The Knowledge Engineering Review (2022), **37**, e7, pp. 1–29
doi:10.1017/S0269888921000163

The Knowledge Engineering Review

CAMBRIDGE UNIVERSITY PRESS

RESEARCH ARTICLE

An online scalarization multi-objective reinforcement learning algorithm: TOPSIS Q-learning

Mohammad Mirzanejad[1], Morteza Ebrahimi[1], Peter Vamplew[2] and Hadi Veisi[1]

[1]Faculty of New Sciences and Technologies, University of Tehran, Tehran, Iran; e-mail: mirzanejad@ut.ac.ir; mo.ebrahimi@ut.ac.ir; h.veisi@ut.ac.ir
[2]School of Engineering, Information Technology and Physical Sciences, Federation University Australia, Ballarat, Australia; e-mail: p.vamplew@federation.edu.au

Received: 25 March 2021; **Revised:** 14 October 2021; **Accepted:** 15 December 2021

Abstract

Conventional reinforcement learning focuses on problems with single objective. However, many problems have multiple objectives or criteria that may be independent, related, or contradictory. In such cases, multi-objective reinforcement learning is used to propose a compromise among the solutions to balance the objectives. TOPSIS is a multi-criteria decision method that selects the alternative with minimum distance from the positive ideal solution and the maximum distance from the negative ideal solution, so it can be used effectively in the decision-making process to select the next action. In this research a single-policy algorithm called TOPSIS Q-Learning is provided with focus on its performance in online mode. Unlike all single-policy methods, in the first version of the algorithm, there is no need for the user to specify the weights of the objectives. The user's preferences may not be completely definite, so all weight preferences are combined together as decision criteria and a solution is generated by considering all these preferences at once and user can model the uncertainty and weight changes of objectives around their specified preferences of objectives. If the user only wants to apply the algorithm for a specific set of weights the second version of the algorithm efficiently accomplishes that.

1 Introduction

Reinforcement learning is a field of machine learning whose main purpose is to create intelligent agents who can learn the optimal policy in the environment by interacting with it without relying on basic knowledge about that environment. By defining the reward in the environment and in return for its various situations, the agent gradually learns which of its actions result in receiving the reward, and then the agent learns the policy that maximizes the reward, and finally given that a large and distinct value is defined for the goal state, the agent is directed to the goal. In single-objective reinforcement learning, the agent pursues only one goal, and the reward is represented by a single real number. In practice, in many real-world problems and applications, there is more than one goal which is often conflicting. In such cases, conventional reinforcement learning methods are often not successful because they oversimplify the problem, leading to unrealistic results and semi-optimal decisions (Moffaert, 2014). In multi-objective reinforcement learning, rewards are written inside a vector where each component of the vector represents the reward associated with a goal. There are algorithms that can learn one or more optimal policies in a multi-objective environment. Some of them require basic knowledge of user preferences and some of them are able to learn more optimal policies without any basic knowledge.

Cite this article: M. Mirzanejad, M. Ebrahimi, P. Vamplew and H. Veisi. An online scalarization multi-objective reinforcement learning algorithm: TOPSIS Q-learning. *The Knowledge Engineering Review* 37(e7): 1–29. https://doi.org/10.1017/S0269888921000163

Many real-world problems are far more complex than can be described by a single scalar signal. Real decision issues require the simultaneous optimization of goals and criteria (Moffaert, 2014). These goals can be related or independent, but often contradict each other. That is, increasing the efficiency of one goal implicitly reduces the efficiency of another, and vice versa. In general, multi-objective reinforcement learning methods are divided into two categories depending on the number of policies they can learn. The first group turns the multi-objective problem into a single-objective problem through various techniques and then uses standard reinforcement learning methods to solve it. It should be noted that these methods learn an average policy in the objective space and cannot be optimal and appropriate for specific preferences on the objectives. The second category calculates a set of optimal policies that cover the entire space of possible priorities on the objectives. In fact, the second group is looking for a set of policies that somehow estimate the Pareto front. In explaining the Pareto front it can be said those are the set of solutions that according to the definition of the optimal solution to the problem, prevail over other members and are considered the most optimal solution available (Roijers et al., 2013). Each of these methods has advantages and disadvantages. Multi-policy methods have the ability to generate several solutions for estimating the Pareto front so that the users can choose the solution they want. The main disadvantage of producing several policies or solutions is its high computational cost which affects the ability to learn online, so single-policy methods perform better in this area (Nguyen et al., 2020). Since there are many real-world problems that need to be addressed online, this study presents a new single-policy algorithm that is expected to perform well in online mode. We use one of the multi-criteria decision-making methods in which the distance with the objectives is considered as the main decision criterion. In this way, we can set a specific policy and direction for the agent so that by considering the objectives in the decision-making process at the same time, it works better than the methods that consider the average of the objectives and other similar methods.

In this paper for the first time, the decision-making problem of choosing an action in each state considering incompatible objectives is solved by a multi-criteria decision-making method, that is, TOPSIS algorithm. Multi-criteria decision-making methods are designed to analyze complex problems with conflicting objectives. Given that in order to optimize many real-world problems, several criteria or multiple objectives must be considered simultaneously and these criteria or objectives may be independent or related to each other and even contradict each other and in such cases often there is no optimal solution and a trade-off must be proposed among the solutions to balance the objectives, this can be seen as a multi-criteria decision-making problem. Because of the commonality of multi-criteria decision-making and multi-objective reinforcement learning, and considering that in TOPSIS the selected alternative should have the minimum distance with the positive ideal solution and the maximum distance with the negative ideal solution, TOPSIS can be used effectively in the decision to choose the next action by the agent in multi-objective reinforcement learning.

The algorithm proposed in this paper is completely different in terms of how to specify preferences on the objectives. In first version of this algorithm, we will see that unlike other scalarization methods, there is no need for the user to specify the preferences on the objectives before starting the learning process and all desired preferences can be fed into the algorithm at once and simultaneously. In fact, the algorithm is executed once for all the preferences of the objectives, while in the previous methods, for each weight preferences, an execution specific to that weight had to be performed. In fact, it is perfectly suitable for circumstances where the user's preferences on the objectives are not precisely defined or are subject to change. The second version of the algorithm is used when the user wants to model the algorithm for a certain set of weights around the objectives.

2 Background: Multi-objective reinforcement learning

In this section, we provide the necessary background information on multi-objective reinforcement learning with a focus on single-policy methods.

In reinforcement learning, the agent is faced with the problem of making decisions in unfamiliar and possibly dynamic environments. In many cases there is more than one objective. Objectives may be

directly related or completely independent of each other, in which case the objectives can be combined into a single objective, then the problem can be considered as a single-objective problem and a policy with the ability to maximize the combined objective should be found. If the objectives are in conflict with each other, no policy can be found that maximizes the effectiveness of all objectives. So either one of the objectives should be preferred or a compromise should be made between them. In this case, there is usually not a single optimal policy, but there is a set of optimal policies (Vamplew et al., 2011). In the case of single-objective reinforcement learning, the overall goal is to learn the optimal decisions in order to maximize a single numerical reward. These decisions are essentially actions that must be taken in specific situations in the environment. This environment is usually defined using a mathematical framework called the Markov decision process (Sutton & Barto, 1998). Reinforcement learning is a way to solve Markov decision problem by learning the optimal policy through interaction with the environment which results in the maximum possible reward in the long run. There is always at least one policy that is better or equal to other policies. The superiority of one policy (π) over another (π') is measured by the expected return of each.

To estimate the value of the agent being in a certain state or the value of the agent performing a particular action in a particular state, value functions are used which are functions of state-action pairs. The optimal value function for state s and the optimal value function for action a in state s are as follows:

$$V^*(s) = \max_{\pi} V^{\pi}\ (s) \tag{1}$$

$$Q^*(s) = \max_{\pi} Q^{\pi}(s, a) \tag{2}$$

During the learning process, the agent updates the value functions by receiving rewards to eventually learn the optimal policy. The Q-Learning algorithm was introduced by Watkins (1989). In this algorithm, a table consisting of state-action pairs (s, a) is stored, each element of which contains an estimated value of $Q^*(s,\ a)$. Q-values are updated according to the following formula:

$$Q_{t+1}(s,\ a) \leftarrow Q_t(s,\ a) + \alpha_t \left[R(s,\ a) + \gamma \max_{a'} Q_t\ (s', a')\ -\ Q_t(s,\ a)\right] \tag{3}$$

In the above formula $\alpha_t \varepsilon\ [0, 1]$ is the learning rate at time t. In each learning episode, the actions are selected according to a specific action selection strategy, and the agent goes to a new state in the environment and receives the corresponding reward, and the Q-value is updated according to the above formula. This process continues until the Q-values converge or for a certain number of episodes. If all elements of the Q-table (state-action pairs) are visited infinitely and the appropriate learning rate is selected, the estimated values of Q will converge to the optimal values (Tsitsiklis, 1994). In fact, the condition of convergence can be met when the difference between two updates of the same Q-value related to a specific state-action pair does not exceed a small threshold or algorithms reaches a predefined number of episodes (Sutton & Barto, 1998).

To model a multi-objective problem as a reinforcement learning problem, we need to define the Multi-Objective Markov Decision Process (MOMDP) in which the reward is not a single scalar but a vector of rewards. The only difference between the standard and Multi-Objective Markov Process definition is in the definition of the reward function such that for a problem with m objectives it is defined as a vector with m dimensions as follows in which $R_i(s,\ a,\ s')$ is the single scalar reward related to the i^{th} objective:

$$R(s, a, \mathrm{s}') \rightarrow \mathbb{R}^m \tag{4}$$

$$R(s,\ a,\ s') = (R_1(s,\ a,\ s'),\ R_2(s,\ a,\ s'), \ldots,\ R_m(s,\ a,\ s')) \tag{5}$$

Expected return is also defined in terms of the vectors:

$$R_t = (R_{t1}, R_{t2}, \ldots., R_{tm}) \tag{6}$$

Each R_{ti} represents the expected return for each objective which is equal to the sum of the rewards for that objective over time and is equal to following formula:

$$R_{ti} = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \tag{7}$$

Here r_t is the reward received at time t and $0 \leq \gamma \leq 1$ is the discount factor that determines the relative importance of future rewards. The closer γ is to 1, the more important the future rewards will be. In the case of finite problems, the sum of the observed rewards is taken into account, and in the case of infinite problems, the infinite sum is taken into account. This definition applies to both episodic and continuous problems. Episodic problems have a final state that ends the episode. After reaching the final state, the environment returns to the initial state. Continuous problems never end because there is no end to them and are not separable into smaller episodes and therefore do not have a final state, so γ cannot be equal to one (Sutton & Barto, 1998).

In Multi-objective reinforcement learning the goal is to solve the Multi-Objective Markov Decision Process (MOMDP) problem and find a policy to maximize the cumulative vectorial reward, which is the reward gained during execution or as expected return.

The value function for the state s and under the policy π is defined as follows (Moffaert, 2014):

$$\begin{aligned} V^{\pi}(s) &= \mathbf{E}_{\pi}\left[R_t \mid S_t = s,\ A_t = a\right] \\ &= \mathbf{E}_{\pi}\left[\sum_{k=0}^{\infty}\sum_{i=1}^{m} \gamma^k r_{ti+k+1} \mid S_t = s\ ,\ A_t = a\right] \end{aligned} \tag{8}$$

In above formula $\mathbf{E}_{\pi}$ represents the value of the random variable given that the agent follows policy π. In fact, the value function is calculated as the expected vectorial return that the agent receives if it starts from state s and selects actions according to the policy π.

The most common measure of policy optimality is called Pareto dominance and is defined as follows (Moffaert, 2014):

The policy π_1 is strictly Pareto dominates π_2 ($\pi_2 \succ \pi_1$) if it is strictly better on at least one objective of policy π_2 , and at the same time it is not strictly worse on the other objectives in all states. The policy π_1 weakly Pareto dominates π_2 ($\pi_2 \succcurlyeq \pi_1$) when there is no objective that for all states, $V^{\pi_1}(s)$ is better than $V^{\pi_2}(s)$. The policies π_1 and π_2 are incomparable if $V^{\pi_1}(s)$ does not Pareto dominate $V^{\pi_2}(s)$ for all states and vice versa. The policy π_1 is non-dominated by the policy π_2 if it either Pareto dominate policy π_2 or is incomparable with that policy. Any solution that is dominated by another solution has less value and the dominant solution is preferred. Thus, Pareto optimal solutions or policies are those that are either strictly dominant or incomparable to any member of the solution set. If this procedure is applied to a set of possible solutions, a non-dominated set of solutions is obtained, which is called the Pareto front.

Multi-objective reinforcement learning algorithms are divided into two categories: single-policy and multi-policy. In this paper, since our proposed method belongs to first category, we will not discuss multi-policy algorithms and focus only on the single-policy algorithms. In the following subsections we will introduce the most common and noteworthy single-policy algorithms.

2.1 Linear scalarization

Single-policy algorithms are similar to single-objective reinforcement learning algorithms in that they learn only one optimal policy. In multi-objective algorithms, however, there are usually more optimal policies on the Pareto front, and it must be specified for the algorithm which of the optimal policies is preferable (Vamplew et al., 2011). One way to prioritize is to use a scalarization function that converts the reward into a numerical value. This leaves us with only one value function that can be achieved by one or more optimal policies. Scalarization functions are defined as follows:

$$v_w = f(v, w) \tag{9}$$

In the above formula, f is a scalarization function that converts the vector v to a scalar. These functions usually use the weight vector w to determine the relative importance of the objectives. The main drawback of single-policy algorithms is that the preference of objectives must be decided in advance.

Weight changes related to objectives can cause a decrease in one objective to a very small extent but a significant increase in another objective. In fact, depending on the nature of the Pareto front, small changes in weights can cause large changes in the policy being learned, and vice versa (Vamplew et al., 2008). The linear scalarization function calculates the inner product of the weight vector w and the value vector V^{π} obtained from the learned value function:

$$V_w^{\pi} = w.V^{\pi} \tag{10}$$

Each element of w is a real number in the range [0,1] that indicates how much a value associated with each objective will contribute to the scalarized value. All of the weights must be positive and sum to 1. For representing the value functions in single-objective learning, the agent stores a table of values of $Q(s, a)$ that represents the expected cumulative discounted reward associated with each pair of state s and action a in the environment. These values of $Q(s, a)$ are updated over time using formula (3). In a multi-objective environment to learn a single-objective policy, the agent stores a Q-vector for each pair of state and action. Each Q-vector contains a $Q(s, a)$ value for each objective. Q-vectors are updated separately for each element. For example, objective O of the Q-vector is updated using the following formula, which is used for multi-objective Q-Learning and is called the bootstrapping rule:

$$Q_o(s,\ a) \leftarrow Q_o(s,\ a) + \alpha_t\ (\mathbf{r}_o + \gamma \max_{f(Q(s',a'),w)} Q_o\ (s', a') -\ Q_o(s, a)) \tag{11}$$

In the above formula $\mathbf{r}_o$ is o^{th} element of the immediate reward vector and $\max_{f(Q(s',a'),w)} Q_o\ (s', a')$ regarding the scalarization function f and the weight vector w gives the maximum scalarized value in the Q vector (Moffaert, 2014).

In order to describe the algorithm, the mechanism of action selection must first be specified. The exploration is done as ϵ -greedy, which ϵ indicates the randomness of the action selection and 1- ϵ the best action based on the greatest value of the value function and the corresponding Q-values to go to the next state. In order to be able to determine the best action to go to the next state in a multi-objective environment, the values of the Q vectors must be converted to a scalarized value. The action selection algorithm is presented in Algorithm 1. The scalarization function f which is linear, calculates the linear combination of the values of a vector with respect to the weight vector w. In fact, in the framework of the multi-objective reinforcement learning algorithm shown in Algorithm 2, it operates according to (12) and eventually based on (13) the action that has the highest weighted sum, that is, the sum of its Q-values in the state s for all objectives is greater than all other actions, is selected as the next action:

$$\text{ScalarizedQ}(s, a) = (s, a).w = \sum_{o=1}^{m} Q_o(s, a)\ .w_o \tag{12}$$

$$greedy_{a'}(s) = \max_{a'} \text{ScalarizedQ(s, } a') \tag{13}$$

To start the algorithm, the agent is in an initial state in each episode. In the next step, the agent selects an action and by adopting that action, observes the state to which it has gone and its reward vector. Using the action selection function, and according to the observed state s', action a' is selected and used to update the Q-table of each objective. Then for all objectives according to the reward vector obtained and the maximum scalarized value, the Q vector is updated by the scalarization function f. The agent then goes to the next state and these operations are repeated until it reaches a final state and a new episode begins. The algorithm terminates after convergence. The pseudo-code of the algorithm is given in Algorithm 2.

In the linear scalarization method, the relative importance of the objectives to each other is specified by w and should be determined before starting the learning process. From a convergence point of view, since this algorithm is based on Q-Learning and only the scalarization function is added as a mechanism for selecting action, it converges in exchange for a certain number of episodes. More theoretical details are available in Sutton and Barto (1998). The Linear scalarization has simplicity and fair speed, but various articles have mentioned that this method is only able to find policies that are in the convex region of the Pareto front (Vamplew et al., 2008; Roijers et al., 2013). Therefore several researchers

Algorithm 1 Scalarized Action Selection (Moffaert, 2014)

ScalarizedQList ← {}
for each action $a \in$ A(s) **do**
ScalarizedQ ← $f(\ Q(s, a), w)$
Append ScalarizedQ to ScalarizedQList
end for
return $\varepsilon - greedy$ ($\max_a$ ScalarizedQList(s, a))

Algorithm 2 Scalarized Multi-Objective Q-learning (Moffaert, 2014)

Initialize $Q(s, a)$
for each episode t **do**
  Initialize state s
  repeat
    $a \leftarrow$ ScalarizedActionSelection(s)
    Take action a and observe state $s' \in S$ and reward r $\in R^m$
    $a' \leftarrow$ ScalarizedActionSelection(s′)
    for each objective o **do**
      $Q_o(s,\ a) \leftarrow Q_o(s,\ a) + \alpha_t\ (\mathbf{r}_o(\mathrm{s}, a) + \gamma \max_{f(Q(s',a'),w)} Q_o(s', a') - \ Q_o(s, a))$
    end for
    $s \leftarrow s'$
  until s in terminal
end for

have proposed the use of nonlinear scalarization methods. The following subsections will review some of these nonlinear approaches to scalarization.

2.2 Threshold lexicographic Q-learning

The TLQ-Learning algorithm uses a combination of thresholding and lexicographic ordering to guide the learning process. This algorithm has the ability to define a threshold value for each objective, which is defined depending on whether the objective should be maximized or minimized. Therefore, this method is mainly suitable for situations where one objective should be maximized and at the same time constraints should be applied to other objectives. After reaching the threshold value on one objective, the agent does not continue to advance on that objective and optimizes the other objective (Gabor et al., 1998).

Due to the limitations on the objectives, the definition of the value function in the action selection mechanism is defined as follows:

$$CQ(s, a)_i = \min\ (CQ(s, a)_i, C_i) \tag{14}$$

The threshold value for i^{th} objective is denoted by C_i. If we have m objectives, we usually set a limit on all of them except the last objective, that is, $C_m = \infty$. Ordering objectives also leads to ordering policies. To compare the policy π with the policy π', we first look at the first objective and choose the policy that works best for that objective. If the cumulative reward of the first objective is the same for both policies, we must compare the second objective, and so on. If the policies have the same value on all objectives, we can choose one of them arbitrarily. Given that most objectives are constrained to their threshold values, the cumulative reward for these goals will be the same for many policies. Therefore a policy can be chosen that satisfies all constraints and maximizes the last objective (Gabor et al., 1998).

Algorithm 3 Superior (Vamplew et al., 2011)

```
if CQ_i(s, a) > CQ_i(s, a')
    return true;
else if CQ_i(s, a) = CQ_i(s, a')
    if i= NumberOfObjectives
        return true;
    else
        return Superior(CQ(s, a), (CQ(s, a'), i + 1);
else
    return false;
```

Algorithm 4 TLQ Action Selection

```
1:  for each action a ∈ A(s) do
2:      IsSuperior ← True;
3:      for each action a' ∈ A(s) do
4:          if a ≠ a'
5:              IsSuperior ← Superior (CQ(s, a), (CQ(s, a'), 1);
6:          if IsSuperior ≠ True
7:              break;
8:      end for
9:      if IsSuperior
10:         return a ∈ A(s)
11: end for
```

The TLQ-Learning algorithm is presented by changing the action selection mechanism of Algorithm 2. Therefore, we must first define its action selection algorithm. To do this, we need to define the Superior algorithm, which compares two operations and their value function in the form of Algorithm 3. Then we can define an action selection algorithm (Algorithm 4) that actually greedily selects action a in state s if it is superior to all other actions according to Algorithm 3. Finally, by defining these two algorithms, the TLQ can be presented in Algorithm 5. The difference between this algorithm and the Linear Scalarization is the use of a different action selection algorithm and the addition of the calculation of $CQ_i(s, a)$ in line 10, which is used to select the action.

The results of experiments on the Linear Scalarization and TLQ-Learning algorithms in Issabekov and Vamplew (2012) indicate that in problems whose structure is suitable for the TLQ-Learning method, since it can detect policies located in concave regions, it outperforms the Linear Scalarization method which is unable to detect them. The convergence of the algorithm cannot be proved formally due to nonlinearity of the scalarization function and the quality of the test results should be relied upon.

2.3 Chebyshev scalarization

This algorithm is based on the L_p norm. L_p calculates the distance between the point x and the utopian point z^* in multi-objective space. L_p is defined to measure the distance of the value of the objective function f from z^* as follows. Note that f is a function of solution x for each objective o:

$$L_p\,(x) = \left(\sum\nolimits_{o=1}^{m} w_o \left|f_o(x) - \; z_o^*\right|^p\right)^{1/p} \tag{15}$$

where m is the number of objectives and w_o is the weight of the objective o. Also, p is a number in the range of $[1, \infty]$. If $p = \infty$, the L_p norm is called the weighted L_∞ or the Chebyshev norm and is defined

Algorithm 5 TLQ-Learning

1: Initialize $Q(s,\ a)$
2: **for** each episode t **do**
3: Initialize state s
4: **repeat**
5: $a \leftarrow$ TLQ Action Selection ()
6: Take action a and observe state $s' \in S$ and reward r $\in R^m$
7: $a' \leftarrow$ TLQ Action Selection ()
8: **for** each objective o **do**
9: $Q_o(s,\ a) \leftarrow Q_o(s,\ a) + \alpha_t\ (\mathbf{r}_o(\mathrm{s,\ a}) + \gamma \max_{f(Q(s',a'),w)} Q_o\ (s', a') - \ Q_o(s, a))$
10: $CQ_o(s, a) \leftarrow \min(Q_o(s, a), C_o)$
11: **end for**
12: s $\leftarrow s'$
13: **until** s in terminal
14: **end for**

according to the following formula:

$$L_\infty(x) = \max_{o=1\ldots m} w_o\ \left|f_o(x) - \ z_o^*\right| \tag{16}$$

by replacing the function $f_o(x)$ with the value of Q associated with each objective and pair of state and action, the L_∞ criterion can be used as a scalarization function in the action selection mechanism in multi-objective learning. Therefore, the scalarized value of Q for state s and action a is defined as follows:

$$\text{ScalarizedQ}\ (s, a) = \max_{o=1\ldots m} w_o\ \left|Q(s, a, o) - \ z_o^*\right| \tag{17}$$

by substituting (17) instead of the scalarization function f in Algorithm 1, the action selection mechanism of the Chebyshev algorithm can be defined. The difference is the choice of the greedy action, which in the standard mode and in Algorithm 1 is the action with the maximum value, but here is the minimum value and is defined as follows:

$$greedy_{a'}(s) = \min_{a'} \text{ScalarizedQ(s, } a') \tag{18}$$

The goal is to get as close as possible to the utopian point z^*, so we must choose the action that has the shortest distance to that point. During the learning process, the ideal point z^* is constantly updated with the best value of each objective up to that point plus a constant value of $\tau \in \mathbb{R}$ and guides the learning process towards the optimal solution. This update is added to Algorithm 2. Finally, assuming that Algorithm 1 and the corresponding action selection mechanism use (17) instead of f in its line 3, the Chebyshev algorithm is defined as Algorithm 6.

As any other algorithm that uses nonlinear function, the Chebyshev nonlinear function lacks the property of addition. Therefore, the Bellman equation is not established for it and it cannot be formally and theoretically proved that the Chebyshev function in combination with a learning algorithm will converge to an optimal policy (Moffaert, 2014). However, in practice, performance and results of the experiments are reported to be very good and satisfactory. As stated by the original paper this approach can find policies regardless of their location and provides better coverage solutions on the front. Also it is less dependent on the weight of the objectives compared to methods that use the Linear Scalarization and due to the nonlinearity of the function used policies that are in the non-convex region can be found (Moffaert et al., 2013).

Algorithm 6 Chebyshev Scalarized Q-Learning Moffaert et al., 2013

Initialize $Q(s, a)$
for each episode t **do**
    Initialize state s
    repeat
        $a \leftarrow$ ScalarisedActionSelection()
        Take action a and observe state $s' \in S$ and reward r $\in R^m$
        $greedy_{a'}(s')$//(18)
        for each objective o **do**
            $Q_o(s, a) \leftarrow Q_o(s, a) + \alpha_t\,(\mathbf{r}_o(\mathrm{s}, \mathrm{a}) + \gamma Q_o(s', greedy_{a'}(s')) - Q_o(s, a))$
        end for
        s $\leftarrow s'$
    until s in terminal
end for

3 Proposed model: TOPSIS Q-learning

In both multi-criteria decision-making and multi-objective reinforcement learning, prioritizing objectives and solutions is the responsibility of the user or so-called decision-maker and plays an important role in the decision-making process. In fact, by determining preferences by the user, the relative importance of objectives to each other is specified. In literature there are three ways to express preferences: prior, posterior, and interactive. Specifically in the prior method, the user sets preferences before the decision-making or learning process (Moffaert, 2014). Most of the previous methods include priorities in the form of parameters in the problem, which leads the search process to a solution that satisfies these priorities. In this regard, similar to the approach taken by all the methods reviewed in Section 2 a scalarization function is used which reduces the multi-objective problem to a single-objective problem by considering the user's preferences as a constraint in order to maintain optimality. As mentioned previously, a scalarization function is a function that converts a vector to a scalar according to a given weight vector. When the user is already aware of the priorities of the objectives in the decision-making problem, there is no need to estimate the Pareto front of the optimal solutions and the weight vectors specifies the compromised or trade-off solutions that somehow specified before the execution of the algorithm by the user. So there is an overlap between the decision-making process and multi-objective reinforcement learning. In the following, we will see how the agent in the decision-making process in the action selection mechanism can use the multi-criteria decision-making method.

We are looking for optimal solutions in MCDM problems. Although there is often no optimal solution due to conflicting objectives or criteria, an optimal solution, also called the ideal solution, is a solution in which the value of each objective is maximized simultaneously. The ideal solution is not actually achievable, but can be used as a guide for evaluating other solutions. MCDM is divided into two sub-categories: Multi-Objective Decision Making (MODM) and Multi-Attribute Decision Making (MADM). MODM includes problems in which the decision space is continuous and MADM deals with problems which have discrete decision space. In this paper, as our decision space in action selection mechanism in MORL is discrete so we use MADM to aid the decision process. The ideal solution in MADM is a hypothetical alternative whose Cartesian product consists of the values of each criterion with the highest preference in the decision matrix.

To reach a trade-off model finding a non-dominated solution which is close to the ideal solution is necessary. A non-dominated solution (also known as a Pareto-optimal solution) is a feasible solution where there is no other solution that could improve on an objective without performing worse on at least one other objective. A non-dominated solution is called a preferred solution when it is selected as the final solution using decision-maker's preference information (Hwang & Yoon, 1981).

There are some methods for MADM which process the criteria or attributes to reach a solution. Two major approaches are non-compensatory and compensatory models. Non-compensatory models are simple to use but lack ability to trade-off between criteria, that is, a decline in the value of one criterion cannot be offset by an improvement in the value of another. In contrast, compensatory models allow trade-offs between criteria. Each alternative in decision matrix with multiple criteria is assigned to a single number or score. There are three approaches to calculate this number and appropriate alternative. (1) Scoring model which selects an alternative with highest score using various weighting methods. (2) Concordance model which performs pair-wise comparisons of alternatives in order to rank them based on concordance. (3) Compromising model which selects alternative based on its closeness to the ideal solution. Since compromising model takes advantage of the concepts of ideal, non-dominated and Pareto-front solutions which are crucial concepts in MORL as well, so it seems we could apply that to assist the agent in decision-making process in action selection mechanism. One of the famous methods of this model is TOPSIS which uses ideal solution concept in its core to rank alternatives (Hwang & Yoon, 1981).

3.1 TOPSIS method

Since we will use TOPSIS in the proposed model, here we introduce this multi-criteria decision-making method that was introduced in 1981 by Hwang and Yoon (1981). In this method, the m alternatives are evaluated by n criteria. In fact, a decision matrix with the form of n × m is constructed. This method is basically based on the two concepts of positive and negative ideal solution. This means that the alternative chosen must have the shortest distance from the positive ideal solution (best possible case) and the longest distance from the negative ideal solution (worst possible case). To solve the problem after forming the decision matrix, the following six steps must be followed (Hwang & Yoon, 1981):

1. Normalizing the decision matrix X: The matrix is normalized in the form of $(x_{ij})_{m\times n}$ as follows:

$$r_{ij} = \frac{x_{ij}}{\sqrt{\sum_{k=1}^{m} x_{kj}^2}},\ i = 1,2,\ldots,m,\ j = 1,2,\ldots,n$$
$$R = (r_{ij})_{m\times n} \tag{19}$$

2. Calculating the weighted normalized decision matrix: We multiply the normalized decision matrix of the previous step in the diagonal matrix of weights $W_{n\times n}$ in the following order:

$$V = R \times W_{n\times n}$$
$$v_{ij} = r_{ij} \cdot w_j, i = 1, 2, \ldots, m, j = 1, 2, \ldots, n$$
$$w_j = W_j / \sum_{k=1}^{n} W_k, j = 1, 2, \ldots, n, \sum_{k=1}^{n} w_i = 1 \tag{20}$$

3. Determining the positive ideal solution and the negative ideal solution:
   Positive ideal solution (v_j^+) = vector containing the best values of each criterion of the matrix V.
   Negative ideal solution (v_j^-) = vector containing the worst values of each criterion of the matrix V.
   The best values for positive criteria are the largest values and for negative criteria are the smallest ones, and the worst values for positive criteria are the smallest values and for negative criteria are the largest ones.
4. Calculating the distance between the alternatives and the positive and negative ideals: To do this, the Euclidean distance of each alternative from the positive ideal and the negative ideal solutions must be calculated in the following order:

$$d_i^+ = \sqrt{\sum_{j=1}^{n} (v_{ij} - v_j^+)^2}, i = 1, 2, \ldots, m \tag{21}$$

$$d_i^- = \sqrt{\sum_{j=1}^{n} (v_{ij} - v_j^-)^2}, i = 1, 2, \ldots, m$$

5. Determining the relative similarity of each alternative to the ideal solution:

$$cl_i = \frac{d_i^-}{d_i^- + d_i^+} \tag{22}$$

6. Ranking: the alternative with a larger cl is preferred.

If distance measure in TOPSIS is defined by the Manhattan distance instead of the Euclidean distance then the Simple Additive Weighting (SAW) method will be a special case of TOPSIS. For proof let assume the distance between A_i and A_k is defined by following formula:

$$d_{ik} = \sum_{j=1}^{n} \left| v_{ij} - v_{ij} \right| \text{ i, k} = 1, 2, \ldots, \text{m i} \neq \text{k} \tag{23}$$

As Manhattan distance has the following relationship:

$$d_i^* + d_i^- = d_*^- = K, i = 1, 2, \ldots, m (K > 0 \text{ and Constant}) \tag{24}$$

So alternatives with the shortest distance to the positive ideal solution have the longest distance to the negative ideal solution. And for $i = 1, 2, \ldots, m$ relative closeness can be defined as $C_i^* = d_i^- / d_*^-$. If alternative A^+ is defined by $A^+ = \{ A_i \mid \max C_i^* \}$ and finally it is proved (Yoon, 1980; Hwang & Yoon, 1981):

$$A^+ = \{ A_i \mid \max_i \sum_{j=1}^{n} v_{ij} \} = \{ A_i \mid \max_i C_i^* \} \tag{25}$$

SAW method functions exactly like Linear Scalarization to select an alternative. Both choose an alternative which has the maximum average outcome. As we stated above if we replace Euclidean distance measure with Manhattan distance, TOPSIS will be converted to SAW or weighted sum model. The critical difference between these two measures is trade-off which explained in Hwang and Yoon (1981), Keeney and Raiffa (1976) MacCrimmon and Toda (1969), and MacCrimmon and Wehrung (1977) and we will review it here.

Ratio of the change in one criterion that exactly negates a change in another criterion is called trade-off or marginal rate of substitution (MRS). Contours on a given value function are called indifference curves. It is assumed each alternative located on an indifference curve is preferred equally by a decision-maker. Given the indifference curves the trade-off or MRS at any point is equal to negative reciprocal of the slope at that point. If indifference curve is defined by $\text{f}(v_1, v_2) = \text{c}$ where f is a value function and c is a constant and v_1 and v_2 are coordination of a point through which curve passes. To calculate MRS we have the following:

$$\lambda = -\frac{dv_1}{dv_2}|_{(v_1, v_2)} = \frac{\partial f}{\partial v_2} / \frac{\partial f}{\partial v_1}|_{(v_1, v_2)} \tag{26}$$

Since value function of SAW or TOPSIS with Manhattan distance measure is equal to $\text{f}(v_1, v_2) = v_1 + v_2$, MRS is obtained by $\lambda = 1$. So it can be concluded that trade-off (MRS) in SAW is constant between criteria and indifference curves are straight lines with the slope of -1. This type of trade-off considers local MRS as the global one as well.

To calculate trade-off in TOPSIS with Euclidean distance measure we have to calculate λ using value function:

$$f(v_1, v_2) = \frac{d_i^-}{d_i^- + d_i^*} = \text{c} = \frac{\sqrt{(v_{1-} - v_1^-)^2 + (v_{2-} - v_2^-)^2}}{\sqrt{(v_{1-} - v_1^-)^2 + (v_{2-} - v_2^-)^2} + \sqrt{(v_{1-} - v_1^*)^2 + (v_{2-} - v_2^*)^2}} \tag{27}$$

$$\lambda = \frac{d_i^{-2}(v_2^* - v_2) + d_i^{*2}(v_2 - v_2^-)}{d_i^{-2}(v_1^* - v_1) + d_i^{*2}(v_1 - v_1^-)} \tag{28}$$

For the point at where distances to the positive and negative ideal solutions are the same (i.e. $d_i^* = d_i^-$) λ and value function equal the following:

$$\lambda = \frac{v_2^* - v_2^-}{v_1^* - v_1^-} \tag{29}$$

$$\mathrm{c}d_i^* - (1 - \mathrm{c})d_i^- = 0, 0 < \mathrm{c} < 1 \tag{30}$$

As a conclusion in TOPSIS the property of diminishing MRS is in effect which allows trade-off between criteria more naturally and flexible than weighted sum method (and consequently the Linear Scalarization) that the MRS is a constant and doesn't follow this property. Also for some specific values of relative closeness to the positive and negative ideal solutions (C_i^*) indifference curves are convex or concave to the preference origin. The former is more common and the latter is suitable for situations in which decision-maker for any reason is inclined to negative ideal solution rather than positive one (MacCrimmon & Toda, 1969; Hwang & Yoon, 1981). Another advantage of using TOPSIS in our MORL model is that it recognizes the lack and not only the presence of criteria.

3.2 Using TOPSIS in the proposed model

In the proposed model, if we consider TOPSIS as an action selection function, the standard form of this function could be as follows:

$$topsisActionSelection\ (actions, C_1(s), \ldots, C_N(s), w) \tag{31}$$

where actions are an array of alternatives to select, that is, the same agent's actions, C_1 and C_N are the criteria related to all actions in state s for the preference set 1 and preference set N respectively containing corresponding Q-values and weight preferences. That is, all weight preferences are given to the action selection function as decision criteria at once, and unlike other previous scalarization models, there is no need for the user to choose the weight preferences. w is the weight vector by which the relative importance of each of the decision criteria or weight preferences can be determined. Normally we define the TOPSIS weight vector in the uniform way (with equal weights sum to one). This function is called when the agent has to decide to select the next action. According to Section 3.1 the decision matrix is formed first, where actions are decision or selection alternatives and Q-values in state s related to different weight preferences are the decision criteria. Then steps 1 to 6 of TOPSIS Method (Section 3.1) are performed and the output of the function is the ranking of the alternatives, which is usually the alternative with the rank of 1 will be chosen. Hence the greedy action which is obtained from scalarization function is generated as follows:

$$greedy_a = \max Q_i\ (s, a)\ \ i = 1 \ldots n \tag{32}$$

Suppose that each of the decision criteria given to TOPSIS is a combination of objectives with different weights. In this case, each criterion can be defined as follows:

$$C_j(s, a) = \max_i w_1 Q_1\ (s, a), \ldots, w_i Q_i(s, a), \ldots,\ w_n Q_n(s, a) \tag{33}$$

$$\sum_{i=1}^{n} w_i = 1$$

Each criterion is denoted by C_j and $\mathrm{j} = 1, 2, \cdot, \mathrm{m}$ is the number of criteria. The total weight of the objectives must also be equal to one. In fact, the objective that is most valuable is selected according to the state and action pair in Q table as well as its weight coefficient. Of course, this selection is updated and changed according to the change of Q-values in each episode. Also note that each of the criteria contains a different weight combination according to the objectives coefficients. In some cases, despite

the use of single-policy and scalarization methods and the provision of weights before the start of the decision-making and learning process, the preferences for the user may not be completely definite or are subject to change. There has been some research on designing a framework that can include uncertainty and changes in the preferences of the objectives in the field of multi-objective reinforcement learning (Roijers et al., 2017, 2018, 2020; Vamplew et al., 2017) and in this study we want to address this issue in a different way. The problem can be modeled with the help of TOPSIS as presented in the first version of the algorithm of our proposed model. As we will show in the following, a different weight combination of objectives can be created with each criterion to cover changes and uncertainty of user preferences. These weight combinations for each objective are slightly apart from each other (for example 0.1).

The user may also want to train and use the model for a specific set of weights. Each objective, that is, its Q table multiplied by related weight is given to the TOPSIS function as a criterion and finally TOPSIS algorithm carries out the operation to select the next action for the agent. Unlike the first version, which is weight-independent, the scalarization function of the second version must take into account the user's preference weights. So the greedy action generated from scalarization function for the second version is presented as follows:

$$greedy_a = \max_i w_i Q_i\,(s, a) \quad i = 1 \ldots n \tag{34}$$

3.3 TOPSIS Q-learning algorithm

This algorithm is based on Q-Learning and its structure is similar to the Linear Scalarization, except that its action selection mechanism is replaced by TOPSIS. As mentioned earlier, in the first version of proposed algorithm, unlike other single-policy methods in which weights must be set as preferences by the user before starting the learning process, these weight combinations are automatically generated as decision criteria in the algorithm at once and simultaneously and do not need to be determined by the user, and with only one run and taking into account all these weight combinations, the solution is produced. These weights are used as Q-coefficients for the objectives to form weight preferences of the objectives associated with them. Each weight combination is considered as a decision criterion for TOPSIS so that the user can model the conditions of uncertainty and weight changes of objectives around their preferences over objectives. In the first version of the algorithm using the TOPSIS action selection mechanism, the values of the Q Table related to each objective in state s alongside corresponding weights are given to TOPSIS as a criterion for all actions in that state to select the optimal action (line 5 of Algorithm 7). Also, to update the Q table of each objective, the TOPSIS action selection mechanism is used, that is, the value of the Q table for each objective according to the received reward and towards the Q-value related to the optimal action of the next state, which is determined by TOPSIS for all objectives is updated (line 9 of Algorithm 7). An overview of the first version of the algorithm is given in the form of Algorithm 7.

In the second version of the algorithm the agent invokes the TOPSIS function as the action selection mechanism when deciding to move to the next state. TOPSIS considers all criteria as shown in line 8 of Algorithm 8 while ε-greedy selection. In fact steps 1 to 6 of Section 3.1 are performed and finally the alternative with a highest rank is selected as the preferred action and agent goes to the next state s' (line 9). In line 10 greedy function is applied on TOPSIS in state s' and action a' is selected to update the Q-Learning formula in line 11. The Q-values and criteria are then updated and the agent goes to the next state. The choice of learning and environment-related parameters will be discussed in Section 4. The second version of the algorithm is presented in Algorithm 8. The subtle distinction between criteria of two versions of algorithms is that the first version handles different combinations of weight preferences while the second one directly deals with the objectives. N in C_N in Algorithm 7 refers to the number of combinations of weights and in Algorithm 8 it refers to the number of objectives.

To prove the convergence, it should be said that although the algorithm is based on Q-Learning, but because the TOPSIS method is basically based on the concept of its distance function (i.e. calculating the distance of alternatives from the positive and negative ideal solution) and the distance function in

Algorithm 7 TOPSIS Q-Learning – First Version

Initialize $Q(s, a)$
for each episode t **do**
  Initialize state s
  repeat
    $a \leftarrow \epsilon -$ greedy(*topsisActionSelection*(*actions*, $C_1(s), \ldots, C_N(s), w$))
    Take action a and observe state $s' \in S$ and reward r $\in R^m$
    $a' \leftarrow$ greedy(*topsisActionSelection*(*actions*, $C_1(s), \ldots, C_N(s), w$))
    for each objective o **do**
      $Q_o(s,\ a) \leftarrow Q_o(s,\ a) + \alpha_t\ (\mathbf{r}_o(\mathrm{s, a}) + \gamma\ Q_o(s', a') -\ Q_o(s, a))$
    end for
    for each criterion C_j **do**
      $C_j \leftarrow \max_{o=i\ldots m} w_o Q_o(s)\ \ (j = 1 \ldots . n)$
    end for
    s $\leftarrow s'$
  until s is terminal
end for

Algorithm 8 TOPSIS Q-Learning – Second Version

Initialize $Q(s, a)$
for each episode t **do**
  Initialize state s
  repeat
    $a \leftarrow \epsilon -$ greedy(*topsisActionSelection*(actions $\in A(s)$, $C_1(s), \ldots, C_N(s), w$))
    Take action a and observe state $s' \in S$ and reward r $\in R^m$
    $a' \leftarrow$ greedy(*topsisActionSelection*(actions $\in A(s')$, $C_1(s), \ldots, C_N(s), w$))
    for each objective o **do**
      $Q_o(s,\ a) \leftarrow Q_o(s,\ a) + \alpha_t\ (\mathbf{r}_o(\mathrm{s, a}) + \gamma\ Q_o(s', a') -\ Q_o(s, a))$
      $C_o(s, a) \leftarrow\ w_o Q_o(s, a)$
    end for
    s $\leftarrow s'$
  until s is terminal
end for

TOPSIS is nonlinear and lacks additivity, so the Bellman equation does not hold for it. As a result, like other methods that combine nonlinear scalarization functions with Q- Learning, the convergence of the algorithm into an optimal policy is not theoretically provable and must rely on practical and experimental results.

In fact, using nonlinear scalarization functions such as Chebyshev, TLQ, etc. in combination with Q-Learning, the basic assumption in the Bellman equation, namely the property of additivity of returns, is no longer established. Therefore, there is no guarantee that the agent will converge to the optimal policy (i.e. in terms of numerical rewards during the episode). The question also arises as to whether the algorithm will converge at all in some cases where the agent meets a state again or the states and rewards are random. In general, to ensure convergence under such conditions, the agent must store the sum of the rewards it has earned so far in the current episode and in each state by adding that sum of rewards to the values of $Q(s, a)$ and scalarization of the yielded sum in the action selection should be done. Also the decisions about Q-values should also be made based on the values of accumulated rewards and the current state (Geibel, 2006; Roijers et al., 2013).

4 Experiments and results

It is obvious that the research problem does not end with the presentation of the model and the model must be tested. The test is performed in different environments with parameters specific to that environment and specific configuration related to the algorithm to achieve the highest possible efficiency or the optimal solution in the solution space. In order to evaluate multi-objective reinforcement learning algorithms, the solution obtained online may be satisfactory, but not offline, and vice versa. An algorithm that is suitable for one environment may be unsuitable and ungeneralizable for another environment. Single-policy algorithms are suitable for online learning. The effectiveness of online learning can be assessed in two ways: the hypervolume of the accumulated rewards and regret measure. To use the regret measure, the Pareto front of the problem or the optimal policy must be known in advance. For many problems where the optimal policy is unknown, the hypervolume of the accumulated reward can be calculated without knowing the optimal policy. In situations where we are not aware of the Pareto front of the problem, the online hypervolume measure is the best option, but if we know the solution, the regret measure can provide more accurate information (Vamplew et al., 2011). However, the most common evaluation method in most papers is the hypervolume measure.

In the definition of hypervolume, we can say that by considering a reference point r over which the whole Pareto front (solution set S) or our estimation of the Pareto front dominates, the space between r and the Pareto front is defined as hypervolume. The more space there is, the larger the hypervolume and the better the algorithm works. The advantage of hypervolume is that it provides only a number to evaluate the quality of a set of reward vectors, and this number represents an improvement in all three characteristics of the Pareto front: accuracy, extent, and diversity. It is also the only indicator that is strictly monotonic with the Pareto dominance relationship, meaning that whenever a better solution is found or the Pareto front improves, the hypervolume value also increases.

The definition of regret in multi-objective learning is as follows:

$$\mathbf{R}_T = T\rho^* - \sum_{t=0}^{T-1} r_t \tag{35}$$

where $\mathbf{R}_T$ is the regret vector, T is the number of time steps during which the regret measure is calculated, r_t is the reward vector received at time t, and ρ^* is the mean reward vector for following an optimal policy. Here, if the regret vector of one algorithm dominates another, the dominant algorithm is clearly in preference. There is also the issue of non-comparability here, and a method must be provided to determine which algorithm better meets the user's preferences. Using the length of the regret vector leads to the choice of dominated policies, so the length of the nonnegative elements of the regret vector is used as follows (Vamplew et al., 2011):

$$\mathbf{R}_S = \sqrt{\sum_{j=0}^{N} (max(0, \mathbf{R}_{T,j}))^2} \tag{36}$$

The point to be considered is to determine an appropriate member of the solution set (i.e. Pareto front) as the target policy so that the calculation of ρ^* in formula (35) can be performed in return. The paper Vamplew et al. (2011) suggests that single-policy algorithms should be evaluated by human agents in terms of their application, and in this context, user-based practical test methods are presented and discussed. The interested reader can refer to it for more details.

There are two approaches to examining how the algorithm has performed; one is to review the rewards collected during learning and the other is the rewards received after the completion of the learning process and the applying of the final policy by the agent. The point to be made about rewards while learning is that algorithms may produce different results depending on the extent of the search space and the rate of exploration if repeated with the same learning parameters, therefore in papers, the discussion of repeating experiments and averaging over the results has been discussed. Those rewards or solutions that dominate others on the Pareto front are selected, which are used in most papers as a criterion for judging the performance of the algorithm and eventually for evaluating the algorithm, the amount of hypervolume generated by these dominant rewards should be considered. The outcome of each complete execution of each algorithm is a set of online non-dominated rewards gathered during training

(i.e. Pareto Front). But if an agent discovers a particular reward during learning that may be due to the fact it performed some random actions rather than it having actually learned a policy which would allow it to reliably achieve that reward. It is only meaningful to say an agent has found a particular reward if its offline policy actually reaches that reward. We cannot rely only on online rewards alone which could be randomly encountered during learning and they must be validated using some policy following test and finally be extracted as offline rewards. So we set the final offline rewards generated after learning process is finished as the basis for calculating the hypervolume. In fact, in our approach online rewards should be repeated in offline mode.

According to Moffaert and Nowé (2014) and Roijers et al. (2021), applying arg-max operator on all actions in current state is more suitable for single-objective reinforcement learning and for multi-objective learning we need to define an adapted version of greedy action selection. With inspiration of the initial part of Roijers et al. (2021) which selects the action minimizing the Manhattan distance of minimum Q-value of current state and desired value vector (i.e. reward or solution vector), we developed a new algorithm to search for a policy in offline mode. As Algorithm 9 demonstrates the process, each time a value vector is extracted from queue of online generated Pareto front and after selecting next action according to above mentioned rule, agent sees potentially next state s'. There are some extra operations to prevent agent to get stuck in the loop and make it to get converged. First we need to mark visited states and crossing off them from the list of states which agent will go to by assigning them a big Q-value. Lines 8 to 21 of Algorithm 9 do that. Also in research it was found that there may be some duplicate Q-values which causes agent to get stuck in the infinite loop or navigate it to a wrong direction. So lines 22 to 30 agent looks up the next states of states in which there are duplicate Q-values and continues to search if there are duplicate values in subsequent states. Finally agent goes to the next state. For some problems agent does not need to look back to gather rewards and needs to proceed in depth in search space so there is no need to gravitate toward those states which visited before, so visited states is marked for this kind of problems and line 35 which nullifies the respective array is not executed.

It is also challenging to evaluate multi-objective algorithms without using benchmark problems. Prior to the presentation of the benchmark problems, the researchers mentioned the results of problems in their papers that had nothing in common with each other and could not be compared. Also, since the Pareto Front is not specified for many problems, there is no basis for performance appraisal. Therefore, the existence of standard problems to evaluate the performance of algorithms and compare them with each other seems necessary (Vamplew et al., 2011). There are some instances of benchmark problems; here we mention two cases that the proposed model has been implemented to solve them.

4.1 Deep sea treasure

This problem and its environment, proposed by Vamplew et al. (2011), is an episodic problem in which a submarine is looking for treasure in the deep sea. The problem has two objectives, one is to obtain more valuable treasures and the other is to minimize the time to collect those treasures. The environment of this problem is a gridworld consisting of 11 columns and 10 rows. As the distance from the initial state increases, the value of the treasures on the sea floor increases. The agent can move right, left, up, and down in this environment. With each move, the agent receives a reward vector with two members, one for the amount of treasure discovered and the other for the time. The reward for the treasure is zero where no treasure is discovered and is equal to the value of the treasure on steps where the treasure is found. The time with each move of the agent is considered as a penalty with a value of -1. The values of the other entries in the table are zero. Each episode either ends with the discovery of one of the treasures or after the agent has made the maximum allowed movement in each episode (maximum 1000 steps).

Figure 1 shows the problem environment: A represents the agent, black cells represent the seabed, and numbers in gray cells represent the value of the treasures. As Vamplew has suggested in Vamplew et al. (2011), this problem is designed to illustrate the limitations of the scalarization methods. Figure 2 demonstrates the Pareto front of the problem and indicates that the front is globally concave.

Algorithm 9 Policy Search in Offline Mode

Input: Q table and the Pareto front (i.e. online non-dominated solutions) generated during learning, and an initially empty visitedStates array.

Output: An array representing a set of solutions

$P \leftarrow$Insert the Pareto front into a Queue (P) and sort it based on preference or value $(p_1 < p_2 < \ldots < p_n)$
Initialize state s
while s is not terminal or **for** some specified steps:
    while Queue is not empty:
        $V \leftarrow P.dequeue()$// V stands for a solution (i.e. value-vector) in the Pareto front
        a $\leftarrow argmin_a min_{q \in Q(s,a)} |q - V|$
        $s' \leftarrow Q(s, a)$
    $qValueChangeCounter \leftarrow 0$
        while $s'$is in visitedStates:
            for each action a in s:
                $s' \leftarrow Q(s, a)$
                if $s'$is in visitedStates:
                    $Q(s, a) \leftarrow Q(s, a) + \infty$
                    $qValueChangeCounter \leftarrow qValueChangeCounter + 1$
                end if
            end for
            $a \leftarrow argmin_a min_{q \in Q(s,a)} |q - V|$
            if $qValueChangeCounter =$ number of possible actions in s:
                break // exit latest while loop
            end if
        end while
        if for at least two different actions in s there is the same q value such that: $argmin_a \; min_{q \in Q(s',a)} |q - V| = a_{i..n}$
            $s'' \leftarrow s'$
            repeat
                for each action a in s:
                    $a \leftarrow armi_a min_{q \in Q(s'',a)} |q - -V|$
                end for
                $s'' \leftarrow Q(s'', a)$
            until there are no different actions in s'' such that: $argmin_a min_{q \in Q(s'',a)} |q - V| = a_i..n$
                end if
            $s \leftarrow s'$
            visitedStates $\leftarrow$ visitedStates $+ s$
            if any objective found:
                solutions solutions + objective
                visitedStates empty // this part is optional and more suitable for problems in which agent has to look back to search for more goals before it reaches the final state.
                end if
    end while
end while
return solutions

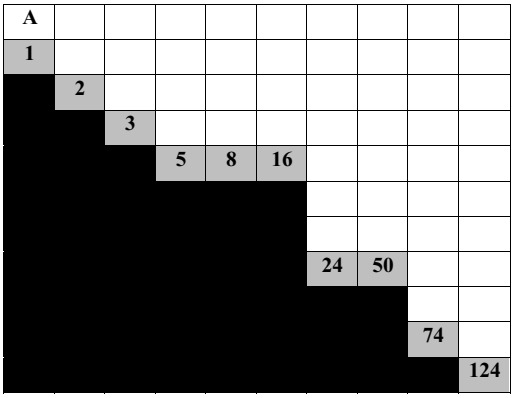

Figure 1. *Deep sea treasure.*

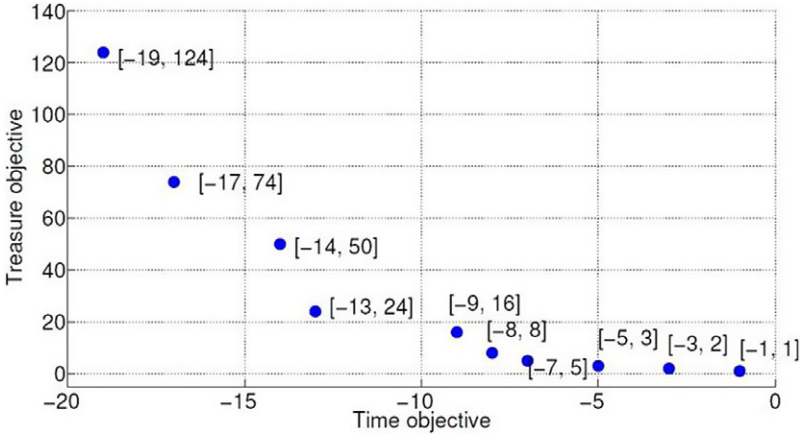

Figure 2. *Pareto front of deep sea treasure.*

When assessing single-policy methods, they must be evaluated across a range of different weight settings. For the algorithms considered in this study, the weight for each objective lies in the range of 0 to 1, and their sum is equal to 1. So if we sample this weight space at a distance of 0.1, there are a total of 11 different sets of weight coefficients. In first version of proposed algorithm (Algorithm 9), according to (37) all 11 weight preferences are given to the algorithm as criteria together and simultaneously and there is no need for the user to specify them.

It should be noted that in linear and Chebyshev scalarization methods, the weights associated with each objective are set in the scalarization functions defined in (12) and (17) respectively:

$$\mathbf{C_1}[s,a] = MAX(0.0 \quad \mathbf{Q_1}[s,a], 1.0 \quad \mathbf{Q_2}[s,a])$$
$$\mathbf{C_2}[s,a] = MAX(0.1 \quad \mathbf{Q_1}[s,a], 0.9 \quad \mathbf{Q_2}[s,a])$$
$$\mathbf{C_3}[s,a] = MAX(0.2 \quad \mathbf{Q_1}[s,a], 0.8 \quad \mathbf{Q_2}[s,a])$$
$$\mathbf{C_4}[s,a] = MAX(0.3 \quad \mathbf{Q_1}[s,a], 0.7 \quad \mathbf{Q_2}[s,a])$$
$$\mathbf{C_5}[s,a] = MAX(0.4 \quad \mathbf{Q_1}[s,a], 0.6 \quad \mathbf{Q_2}[s,a])$$
$$\mathbf{C_6}[s,a] = MAX(0.5 \quad \mathbf{Q_1}[s,a], 0.5 \quad \mathbf{Q_2}[s,a])$$
$$\mathbf{C_7}[s,a] = MAX(0.6 \quad \mathbf{Q_1}[s,a], 0.4 \quad \mathbf{Q_2}[s,a])$$
$$\mathbf{C_8}[s,a] = MAX(0.7 \quad \mathbf{Q_1}[s,a], 0.3 \quad \mathbf{Q_2}[s,a])$$

$$\begin{aligned}
\mathbf{C_9}[s,a] &= MAX(0.8 \quad \mathbf{Q_1}[s,a], 0.2 \quad \mathbf{Q_2}[s,a]) \\
\mathbf{C_{10}}[s,a] &= MAX(0.9 \quad \mathbf{Q_1}[s,a], 0.1 \quad \mathbf{Q_2}[s,a]) \\
\mathbf{C_{11}}[s,a] &= MAX(1.0 \quad \mathbf{Q_1}[s,a], 0.0 \quad \mathbf{Q_2}[s,a])
\end{aligned} \tag{37}$$

In the second version of proposed algorithm (Algorithm 9), for example, if the user has a weight preference of 0.2 for the time and 0.8 for the treasure, Algorithm 8 forms two decision criteria for TOPSIS as follows:

$$\begin{aligned}
\mathbf{C_1}[s,a] &= 0.2 \quad \mathbf{Q_1}[s,a] \\
\mathbf{C_2}[s,a] &= 0.8 \quad \mathbf{Q_2}[s,a]
\end{aligned} \tag{38}$$

The two criteria C_1 and C_2 for each decision or action that the agent has to choose are thereby quantified, and as there are four actions in the DST environment, the 4×2 decision matrix is formed.

$Q1\ [s,\ a]$ is the value of Q table related to the time objective and associated state s and action a and $Q2\ [s,\ a]$ is the value of table Q related to the treasure objective and associated state s and action a.

In this section, we compare the performance of the TOPSIS Q-Learning algorithm with the Linear Scalarization and the Chebyshev algorithms in DST environment. It should be noted that changes in exploration rate, number of episodes and learning rate affect the results and the results may differ from what is stated in other publications. In other papers, such as Moffaert et al. (2013), the number of episodes was considered less, but the Pareto front did not appear in the output. Here, because we are interested in having a solution, we considered the number of episodes to be high. All three algorithms were implemented with learning parameters including number of episodes = 10 000, maximum time step per episode = 1000, learning rate (α) = 0.9, discount factor (γ) = 0.8. Exploration rate (ϵ) decays during learning meaning that the exploration rate at the beginning is 100% and it gradually decreases to zero by the end of the algorithm.

The three algorithms of Linear Scalarization, Chebyshev and TOPSIS Q-Learning were run 10 times for each of the 11 weight preferences, which is equal to 110 times. It is worth mentioning since all 11 weight preferences are injected to the first version of the TOPSIS algorithm at once and we don't need to run algorithm for each combination of weights separately, it is significantly cost-effective. However we ran the first version of the TOPSIS as many times as other algorithms to have a fair comparison. In the first version of the TOPSIS algorithm, each weight preference uses other weights to support uncertainty and variation. The criteria for the first version of the algorithm are formed as (37) and for the second version they are like (38).

Then the hypervolume of offline rewards is calculated and averaged for the above specified number of experiments (i.e. runs). The hypervolume and also the regret mean is shown in Table 1. For the regret measure according to (35) after each episode, rewards are accumulated. Once the algorithm terminates, the accumulated reward is subtracted from the product of the total number of episodes and the optimal solution. Optimal solution is calculated as follows: first choose the maximum and minimum bounds for each objective either from true front or estimated from previous experiments, second create a hyperplane passing through the minimum and maximum bounds of the front, third calculate a point called P_h on the hyperplane for a given set of preference weights and specify them as extrema of hyperplane, forth find the point P_t on the Pareto front which minimizes $|P_h - P_t|$ (Vamplew et al., 2011).

In order to include all weights in the regret measure, the first version of the TOPSIS algorithm was run for all 11 weights so that all weights were taken into account in calculating the amount of the regret and especially the mean reward of each optimal solution defined in the regret formula.

Table 1 shows the average performance of the algorithms. Since LS and Chebyshev algorithms indicated poor hypervolume results with Algorithm 9 policy search, their greedy operators also have been included. It can be seen that both versions of the TOPSIS algorithms have higher values of hypervolume than the other two algorithms. But in terms of the regret measure Chebyshev algorithm outperformed the other two algorithms. In the case of cardinality both TOPSIS versions found maximum possible solutions in the Pareto front considering the total runs of the experiments.

Table 1. *Statistics of average performance of algorithms in DST environment*

Algorithm	Offline Hypervolume Mean	Regret Mean	Cardinality
LS (Algorithm 9 policy search)	161.82	509 684.578	3
LS (Arg max-greedy operator)	831.98	509 321.67	9
Chebyshev (Algorithm 9 policy search)	497.58	297 118.91	3
Chebyshev (Arg min-greedy operator)	899.22	**297 118.91**	5
TOPSIS Q-Learning v.1 (Algorithm 9 policy search)	**1122.56**	413 767.35	**10**
TOPSIS Q-Learning v. 2 (Algorithm 9 policy search)	982.93	531 145.1	**10**

Table 2. *T-test for comparing LS and Chebyshev algorithms average offline hypervolumes with TOPSIS Q-Leaning algorithms in DST environment*

Algorithms / Statistics	Linear Scalarization (Arg max-greedy operator) vs.		Chebyshev (Arg min-greedy operator) vs.	
	TOPSIS Q-Learning version 1	TOPSIS Q-Learning version 2	TOPSIS Q-Learning version 1	TOPSIS Q-Learning version 2
Degree of freedom	110	197	110	215
Hypotheses (for each comparison)	$H_0 = \mu_{LS} = \mu_{TOPSIS}$ $H_A = \mu_{LS} < \mu_{TOPSIS}$		$H_0 = \mu_{Chebyshev} = \mu_{TOPSIS}$ $H_A = \mu_{Chebyshev} < \mu_{TOPSIS}$	
$P(T <= t)$ one-tail	4.94E-14	1.97E-4	3.17E-13	0.01
Calculated t-value	−8.51	−3.61	−8.15	−2.29
Critical t-value one-tail	1.65	1.65	1.66	1.65

In order to be able to infer the results of Table 1 statistically, we can use the t-test to compare the means of the hypervolumes and the regrets of the algorithms to see if there is a significant difference between them. The results of t-test related to comparisons of best hypervolume values of the Linear Scalarization and the Chebyshev algorithms (i.e. acquired by the greedy operators) with two versions of the TOPSIS Q-Learning algorithm are presented in Table 2. As shown in Table 2 null hypothesis (H_0) is considered as the equality of means of the hypervolumes of the LS and the TOPSIS Q-Learning, and alterative hypothesis (H_A) is the assumption of superiority or inferiority of TOPISIS Q-Learning algorithms against LS or Chebyshev algorithms and the same definition also applies to Table 3 and for the regret values. Our significance level is $\alpha = 0.05$ and according to t-test procedure if p-value (p-value is the probability that the results obtained from our data occurred by chance) is less than α or absolute value of calculated t-value is greater than critical t-value, the null hypothesis is rejected and according to the alternative hypothesis, it can be statistically inferred that the mean of the hypervolume of the compared algorithms is significantly less than TOPSIS Q-Learning. The same principle applies to Table 3 for the regret measure. As it can be seen in Table 2 absolute values of all calculated t-values are greater than corresponding critical t-values and consequently all p-values are less than corresponding $\alpha = 0.05$, so null hypothesis is rejected and we can infer that both versions of the TOPSIS Q-Learning algorithms performed significantly better than the other two algorithms in terms of the hypervolume mean. Table 3 demonstrates that since the absolute value of calculated t-value related to comparison of the Linear Scalarization and first version of TOPSIS algorithm is greater than its corresponding critical t-value and p-value of t-test is less than $\alpha = 0.05$, null hypothesis is rejected and it can be inferred the average regret values of first version of the TOPSIS algorithm is significantly less than the Linear Scalarization algorithm. But about comparison of second version of the TOPSIS Q-Learning and the

Table 3. *T-test for comparing LS and Chebyshev algorithms average regret with TOPSIS Q-Learning algorithms in DST environment*

Algorithms / Statistics	Linear Scalarization (Arg max-greedy operator) vs.		Chebyshev (Arg min-greedy operator) vs.	
	TOPSIS Q-Learning version 1	TOPSIS Q-Learning version 2	TOPSIS Q-Learning version 1	TOPSIS Q-Learning version 2
Degree of freedom	209	209	214	213
Hypotheses (for each comparison)	$H_0 = \mu_{LS} = \mu_{TOPSIS}$ $H_A = \mu_{LS} > \mu_{TOPSIS}$		$H_0 = \mu_{Chebyshev} = \mu_{TOPSIS}$ $H_A = \mu_{Chebyshev} < \mu_{TOPSIS}$	
$P(T <= t)$ one-tail	0.01	0.31	0.0046	1.9E-7
Calculated t-value	2.21	−0.5	−2.63	−5.24
Critical t-value one-tail	1.65	1.65	1.65	1.65

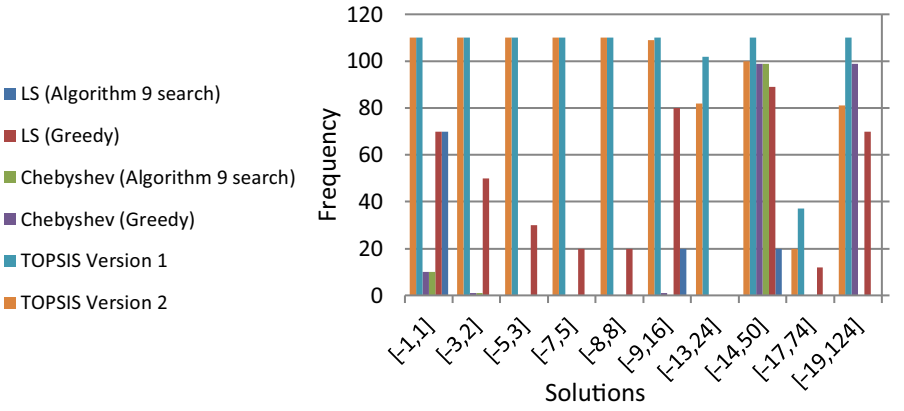

Figure 3. *Chart of frequency of all offline results discovered after learning process is finished in DST environment.*

Linear Scalarization there is no significant difference between regret values of these algorithms considering lacking enough evidence to reject the null hypothesis. In the case of comparison of the Chebyshev algorithm with both versions of the TOPSIS Q-Learning, null hypothesis is rejected because as can be seen in the Table 3 absolute values of calculated t-values are greater than critical t-values and p-values are less than $\alpha = 0.05$, so according to alternative hypothesis ($H_A = \mu_{Chebyshev} < \mu_{TOPSIS}$) Chebyshev algorithm has much less regret values.

As mentioned before it is necessary for the agent to replicate the results achieved during learning in offline mode to ensure that the rewards are nonrandom and obtained policy is reliable. Even though in some runs of LS and TOPSIS algorithms more rewards have been acquired, in Table 4 offline results which occurred in every run of the algorithms have been presented and more details has been provided in Figure 3.

As it can be seen in Table 4 and Figure 3 LS algorithm could not find any desirable solution with Algorithm 9 policy search except for weight [0,1]. But with greedy operator it found all solutions except [−13, 24]. Solution with the value [−17, 74] is absent from Table 4 because it failed to be present in all runs of a specific weight. LS is able to find closest optimal Pareto front alongside other solutions at all weights except [0.3, 0.7], [0.4, 0.6], and [0.8, 0.2].The Chebyshev with Algorithm 9 search policy finds closest optimal policy only at weights [0.5, 0.5], [0.6, 04], [0.7, 03] and [0,1]. The Chebyshev with greedy policy could not find solution corresponding with weights [0.3, 07], [0.4, 0.6], [0.8, 0.2], [0.9, 0.1]. As

Table 4. *Offline results occurred in every run of all algorithms in DST environment*

Weight [time, treasure]	Closest Optimal pareto front	LS (Algorihm 9 policy search)	LS (Arg max-greedy operator)	Chebyshev (Algorithm 9 policy search)	Chebyshev (Arg min-greedy operator)	TOPSIS Q-Learning Version 1	TOPSIS Q-Leaning Version 2
[0,1]	[−19, 124]	[−14, 50]	[−14, 50], [−19, 124]	[−15, 24], [−14, 50]	[−14, 50], [−19, 124]	[−1, 1], [−3, 2], [−5, 3], [−7, 5], [−8, 8], [−9, 16], [−13, 24], [−14, 50], [−19, 124]	[−1, 1], [−3, 2], [−5, 3], [−7, 5], [−8, 8], [−9, 16], [−14, 50], [−19,124]
[0.1,0.9]	[−19, 124]	[−14, 50]	[−14, 50], [−19, 124]	[−15, 24], [−14, 50]	[−14, 50], [−19, 124]	[−1, 1], [−3, 2], [−5, 3], [−7, 5], [−8, 8], [−9, 16], [−13, 24], [−14, 50], [−19, 124]	[−1, 1], [−3, 2], [−5, 3], [−7, 5], [−8, 8], [−9, 16], [−14, 50]
[0.2,0.8]	[−19, 124]	[−9, 16]	[−9, 16], [−14, 50], [−19, 124]	[−15, 24], [−14, 50]	[−14, 50], [−19, 124]	[−1, 1], [−3, 2], [−5, 3], [−7, 5], [−8, 8], [−9, 16], [−13, 24], [−14, 50], [−19, 124]	[−1, 1], [−3, 2], [−5, 3], [−7, 5], [−8, 8], [−9, 16], [−14, 50]
[0.3,0.7]	[−17, 74]	[−9, 16]	[−9, 16], [−14, 50], [−19, 124]	[−15, 24], [−14, 50]	[−14, 50], [−19, 124]	[−1, 1], [−3, 2], [−5, 3], [−7, 5], [−8, 8], [−9, 16], [−13, 24], [−14, 50], [−19, 124]	[−1, 1], [−3, 2], [−5, 3], [−7, 5], [−8, 8], [−9, 16], [−14, 50]
[0.4,0.6]	[−17, 74]	[−1, 1]	[−1, 1], [−9, 16], [−14, 50], [−19, 124]	[−15, 24], [−14, 50]	[−14, 50], [−19, 124]	[−1, 1], [−3, 2], [−5, 3], [−7, 5], [−8, 8], [−9, 16], [−13, 24], [−14, 50], [−19, 124]	[−1, 1], [−3, 2], [−5, 3], [−7, 5], [−8, 8], [−9, 16], [−14, 50]
[0.5,0.5]	[−14, 50]	[−1, 1]	[−1, 1],[−3, 2], [−9, 16], [−14, 50]	[−15, 24], [−14, 50]	[−14, 50], [−19, 124]	[−1, 1], [−3, 2], [−5, 3], [−7, 5], [−8, 8], [−9, 16], [−13, 24], [−14, 50], [−19, 124]	[−1, 1], [−3, 2], [−5, 3], [−7, 5], [−8, 8], [−9, 16], [−13, 24], [−14, 50]

Table 4. *Offline results occurred in every run of all algorithms in DST environment*

Weight [time, treasure]	Closest Optimal pareto front	LS (Algorihm 9 policy search)	LS (Arg max-greedy operator)	Chebyshev (Algorithm 9 policy search)	Chebyshev (Arg min-greedy operator)	TOPSIS Q-Learning Version 1	TOPSIS Q-Leaning Version 2
[0.6,0.4]	[−14, 50]	[−1, 1]	[−1, 1],[−3, 2], [−9, 16], [−14, 50], [−19, 124]	[−15, 24], [−14, 50]	[−14, 50], [−19, 124]	[−1, 1], [−3, 2], [−5, 3], [−7, 5], [−8, 8], [−9, 16], [−13, 24], [−14, 50], [−19, 124]	[−1, 1], [−3, 2], [−5, 3], [−7, 5], [−8, 8], [−9, 16], [−14, 50]
[0.7,0.3]	[−14, 50]	[−1, 1]	[−1, 1],[−3, 2], [−5, 3], [−9, 16], [−14, 50]	[−15, 24], [−14, 50]	[−14, 50], [−19, 124]	[−1, 1], [−3, 2], [−5, 3], [−7, 5], [−8, 8], [−9, 16], [−13, 24], [−14, 50], [−19, 124]	[−1, 1], [−3, 2], [−5, 3], [−7, 5], [−8, 8], [−9, 16], [−14, 50]
[0.8,0.2]	[−13, 24]	[−1, 1]	[−1, 1],[−3, 2], [−5, 3], [−7, 5], [−8, 8], [−9, 16]	[−15, 24], [−14, 50]	[−14, 50], [−19, 124]	[−1, 1], [−3, 2], [−5, 3], [−7, 5], [−8, 8], [−9, 16], [−13, 24], [−14, 50], [−19, 124]	[−1, 1], [−3, 2], [−5, 3], [−7, 5], [−8, 8], [−9, 16], [−13, 24], [−14, 50]
[0.9,0.1]	[−9, 16]	[−1, 1]	[−1, 1],[−3, 2], [−5, 3], [−7, 5], [−8, 8], [−9, 16]	[−15, 24], [−14, 50]	[−14, 50], [−19, 124]	[−1, 1], [−3, 2], [−5, 3], [−7, 5], [−8, 8], [−9, 16], [−13, 24], [−14, 50], [−19, 124]	[−1, 1], [−3, 2], [−5, 3], [−7, 5], [−8, 8], [−9, 16], [−14, 50]
[0,1]	[−1, 1]	[−1, 1]	[−1, 1]	[−1, 1]	[−1, 1]	[−1, 1], [−3, 2], [−5, 3], [−7, 5], [−8, 8], [−9, 16], [−13, 24], [−14, 50], [−19, 124]	[−1, 1], [−3, 2], [−5, 3], [−7, 5], [−8, 8]

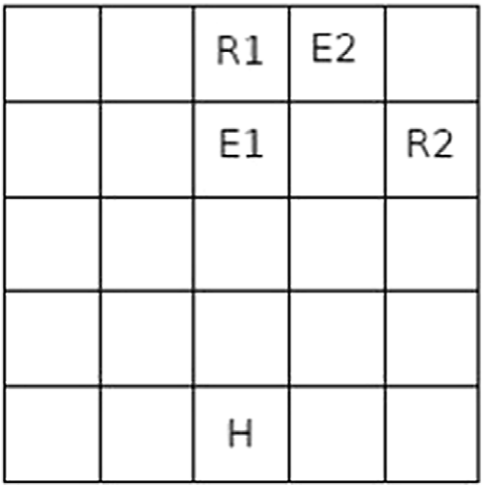

Figure 4. *Resource gathering problem environment (Gabor et al., 1998). H = home, R1 = gold resource, R2 = gem resource E1& E2 = enemy.*

it is observable in Figure 3 both Chebyshev experiments failed to generate different optimal results but [−14, 50] and [−19, 124]. The TOPSIS Q-Learning version 1 managed to find all optimal policies except for weights with values [0.3, 07] and [0.4, 0.6]. The algorithm also finds all other solutions except [−17, 74] alongside the closest optimal solution in every run of the experiments. Figure 3 demonstrates the TOPSIS Q-learning version 1 could find all optimal solutions during the experiments, although it found [−17, 74] less than the others because this reward did not occur in every run. The TOPSIS Q-Learning version 2 is unable to reach the optimal solution in every run at weights [0.1, 0.9], [0.2, 08], [0.3, 0.7], and [0.4, 0.6], as it can be seen in Figure 3 the algorithm found all solutions but [−17, 74] generated considerably less than the others. Looking at Figure 3 it can be said that in terms of spread among the solution space, the TOPSIS Q-learning version 1 has been more successful than the other algorithms as corresponding hypervolume value as a reflector of diversity in the Pareto front confirms this claim as well and all solutions but [−17, 74] are generated almost equally.

In order to ensure that the deepest treasure is found, the exploration parameter must be set to a high value number. In the experiment performed in Vamplew et al. (2017), the exploration rate ($\varepsilon = 0.8$) or higher is used, which is decayed during learning. It should be mentioned these results are related to the TLQ algorithm, but the authors of the article observed similar results for the LS algorithm (Roijers et al., 2017). Therefore, if the exploration is not sufficient enough for the algorithm to find the most valuable treasure located in the convex area of the Pareto front, the algorithm will converge to one of the available solutions in the concave region. But it should be noted that the solutions located in the concave region are not really optimal, because the agent has not actually succeeded in finding the original optimal solution, which is located in a convex hull and is the most valuable. At the same time, the solution found is not entirely reliable, because the agent may reach the same solution in the concave region in a percentage of the tests, while ideally always converges to that truly optimal solution.

4.2 Resource gathering

The environment of the problem as shown in Figure 4 is a table with 5 rows and 5 columns. The agent is in the home location and is allowed to move one of the four directions up, down, left, and right at any time. The resources that are gold and gem here are in specific places in the table, and the task of the agent is to collect one or both of the resources and return home. The agent receives a (+1) reward for each resource it returns home with.

There are also two locations in the environment where the enemy may attack the agent with a 10% probability. In case of an attack, the agent loses a resource if it has any and is sent home with a penalty of −1. The reward vector is created as [attack, gold, gem] that there will be four possible states when returning to the home and a zero reward state [0, 0, 0] in other time steps (Gabor et al., 1998):

- [−1, 0, 0]: Agent being attacked
- [0,0,1]: Return to the home with gold and without gem
- [0,0,1]: Return to the home with gem and without gold
- [0,1,1]: Return to the home with both gold and gem

This problem is mostly continuous and non-episodic in nature (Vamplew et al., 2011) but since all rewards are concentrated in the home location, its structure can be considered and implemented episodically.

Most of the researches discuss the total of rewards taken while learning and discovering the optimal policies to reach the resources. In the implementation of the algorithm, according to what was suggested in some papers, the total reward received during each episode is divided by the number of allowed steps in each episode (here the number of allowed steps is considered 100). Given that the number of possible combinations of weight preferences for the problem with 0.1 distances between them is 64, we are interested to use the TOPSIS Q-Learning version 2 to apply each specific weight preference of the entire objective space to the problem. For example if the preference of the first objective, that is, the probability of the enemy not attacking, is assumed to be 0.1 and the preference of the second and third objectives, that is, gold and gem to be equal to 0.1 and 0.8 respectively, criteria will be like as follows:

$$\begin{aligned} \mathbf{C_1}[s,a] &= \underline{0.1}\ \mathbf{Q_1}[s,a] \\ \mathbf{C_2}[s,a] &= \underline{0.1}\ \mathbf{Q_2}[s,a] \\ \mathbf{C_3}[s,a] &= \underline{0.8}\ \mathbf{Q_3}[s,a] \end{aligned} \tag{39}$$

In this section, the TOPSIS Q-Learning algorithm is compared only with the LS algorithm and the comparison with the Chebyshev algorithm is omitted, because in Van Moffaert's doctoral dissertation, a comparison was made between the Linear Scalarization and the Chebyshev algorithms and it has been observed that Linear Scalarization has been more successful in terms of obtaining the hypervolume as well as in terms of optimal policies (Moffaert, 2014). According to a report in the same study, the Chebyshev algorithm was not able to discover a policy that could collect both gold and gem sources, while as we will see, both the TOPSIS and the Linear Scalarization algorithms (with Algorithm 9 policy search) are able to do that. Performance result of the TOPSIS Q-Learning algorithm is presented in Table 5. Different weight preferences are given to the second version of the TOPSIS Q-Learning algorithm as decision criteria analogous to relations defined in (39). The hypervolume measure whose reference point is [−2, −1, −1] was applied to final offline rewards to compare two algorithms performance. Also, because the use of the regret measure and determining the maximum and minimum bounds for each objective in a three-dimensional environment is more complex than the two-dimensional environment presented in the paper of Vamplew et al. (2011) it was not used for the problem. Instead, the hypervolume was calculated for cumulative online reward vectors to assess algorithms online performance.

Both Algorithms were run 10 times for each weight preference set in which their elements sum equals to 1 and each weight element is greater than zero, so total number of runs would be 320. Both algorithms were run with the following learning parameters. Number of episodes = 100, maximum steps per episode = 100, learning rate (α) = 0.1, discount factor (γ) = 0.95, exploration rate (ϵ) = 0.9. As demonstrated in the Figure 5, LS with greedy operator gained minimum penalty rewards, that is, [−1, 0, 0] and maximum gem reward but it could not obtain rewards containing both gold and gem. LS with Algorithm 9 policy search managed to discover [0,1,1] reward including gold and gem at the same time. The TOPSIS Q-Learning gained the most penalties and the least solo gold rewards. But it obtained maximum rewards of gems and dominated reward, that is, [0,1,1] compared to LS.

Table 5. *Average performance of algorithms in resource gathering problem*

Algorithm	Offline Hypervolume Mean	Average Online Vectorial Rewards	Hypervolume of Online Cumulative Vectorial Rewards
Linear Scalarization (Algorithm 9 policy search)	**4.553**	[−**0.33**, 53, 34]	3.64
Linear Scalarization (Greedy operator)	3.893		
TOPSIS Q-Learning Version 2	4.548	[−0.38, **55**, **37**]	3.62

Table 6. *T-test for statistically comparing Linear Scalarization algorithm average offline and online hypervolume with TOPSIS Q-Leaning algorithm in DST environment*

Algorithms / Statistics	Offline Hypervolume of TOPSIS Q-Learning Version 2 vs		Hypervolume of Online Cumulative Rewards of TOPSIS Q -Learning Version 2 vs.
	LS (Algorithm 9 policy search)	LS (Arg-max operator)	LS
Hypotheses	$H_0 = \mu_{LS} = \mu_{TOPSIS}$	$H_0 = \mu_{LS} = \mu_{TOPSIS}$	$H_0 = \mu_{LS} = \mu_{TOPSIS}$
(for each comparison)	$H_0 = \mu_{LS} > \mu_{TOPSIS}$	$H_A = \mu_{LS} < \mu_{TOPSIS}$	$H_A = \mu_{LS} < \mu_{TOPSIS}$
Degree of freedom	569	423	585
$P(T <= t)$ one-tail	0.48	6.24E-17	0.42
Calculated t-value	0.046	−8.63	−0.21
Critical t-value one-tail	1.65	1.65	1.65

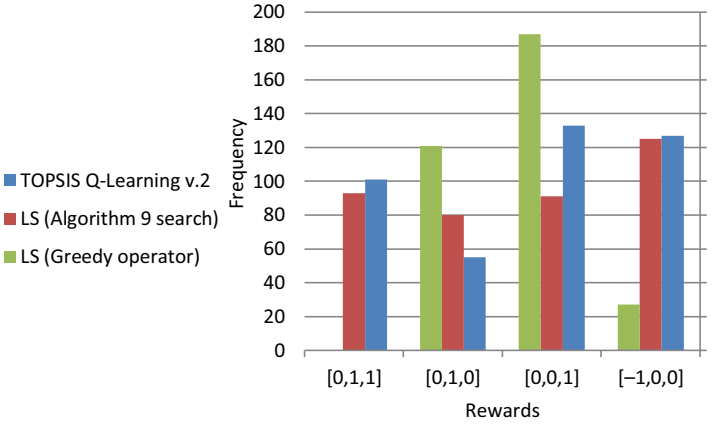

Figure 5. *Chart of frequency of offline rewards for TOPSIS Q-Learning and Linear Scalarization in resource gathering problem.*

The hypervolume of offline rewards calculated after complete execution of the algorithms for above specified experiment repeat times is presented in Table 5. It can be seen here that unlike the DST environment, the LS algorithm performed better with Algorithm 9 policy search compared to using the greedy operator. The LS algorithm has a larger hypervolume with a small difference compared to the TOPSIS Q-Learning. So we should refer to t-test to see how significant this comparison is. The average

vectorial rewards show that the TOPSIS Q-Learning algorithm has performed better than the Linear Scalarization in gaining gold and gem resources, although earning more rewards comes at a higher cost meaning receiving more penalties and gaining returns of $[-1, 0, 0]$ during learning. However, to ensure that the relationship follows Pareto dominance, the hypervolume has been calculated in Table 5 and it shows LS has a slightly larger hypervolume. But considering p-value and calculated t-value, null hypothesis is not rejected and t-test proves this difference is not significant. So these algorithms are almost on the same level in terms of online performance.

The t-test was performed with the significance level (α) of 0.05. According to the p-value of the LS implemented with Algorithm 9 policy search which is greater than α and absolute value of calculated t-value which is smaller than critical t-value, null hypothesis is not rejected and we can conclude that the TOPSIS Q-Learning and the LS have the close performance in terms of average offline hypervolume in resource gathering environment. The TOPSIS Q-learning outperformed the LS greedy version according to p-value and calculated t-value.

5 Conclusion

In this research, a model was presented that greatly reduces the issue of the user being bound to determine all the weights or preferences and leaves this task to the algorithm. An online single-policy algorithm which is called TOPSIS Q-Learning was presented that is based on Q-Learning and its structure is similar to the Linear Scalarization, with the difference that the action selection mechanism is replaced by the TOPSIS in such a way that the values in Q Table for each objective in state s and for all actions in that state are given to the TOPSIS as criteria to select the optimal action. Also, to update the Q table for each objective, the TOPSIS action selection mechanism has been used. In first version of the proposed algorithm, all the weight preferences are generated automatically as decision criteria within the algorithm and there is no need for them to be determined by the user and with one run and considering all these preferences, the solution is generated. Since it contains whole spectrum of possible weights in objective space to support uncertainty it is suitable for circumstances in which the preferences for the user may not be completely definite or subject to change. The user may also want to train and use the model for a specific set of weights. In this regard in the second version of our algorithm, each weight preference is considered as a decision criterion for the TOPSIS so that the user can model the weight preferences around their priorities toward the objectives. We also developed a new policy search algorithm for offline mode especially designed for multi-objective space to replace arg-max operator which is more suitable for single-objective RL. It managed to discover considerable number of solutions in the Pareto front regarding closest optimal solution at every run.

To measure efficiency of the proposed algorithm and significance of the difference in the results of the compared algorithms the t-test was performed and it was found in DST environment the TOPSIS Q-Learning algorithms outperformed the LS and the Chebyshev algorithms in terms of hypervolume and consequently diversity of the Pareto front, but in the case of regret the Chebyshev algorithm has smaller values compared to other algorithms. In RG environment the TOPSIS Q-Learning algorithm and the LS implemented with Algorithm 9 policy search have the same performance according to the closeness of their hypervolume values and the t-test results. The TOPSIS Q-learning and LS with Algorithm 9 policy search were able to discover the rewards containing both gold and gem while LS with greedy operator was not. Also it is shown that the LS algorithm with Algorithm 9 policy search performed significantly better compared to offline greedy operator in RG environment.

Since algorithms in convex environments tend to generate more solutions located in convex regions, if according to user-defined preferences or regarding the situations in which all the priorities are considered, solutions could be evenly distributed on all regions, especially those located in non-convex ones, it can be said that the algorithm has performed well in terms of coverage of rewards or diversity of solutions. In this regard, the first version of the TOPSIS Q-Learning was able to perform well.

For future work, since adjusting learning parameters plays an important role in converging and finding optimal policies, it can be very useful to provide a model in this regard, that is, hyperparameter adjustment in the field of multi-objective reinforcement learning algorithms. Also, improving the convergence speed of the algorithm using deep learning methods and more adaptation to stochastic environments are the subjects of researches that can be considered.

Declaration of competing interests

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

Barrett, L. & Narayanan, S. 2008. Learning all optimal policies with multiple criteria. In *Proceedings of the 25th International Conference on Machine Learning*, New York, NY, USA, pp. 41–47.

Gabor, Z., Kalmar, Z. & Szepesvari, C. 1998. Multi-criteria reinforcement learning. In *The Fifteenth International Conference on Machine Learning*, San Francisco, CA, USA, pp. 197–205.

Geibel, P. 2006. Reinforcement learning for MDPs with Constraints. In *Machine Learning: ECML 2006*, Lecture Notes in *Computer Science*, vol. **4212**.

Hwang, C. L. & Yoon, K. 1981. *Multiple Attribute Decision Making: Methods and Applications*, Lecture Notes in Economics and Mathematical *Systems*. Springer-Verlag.

Hwang, C. L. & Yoon, K. 1981. *Multiple Attribute Decision Making: Methods and Applications*. Springer-Verlag.

Issabekov, R. and Vamplew, P. 2012. An empirical comparison of two common multiobjective reinforcement learning algorithms. In *AI 2012: Advances in Artificial Intelligence.* Lecture Notes in Computer Science, vol. **7691**, pp. 626–636.

Keeney, R. L. & Raiffa, H. 1976. *Decision with Multiple Objectives: Preferences and Value Tradeoffs*. Wiley.

MacCrimmon, K. R. & Toda, M. 1969. The experimental determination of indifference curves. *The Review of Economic Studies*, **36**(4), 433–450.

MacCrimmon, K. R. & Wehrung, D. A. 1977. *Trade-off Analysis: The Indifference and Preferred Proportions Approaches, Conflicting Objectives in Decisions*. Wiley, pp. 123–147.

Moffaert, K. V. 2014. Multi-criteria reinforcement learning for sequential decision making problems, Ph.D. dissertation, Dept. Comput. Sci., Vrije Universiteit Brussel., Brussels, Belgium.

Moffaert, K. V., Drugan, M. M. & Nowé, A. 2013. Scalarized multi-objective reinforcement learning: Novel design techniques. In IEEE ADPRL, Singapore, pp. 191–199.

Moffaert, K. V. & Nowé, A. 2014. Multi-objective reinforcement learning using sets of pareto dominating policies. *Journal of Machine Learning Research* **15**, 3483–3512.

Nguyen, T. T., Nguyen, N. D., Vamplew, P., Nahavandi, S., Dazeley, R. & Lim, C. P. 2020. A multi-objective deep reinforcement learning framework. *Engineering Applications of Artificial Intelligence* 96.

Roijers, D. M., Röpke, W., Nowe, A. & Radulescu, R. 2021. On following pareto-optimal policies in multi-objective planning and reinforcement learning. *Paper Presented at Multi-Objective Decision Making Workshop 2021.*

Roijers, D. M., Vamplew, P., Whiteson, S. & Dazeley, R. 2013. A survey of multi-objective sequential decision-making. *Journal of Artificial Intelligence Research* **48**(1), 67–113.

Roijers, D. M., Zintgraf, L. M., Libin, P. & Nowé, A. 2018. Interactive multi-objective reinforcement learning in multi-armed bandits for any utility function. In ALA Workshop at FAIM, vol. 8.

Roijers, D. M., Zintgraf, L. M., Libin, P., Reymond, M., Bargiacchi, E. & Nowé, A. 2020. Interactive multi-objective reinforcement learning in multi-armed bandits with gaussian process utility models. In *ECML-PKDD 2020: Proceedings of the 2020 European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases*.

Roijers, D. M., Zintgraf, L. M. & Nowé, A. 2017. Interactive thompson sampling for multi-objective multi-armed bandits. In *Algorithmic Decision Theory, ADT 2017*, Lecture Notes in Computer Science, vol. **10576**. Springer.

Sutton, R. S. and Barto, A. G. 1998. *Reinforcement Learning: An Introduction*. *Adaptive Computation and Machine Learning*. MIT Press.

Tsitsiklis, J. N. 1994. Asynchronous stochastic approximation and q-learning. *Journal of Machine Learning* **16**(3), 185–202.

Vamplew, P., Dazeley, R., Berry, A., Issabekov, R. & Dekker, E. 2011. Empirical evaluation methods for multiobjective reinforcement learning algorithms. *Machine Learning* **84**, 51–80.

Vamplew, P., Dazeley, R. & Foale, C. 2017. Softmax exploration strategies for multiobjective reinforcement learning. *Neurocomputing*, **263**, 74–86.

Vamplew, P., Issabekov, R., Dazeley, R., Foale, C., Berry, A., Moore, T. & Creighton, D. 2017. Steering approaches to Pareto-optimal multiobjective reinforcement learning. *Neurocomputing* **263**, 26–38.

Vamplew, P., Yearwood, J., Dazeley, R. & Berry, A. 2008. On the limitations of scalarization for multi-objective reinforcement learning of Pareto fronts. In *AI 2008: Advances in Artificial Intelligence.* Lecture Notes in Computer Science, vol. **5360**, pp. 372–378.

Watkins, C. 1989. *Learning from delayed rewards*, Ph.D. thesis, University of Cambridge, England.

Yoon, K. 1980. *Systems selection by multiple attribute decision making*, Ph.D. Dissertation, Kansas State University, Manhattan, Kansas.