediate and future returns [9], this invalidates the Bellman equation for most RL algorithms. To deal with this issue, it is necessary to augment the environment state with the cumulative reward vector (CRV) which, intuitively, is equal to the cumulative sum of the reward vectors provided by an environment E at all steps prior to a given one. During learning, the utility of the CRV can be calculated at each timestep and then the per-timestep delta in utility may be determined. This value is then stored in the replay buffer for our DQN agents and is used for gradient computations as discussed in Section 2.2.

In the remainder of this work utility functions will be used to map reward vectors, CRVs[1] or state-values $\mathbf{v}$ to scalar values. The three entities have the same dimensionality and live in the same space, thus no changes to the definition of a utility function are required to switch between the three usages.

2.3 Tunable Agents

The motivational idea behind training tunable agents is that the choice of a utility function is arbitrary, and different real-world agents could value objectives differently. Furthermore, in certain contexts it could be undesirable for a trained agent to exhibit static behaviour at runtime. Observing these problems, Källström and Heintz [6] proposed a new framework to train agents that are capable of learning many different policies that maximize for different utility functions; at runtime these agents are capable of changing their behaviour based on the utility function that is provided to them.

2.3.1 The training process. Instead of training an agent with a fixed utility, a set $\mathcal{U}$ of utility functions is constructed and the agent is trained with utility functions sampled from that set[2]. A simple example of what this set could look like, is the linear utilities case where

$$\mathcal{U} = \{u \colon \mathbb{R}^n \to \mathbb{R} \mid \exists\, \mathbf{w} \in \mathbb{R}^n \colon\ u(\mathbf{V}) = \mathbf{w} \cdot \mathbf{V}\} \tag{8}$$

Let P be a probability distribution defined over the set $\mathcal{U}$. During training at the beginning of every episode a sample $u_i \in \mathcal{U}$ is produced following the distribution P. The i-th episode is then simulated using the utility u_i to map the per-timestep reward vectors to per-timestep utilities that the agent can then learn to maximize as in the traditional RL setting.

Of course, the specific utility function u_i that is used at the i-th episode plays a crucial role in determining what the optimal policy to be followed should be at that episode. It is therefore useful to provide this function (or a representation of it) as input to the agent, so that it can best decide what the optimal actions to take are when trying to maximize the utility function u_i at the i-th episode.

2.3.2 Important notes. When comparing the work and terminology adopted in this paper with [6], there are a couple key differences that must be addressed.

To start with, [6] dealt only with linear utility functions which are uniquely identifiable by the weights used on each reward-triggering event. These weights are what they refer to as *preferences* though, in a more general context[3], the term is too restrictive and it is preferable to refer to these as a *utility function parametrization* (UFP). Given that the experiments conducted in this work make exclusive use of the DQN architecture, the UFP should be thought of as a vector that uniquely identifies a utility function from the set $\mathcal{U}$ on which the agent is trained. This vector representation is fed to the agent as part of its observations and the agent should learn

[1] Note that with, a slight abuse of notation, reward vectors and CRVs will be both denoted by $\mathbf{r}$; which one is being referred to at any given point will be clear from the context.

[2] It is important to note that an agent trained on a set $\mathcal{U}$, should only be provided with utility functions $u \in \mathcal{U}$ at runtime.

[3] i.e. with arbitrary utility functions.

to output Q-values that estimate the utility of the expected return for the given UFP and state observations.
It is also necessary to establish consistency in the usage of the term *reward*; in the traditional RL setting the reward fed by the environment is also stored in the replay buffer along with all the transitions and will be used to compute the gradients to update the agent. In MORL however, the term is used when talking about the vector fed to the agent by the environment which is different from what gets used in the gradient computations. In accordance to this, within the scope of this work (unless otherwise specified) *reward* will refer to the vector of rewards for each objective, whilst the term *utility* will be used when talking about the scalar value that is used in gradient computations and that the learning algorithm actually seeks to maximize.

3 EXPERIMENTAL DESIGN

This work follows the framework (and its DQN implementation) proposed by [6]. The objective is to extend this to families of non-linear utility functions ranging from threshold utilities to a relatively new family of utility functions that has recently been defined in [3]. The code used for running the environment, training the agent and evaluating can all be found at a public GitHub repository[4].

3.1 Gathering environment

The gathering environment shown in Figure 1 used for our experiments is very similar to the one provided by [6], with some minor tweaks to adjust undesirable behaviour; specifically the end of episode criterion was updated to reflect what it would have to be like for the non-linear utilities used.

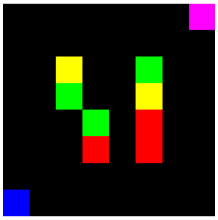

Figure 1: Example initial state of the gathering environment.

The environment consists of an 8x8 grid containing 8 items to be collected; these items are randomly positioned at the start of every episode within the 4x4 sub-grid located at the center. While the location of items is random, their numbers are fixed; there are always 3 red items, 3 green items and 2 yellow items. The colours for these items are meaningless and serve only as a placeholder for different objectives that an agent could be interested in.
An episode in this environment ends after 30 steps or when no *valuable* items remain in it. An item is valuable if there is a positive utility to be gained in taking it. Note that this end-episode condition is utility dependant, so an agent might terminate or not a certain episode depending on what utility function has been sampled for that episode.

[4]https://github.com/FMalerba/tunable-agents-MORL

3.1.1 Agents and movement. The agent to be trained is represented by the blue square and always starts in the bottom left corner, whilst a second agent always starts at the opposite corner. This second agent is a deterministic one; it is not trained and implements a very simple heuristic policy. The deterministic agent is only interested in the red items, it will determine which is the closest to itself and will then proceed to adjust the x-coordinate and then the y-coordinate in order to reach it. Once all red items have been taken, the deterministic agent will stay still in whatever position it happens to be at that timestep and wait the end of the episode.
The players can move in the canonical 4 directions within the grid or they can take no action and stay still. Attempting to move beyond the boundary of the grid will result in no change in position for the agent and a wall penalty will be given. Whenever a player enters the cell occupied by an item it will consume that item, even if it is not interested in it. Both agents can be in the same cell at the same time and will not interact in any way with one another. In cases where both agents enter the same cell containing an item, the deterministic agent "arrives first" and consumes the item.

There are 6 different objectives which the environment keeps track of: time and wall penalties, item collection for the agent (differentiated by colour) and a last entry for the other agent taking a red item specifically. The environment keeps track of these objectives in a CRV $\mathbf{r} \in \mathbb{N}^6$. As discussed in Section 2.2, at each time-step the CRV is mapped via the utility function to obtain a scalar utility; the difference between this value and the equivalent one obtained at the previous time-step is stored in the replay buffer as the time-step scalar reward to be used for gradient computations.
The environment can feed up to three different observations to the agent. Firstly, a view of the status of the gridworld is provided; this is fed to the agent as an RGB image not dissimilar by the one in Figure 1. Following [6], three such images - corresponding to the current time-step and the previous two - are stacked together and fed to the CNN component of the agent's model. For the initial time-steps, the undefined frames are set to 0 on all pixels (black image).
As discussed in Section 2.3, the agent also receives a UFP as a vector. The dimensionality of this vector varies depending on the utility function set that is used to train the specific agent. Finally, the agent may or may not be provided with the cumulative rewards vector depending on the experiment being conducted. As seen in Section 2.2, this should actually be required for non-linear utilities, but both approaches were tested for comparisons.

3.2 Agent model

Various different agent models were used across all of the experiments run; however, the core structure behind each one was identical. As previously mentioned, a DQN approach was followed, where a neural network outputs a Q-value for every action possible in the current state and the arg max of these Q-values is used as the chosen action for the time-step. Exploration is enforced with the use of an ϵ-greedy algorithm that forces the agent to take a random action with probability $1 - \epsilon$.
The general structure of the NN follows that used in [6]: the three RGB frames are fed as input to a convolutional neural network composed of two convolutional layers with a 3x3 kernel and 256

filters each. The features that are obtained are then flattened and concatenated with the UFP and (optionally, depending on the specific experiment) with the CRV; this long vector of inputs is then passed to a fully-connected neural network with varying number of layers and nodes per layer. The ReLU activation function is used to add non-linearity at all layers except the last one which is simply a 5-dimensional, linear, fully-connected output[5]. A graphical representation of this general structure is provided in Figure 2.

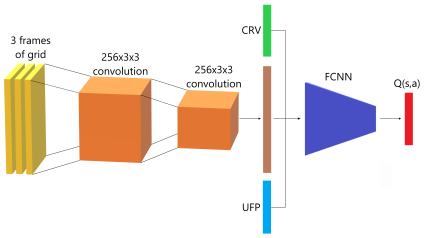

Figure 2: Architecture for tunable agents in this work.

Four different agent models were used for training and evaluation in all the experiments. The only difference between them was the number of layers and nodes per layer in the FCNN part of the model. Within the scope of this work, they will be identified with the following shortened names:

- *Tiny*: identifying a model with two hidden layers with 64 nodes each.
- *Small*: identifying a model with three hidden layers with 128, 128 and 64 nodes respectively.
- *Medium*: identifying a model with five hidden layers with 256, 128, 128, 64 and 64 nodes respectively.
- *Large*: identifying a model with six hidden layers with 512, 256, 256, 128, 128 and 64 nodes respectively.

3.3 Utility function sets

Following the work done in [6], the first set of utility functions used was

$$\mathcal{U}_L = \left\{ u\colon \mathbb{R}^6 \to \mathbb{R} \;\middle|\; \begin{array}{l} \exists\, \mathbf{w} \in [-20, 20]^6\colon\ u(\mathbf{r}) = \mathbf{w}\cdot\mathbf{r}, \\ w_1 = -1 \wedge w_2 = -5 \end{array} \right\} \tag{9}$$

This is what will be referred to as the set of linear utility functions. During training, the utility functions in $\mathcal{U}_L$ were sampled with a discrete uniform probability distribution with step 5; i.e. the weights could only take values in $\{-20, -15, -10, -5, 0, 5, 10, 15, 20\}$. Note that the first two weights have a fixed value of -1 and -5; they respectively indicate the penalty for time and hitting a wall.
A second set of utility functions is that of threshold utilities; formally it is defined as

$$\mathcal{U}_{Th} = \left\{ u\colon \mathbb{R}^6 \to \mathbb{R} \;\middle|\; \begin{array}{l} \exists\, \mathbf{w} \in [-20, 20]^6, \mathbf{t} \in \mathbb{N}^6_{\leq 3}\colon \\ u(\mathbf{r}) = \sum_{i=1}^{n} (w_i r_i \cdot \mathbb{1}_{r_i \geq t_i}), \\ w_1 = -1 \wedge w_2 = -5 \wedge t_1 = t_2 = 0 \end{array} \right\} \tag{10}$$

[5]The dimensionality of the output must match the number of available actions for a DQN model.

Once again, during training the weights were sampled with a discrete probability distributions with step 5, whilst the thresholds were sampled discretely in $\mathbb{N}_{\leq 3}$. Note that the weights for time and wall penalties were fixed as before and their respective thresholds were also fixed at 0 (meaning there would always be a penalty). It is important to highlight that $\mathcal{U}_L \subset \mathcal{U}_{Th}$ since taking all thresholds to be 0 yields a utility function u that is mathematically equivalent to a linear function; the reverse does not hold.
An intuitive variation on threshold utilities is that of their corresponding dual threshold utility function. These functions are conceptually very similar to threshold utilities with the exception of increasing prior to reaching the threshold and then remaining constant. Formally, their set is defined as

$$\mathcal{U}_{DTh} = \left\{ u\colon \mathbb{R}^6 \to \mathbb{R} \;\middle|\; \begin{array}{l} \exists\, \mathbf{w} \in [-20, 20]^6, \mathbf{t} \in \mathbb{N}^6_{\leq 31}\colon \\ u(\mathbf{r}) = \sum_{i=1}^{n} (w_i r_i \cdot \mathbb{1}_{r_i \leq t_i} + w_i t_i \cdot \mathbb{1}_{r_i > t_i}), \\ w_1 = -1 \wedge w_2 = -5 \wedge t_1 = t_2 = 31 \end{array} \right\} \tag{11}$$

Sampling within $\mathcal{U}_{DTh}$ is done identically to $\mathcal{U}_{Th}$, though the dual threshold for time and wall penalties are now set to 31 (to ensure that a penalty is always applied). Again, it should be noted that $\mathcal{U}_L \subset \mathcal{U}_{DTh}$ by setting all dual thresholds to a sufficiently high value (in this setting $t_i \geq 31 \quad \forall\, i$ works).
The last type of utility functions used in this work was first introduced in [3], and will be referred to within the scope of this work as *target utilities*. Their set is defined as

$$\mathcal{U}_{Ta} = \left\{ u\colon \mathbb{R}^6 \to \mathbb{R} \;\middle|\; \begin{array}{l} \exists\, \mathbf{r}_\dagger \in \mathbb{N}^6_{\leq 31}\colon \\ u(\mathbf{r}) = \arg\max_{c \in \mathbb{R}}\colon \mathbf{r} - c\dfrac{\mathbf{r}_\dagger}{|\mathbf{r}_\dagger|} \geq 0, \\ r_{\dagger 1} = r_{\dagger 2} = 31 \end{array} \right\} \tag{12}$$

The vector $\mathbf{r}_\dagger$ is referred to as the target vector, though it is important to point out that the scalar utility value is not maximized when the target is reached, but rather increases continuously along the target's direction. Also note that for this type of utility function to work properly all entries must be increasing and non-negative; to this aim, the first two entries of the CRV fed are multiplied by -1 and 31 is then added to them[6]. Sampling was done with a uniform discrete distribution in in $\mathbb{N}_{\leq 3}$ for the non-fixed entries of the target.
In all cases, attention was paid to avoid the marginal cases in which utility functions with no possible positive feedback would be sampled. As explained in Section 3.1, the condition for terminating an episode is utility-dependent and all-negative utility functions would simply result in the immediate termination of the episode.

3.4 Evaluation

When training several agents on entirely different sets of utility functions the problem of evaluating and comparing them is non-trivial. The difficulties in carrying out this task arise from several different considerations, and there is no current literature (to the authors' knowledge) that comprehensively compares the effects of learning under a range of different non-linear utility functions.

[6]Their maximum negative value is 31, hence adding 31 to ensure they are always non-negative.

A first important fact to realize is that utility functions are not simply some other information given to the agent to influence its behaviour; the utility function effectively decides what *success* means for the agent. Therefore when training an agent on different utility functions, one is actually training it on different games with different goals. Comparing tunable agents which have been trained on different sets of utility functions is yet more arduous as these agents may well have been trained to pursue different sets of objectives, and none of these is in absolute terms superior to the others.

Furthermore, even if one was to evaluate an agent on a sufficiently large number of episodes with utilities sampled from a set $\mathcal{U}$ with probability distribution P, the empirical distribution $U_{\mathcal{U},P}$ of end-of-episode scalar utility values obtained would be dependent on $\mathcal{U}$ and P. Consequently, any possible statistic T computed on $U_{\mathcal{U},P}$ (e.g. mean or standard error) would also be dependent on them, and would be fundamentally incomparable with statistics derived from any other empirical distribution $U'_{\mathcal{U}',P'}$ where $\mathcal{U} \neq \mathcal{U}' \ \vee \ P \neq P'$. Two simple examples of this issue are provided below.

Example 1. Let $\mathcal{U}$ be defined similarly to Equation 8 as

$$\mathcal{U} = \{u \colon \mathbb{R}^n \to \mathbb{R} \mid \exists\, \mathbf{w} \in [-20, 20]^n \colon \ u(\mathbf{V}) = \mathbf{w} \cdot \mathbf{V}\}$$

Where the weights have been restricted to all being in the range $[-20, 20]$. Let $\mathcal{U}'$ similarly be

$$\mathcal{U}' = \{u \colon \mathbb{R}^n \to \mathbb{R} \mid \exists\, \mathbf{w} \in [-40, 40]^n \colon \ u(\mathbf{V}) = \mathbf{w} \cdot \mathbf{V}\}$$

And let P and P' be the uniform distributions over the two ranges of $\mathcal{U}$ and $\mathcal{U}'$. Consider a fixed policy[7] π that takes a utility u as part of its observations of the environment E and behaves so as to maximize u. Sampling N episodes from E following policy π with utilities sampled from $\mathcal{U}$ and $\mathcal{U}'$ according to P and P', one obtains two empirical distributions U and U' of end-of-episode scalar utility. Given the definitions set above one would expect the empirical means over U and U' to be

$$\frac{1}{N}\sum_{x \in U} x \simeq \frac{1}{2}\left(\frac{1}{N}\sum_{x \in U'} x\right)$$

Example 2. Let $\mathcal{U}$ be defined as

$$\mathcal{U} = \{u \colon \mathbb{R}^n \to \mathbb{R} \mid \exists\, \mathbf{w} \in [-20, 0)^n \colon \ u(\mathbf{V}) = \mathbf{w} \cdot \mathbf{V}\}$$

Where the weights have been restricted to all being in the range $[-20, 0)$. Let $\mathcal{U}'$ similarly be

$$\mathcal{U}' = \{u \colon \mathbb{R}^n \to \mathbb{R} \mid \exists\, \mathbf{w} \in (0, 20]^n \colon \ u(\mathbf{V}) = \mathbf{w} \cdot \mathbf{V}\}$$

And let P and P' be the uniform distributions over the two ranges of $\mathcal{U}$ and $\mathcal{U}'$. Following the same procedure as in the previous example the empirical means over U and U' are going to be

$$\frac{1}{N}\sum_{x \in U} x < 0 < \frac{1}{N}\sum_{x \in U'} x$$

In both of the examples above, it is important to observe that the same policy π is expected to yield different average utility values even though the behaviour could be unchanged between the two utility function sets $\mathcal{U}$ and $\mathcal{U}'$.

[7] This would be a trained tunable agent that operates using a greedy policy in our setting.

To complicate matters further, evaluating a tunable agent trained on $(\mathcal{U}, P)$ by feeding it approximations of utility functions coming from $(\mathcal{U}', P')$ cannot really be done in a rigorous way.

3.4.1 Metrics. In light of the issues presented above, two distinct methods to meaningfully evaluate agents and compare their behaviours were chosen. To start with, simple results for average end-of-episode utility values are shown; these present the aforementioned problems and are mostly useful only for evaluations between agents that are tested on the same tuple $(\mathcal{U}, P)$. As observed in Section 3.3, $\mathcal{U}_L \subset \mathcal{U}_{Th}$ and it is therefore possible to feed linear utilities to agents trained on threshold utilities so these results are also presented and discussed.

The second type of results presented looks instead at the end-of-episode CRVs that the agent obtains. Given that the object of study are tunable agents whose objective is to learn maximizing policies for many different utility functions, it makes sense to analyze the diversity of reward vectors that they are capable of achieving.

Of course, sheer diversity is not a necessarily desirable feature thus the partial ordering introduced in Definition 1 is used to measure the number of non-dominated reward vectors. These vectors are of interest because of the utility functions' monotonicity property defined in Equation 3; this implies that non-dominated vectors must provide non-dominated utility values for some utility function u. This metric is therefore relevant by representing the ability of an agent to not only display varied behaviour, but also behaviour that the agent itself cannot improve upon. Most importantly, sets of non-dominated vectors can also be compared between agents that have been trained and/or evaluated on different $(\mathcal{U}, P)$ tuples, since domination of vectors implies domination of utility values regardless of the utility function being used. The comparisons between sets of non-dominated vectors coming from different agents, informs on how much of the intra-agent-optimal behaviour that is learnt, is also inter-agent-optimal.

In the coming sections, when using the term non-dominated CRVs it is implied that these are non-dominated only at the agent-level; i.e. the agent that produced them has not shown to be able to improve upon them. Considerations and results are also presented regarding *combined non-dominated CRVs* which are instead non-dominated among the entirety of behaviour displayed by all the agents.

3.4.2 Fixed environment. A key observation to be mindful of is that the end-of-episode reward vector that an agent is able to achieve, does not depend solely on the agent's choices, but it is also affected by the specific random initialization of the environment. As explained in Section 3.1, the coloured items that the agent seeks are randomly distributed in the center of the grid. The presence of the deterministic agent makes it so that in certain cases some items might be unobtainable no matter what actions the DQN agent takes. To control this variability, a fixed environment was constructed (shown in Figure 3) and agents were evaluated with the aforementioned reward vector metrics on it. In selecting this environment, attention was paid to create one where the agent has complete

control over whether to take any item or not; with the correct behaviour, the agent can choose to take all the items, only some or it can also choose to leave some red items to the deterministic agent[8].

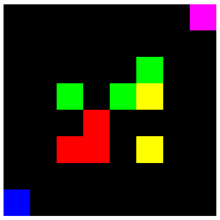

Figure 3: Image representation of the fixed environment.

It is important to highlight two key features of this evaluation. Firstly, the samples obtained in the fixed environment vary only by agent being evaluated and utility function provided at the beginning of the episode. Secondly, the utility functions sampled are fixed for each set; i.e. a fixed discrete subset of utility functions was sampled from the different sets of utility functions described in Section 3.3. All agents being evaluated on the fixed environment with a given utility function set, would therefore be exposed to the same utility functions. Note that this does not mean that all agents were evaluated on the same utility function subsets; this is because certain agents could not be shown utility functions belonging to other sets[9].
A problem that arises specifically with agents trained with $\mathcal{U}_{Ta}$; because of the nature of these utility functions, there are substantially fewer functions that can be sampled. When sampling random environments this is not a hindrance to obtaining a varied dataset, since the randomness in the environment can offset this issue. However, on a fixed environment and operating with the same greedy policy, there are very few samples that can be taken from an agent. Therefore, target agents had about 1000 times less episodes sampled for them on the fixed environment. Their results are still presented, but they aren't really comparable to the agents evaluated on the other sets because of this huge discrepancy.
Evaluating on a single fixed environment has definitely some drawbacks because it unfairly puts much weight on this specific environment, but - given the large number of experiments - it would have been too high a computational burden to sample significantly more fixed environments and multitudes of utilities for each one. Agents are however not specially trained for this environment so they all start from a common ground on this point.

4 EXPERIMENTAL RESULTS

It is important to distinguish between training and evaluation in the results that will be presented in this section. For training, three hyper-parameters with discrete values were taken:

- CRV flag: taking values `true`/`false` and determining whether the agent to be trained would receive the CRV as input. If `true`, the CRV would be passed to the agent as shown in Figure 2; if `false`, it would simply be excluded from the concatenation occurring prior to the FCNN.
- Agent model: the four models that were used are described in Section 3.2.
- Utility function set: the set of utility functions (and corresponding probability distribution P) with which the agent was trained. They are described in greater detail in Section 3.3.

The Cartesian product of these three hyper-parameters was constructed and experiments for every element of it were run six different times; i.e. six different agents would be initialized and trained for every choice of model, utility function set, and CRV flag.
After having trained all these different agents, each one was tested using its greedy policy in two different ways:

- Sampling $100k$ random episodes and storing the end-of-episode utility value.
- Sampling $\sim 180k$ episodes from the fixed environment (see Section 3.4.2) and storing the end-of-episode CRV[10].

The results of these testing procedures where then used to compute the metrics discussed in Section 3.4.1. Averages and standard errors were computed over the six different agents that were trained on all configurations. The tables to follow will denote the experiments with dash-separated abbreviations; *CR* is used to identify experiments where the CRV was fed as input to the agent, whilst the other abbreviations denote the utility set with which the agent was trained[11]. Furthermore, as discussed in Section 3.3, agents trained on $\mathcal{U}_{Th}$ and $\mathcal{U}_{DTh}$ can be fed functions that are mathematically equivalent to functions in $\mathcal{U}_L$; there are therefore evaluations where the results of this are shown and are denoted by having "L-" prepended to the utility set on which the agent was trained [12].
In the coming sections, the expressions "linear agents", "threshold agents", etc. will be used as shorthand to refer to agents that were trained on those utility function sets.

4.1 Average utility results

The results for average end-of-episode utilities across models are shown in Table 1. For the reasons discussed in Section 3.4, the results obtained for different sets of utilities should not be compared.
Larger models improve performance over smaller ones for linear agents, but the effect is mixed or opposite for non-linear ones. This is due to the fact that with larger models agents seem to get stuck in sub-optimal policies and go through a much noisier training process. This is a problem that in rare cases plagues linear agents too, but for non-linear agents becomes more pronounced. Threshold agents in particular do not seem to be as affected by this as dual threshold and target agents; this is likely due to the fact that threshold utilities in this environment should lead the agent to behave very similarly to the linear case. Dual threshold and target utilities are instead more radically different and this could lead to increased complexity in figuring out what behaviour is optimal. It

[8] Remember that as stated in Section 3.1.1, the deterministic agent is only interested in red items.
[9] More detailed discussions of this can be found in Sections 3.3 and 3.4
[10] As noted in Section 3.4.2, agents trained on target utilities are actually evaluated on a much lower number of episodes and their fixed environment results are thus not comparable with the other agents.
[11] They follow the naming schemes in the subscript of $\mathcal{U}$ in section 3.3.
[12] e.g. L-Th identifies an agent trained on threshold utilities that was evaluated on linear ones.

Model Setting	Tiny	Small	Medium	Large
L	12.8 (±0.3)	15.8 (±0.2)	20.9 (±0.3)	24.3 (±0.1)
CR-L	13.2 (±0.4)	16.3 (±0.3)	19.9 (±0.2)	22.1 (±0.2)
Th	4.9 (±1.3)	9.6 (±0.6)	11.3 (±0.5)	12.9 (±0.4)
CR-Th	9.0 (±0.6)	9.9 (±0.4)	10.8 (±0.6)	11.6 (±0.6)
DTh	6.8 (±0.4)	8.2 (±0.2)	9.6 (±0.3)	4.4 (±4.6)
CR-DTh	7.3 (±0.3)	8.8 (±0.3)	8.1 (±0.4)	8.9 (±0.5)
Ta	16.9 (±0.7)	17.4 (±0.2)	13.2 (±1.0)	0.5 (±0.0)
CR-Ta	17.1 (±0.4)	16.8 (±0.3)	10.7 (±1.9)	4.4 (±1.3)
L-Th	8.5 (±0.9)	11.2 (±0.4)	12.6 (±0.6)	13.9 (±0.8)
CR-L-Th	11.6 (±0.5)	12.3 (±0.5)	13.3 (±0.4)	13.9 (±0.9)
L-DTh	9.3 (±0.7)	11.7 (±0.4)	14.4 (±0.2)	7.3 (±5.5)
CR-L-DTh	10.6 (±0.5)	12.9 (±0.7)	11.5 (±0.5)	12.0 (±1.0)

Table 1: Average end-of-episode utility results over randomly initialised environments.

is likely that spending more time in testing different training hyperparameters (duration, learning rate, regularization, etc.) could solve this issue; the asymmetry between linear and non-linear agents is, however, indicative of the increased difficulty of solving this task even in a relatively simple environment.

It is also worth noting that threshold and dual threshold agents are not competitive with linear agents when evaluated on linear utilities. This is to be expected since linear agents could specialize on those utilities during training, however the discrepancy grows substantially for larger models. Non-linear agents thus seem only to be capable of finding some general policies to apply for linear utilities, and fail to improve and differentiate it even when given more complexity.

Providing the CRV as input to the agent has mixed results; sometimes it affects performance slightly whilst in other cases it has a larger impact. For linear agents, a minor impact is expected since the CRV is not really necessary; it is thus just a further input that needs to be processed by the model. For the smallest model it has a positive impact on threshold agents. It is possible that larger models find ways to make up for this lack of information which is not accessible to the smaller model. Given that the initial number of each item is an invariant for all episodes, one possible way that an agent might compensate the lack of CRV information is to estimate who (either itself or the deterministic agent) took each item that is missing at the current time-step. The CRV input also has a moderate impact on helping the larger models for dual threshold and target agents; providing it seems to mildly aid their training, and they more often manage to not get stuck in sub-optimal policies.

4.2 Fixed environment results

Table 2 shows the number of unique end-of-episode CRVs that agents were able to produce; this is just a metric for dispersion of the models' behaviour since uniqueness of policies is not generally desirable in and of itself. The number of these unique vectors that are non-dominated among the displayed behaviour is displayed in Table 3.

Table 2 shows data that is very noisy in general. Non-linear utilities (with the already discussed exception of the target utility) give rise to a significantly higher number of unique policies, particularly so for larger models. Table 3 though shows that this high variability does not translate to much larger numbers of non-dominated CRVs.

Model Setting	Tiny	Small	Medium	Large
L	155.0 (±12.8)	219.7 (±46.7)	259.0 (±36.9)	254.0 (±17.3)
CR-L	216.0 (±18.2)	241.5 (±21.1)	291.0 (±17.4)	343.7 (±24.1)
Th	147.0 (±16.2)	271.3 (±45.1)	450.2 (±52.7)	501.7 (±45.9)
CR-Th	186.2 (±26.3)	226.3 (±17.6)	406.8 (±34.2)	465.2 (±28.6)
DTh	277.2 (±19.3)	404.0 (±81.1)	378.0 (±47.3)	270.3 (±52.4)
CR-DTh	334.7 (±31.9)	402.3 (±18.8)	367.8 (±29.1)	456.7 (±44.9)
Ta	22.2 (±4.7)	20.3 (±2.9)	28.8 (±3.8)	15.2 (±3.8)
CR-Ta	28.7 (±4.5)	20.5 (±2.2)	23.2 (±3.6)	24.0 (±5.4)
L-Th	116.8 (±14.4)	221.5 (±35.3)	314.2 (±29.8)	378.2 (±36.9)
CR-L-Th	144.8 (±22.8)	190.7 (±12.3)	289.0 (±23.3)	358.0 (±19.4)
L-DTh	152.2 (±16.7)	169.0 (±8.9)	212.7 (±31.7)	174.0 (±36.5)
CR-L-DTh	171.5 (±22.5)	200.0 (±12.8)	249.3 (±16.6)	296.3 (±21.3)

Table 2: Number of unique end-of-episode CRVs in the fixed environment.

Model Setting	Tiny	Small	Medium	Large
L	14.8 (±0.7)	17.3 (±1.3)	16.7 (±0.5)	18.2 (±0.6)
CR-L	16.0 (±0.5)	16.5 (±0.7)	18.8 (±0.8)	20.5 (±1.3)
Th	13.5 (±0.9)	16.5 (±0.5)	20.5 (±1.3)	20.2 (±0.8)
CR-Th	16.3 (±0.8)	15.0 (±1.0)	18.0 (±1.1)	20.3 (±0.6)
DTh	23.5 (±1.3)	27.3 (±1.7)	25.2 (±1.6)	20.3 (±3.5)
CR-DTh	25.0 (±1.4)	24.0 (±1.4)	23.2 (±1.0)	25.7 (±1.1)
Ta	8.8 (±1.4)	9.0 (±0.8)	9.7 (±0.7)	3.0 (±0.5)
CR-Ta	10.3 (±1.6)	8.5 (±0.8)	7.8 (±0.8)	6.3 (±1.5)
L-Th	13.5 (±1.1)	16.8 (±0.8)	18.5 (±0.9)	18.0 (±0.7)
CR-L-Th	13.8 (±0.4)	15.5 (±0.8)	16.2 (±1.1)	19.8 (±0.8)
L-DTh	14.8 (±0.6)	16.8 (±1.0)	18.3 (±1.4)	14.5 (±2.3)
CR-L-DTh	14.0 (±1.1)	15.5 (±0.9)	14.3 (±0.9)	18.5 (±0.5)

Table 3: Number of unique non-dominated end-of-episode CRVs in the fixed environment.

The data in this table is quite noisy too, but it can be seen that dual threshold agents in particular do manage to reach a larger number of non-dominated CRVs (when they don't under-perform because of getting stuck in sub-optimal policies). This is probably due to how the end-of-episode condition is determined; linear and threshold agents either have to collect all items of a certain colour (if they have a positive weight for that colour) or none (if the weight is non-positive) in order to end the episode. Dual threshold agents can instead only take *some* of the items of a given colour and save on time, which unlocks new possible end-of-episode reward vectors. This feature is also there for target agents though their sampling problem[13] does not allow it to emerge from the data in this table.

The percentage of times that the agent reached a combined non-dominated end-of-episode CRV is given in Table 4. The numbers are quite low across the board, but dual threshold and target agents and definitely stand out with significantly higher percentages. Table 5 shows however that their results are mixed; while they do reach often combined non-dominated CRVs, they often reach the wrong one for the utility function that they are operating under. Linear and threshold agents instead make less of these mistakes when they reach non-dominated CRVs.

Finally, the percentage coverage of combined non-dominated CRVs for each agent is given in Table 6. Following the results shown in the previous tables, dual threshold and target agents generally cover a wider range of these, particularly for the intermediate model sizes.

[13] Discussed in Section 3.4.2

Model Setting	Tiny	Small	Medium	Large
L	11.3 (±3.0)	8.8 (±1.0)	13.5 (±1.8)	18.0 (±1.1)
CR-L	10.2 (±2.2)	8.4 (±0.5)	13.7 (±2.6)	12.9 (±0.9)
Th	11.4 (±3.9)	12.8 (±2.9)	10.8 (±2.3)	12.4 (±0.8)
CR-Th	8.3 (±1.9)	9.7 (±1.5)	13.1 (±2.2)	12.3 (±1.4)
DTh	17.3 (±3.6)	12.6 (±3.1)	30.6 (±2.4)	28.0 (±6.5)
CR-DTh	24.0 (±2.8)	19.1 (±3.0)	31.6 (±2.2)	22.7 (±1.9)
Ta	16.5 (±9.1)	21.0 (±7.6)	28.0 (±4.6)	0.0 (±0.0)
CR-Ta	11.5 (±3.9)	22.3 (±8.6)	25.7 (±7.5)	10.4 (±4.3)
L-Th	17.6 (±5.9)	10.9 (±2.0)	9.4 (±2.1)	12.0 (±1.8)
CR-L-Th	10.5 (±2.5)	9.7 (±1.6)	11.3 (±1.7)	11.1 (±2.0)
L-DTh	5.4 (±1.7)	6.8 (±3.8)	13.1 (±2.2)	11.8 (±2.6)
CR-L-DTh	11.5 (±4.4)	8.6 (±2.9)	17.3 (±3.2)	13.0 (±2.6)

Table 4: Percentage of episodes where a combined non-dominated end-of-episode CRVs was achieved in the fixed environment.

Model Setting	Tiny	Small	Medium	Large
L	34.5 (±7.8)	58.2 (±7.4)	62.4 (±4.3)	64.2 (±4.1)
CR-L	40.5 (±3.7)	67.4 (±4.8)	60.3 (±7.6)	64.3 (±5.7)
Th	23.0 (±7.5)	50.7 (±9.3)	56.8 (±7.0)	63.7 (±4.0)
CR-Th	36.3 (±7.3)	39.5 (±6.6)	56.4 (±7.0)	57.6 (±3.6)
DTh	36.7 (±2.3)	38.7 (±3.8)	38.7 (±2.8)	33.4 (±2.5)
CR-DTh	35.0 (±2.7)	44.5 (±3.7)	35.4 (±2.3)	32.7 (±3.1)
Ta	14.3 (±5.4)	11.0 (±4.0)	22.3 (±4.9)	0.0 (±0.0)
CR-Ta	10.3 (±2.9)	9.3 (±4.3)	16.4 (±4.7)	10.4 (±3.8)
L-Th	24.0 (±8.8)	46.2 (±11.3)	47.6 (±7.1)	41.2 (±6.7)
CR-L-Th	26.9 (±4.3)	44.8 (±7.2)	54.0 (±8.1)	49.5 (±5.8)
L-DTh	26.0 (±10.5)	42.9 (±11.1)	32.2 (±5.1)	28.1 (±7.7)
CR-L-DTh	25.1 (±8.6)	41.4 (±6.2)	22.4 (±4.8)	24.3 (±3.5)

Table 5: Percentage of episodes (among those with a non-dominated CRV) where the correct combined non-dominated end-of-episode CRV is reached in the fixed environment.

Model Setting	Tiny	Small	Medium	Large
L	30.4 (±4.5)	25.5 (±3.2)	38.2 (±5.8)	61.1 (±6.2)
CR-L	25.5 (±3.7)	24.7 (±5.0)	45.3 (±5.6)	43.9 (±7.7)
Th	30.8 (±3.5)	25.2 (±3.7)	23.9 (±4.2)	42.3 (±2.9)
CR-Th	27.3 (±4.9)	23.3 (±4.0)	35.4 (±3.1)	39.8 (±6.3)
DTh	29.6 (±4.5)	24.2 (±3.7)	51.5 (±2.2)	56.5 (±11.5)
CR-DTh	33.6 (±3.4)	33.4 (±2.8)	53.6 (±1.9)	59.8 (±3.8)
Ta	43.6 (±11.9)	52.0 (±10.6)	61.1 (±5.7)	0.0 (±0.0)
CR-Ta	40.2 (±7.4)	51.0 (±13.2)	64.8 (±10.4)	44.0 (±15.8)
L-Th	33.8 (±4.8)	20.8 (±3.6)	20.1 (±3.9)	37.5 (±2.5)
CR-L-Th	33.0 (±6.7)	20.1 (±2.9)	33.5 (±4.9)	30.2 (±5.4)
L-DTh	17.4 (±2.9)	14.9 (±4.5)	40.1 (±1.9)	54.8 (±10.6)
CR-L-DTh	26.3 (±2.6)	22.8 (±4.9)	47.8 (±1.3)	54.0 (±4.2)

Table 6: Percentage of the non-dominated vectors from Table 7 that were found by the agent.

For reference, the combined non-dominated CRVs that were found across all agents in the fixed environment are listed in Table 7.

[5, 0, 0, 2, 0, 0]	[10, 0, 1, 0, 2, 3]	[12, 0, 3, 2, 1, 1]
[5, 0, 0, 2, 0, 0]	[10, 0, 1, 1, 2, 2]	[12, 0, 3, 3, 1, 0]
[7, 0, 1, 1, 0, 0]	[10, 0, 2, 2, 1, 1]	[13, 0, 2, 2, 2, 1]
[8, 0, 0, 2, 1, 1]	[10, 0, 2, 3, 1, 0]	[13, 0, 2, 3, 2, 0]
[8, 0, 1, 2, 0, 1]	[10, 0, 3, 0, 1, 3]	[13, 0, 3, 0, 2, 3]
[9, 0, 0, 0, 2, 2]	[10, 0, 3, 1, 1, 2]	[13, 0, 3, 1, 2, 2]
[9, 0, 1, 2, 1, 1]	[11, 0, 1, 2, 2, 1]	[14, 0, 3, 2, 2, 1]
[9, 0, 1, 2, 1, 1]	[11, 0, 1, 3, 2, 0]	[15, 0, 3, 3, 2, 0]
[9, 0, 1, 3, 1, 0]	[11, 0, 2, 0, 2, 3]	
[9, 0, 2, 1, 1, 2]	[11, 0, 2, 1, 2, 2]	

Table 7: The combined set of non-dominated vectors found in the fixed environment by agent using all utility function type and UFP combinations.

5 CONCLUSION AND FUTURE WORK

In this paper, a first attempt at training tunable agents with non-linear utility functions was made. Several key challenges in training and evaluating with such a framework were identified and discussed, and possible solutions were proposed. The approaches and experiments presented are by no mean exhaustive in terms of the possible strategies that could be adopted; this research topic could and should be approached with a plethora of different methods, while paying close attention to the advantages and shortcomings of each one.

In the results shown in Section 4, non-linear agents have proven their potential to expand the range of valuable non-dominated vectors that can be reached and of using them more consistently than linear agents. Their usage of them is however still plagued with mistakes and challenges in the training process. The target utility seems particularly adept at uncovering combined non-dominated CRVs, though its performance is lacklustre when it comes to using them effectively.

Future work should focus on both expanding the sets of non-linear utilities, while finding ways of not having the agents lost in their complexity. One possible way to attempt this would be to separate the agent's model into two stages: a first one is responsible for predicting the vector Q-values (instead of their utility-mapped counterparts), whilst the second would be responsible of determining an approximation of the utility function (via a FCNN). The model thus constructed would then have two outputs and two different gradients flowing through parts of it.

REFERENCES

[1] Robert Aumann. 1987. Correlated Equilibrium as an Expression of Bayesian Rationality. *Econometrica* 55, 1 (1987), 1–18. https://EconPapers.repec.org/RePEc:ecm:emetrp:v:55:y:1987:i:1:p:1-18

[2] Kalyanmoy Deb and Deb Kalyanmoy. 2001. *Multi-Objective Optimization Using Evolutionary Algorithms.* John Wiley & Sons, Inc., USA.

[3] Conor Francis Hayes, Mathieu Reymond, Diederik Marijn Roijers, Enda Howley, and Patrick Mannion. 2021. Risk Aware and Multi-Objective Decision Making with Distributional Monte Carlo Tree Search. In *Proceedings of the Adaptive and Learning Agents Workshop (at AAMAS 2021).* https://arxiv.org/abs/2102.00966

[4] Conor F. Hayes, Roxana Rădulescu, Eugenio Bargiacchi, Johan Källström, Matthew Macfarlane, Mathieu Reymond, Timothy Verstraeten, Luisa M. Zintgraf, Richard Dazeley, Fredrik Heintz, Enda Howley, Athirai A. Irissappane, Patrick Mannion, Ann Nowé, Gabriel Ramos, Marcello Restelli, Peter Vamplew, and Diederik M. Roijers. 2021. A practical guide to multi-objective reinforcement learning and planning. *arXiv preprint arXiv:2103.09568* (2021). https://arxiv.org/abs/2103.09568

[5] Leslie Pack Kaelbling, Michael L. Littman, and Andrew W. Moore. 1996. Reinforcement Learning: A Survey. *CoRR* cs.AI/9605103 (1996). https://arxiv.org/abs/cs/9605103

[6] Johan Källström and Fredrik Heintz. 2019. Tunable Dynamics in Agent-Based Simulation using Multi-Objective Reinforcement Learning.

[7] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (Feb. 2015), 529–533. http://dx.doi.org/10.1038/nature14236

[8] J.F. Nash. 1951. Non-cooperative Games. *Annals of Mathematics* 54, 2 (1951), 286–295.

[9] Diederik M. Roijers, Denis Steckelmacher, and Ann Nowé. 2020. Multi-objective reinforcement learning for the expected utility of the return. 2018 Adaptive Learning Agents, ALA 2018 - Co-located Workshop at the Federated AI Meeting, FAIM 2018 ; Conference date: 14-07-2018 Through 15-07-2018.

[10] D. M. Roijers, P. Vamplew, S. Whiteson, and R. Dazeley. 2013. A Survey of Multi-Objective Sequential Decision-Making. *Journal of Artificial Intelligence Research* 48 (Oct 2013), 67–113. https://doi.org/10.1613/jair.3987

[11] Roxana Rădulescu, Patrick Mannion, Diederik Roijers, and Ann Nowe. 2019. Multi-objective multi-agent decision making: a utility-based analysis and survey. *Autonomous Agents and Multi-Agent Systems* 34 (12 2019). https://doi.org/10.1007/s10458-019-09433-x

[12] Roxana Rădulescu, Patrick Mannion, Yijie Zhang, Diederik M. Roijers, and Ann Nowé. 2020. A utility-based analysis of equilibria in multi-objective normal-form games. *The Knowledge Engineering Review* 35 (2020). https://doi.org/10.1017/s0269888920000351

[13] Richard S. Sutton and Andrew G. Barto. 2018. *Reinforcement Learning: An Introduction* (second ed.). The MIT Press. http://incompleteideas.net/book/the-book-2nd.html

[14] Peter Vamplew, John Yearwood, Richard Dazeley, and Adam Berry. 2008. On the Limitations of Scalarisation for Multi-objective Reinforcement Learning of Pareto Fronts. 372–378. https://doi.org/10.1007/978-3-540-89378-3_37

[15] H. Yu and H. M. Liu. 2013. Robust Multiple Objective Game Theory. *Journal of Optimization Theory and Applications* 159, 1 (October 2013), 272–280. https://doi.org/10.1007/s10957-012-0234-z

[16] Luisa Zintgraf, Timon Kanters, Diederik Roijers, Frans Oliehoek, and Philipp Beau. 2015. Quality Assessment of MORL Algorithms: A Utility-Based Approach.

[17] 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. arXiv:1802.07606 [cs.LG]