Pruning Dominated Policies in Multiobjective Pareto Q-learning***

Lawrence Mandow and José-Luis Pérez-de-la-Cruz

Universidad de Málaga, Andalucía Tech, Departamento de Lenguajes y Ciencias de la Computación, Málaga, España.
lawrence|perez@lcc.uma.es

Abstract. The solution for a Multi-Objetive Reinforcement Learning problem is a set of Pareto optimal policies. MPQ-learning is a recent algorithm that approximates the whole set of all Pareto-optimal deterministic policies by directly generalizing Q-learning to the multiobjective setting. In this paper we present a modification of MPQ-learning that avoids useless cyclical policies and thus improves the number of training steps required for convergence.

1 Introduction

The problem of solving Multi-Objective Markov Decision Processes (MOMDPs) and more concretely the application of Reinforcement Learning techniques to this problem (MORL) has raised some interest in recent literature [1]. MORL algorithms can be single-policy and multi-policy [2]. The later try to approximate part or the whole set of Pareto optimal policies. There are currently very few multi-policy MORL algorithms; among them we will consider in this paper MPQ-learning [3]. It is an off-policy temporal-difference method. Off-policy methods are particularly interesting in multiobjective reinforcement learning, since they allow to learn a set of Pareto-optimal policies simultaneously. MPQ-learning was shown to solve standard problems from the benchmark proposed in [4]. However, there are efficiency issues due to the nature of the algorithm, that aims to learn *all* deterministic optimal policies (including non stationary ones). In this paper we propose a modification of MPQ-learning that controls the generation of cycles during the learning process and thus improves greatly its efficiency in terms of training steps.

The paper is structured as follows: in section 2 MPQ-learning is summary described and some problematic features are identified and discussed. Then we describe the proposed modifications (section 4) and present and discuss the results obtained in a set of experiments (section 5). Finally some conclusions are drawn.

* Supported by: the Spanish Government, Agencia Estatal de Investigación (AEI) and European Union, Fondo Europeo de Desarrollo Regional (FEDER), grant TIN2016-80774-R (AEI/FEDER, UE); and Plan Propio de Investigación de la Universidad de Málaga - Campus de Excelencia Internacional Andalucía Tech.

** The final authenticated version is available online at https://doi.org/10.1007/978-3-030-00374-6_23

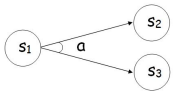

Fig. 1: Sample non-deterministic transition from state s_1 through action a.

2 MPQ-learning

2.1 The Algorithm

MPQ-learning [3] is an extension of Q-learning to multiobjective problems. The goal is to obtain the set of all Pareto-optimal deterministic policies. These include both stationary and non-stationary policies, as nonstationarity is an essential feature of multi-objective reinforcement learning problems.

A detailed description of MPQ-learning can be found in [3]. Let us briefly describe here the rationale of the algorithm in an intuitive way. A basic underlying idea of the algorithm (and also of the improvements presented later) is to reduce the set of candidate policies applying the principle of optimality. Just like Q-learning, MPQ-learning is an off-policy temporal-difference method. An agent interacts with the environment and learns the optimal expected long term rewards of its actions. Without loss of generality we shall assume that all rewards are to be maximized. Sets of labeled vectors $\mathbb{Q}(s,a)$ are learned for each state-action pair. Each labeled vector $\boldsymbol{q} \in \mathbb{Q}(s,a)$ estimates the expected vector reward when a particular Pareto-optimal policy is followed after choosing action a in state s. Additionally, set $\mathbb{V}(s')$ denotes the set of all Pareto-optimal vector estimates for all actions available to state s'. These sets are the multi-objective analogues of the scalar Q and V sets used in Q-learning.

An important feature of MPQ-learning is that it learns a partial model of the environment. This is in contrast with Q-learning, which is a model-free algorithm. The partial model is necessary for two main reasons. In the first place, it helps to reduce the number of candidate policies tracked by the algorithm. Additionally, once learning is over, the model can be used to recover the actions associated to a particular Pareto-optimal policy chosen by the agent. Therefore, in MPQ-learning each vector estimate $\boldsymbol{q} \in \mathbb{Q}(s,a)$ is labeled with a set of *indices* P, where each index $(s',i) \in P$ indicates that $\boldsymbol{q}$ is updated from the i-th vector in $\mathbb{V}(s')$.

Let us assume that at step n, transition $s, a, \boldsymbol{r'}, s'$ is stochastically performed by the agent. The updating expression for MPQ-learning is,

$$\mathbb{Q}_n(s,a) = \begin{cases} \mathbb{N}_{n-1}(s,a) \cup \mathbb{U}_{n-1}(s,a) \cup \mathbb{E}_{n-1}(s,a) \\ \qquad \text{if } s = s_n \wedge a = a_n \\ \mathbb{Q}_{n-1}(s,a) \\ \qquad \text{otherwise} \end{cases} \tag{1}$$

The updated Q-set results from the union of three sets: $\mathbb{N}$, $\mathbb{U}$ and $\mathbb{E}$. These depend on the nature of the transition $s, a, \boldsymbol{r}, s'$. Let us consider the sample stochastic transition shown in figure 1.

Let us assume that action a is selected *for the first time* from state s_1, and the agent transitions to state s_2 obtaining some vector reward $\boldsymbol{r}$. Let us further assume that currently there is a single optimal estimate for s_2, i.e. $\mathbb{V}(s_2) = \{\boldsymbol{q_1^2}\}$. According to the principle of optimality, optimal policies in s_1 can be obtained by combining optimal policies from s_2 and s_3. Since $\mathbb{Q}_{n-1}(s_1, a)$ is empty, a *new* estimate is added to $\mathbb{Q}(s_1, a)$ for each estimate in $\mathbb{V}(s_2)$. In this case $\mathbb{Q}_n(s_1, a) = \{(\boldsymbol{q_1^1}, (s_2, 1))\}$, indicating that $\boldsymbol{q_1^1}$ is related to vector 1 in $\mathbb{V}(s_2)$ as indicated by the formula,

$$\begin{aligned}\mathbb{N}_{n-1}(s,a) = \{((1-\alpha_n)\boldsymbol{q} + \alpha_n[\boldsymbol{r}_n + \gamma \boldsymbol{v}_j], P \cup \{(s', j)\} \mid \\ (\boldsymbol{q}, P) \in \mathbb{Q}_{n-1}(s,a) \wedge \boldsymbol{v}_j \in \mathbb{V}_{n-1}(s') \\ \wedge\, s' \not\sqsubset \mathbb{Q}_{n-1}(s,a)\}\end{aligned} \tag{2}$$

where $s' \not\sqsubset \mathbb{Q}_{n-1}(s, a)$ denotes that state s' does not participate in any index in $\mathbb{Q}_{n-1}(s, a)$, and α_n and γ are the usual *learning rate* and *discount factor* respectively.

Notice that if transition $s_1, a, \boldsymbol{r}, s_3$ is performed afterwards *for the first time*, and there is just one optimal estimate in $\mathbb{V}(s_3) = \{\boldsymbol{q_1^3}\}$, then $\boldsymbol{q_1^1}$ would be updated from $\boldsymbol{q_1^3}$ and its set of indices extended to $(s_2, 1), (s_3, 1)$, indicating that $\boldsymbol{q_1^1}$ is obtained combining estimate 1 from $\mathbb{V}(s_2)$, and estimate 1 from $\mathbb{V}(s_3)$.

When the transition $s_1, a, \boldsymbol{r}, s_2$ is repeated, vector $\boldsymbol{q_1^1}$ will be *updated* from $\boldsymbol{q_1^2}$, provided both are still part of $\mathbb{Q}(s_1, a)$ and $\mathbb{V}(s_2)$ respectively. Analogously for transition $s_1, a, \boldsymbol{r}, s_3$. The general expression for this situation is given by the formula,

$$\begin{aligned}\mathbb{U}_{n-1}(s,a) = \{((1-\alpha_n)\boldsymbol{q} + \alpha_n[\boldsymbol{r}_n + \gamma \boldsymbol{v_j}], P) \mid \\ (\boldsymbol{q}, P) \in \mathbb{Q}_{n-1}(s,a) \wedge (s', j) \in P \\ \wedge\, \boldsymbol{v}_j \in \mathbb{V}_{n-1}(s')\}\end{aligned} \tag{3}$$

Finally, let us consider that transition $s_1, a, \boldsymbol{r}, s_2$ is repeated one more time, but now $\mathbb{V}(s_2) = \{\boldsymbol{q_1^2}, \boldsymbol{q_2^2}\}$ is populated by two estimates: the previous one $\boldsymbol{q_1^2}$ (with possibly now a more accurate value), and a new, or *extra* one $\boldsymbol{q_2^2}$ that was not present the last time transition $s_1, a, \boldsymbol{r}, s_2$ was traversed. As described in formula 3, the value of estimate $\boldsymbol{q_1^1}$ will be rigthfully updated with that of $\boldsymbol{q_1^2}$, as indicated by the indices of the former. However, an additional estimate will be included in $\mathbb{Q}(s_1, a)$ reflecting a possibly new optimal policy that can be followed combining results obtained from the second vector in $\mathbb{V}(s_2)$, and the first one (and only known to date) in $\mathbb{V}(s_3)$. This operation is given in the general case by the formula,

$$\begin{aligned}\mathbb{E}_{n-1}(s,a) = \{(\alpha_n[\boldsymbol{r}_n + \gamma \boldsymbol{v_j}], (P \setminus s') \cup \{(s', j)\}) \mid \\ \boldsymbol{v}_j \in \mathbb{V}_{n-1}(s') \;\wedge\; s' \sqsubset \mathbb{Q}_{n-1}(s,a) \wedge \\ (s', j) \not\sqsubset \mathbb{Q}_{n-1}(s,a) \wedge \exists \boldsymbol{q}(\boldsymbol{q}, P) \in \mathbb{Q}_{n-1}(s,a)\}\end{aligned} \tag{4}$$

where $(s', j) \not\sqsubset \mathbb{Q}_{n-1}(s, a)$ denotes that index (s, a) does not appear in the index set of any vector in $\mathbb{Q}_{n-1}(s, a)$.

Let us complete the description of the algorithm with the formal definition of the $\mathbb{V}(s)$ sets,

$$\mathbb{V}(s) = \mathrm{ND} \bigcup_{a \in A} \{\boldsymbol{q} \mid (\boldsymbol{q}, P) \in \mathbb{Q}(s, a)\} \tag{5}$$

where $ND(\mathcal{X})$ denotes the set of nondominated, (or Pareto-optimal) vectors in set $\mathcal{X}$.

In summary, optimal policies for each state-action pair are obtained combinig optimal policies from reachable states. MPQ-learning incrementally builds a partial model of the environment through sets of indices, ensuring that each estimate is updated incrementally always from the same (optimal) policies. This reduces the number of combinations of policies tracked by the algorithm, increasing its efficiency.

2.2 Efficiency issues with MPQ-learning

By considering the algorithm at work, some issues can be detected. First of all, the "update" operation $\mathbb{U}$ is defined in a very conservative way respect to the use of memory. Notice that in the former definition of the $\mathbb{U}(s, a)$ set only optimal values in successors s' of s (i. e., values in $\mathbb{V}(s')$) are considered. In this way, new and potentially useful information in s' is not propagated to s.

Related to this point is the generation of useless cyclical (nonstationary) policies in intermediate steps of the algorithm. Indeed, since some of them can be optimal, the consideration of cyclical policies is unavoidable and, in fact, necessary. However, many cyclical policies appearing during the execution are mere artifacts due to the delayed propagation of true Q-values. These policies will be eventually discarded, since they will have –sooner or later– a dominated Q-value; but until that moment, they populate $\mathbb{Q}$ sets and slow down greatly the execution of the algorithm.

Finally, these spurious policies and Q-values difficult considerably an adequate finalization of the algorithm. At any given moment, it is difficult to identify them inside $\mathbb{Q}$ sets and hence it is difficult to output an useful approximation of the real set of optimal policies.

3 Example

Let us illustrate the concerns raised in section 2.2. Let us consider a simple state space consisting of only two states, s_1 and s_2. Episodes start always at s_1 and terminate at s_2. There are two actions available at s_1. Action a_1 loops back deterministically to s_1, and the agent receives a vector reward of $(-1, 0)$. Action a_2 leads deterministically to s_2, and the agent receives a vector reward of $(-1, 0)$. At s_2 the episode ends, and the agent receives a reward of $(-1, 1)$. Let us assume all $\mathbb{Q}(s, a)$ sets are initialized with a zero vector, and $\mathbb{V}(s_2) = \{\boldsymbol{q_1^2} = (-1, 1)\}$. The agent maximizes all rewards.

There is just one optimal policy in this problem with value $(-2, 1)$, i.e. at s_1 the agent chooses action a_2 leading to s_2. However, there are infinitely many dominated nonstationary policies: looping with a_1 a number of times and then choosing a_2.

Let us assume that the process starts, and fixed values of $\alpha = 0'1$ and $\gamma = 1$ are used. We further assume that the agent chooses action a_2 leading to transition

$s_1, a_2, \boldsymbol{r}, s_2$. Through application of the new rule in MPQ-learning (see formula 2), and since $\mathbb{V}(s_2) = \{\boldsymbol{q_1^2} = (-1, 1)\}$, a new value is calculated in $\mathbb{Q}_1(s_1, a_2) = \{(\boldsymbol{q_1^1}, (s_2, 1)\}$. The new vector $\boldsymbol{q_1^1}$ stands for the optimal policy, and is linked to the first (and only) estimate in s_2. With the provided data, the initial value for the estimate would be $\boldsymbol{q_1^1} = (-0.2, 0.1)$.

Let us assume the process starts again and the agent performs transition $s_1, a_1, \boldsymbol{r}, s_1$. Now $\mathbb{Q}_2(s_1, a_2) = \mathbb{Q}_1(s_1, a_2)$, but through application of the *new* rule, and since $\mathbb{V}(s_1) = \{\boldsymbol{q_1^1} = (-0.2, 0.1)\}$, a new value is calculated in $\mathbb{Q}_2(s_1, a_1) = \{(\boldsymbol{q_2^1}, (s_1, 1)\}$. The new estimate $\boldsymbol{q_2^1}$ is linked through its index to the first estimate in s_1, which in turn is linked to the first estimate in s_2. In other words, the new estimate stands for a nonstationary policy that loops once in s_1 and then proceeds to s_2. However, with the provided data, the initial value of this estimate would be $\boldsymbol{q_2^1} = (-0.12, 0.01)$, which is nondominated with the value of $\boldsymbol{q_1^1}$.

Let us assume the process starts again and the agent performs transition $s_1, a_1, \boldsymbol{r}, s_1$ once again. Now $\mathbb{V}(s_1) = \{\boldsymbol{q_1^1}, \boldsymbol{q_2^1}\}$. Therefore, two different operations will be carried out to obtain $\mathbb{Q}_3(s_1, a_1)$. First, estimate $\boldsymbol{q_2^1}$ will be *updated* according to its index (formula 3). But additionally, an *extra* estimate $\boldsymbol{q_3^1}$ will be created (formula 4), since there is a new element in $\mathbb{V}(s_1)$. The new estimate will have an index $(s_1, 2)$, and stands for the nonstationary policy consisting of looping twice in s_1 and then proceeding to s_2. The calculated initial value for $\boldsymbol{q_3^1}$ would be $(-0'112, 0, 001)$ which is again nondominated in $\mathbb{V}(s_1)$.

Now, each time the transition $s_1, a_1, \boldsymbol{r}, s_1$ is performed, an extra vector will be added to $\mathbb{Q}(s_1, a_1)$, and a long chain of dependencies between its estimates will be created ($\boldsymbol{q_2^1} \leftarrow \boldsymbol{q_3^1} \leftarrow \ldots \leftarrow \boldsymbol{q_k^1}$).

Let us assume that, after some experience in the environment is accumulated, the value of $\boldsymbol{q_2^1}$ eventually converges to its true value and is found to be dominated in $\mathbb{V}(s_1)$. The update rule in MPQ-learning would then remove $\boldsymbol{q_3^1}$ (i.e. the two loop policy) the next time transition $s_1, a_1, \boldsymbol{r}, s_1$ is performed. Two observations are in order here. In the first place, nothing prevents in the future the reconsideration of the two-loop policy provided an optimistic estimate happens to be nondominated again in $\mathbb{V}(s_1)$. Additionally, a set of *broken* estimates is left in $\mathbb{V}(s_1)$, since they not only depend on an estimate that was found to be dominated, but that was even removed from the Q-set.

In the next section we propose a new mechanism to tackle this kind of situation, noting that, once an estimate is found to be dominated, *all* estimates that depend on it through their indices cannot lead to nondominated policies, even if their current estimates happen to be locally nondominated.

4 Pruning MPQ-learning

According to the aforementioned problems, the following modifications are proposed and implemented in the MPQ algorithm.

Nondiscriminating update. Updating of Q-values depending of preexisting continuations are performed for *all* Q-values in $\mathbb{Q}(s', a)$. More concretely, the general expression for updating is not defined as in 3, but as,

$$\mathbb{U}_{n-1}(s,a) = \{((1-\alpha_n)\boldsymbol{q} + \alpha_n[\boldsymbol{r}_n + \gamma \boldsymbol{v_j}], P) \mid (\boldsymbol{q}, P) \in \mathbb{Q}_{n-1}(s,a) \wedge (s', j) \in P \wedge \boldsymbol{v}_j \in \cup_a \mathbb{Q}_{n-1}(s', a)\} \quad (6)$$

Suspended Q-values. In order to avoid the generation of more and more cyclical policies, every Q-value can be labelled as "suspended". A suspended value in never considered for the operations "new" and "extra". In this way, the definitions are no more 2 and 4. They are now,

$$\mathbb{N}_{n-1}(s,a) = \{((1-\alpha_n)\boldsymbol{q} + \alpha_n[\boldsymbol{r}_n + \gamma \boldsymbol{v}_j], P \cup \{(s', j)\} \mid (\boldsymbol{q}, P) \in \mathbb{Q}_{n-1}(s,a) \wedge \boldsymbol{v}_j \in \mathbb{V}_{n-1}(s') \wedge \neg \text{suspended}(\boldsymbol{v}_j) \wedge s' \not\sqsubset \mathbb{Q}_{n-1}(s,a)\} \quad (7)$$

and

$$\mathbb{E}_{n-1}(s,a) = \{(\alpha_n[\boldsymbol{r}_n + \gamma \boldsymbol{v_j}], (P \setminus s') \cup \{(s', j)\}) \mid \boldsymbol{v}_j \in \mathbb{V}_{n-1}(s') \ \wedge \ s' \sqsubset \mathbb{Q}_{n-1}(s,a) \wedge \neg \text{suspended}(\boldsymbol{v}_j) \wedge (s', j) \not\sqsubset \mathbb{Q}_{n-1}(s,a) \wedge \exists \boldsymbol{q} (\boldsymbol{q}, P) \in \mathbb{Q}_{n-1}(s,a)\} \quad (8)$$

Suspension. Additional rules must be defined in order to (i) label and (ii) unlabel Q-values $\boldsymbol{q} \in Q(s,a)$ as "suspended",

(i.1) A Q-value is labelled as "suspended" when it is dominated by another Q-value in the state. Formally, the condition is as follows: $\forall s, a, \boldsymbol{q} \in Q(s,a)$, whenever $\mathbb{V}(s)$ is modified, if $\boldsymbol{q}.\text{suspended} = \text{false} \wedge \boldsymbol{q} \notin \mathbb{V}(s)$, then $\boldsymbol{q}.\text{suspended} \leftarrow \text{true}$. (i.2) A Q-value is labelled as "suspended" whenever any of their indices is labelled as "suspended". This propagation is done recursively. In practice, this implements the idea that, if an estimate is found to be suspended, then any estimate depending on it is suspended as well.

(ii) A Q-value is unlabelled as "suspended" when it ceases to be dominated by another Q-value in the state and none of the estimates referenced by its indices is suspended. Formally, the condition is as follows: for every state s, whenever $\mathbb{V}(s)$ is modified, $\forall a, \boldsymbol{q} \in Q(s,a)$, if $\boldsymbol{q}.\text{suspended} = \text{true} \wedge \boldsymbol{q} \in \mathbb{V}(s) \wedge \forall (s', j)$ index of $\boldsymbol{q}, \boldsymbol{q}_j^{s'}$**.suspended = false** then $\boldsymbol{q}.\text{suspended} \leftarrow \text{false}$.

5 Experimental results and discussion

Both the MPQ-learning algorithm, and the alternative described in this paper (MPQ2), were implemented and tested over a set of sample problems. These are based on the standard Deep Sea Treasure (DST) problem proposed by [5].

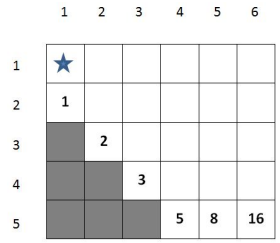

Fig. 2: Sample environment for deep sea treasure. Treasure values are indicated by values inside the cells.

More precisely, we consider a grid world like the one shown in figure 2. The agent controls a submarine that searches for treasures. These are found on the sea bed. Each grid cell represents a valid state of the agent. Grids are referenced according to their row (numbered top to bottom starting at 1), and column (numbered from left to right starting at 1). The agent is allowed to move in four directions (up, down, left or right) from its current cell to an adjacent one. Obstacles (represented as black cells) are unreachable for the agent. If an action tries to move the agent to an unreachable state or outside the grid, then the state remains unchanged. The start position is always $(1, 1)$, and each episode ends whenever a treasure position is reached. The agent receives a vector reward with two components. The first one is a negative reward of -1 each time it moves. The second one is the value of the treasure found, or 0 if no treasure is available in the cell. The goal of the agent is to find all Pareto optimal policies that maximize both rewards.

Our experiments consider a number of subproblems of increasing difficulty. In problem P_i the state space is made up only of the first i columns of the grid shown in figure 2. This way we obtain a sequence of problems such that, for larger i we obtain a larger state space and a larger set of nondominated policies. Notice that in these problems all Pareto-optimal policies are stationary.

In our implementation, the sets of optimal policies for each state $\mathbb{V}(s)$ are calculated in such a way that whenever two vectors are equal, we prefer the one that first entered $\mathbb{V}(s)$, i.e. the one with smaller index value. This way we try to ensure that, once values stabilize and several policies can propagate the same values, the algorithm consistently prefers the same one.

A limit of 700 actions per episode was set. We used the multiobjective ϵ-greedy behavior policy described by [3]. This basically calculates the ratio of nondominated vectors that each $\mathbb{Q}(s, a)$ contributes to $\mathbb{V}(s)$. With probability $(1 - \epsilon)$ each action a is selected with a probability proportional to the ratio of its $\mathbb{Q}(s, a)$ set. With probability ϵ each action is selected randomly with equal probability. The following parameters were used: Learning rate $\alpha = 0'1$; Discount rate $\gamma = 1$; Exploration probability $\epsilon = 0'4$.

Both algorithms were run until the estimates of the initial state ($\mathbb{V}(1, 1)$) approximated the values of all optimal policies (which are readily known beforehand in this problem set) with a precision up to 1%. One hundred agents were run for each problem, and the number of training steps, episodes and maximum number of estimates were recorded for each run. All agents for both algorithms reached the termination criterion.

		Trainig st.			Epi.	# estimates		
	Prob.	avg.	min	max	avg	avg.	min	max
MPQ	1	188'99	128	258	44'00	133'56	51	255
	2	2310'64	1540	3049	229'64	502'18	329	721
	3	7756'59	6116	11553	507'13	1360'21	1012	2252
	4	22359'75	14838	30403	1046'93	2891'84	2364	3357
	5	35085'68	26367	45286	1371'28	5222'02	4517	5956
	6	50015'24	38679	78576	1740'66	7767'18	6844	9569

		Trainig st.			Epi.	# estimates		
	Prob.	avg.	min	max	avg	avg.	min	max
MPQ2	1	136'86	118	169	44'00	95'95	27	186
	2	1031'34	886	1384	253'51	599'24	254	1324
	3	3519'36	2793	4665	633'51	2246'98	1352	3867
	4	8150'70	6448	9623	938'02	7433'49	4648	10652
	5	12110'93	9923	15281	968'05	15834'05	11689	21402
	6	18392'22	15719	25634	1205'79	30682'54	23763	38454

Table 1: no. of episodes, training steps and vector estimates for MPQ (top) and MPQ-2 (bottom).

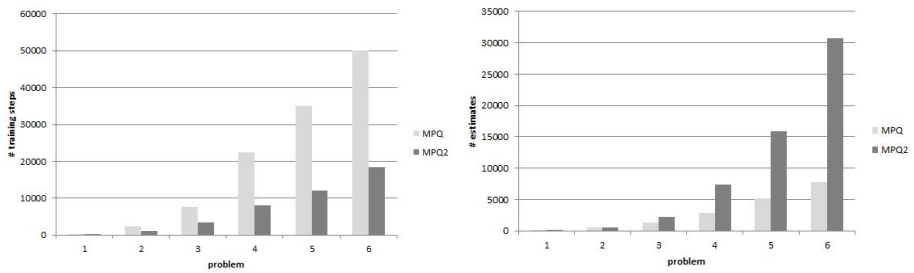

Fig. 3: Comparison of the average number of training steps(left) and maximum number of vector estimates (right) required by MPQ and MPQ2 for the deep sea treasure subproblems.

Table 1 summarizes the data gathered for MPQ (top) and MPQ2 (bottom). Figure 3 (left) compares the average value of overall training steps obtained by both algorithms for the different problems. Figure 3 (right) presents the analogous comparison for the maximum number of vector estimates stored in the $\mathbb{Q}(s, a)$ sets.

The results show an important trade-off between the number of training steps and the number of vector estimates required by both algorithms. This is to be expected, given the different design criteria for both algorithms. MPQ tries to reduce as much as possible the number of estimates. On the other hand, the main concern in MPQ2 is pruning dominated policies (and dominated nonstationary policies in particular), so that exploration can concentrate on the promising policies.

Figure 4 further illustrates the behavior and trade-off between both algorithms. The graphic displays the number of estimates stored by both algorithms on a particular run of problem P_6. The number of estimates kept by all $\mathbb{Q}(s, a)$ sets was recorded every 500 training steps. The growth in the number of policy estimates in MPQ2 is mono-

tonic non-decreasing, since once an estimate is created, it is never discarded. If the estimate is suspended or found to be locally dominated, it is prevented from generating new or extra estimates in neighboring states. These values help MPQ2 limit the number of nonstationary policies. Therefore the exploration of the state space quickly concentrates on promising policies, achieving convergence in a small number of steps. In this particular instance, MPQ2 requires 16331 training steps distributed into 1052 episodes, and stores 28448 estimates.

In contrast, the number of policies tracked by MPQ can vary widely during the exploration of the state space. The only limit to the number of nonstationary policies considered is local dominance. Therefore, MPQ has to establish the dominance of each policy on an individual basis. However, once a policy is found dominated, it is forgotten, and is likely to reappear again and again. In practice, the number of local nonstationary nondominated estimates can be very high, making it hard for the behavior policy to concentrate on the interesting learned policies. This increases considerably the number of training steps required by MPQ. In this particular instance, MPQ requires 48650 training steps distributed into 1688 episodes, and stores a peak of 8291 estimates.

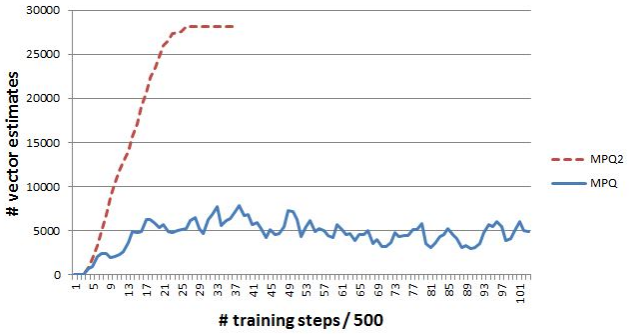

Fig. 4: Number of vector estimates kept by MPQ and MPQ2 agents on a sample run.

6 Conclusions and future work

The MPQ-learning algorithm is an interesting framework to investigate and analyze the phenomena of multi-policy temporal-difference learning. In this paper we address one such penomenon, the presence of dominated policies, and particularly the challenge of controlling the growth of dominated nonstationary policies. These frequently present nondominated estimates due to the delayed propagation of true Q-values, and their dominated nature can only be revealed after costly learning. This paper describes a variant of MPQ-learning aimed at pruning dominated policies. The pruning mechanism, called suspension, helps to reduce the number of nonstationary policies tracked

by the algorithm. The performance of both algorithms is evaluated over sample problem instances based on a standard benchmark.

MPQ-learning does not incorporate a special treatment for nonstationary policies. These are left to grow and eventually disappear whenever learned to be dominated. MPQ is very conservative with the use of memory, deleting candidate policy estimates as soon as they are found to be dominated. This results in an increase in the number of update steps required by the algorithm, since valuable information can be lost due to local oscilations in dominance. Additionally, dominated nonstationary policies tend to reapear quickly after being discarded, and require learning their values each time, frequently to be dominated again and again.

The alternative contributed in this paper (MPQ2) is monotonic in the number of policy estimates considered, i.e. once an estimate is added to the $\mathbb{Q}(s, a)$ sets, it is never removed. The algorithm keeps updating its value even when it is locally dominated or suspended. In such cases, it will never be used to generate new or extra estimates, since new nondominated estimates can only be achieved through the combination of non-dominated ones. Suspension is a recursive procedure, such that, if a particular estimate is suspended, all the estimates that depend on it will be suspended as well. However, if after some time the value is found to be locally nondominated again and suspension is lifted, it can immediatelly contribute to the determination of optimal policies. The experiments show that this strategy significantly reduces the number of training steps in the algorithm. The number of steps per episode is also considerably reduced, indicating that they are shorter and more focused on the interesting policies. The price to pay is an increase in space requirements.

Another important advantage of MPQ2 is that it clearly identifies the set of learned Pareto-optimal policies. Once the estimate of a policy converges to a dominated value, it is kept suspended. Through the recursive suspension mechanism, all such policies are eventually identified and suspended. In consequence, the subsets of un-suspended estimates in the $\mathbb{V}(s)$ sets end up populated only with the estimates of Pareto optimal policies. In MPQ, on the contrary, even when the optimal policies have been found, temptative nonstationary policy estimates emerge in the $\mathbb{V}(s)$ sets to be discarded cyclically again and again.

Pruning dominated policies in general, and dominated nonstationary policies in particular, remains an important challenge in multiobjective temporal-difference learning. The evaluation of the alternatives here presented over more ambitious and diverse benchmark problems is an interesting avenue of future research. It would also be interesting to reduce the memory requirements of MPQ2 in a cost-effective way.

References

1. Madalina Drugan, Marco Wiering, Peter Vamplew, and Madhu Chetty. Editorial: Special issue on multi-objective reinforcement learning. *Neurocomputing*, 263:1–2, 2017.
2. Diederik M. Roijers, Peter Vamplew, Shimon Whiteson, and Richard Dazeley. A survey of multi-objective sequential decision-making. *Journal of Artificial Intelligence Research (JAIR)*, 48:67–113, 2013.

3. Manuela Ruiz-Montiel, Lawrence Mandow, and Jose-Luis Perez-de-la Cruz. A temporal difference method for multi-objective reinforcement learning. *Neurocomputing*, 263:15–25, 2017.
4. Peter Vamplew, Richard Dazeley, Adam Berry, Rustam Issabekov, and Evan Dekker. Empirical evaluation methods for multiobjective reinforcement learning algorithms. *Machine Learning*, 84(1-2):51–80, 2011.
5. Peter Vamplew, John Yearwood, Richard Dazeley, and Adam Berry. On the limitations of scalarisation for multi-objective reinforcement learning of pareto fronts. In *AI 2008: Advances in Artificial Intelligence*, chapter 37, pages 372–378. Springer, 2008.