Risk-Sensitivity Through Multi-Objective Reinforcement Learning

Kristof Van Moffaert, Tim Brys and Ann Nowé

***Abstract*—Usually in reinforcement learning, the goal of the agent is to maximize the expected return. However, in practical applications, algorithms that solely focus on maximizing the mean return could be inappropriate as they do not account for the variability of their solutions. Thereby, a variability measure could be included to accommodate for a risk-sensitive setting, i.e. where the system engineer can explicitly define the tolerated level of variance. Our approach is based on *multi-objectivization* where a standard single-objective environment is extended with one (or more) additional objectives. More precisely, we augment the standard feedback signal of an environment with an additional objective that defines the variance of the solution. We highlight that our algorithm, named *risk-sensitive Pareto Q-learning*, is (1) specifically tailored to learn a set of Pareto non-dominated policies that trade-off these two objectives. Additionally (2), the algorithm can also retrieve every policy that has been learned throughout the state-action space. This in contrast to standard risk-sensitive approaches where only a single trade-off between mean and variance is learned at a time.**

I. Introduction

Reinforcement learning covers a wide range of techniques that aim to solve sequential decision making problems. Traditionally, the goal of the agent is to learn a policy that maximizes the expected accumulated reward over time, i.e., its return. Over the years, researchers have attempted to introduce the notion of *risk* into this learning process. By risk we mean the deviation of the observed return from the expected return, due to the stochastic nature of the environment. For instance, in the case of an agent learning to drive a car from point A to point B, it might not be appropriate to solely focus on the policy that minimizes the average time between these two points, but also to consider the amount of times this 'fast' behaviour results in collisions. In such situations, the system engineer might be interested in policies that result in a slight decrease in performance but are less *risky* and more *robust*. Over the years, many definitions of risk have been proposed. The variance in the solution quality is typically taken as a measure of risk [18]. Risk-sensitivity has been previously researched in reinforcement learning literature [7], [1], [11]. The common approach is to learn a single policy that either maximizes the mean reward objective while satisfying constraints defined in terms of the policy's variance, or that maximizes a weighted linear combination of mean and variance. However, defining these constraints or weights a priori might not be straightforward and the semantics of the weight parameter might be hard to grasp. Also, when the preferences of the system designer change, the algorithm would have to re-learn as every experiment only learns a single-policy at the time.

Kristof Van Moffaert, Tim Brys and Ann Nowé are with the Department of Computer Science, Vrije Universiteit Brussel, Brussels, Belgium (email: {kvmoffae,timbrys,anowe}@vub.ac.be).

978-1-4799-7492-4/15/$31.00 ©2015 IEEE

Main contributions. We propose to add this risk measure as an additional objective to the standard, single-objective problem through the process of *multi-objectivization* [16]. Multi-objectivization means creating multiple feedback signals for what is underneath still a single-objective problem. In our case, we are interested in both the expected quality of solutions, as well as the variability in their observed quality. We present *risk-sensitive* Pareto Q-learning, a reinforcement learning algorithm that does not use weight parameters or constraints in its learning process, but rather learns in parallel a set of Pareto optimal policies that trade-off solution quality and variance, encoded in the multi-objectivized environment. This way, a *multi-policy* algorithm is constructed that uses a '*train one time, test many times*'-principle, meaning that the algorithm does not need to re-learn once new preference information is available.

Outline. This paper is organized as follows. In Section II, we describe necessary concepts such as single and multi-objective reinforcement learning, and multi-objectivization. Subsequently, in Section III, we present the risk-sensitive Pareto Q-learning algorithm and we discuss its design specifications. Furthermore, in Section IV, we empirically evaluate the algorithm on two benchmark environments and discuss the results. Finally, in Section V, we summarize the paper and form conclusions.

II. Related work

A. Reinforcement learning

A reinforcement learning [19] environment is typically described by means of a Markov Decision Process (MDP). An MDP can be described as follows. Let $S = \{s_1, \ldots, s_N\}$ be the state space of a finite Markov chain $\{x_l\}_{l\geq 0}$ and $A = \{a_1, \ldots, a_r\}$ the action set available to the learning agent. Each combination of current state s_i, action choice $a_i \in A_i$ and next state s_j has an associated transition probability $T(s_j|s_i, a_i)$ and immediate reward $R(s_i, a_i)$. The goal is to learn a policy π, which maps each state to an action so that the expected accumulated future discounted reward J^π is maximized:

$$J^\pi \equiv E\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, \pi(s_t))\right] \tag{1}$$

where $\gamma \in [0, 1)$ is the discount factor and expectations are taken over stochastic rewards and transitions. This goal can also be expressed using Q-values which explicitly store the expected discounted reward for every state-action pair. The optimal Q^*-values are defined as follows.

$$Q^*(s, a) = R(s, a) + \gamma \sum_{s'} T(s'|s, a) \max_{a'} Q^*(s', a') \tag{2}$$

Watkins described an algorithm to iteratively approximate Q^*. In the Q-learning algorithm [26], a Q-table consisting of state-action pairs is stored. Each entry contains a value for $\hat{Q}(s,a)$ which is the learner's current estimate about the actual value of $Q^*(s,a)$. The $\hat{Q}$-values are updated according to the following update rule:

$$\hat{Q}(s,a) \leftarrow (1-\alpha_t)\hat{Q}(s,a) + \alpha_t[r + \gamma \max_{a'} \hat{Q}(s',a')] \quad (3)$$

where α_t is the learning rate at time step t and r is the immediate reward received for performing action a in state s. Provided that all state-action pairs are visited infinitely often and a suitable evolution for the learning rate is chosen, the estimates, $\hat{Q}$, will converge to the optimal values, Q^* [20].

B. Multi-objective reinforcement learning

Multi-objective reinforcement learning (MORL) is an extension to standard reinforcement learning where the environment consists of two or more feedback signals, i.e.

$$\mathbf{R}(s_i,a_i) = [R_1(s_i,a_i),\ldots,R_m(s_i,a_i)] \quad (4)$$

where m is the number of objectives. In MORL, a solution is a policy π, evaluated by its expected return $\mathbf{J}^\pi$, a vector of expected discounted returns for each objective. Thus,

$$\mathbf{J}^\pi \equiv \left[E\left[\sum_{t=0}^{\infty} \gamma^t R_1(s_t,\pi(s_t))\right], \ldots, E\left[\sum_{t=0}^{\infty} \gamma^t R_m(s_t,\pi(s_t))\right] \right] \quad (5)$$

Since the environment now consists of multiple objectives, conflicts can arise when trying to simultaneously optimize the objectives. In such cases, trade-offs between these objectives have to be learnt, resulting in a set of policies. A policy x_1 is said to strictly dominate another policy x_2, i.e. $x_2 \prec x_1$, if the performance on each objective of x_1 is not strictly less than the corresponding performance of x_2 and performance in at least one objective is strictly greater. If x_1 improves on x_2 on some objective and x_2 also improves on x_1 on one or more objectives, x_1 and x_2 are said to be incomparable. The set of non-dominated policies is referred to as the *Pareto front*. In [23], a general framework for MORL algorithms was proposed that extends the scalar $\hat{Q}$-values to $\mathbf{Q}$-vectors that store a $\hat{Q}$-value for each objective, i.e.

$$\hat{\mathbf{Q}}(s,a) = \left[\hat{Q}_1(s,a),\ldots,\hat{Q}_m(s,a) \right] \quad (6)$$

Current approaches in MORL often use *scalarization* functions [23], [22] to reduce the dimensionality of the underlying multi-objective environment to a single scalar. Scalarization functions often imply that an objective o is associated with a weighted coefficient, which allows the user some control over the nature of the policy found by the system, by placing greater or lesser emphasis on each objective. In a multi-objective environment, this trade-off is parametrized by $w_o \in [0,1]$ for objective o and $\sum_{o=1}^{m} w_o = 1$. In most cases, a linear combination of the objectives is considered, i.e. $\sum_{o=1}^{m} w_o \cdot \hat{\mathbf{Q}}_o(s,a)$. The linear scalarization function is intuitive and can easily be employed to solve multi-objective problems with traditional single-objective solution techniques. As a downside, the linear scalarization function is limited to learning only a subset of the Pareto front, i.e., the solutions that are optimal for a convex combination of the objective values and their corresponding weights. Hence, solutions situated in non-convex parts of the objective space cannot be discovered [21].

C. Multi-Objectivization and Risk-Sensitivity

Multi-objectivization is a mechanism that converts a single-objective problem into a multi-objective problem in order to improve performance on the original objective, as measured by solution quality, time to solution, or some other measure [16]. This idea has received most attention in evolutionary computation literature where it is divided into two major categories depending on the type of objectives added to the original problem. Firstly, multi-objectivization can be employed to decompose the original, single objective into multiple objectives [16], [12], [14], A good example is training decision trees using the misclassification of each individual class separately instead of the total misclassification [10]. Secondly, multi-objectivization can add extra objectives [15], [3] to improve the performance of the solution technique. For example, in Genetic Programming (GP), Bleuler et al. [2] and de Jong et al. [9] show that adding a second objective to minimize program size resulted in the evolution of smaller *and* more accurate programs. This extra objective basically encodes Occam's razor, and addresses the typical GP issue of evolving large and too complex programs. Other examples are using the number of recursive calls to a procedure and the number of iterations in loops to optimize running time when generating programming competition tasks [6], or turning a constrained problem into an unconstrained one with extra objectives encoding those constraints [25], [17]. Multi-objectivization was introduced in reinforcement learning to incorporate multiple pieces of heuristic knowledge in order to speed up learning [4], [5].

In the context of risk-sensitivity, a second objective related to the variance of the return [18] or with its worst value [13], [8] is added. Several approaches have been proposed for reinforcement learning [7], [1], [11] that learn either to maximize the mean reward objective while satisfying constraints defined in terms of the policy's variance, or that maximize a weighted linear combination of mean and variance. Thus, when the level of tolerance changes, the algorithm is required to re-learn from scratch and significant amounts of computational effort are wasted.

III. Risk-Sensitive Pareto Q-learning

In the previous sections, we have elaborated on standard approaches for risk-sensitive multi-objective reinforcement learning that employ a linear scalarization function. The limitation of these approaches is the fact that only a single policy is retrieved at a time, which lies in a specific area of the Pareto front. In this section, we aim to overcome these issues by presenting the *risk-sensitive Pareto Q-learning* (RSPQ) algorithm. This algorithm is an extension to the Pareto Q-learning algorithm [24] that does not learn a single policy at a time, but instead it learns a set of non-dominated trade-off policies, i.e., policies that yield a non-dominated trade-off of expected mean reward and expected variance on that same reward. In the internal working of the algorithm it stores and evolves a set of $\hat{Q}$-vectors, i.e., a $\hat{Q}_{set}$ for each state-action pair. This mechanism requires the original Q-learning update rule of Equation 3 to be augmented to work with sets of vectors of $\hat{Q}$-values.

In the following subsections, we will elaborate on the design specifications of the RSPQ algorithm. In Section III-A, we will present the internal mechanisms of the algorithm to learn multiple policies at the same time. In Section III-B, we will propose an action selection strategy that can be used to explore the state and action space based on the fruitfulness of the $\hat{Q}_{set}$ of each state-action pair. In Section III-C, we will elaborate on a *tracking* mechanism that extracts the actions in the state space corresponding to a specific trade-off policy. Once the system designer selects a certain tolerance level, the policy that maximizes the constrained expected reward is recovered from the set of learned policies. This tracking mechanism can then retrieve the corresponding state-action sequence that approaches the estimated mean reward of that policy over time.

A. Learning multiple policies at the same time

Before we proceed to the specifics of learning in risk-sensitive environments, we first analyse the process of updating sets of vectorial estimates over time. In the single-objective Q-learning algorithm, the bootstrapping rule updates an estimate of an (s, a)-pair based on the reward and an estimate of the value of the next state [26]. The update rule guarantees that the estimated $\hat{Q}$-values converge to their expected future discounted reward, even when the environment has a stochastic transition function. In this section, we analyze the problem of bootstrapping sets of vectors, which is required to learn multiple policies at the same time.

The set-based bootstrapping problem boils down to the general problem of updating the set of vectors of the current state-action (s, a)-pair with an observed reward vector $\mathbf{r}$ and a set of non-dominated vectors of the next state, $ND(\cup_{a'} \hat{Q}_{set}(s', a'))$ over time. The difficulty in this process arises from the lack of *correspondence* between the vectors in the two sets, i.e., it is not clear which vector of the set of the current (s, a)-pair to update with which vector in s'. This correspondence is needed to perform a pairwise update of each vector in $\hat{Q}_{set}(s, a)$ with the corresponding vector (if any) in the other set (see Figure 1). Our solution to this problem consists of decomposing the Q-values into separate components that can be learned individually. For simplicity reasons, we first limit ourselves to environments with deterministic transition functions. We then proceed to highlight the minimal extensions to the algorithm to also cover stochastic transitions.

In standard, single-objective Q-learning (Eq. 3), $\hat{Q}$-values store the sum of the estimated value of the immediate reward plus the future discounted reward. Our idea consists of storing this information separately. We use $\overline{\mathfrak{R}}(s, a)$ to denote the average observed immediate reward vector of (s, a) and $ND_t(s, a)$ the set of non-dominated vectors in the next state of s that is reached through action a at timestep t. The next state of s is determined by observing the transitions during learning. By storing $\overline{\mathfrak{R}}(s, a)$ and $ND_t(s, a)$ separately, we

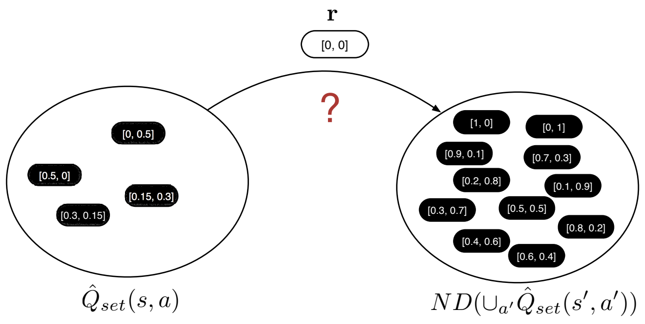


Fig. 1. Set-based bootstrapping: the problem of updating over time the set of vectors of the current state-action pair with the observed reward vector and the optimal vectors of the next state. There is no explicit correspondence between the elements in both sets, so as to perform a pairwise update.

allow them to converge separately as well. This way, no explicit correspondence between the two sets is required and the current set of non-dominating policies at timestep t, $ND_t(s, a)$ is allowed to evolve over time. The $\hat{Q}_{set}$ of (s, a) can be calculated *at run time* by performing a vector-sum over the average immediate reward vector and the set of discounted Pareto dominating future rewards:

$$\hat{Q}_{set}(s, a) \leftarrow \overline{\mathfrak{R}}(s, a) \oplus \gamma ND_t(s, a). \tag{7}$$

Whenever the action a in s is selected, the average immediate reward vector $\overline{\mathfrak{R}}(s, a)$ is updated and the $ND_t(s, a)$ list is updated using the non-dominated $\hat{\mathbf{Q}}$-vectors in the $\hat{Q}_{set}$ of every action a' in s', i.e., $ND(\cup_{a'} \hat{Q}_{set}(s', a'))$.

We present an algorithmic outline of the Pareto Q-learning algorithm in Algorithm 2. The algorithm starts by initializing the $\hat{Q}_{set}$'s as empty sets. In each episode, an action is selected using a particular action selection strategy (line 5). How we actually perform the action selection based on the $\hat{Q}_{set}$'s will be presented in the subsequent section. Afterwards, the environment transfers the agent to state s' and provides the reward r. Based on this reward, a second reward is created in an incremental fashion that specifies the mean absolute deviation (MAD) of the reward of action a: where $n(s, a)$ is the number of times action a in s was

Algorithm 1 on-line MAD(s,a,r)

```
1: $\Delta \leftarrow (r - \overline{\mathfrak{R}}_1(s, a))$
2: $sum_{AD} = sum_{AD} + \sqrt{\Delta(r - \overline{\mathfrak{R}}_1(s, a))}$
3: **return** $\frac{sum_{AD}}{n(s,a)}$
```

selected, r is the immediate reward obtained after selecting action a in state s, $\overline{\mathfrak{R}}(s, a)$ is the average immediate reward and sum_{AD} is a temporary variable storing the sum of the absolute deviations. As a result, vector $\mathbf{r}$ is created with $\mathbf{r}_1 = r$ and $\mathbf{r}_2 = -MAD(s, a, r)$, because reinforcement learning is typically formulated as a maximization problem. In state s', the non-dominated $\hat{\mathbf{Q}}$-vectors for each action are retrieved at line 8 and are discounted. At line 9, the average immediate reward for each objective, $\overline{\mathfrak{R}}(s, a)$, is iteratively updated given the new reward $\mathbf{r}$ and the number of times that action a was sampled, denoted by $n(s, a)$. The algorithm

Algorithm 2 The Risk-Sensitive Pareto Q-learning algorithm

```
1: Initialize $\hat{Q}_{set}(s,a)$'s as empty sets
2: for each episode $t$ do
3:     Initialize state $s$
4:     repeat
5:         Choose action $a$ from $s$ using a policy derived from the $\hat{Q}_{set}$'s
6:         Take action $a$ and observe state $s' \in S$ and reward vector $\mathbf{r} \in \mathbb{R}^m$
7:
8:         $ND_t(s,a) \leftarrow ND(\cup_{a'} \hat{Q}_{set}(s',a'))$                ▷ Update ND policies of $s'$ in $s$
9:         $\overline{\mathfrak{R}}_1(s,a) \leftarrow \overline{\mathfrak{R}}_1(s,a) + \frac{\mathbf{r}-\overline{\mathfrak{R}}_1(s,a)}{n(s,a)}$                ▷ Update average immediate reward
10:        $\overline{\mathfrak{R}}_2(s,a) \leftarrow -MAD(s,a)$                ▷ Update MAD of immediate reward
11:        $\overline{\mathfrak{R}}(s,a) \leftarrow [\overline{\mathfrak{R}}_1(s,a), \overline{\mathfrak{R}}_2(s,a)]$                ▷ Make the rewards vectorial
12:        $s \leftarrow s'$                ▷ Proceed to next state
13:    until $s$ is terminal
14: end for
```

proceeds until the $\hat{Q}_{set}$'s converge or after a predefined number of episodes.

Although we do not provide a formal proof on the convergence of RSPQL, its convergence can be argued by the observation that the procedure of repeatedly calculating the set of non-dominated vectors, as was applied in White's algorithm [27], is guaranteed to converge and the fact that the convergence of the $\overline{\mathfrak{R}}(s,a)$ is trivial.

The updating principle can also be extended to stochastic environments, where the transition probability $T(s'|s,a) \neq 1$ for some next state s', given state s and action a. In the case of stochastic transition functions, we store the expected immediate and future non-dominated rewards per (s,a,s')-tuple that was observed during sampling, i.e., $\overline{\mathfrak{R}}(s,a,s')$ and $ND_t(s,a,s')$, respectively. By also considering the observed frequencies of the occurrence of next state s' per (s,a)-pair, i.e., $F_{s,a}^{s'}$, we estimate $T(s'|s,a)$ for each (s,a). Hence, we learn a small model of the transition probabilities in the environment, similar to Dyna-Q [19], which we use to calculate a weighted pairwise combination between the sets. To combine a vector from one set with a vector from the other set, we propose the $\mathcal{C}$-operator, which simply weighs them according to the observed transition frequencies:

$$\begin{aligned}\mathcal{C}(\hat{\mathbf{Q}}(s,a,s'), \hat{\mathbf{Q}}(s,a,s'')) &= \frac{F_{s,a}^{s'}}{\sum_{s''' \in S} F_{s,a}^{s'''}} \hat{\mathbf{Q}}(s,a,s') \\ &+ \frac{F_{s,a}^{s''}}{\sum_{s''' \in S} F_{s,a}^{s'''}} \hat{\mathbf{Q}}(s,a,s''). \quad (8)\end{aligned}$$

B. Action selection

The action selection strategy is an adaptation to the traditional ϵ-greedy action selection. In the case of a greedy selection step, the mechanism keeps the partial order of the Pareto relationship and simply considers if an action a has a non-dominated vector across every other action a' or not. In the greedy mode, the approach eliminates any actions which are dominated, and then selects uniformly at random amongst those non-dominated actions. In the random mode, the mechanism selects an action uniformly at random from the entire set of action in the current state.

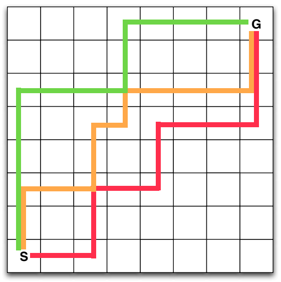
C. Retrieving a desired tolerance level

At any given time, it might be necessary to inspect the set of learnt policies and to select one that matches the criteria of the decision maker. In our setting, the decision maker can specify a *minimal mean level* (MML) after a normalization process. This is a percentage specifying the minimal required level of mean reward that needs to be satisfied. For instance, in the case of an MML of 50%, the algorithm filters the set and only includes the policies within 50% of the best policy w.r.t. the reward and then selects the policy with the smallest mean absolute deviation from this set. In the situations where MML equals 0% and 100%, the algorithm selects the policy with the highest mean reward and the policy with the lowest deviation, respectively.

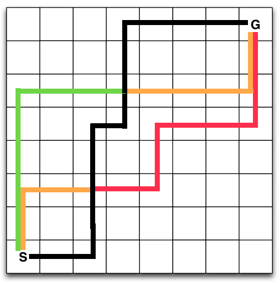
Because of the nature of the multi-policy setting, one needs to select actions *consistently* in order to retrieve a desired policy based on the $\hat{\mathbf{Q}}$-vectors. If one would select actions based on *local* information about the 'local' Pareto front attainable from each action, then there is no guarantee that the cumulative reward vectors obtained throughout the episode will be *globally* Pareto optimal. This process is highlighted in Figure 2 (a) where the state space is an 8×8 grid and three global Pareto optimal policies exist, each given a different colour. In Figure 2 (b), we select actions that are locally non-dominated, i.e, non-dominated within the current state. The black policy is a sequence of locally optimal actions, as it always overlaps with one of the coloured lines. However, the resulting policy is not globally Pareto optimal. Hence, when multiple actions are considered non-dominated, one can not randomly select between these actions when exploiting a chosen balance between criteria. In that case, actions need to be selected consistently.

In order to solve the problem of locally optimal actions that are globally dominated, we define a globally greedy policy as a policy π that consistently *follows* or *tracks* a given expected return vector $\mathbf{V}^\pi(s)$ from a state s so that its return equals $\mathbf{V}^\pi(s)$ in expectation. Therefore, we need to retrieve π, i.e., which actions to select from a start state to a terminal state.

The pseudo-code of the tracking algorithm for environments with deterministic transitions is listed in Algorithm 3.



(a)



(b)

Fig. 2. (a) In this environment, there is a green, yellow and red Pareto optimal action sequence that is globally optimal. (b) Selecting actions that are locally non-dominated within the current state does not guarantee that the entire policy is globally Pareto optimal. Hence, the information about the global Pareto front has been lost in the local Pareto front.

The agent starts in a starting state s of the environment and has to follow a particular policy so as to obtain the expected value of the policy from that state, i.e., $\mathbf{V}^\pi(s)$, at the end of the episode. For each action of the action set A, we retrieve both the averaged immediate reward $\overline{\mathfrak{R}}(s,a)$ and $ND_t(s,a)$, which we discount. If the sum of these two components equals the target vector to follow, we select the corresponding action and proceed to the next state. The return $target$ to follow in the next state s' is then assigned to $\mathbf{Q}$ and the process continues until a terminal state is reached. When the vectors have not entirely converged yet or the transition scheme is stochastic, the equality operator at line 7 should be relaxed. In this case, the action is to be selected that minimizes the difference between the left and the right term. In our experiments, we select the action that minimizes the Manhattan distance between these terms.

Algorithm 3 Track policy π given the expected reward vector $\mathbf{V}^\pi(s)$ from state s

```
1:  $target \leftarrow \mathbf{V}^\pi(s)$
2:  **repeat**
3:      **for** each $a$ in $A$ **do**
4:          Retrieve $\overline{\mathfrak{R}}(s,a)$
5:          Retrieve $ND_t(s,a)$
6:          **for** each $\mathbf{Q}$ in $ND_t(s,a)$ **do**
7:              **if** $\gamma\,\mathbf{Q} + \overline{\mathfrak{R}}(s,a) = target$ **then**
8:                  $s \leftarrow s' : T(s'|s,a) = 1$
9:                  $target \leftarrow \mathbf{Q}$
10:             **end if**
11:         **end for**
12:     **end for**
13: **until** $s$ is not terminal
```

IV. Experimental validation

In this section, we experimentally evaluate the RSPQ-algorithm on two environments with similar characteristics, i.e., worlds where the agent encounters situations that balance mean performance and variance. We first describe these two environments and their optimal trade-off policies. Subsequently, we denote the performance of RSPQ-learning on two performance criteria.

A. Small environment

The environment we propose is a small grid-world environment where the agent needs to find policies that takes it from the start state to the goal state. Upon selecting an action, the environment deterministically transitions the agent to the corresponding neighbouring state. However, the reward it receives is stochastic and depends on the amount of mud in the cell. A visual representation of the environment is provided in Fig. 3.

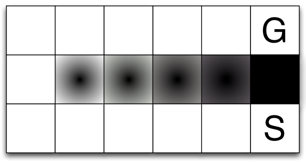
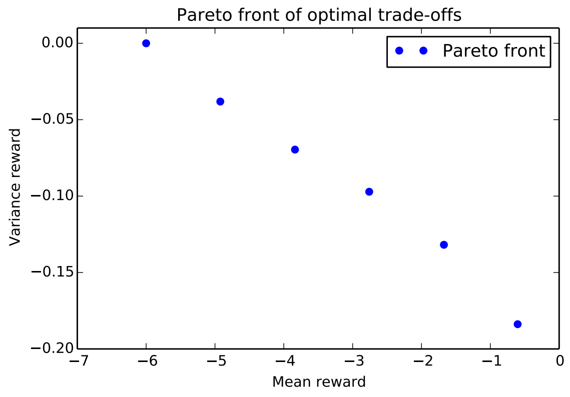
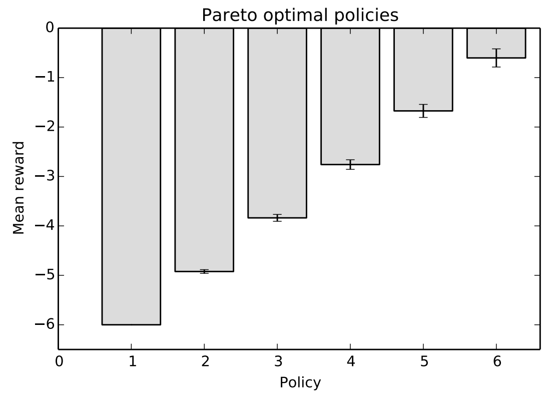


Fig. 3. The small environment contains a single row of muddy states. The mud in these states decreases as the distance from the optimal path to the goal state increases.



(a)



(b)

Fig. 4. The Pareto front of the small environment.

As there are decreasing levels of mud as the distance from the start to the goal increases, there are different trade-off solutions in this environment, i.e. the shortest path is very noisy while the longest path is entirely deterministic. In this environment, the optimal policies correspond to the following sequence of actions; first, the agent selects $i \in [0, 5]$ actions that take it left, followed by 2 upward actions and again i actions that take it right until the goal state has been reached. Depending on the value of i, different trade-offs between mean and variance are struck. The optimal trade-off policies are depicted in a Pareto plot in Fig. 4 (a) and in a sorted bar-chart in Fig. 4 (b). The purpose of the RSPQ-learning algorithm would be to learn and exploit these trade-off solutions simultaneously.

As we are learning multiple policies simultaneously, which potentially may involve different parts of the state space, we found it beneficial to employ *train* and *test* phase, where in the *train* mode, we learn with an ϵ-greedy action selection strategy with decreasing epsilon. At episode eps, we assigned ϵ to be 0.999^{eps} to allow for significant amounts of exploration in early runs while maximizing exploitation in later runs of the experiment. In the *test* mode of the algorithm, we perform multiple greedy policies using Algorithm 3 for 11 thresholds to retrieve a certain MML tolerance of the elements in $ND(\cup_a \hat{Q}_{set}(s_0, a))$ of the start state s_0. In Fig. 5, we average the accumulated returns along the paths and their mean absolute deviation over time for this range of MML thresholds. The results are averaged over 100 trials, each consisting one train and test phase. In the figure, we see that the 11 lines correspond to a trade-off between mean performance and deviation, i.e., the lowest line performs worse on the mean objective, but corresponds to a very stable policy as its mean absolute deviation is zero. On the contrary, the line on the top conforms to the policy with the highest mean reward over time but also to the riskiest policy. The policies between those two extremes leverage a certain trade-off between these two objectives.

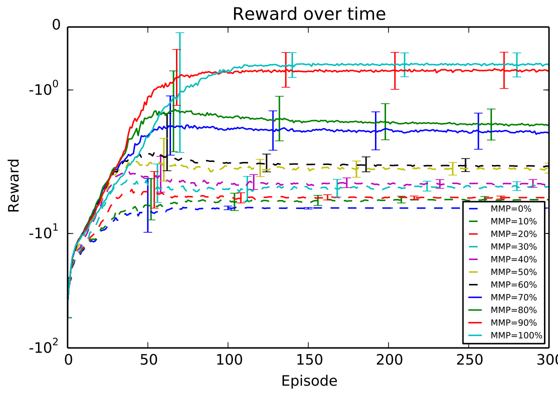


Fig. 5. The learning graphs for 11 different MML trade-off levels.

In Fig. 6, we denote the hypervolume over time of the learned policies in the start state, i.e., $HV(ND(\cup_a \hat{Q}_{set}(s_0, a)))$ where HV is a function that calculates the hypervolume of the set of $\hat{Q}_{set}$ of the start state s_0. We see that over time, the hypervolume of these vectors approaches the hypervolume of the optimal vectors in Fig. 4 (a). This entails that the policies being learned actually converged to the optimal policies. It is important to note that the line is decreasing over time as the $\hat{Q}$-vectors were initialized optimistically.

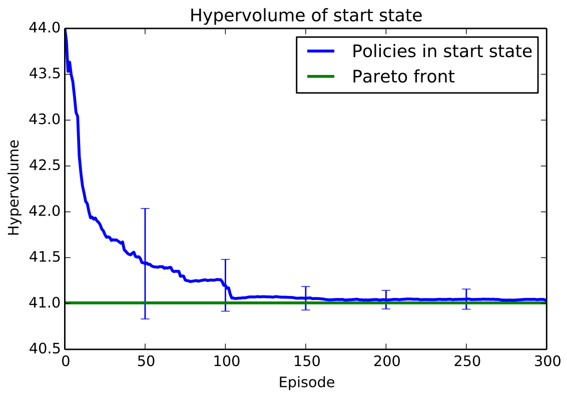


Fig. 6. The hypervolume of the vectorial estimates of the start state approach their optimal values over time.

B. Larger environment

To analyse the capability and the scalability of the RSPQ-learning algorithm, we added additional rows to the environment as depicted in Fig. 7. Because of the enlargement of the environment, the number of optimal policies also increases. An exhaustive sweep through the environment taught us that there are also optimal policies that do not follow a sequence of n consecutive upward actions, where n is the number of rows in the environment. We have found that there are optimal policies in which the agent selects horizontal actions while in the process of making the upward movement. Because of the horizontal actions, the agent observes less or more stochasticity in the reward signal while increasing or decreasing the time needed to reach the goal location, depending on the direction of the movement. These 19 optimal policies are depicted in Fig. 8 (a) and (b) in a Pareto plot and in a sorted bar chart, respectively.

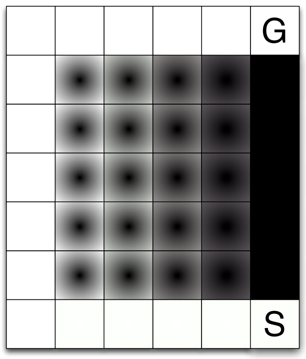
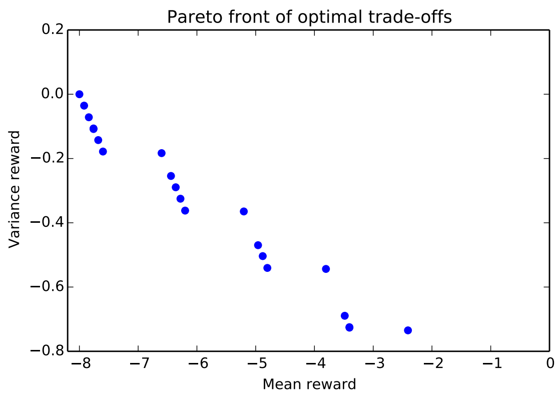
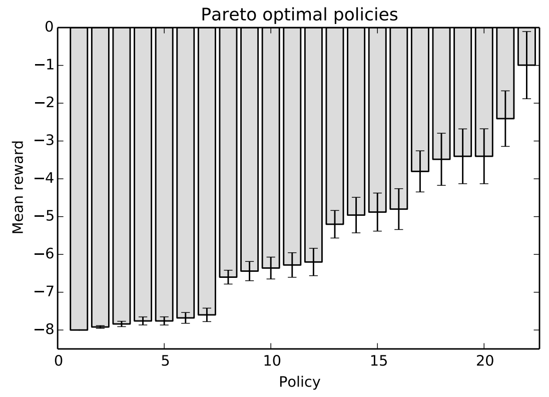


Fig. 7. The large environment contains 5 rows of muddy states. The mud in these states decreases as the distance from the optimal path to the goal state increases.

The outcome of the experiments is presented in Fig. 9 and 10. As we see, RSPQ-learning is able to retrieve the different trade-offs that present a compromise between mean reward and risk. We see that only in this larger environment, the hypervolume of the vectorial estimates in the start state approaches the volume of the policies of Figures 8 (a).



(a)



(b)

Fig. 8. The Pareto front of the large environment.

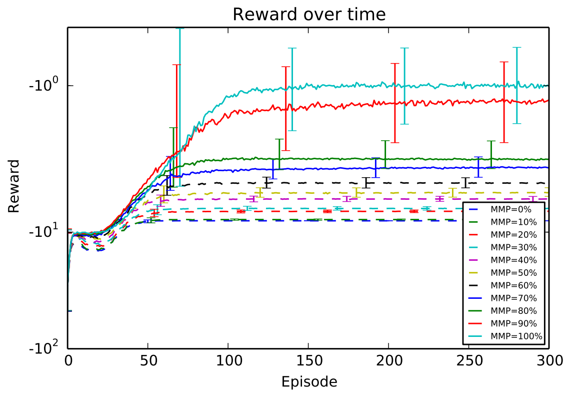


Fig. 9. The learning graphs for several trade-offs between mean reward and risk are learned simultaneously.

V. Conclusions

In this paper, we have analysed methods for combining a notion of risk in the process of learning an optimal policy. Through the process of multi-objectivization, the reward signal is augmented to incorporate an additional objective. This supplementary objective relates to the variance of the original objective and can be used to improve the performance of the solution technique. As a result, the objective space is multi-dimensional and therefore, there usually is no single optimal

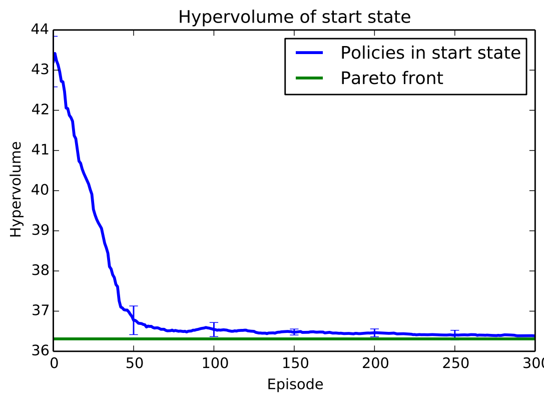


Fig. 10. The hypervolume of the vectors in the start state also approach optimality over time.

policy but a set of optimal trade-offs that leverage mean performance and variance. Previous approaches that attempted to tackle this problem, relied on an a priori scalarization process where the system engineer would provide a threshold before the optimization process takes place. Because the specification of this threshold is far from trivial and in the case of shifting aspiration, the optimization process needs to be repeated multiple times before a desired trade-off is reached. In this paper, we have proposed and analysed RSPQ-learning, an algorithm that simultaneously learns Pareto non-dominated trade-off policies that leverage mean performance and variance. We have also included a mechanism that allows to track a policy with a desired threshold level based on a lexicographic ordering throughout the state and action space. We experimentally evaluated the algorithm on two small environments and we have seen that the tracking mechanism allows to retrieve the trade-off policies through the state space and that their vectors of $\hat{Q}$-values converge to their optimal values. With the development of RSPQ-learning we have scratched the surface for concurrently learning multiple trade-off policies although the experimental environments remain artificial and relatively small. In future work, we would like to analyse RSPQ-learning on challenging and real-life environments. At the same time, it would be interesting to research how function approximation can be adapted to represent sets of $\hat{Q}$-values. A possible idea is to fit the elements in each set through a geometric approach, such as for instance ordinal least-squares. If the environment would consist of two or three objectives, we would be fitting the vectors on a curve or a plane, respectively. In that case, we would be learning the shape of the Pareto front through local interactions that each update parts of this geometric shape.

VI. Acknowledgements

Kristof Van Moffaert is supported by the IWT-SBO project PERPETUAL (grant nr. 110041). Tim Brys is funded by a Ph.D grant of the Research Foundation Flanders (FWO). Together with the Multi-Criteria Reinforcement Learning project (grant G.0878.14N) of the Fonds Wetenschappelijk Onderzoek - Vlaanderen (FWO).

References

[1] P. L. A. and M. Ghavamzadeh. Actor-critic algorithms for risk-sensitive mdps. In C. J. C. Burges, L. Bottou, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26.*, pages 252–260, 2013.

[2] S. Bleuler, M. Brack, L. Thiele, and E. Zitzler. Multiobjective genetic programming: Reducing bloat using spea2. In *Evolutionary Computation, 2001. Proceedings of the 2001 Congress on*, volume 1, pages 536–543. IEEE, 2001.

[3] D. Brockhoff, T. Friedrich, N. Hebbinghaus, C. Klein, F. Neumann, and E. Zitzler. Do additional objectives make a problem harder? In *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 765–772. ACM, 2007.

[4] T. Brys, A. Harutyunyan, P. Vrancx, M. E. Taylor, D. Kudenko, and A. Nowé. Multi-objectivization of reinforcement learning problems by reward shaping. In *IEEE IJCNN*, 2014.

[5] T. Brys, A. Nowé, D. Kudenko, and M. E. Taylor. Combining multiple correlated reward and shaping signals by measuring confidence. In *Twenty-Eighth AAAI Conference on Artificial Intelligence (AAAI-14)*, 2014.

[6] A. Buzdalova, M. Buzdalov, and V. Parfenov. Generation of tests for programming challenge tasks using helper-objectives. In *Search Based Software Engineering*, pages 300–305. Springer, 2013.

[7] D. D. Castro, A. Tamar, and S. Mannor. Policy gradients with variance related risk criteria. In *Proceedings of the 29th International Conference on Machine Learning, ICML 2012, Edinburgh, Scotland, UK, June 26 - July 1, 2012*. icml.cc / Omnipress, 2012.

[8] S. P. Coraluppi and S. I. Marcus. Risk-sensitive and minimax control of discrete-time, finite-state markov decision processes. *Automatica*, 35(2):301–309, 1999.

[9] E. D. de Jong, R. A. Watson, and J. B. Pollack. Reducing bloat and promoting diversity using multi-objective methods. In L. Spector, E. D. Goodman, A. Wu, W. B. Langdon, H.-M. Voigt, M. Gen, S. Sen, M. Dorigo, S. Pezeshk, M. H. Garzon, and E. Burke, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, pages 11–18, San Francisco, California, USA, 7-11 July 2001. Morgan Kaufmann.

[10] J. E. Fieldsend. Optimizing decision trees using multi-objective particle swarm optimization. In *Swarm Intelligence for Multi-objective Problems in Data Mining*, pages 93–114. Springer, 2009.

[11] P. Geibel and F. Wysotzki. Risk-sensitive reinforcement learning applied to control under constraints. *J. Artif. Intell. Res. (JAIR)*, 24:81–108, 2005.

[12] J. Handl, S. C. Lovell, and J. Knowles. Multiobjectivization by decomposition of scalar cost functions. In *Parallel Problem Solving from Nature–PPSN X*, pages 31–40. Springer, 2008.

[13] M. Heger. Consideration of risk in reinforcement learning. In *Proc. of the Eleventh International Machine Learning Conference*, pages 105–111, San Francisco, CA, 1994. Morgan Kaufmann.

[14] M. Jähne, X. Li, and J. Branke. Evolutionary algorithms and multi-objectivization for the travelling salesman problem. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 595–602. ACM, 2009.

[15] M. T. Jensen. Helper-objectives: Using multi-objective evolutionary algorithms for single-objective optimisation. *Journal of Mathematical Modelling and Algorithms*, 3(4):323–347, 2005.

[16] J. D. Knowles, R. A. Watson, and D. W. Corne. Reducing local optima in single-objective problems by multi-objectivization. In *Evolutionary Multi-Criterion Optimization*, pages 269–283. Springer, 2001.

[17] D. K. Saxena and K. Deb. Trading on infeasibility by exploiting constraint?s criticality through multi-objectivization: A system design perspective. In *Evolutionary Computation, 2007. CEC 2007. IEEE Congress on*, pages 919–926. IEEE, 2007.

[18] M. Sobel. The variance of discounted markov decision processes. *Applied Probability*, pages 794–802, 1982.

[19] R. Sutton and A. Barto. *Reinforcement Learning: An Introduction*. Adaptive Computation and Machine Learning. Mit Press, 1998.

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

[21] P. Vamplew, J. Yearwood, R. Dazeley, and A. Berry. On the limitations of scalarisation for multi-objective reinforcement learning of pareto fronts. In *Proceedings of the 21st Australasian Joint Conference on Artificial Intelligence: Advances in Artificial Intelligence*, AI '08, pages 372–378, Berlin, Heidelberg, 2008. Springer-Verlag.

[22] K. Van Moffaert, M. M. Drugan, and A. Nowé. Hypervolume-based multi-objective reinforcement learning. *Lecture Notes in Computer Science, Evolutionary Multi-Criterion Optimization (EMO 2013)*, 2013.

[23] K. Van Moffaert, M. M. Drugan, and A. Nowé. Scalarized Multi-Objective Reinforcement Learning: Novel Design Techniques. In *2013 IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning*. IEEE, 2013.

[24] K. Van Moffaert and A. Nowé. Multi-objective reinforcement learning using sets of pareto dominating policies. *Journal of Machine Learning Research*, 15:3483–3512, 2014.

[25] S. Watanabe and K. Sakakibara. Multi-objective approaches in a single-objective optimization environment. In *Evolutionary Computation, 2005. The 2005 IEEE Congress on*, volume 2, pages 1714–1721. IEEE, 2005.

[26] C. Watkins. *Learning from Delayed Rewards*. PhD thesis, University of Cambridge,England, 1989.

[27] D. J. White. Multi-objective infinite-horizon discounted Markov decision processes. *Journal of Mathematical Analysis and Applications*, 89(2), 1982.