MDP Transformation for Risk Averse Reinforcement Learning via State Space Augmentation

BY

DAVIDE SANTAMBROGIO
B.S, Politecnico di Milano, Milan, Italy, 2017

THESIS

Submitted as partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
in the Graduate College of the
University of Illinois at Chicago, 2020

Chicago, Illinois

Defense Committee:

Prof. Brian Ziebart, Chair and Advisor
Prof. Ian Kash
Prof. Marcello Restelli, Politecnico di Milano

ACKNOWLEDGMENTS

I want to thank my family, my friends and my advisors for the accomplishment.

AB

TABLE OF CONTENTS

TABLE OF CONTENTS (continued)

CHAPTER **PAGE**

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

CVaR	Conditional-Value-at-Risk
VaR	Value-at-Risk
MDP	Markov Decision Process
GPOMDP	Gradient Policy Optimization for Markov Decision Process
RL	Reinforcement Learning
APG	Augmented Policy Gradient
GCVaR	Gradient Conditional-Value-at-Risk
PPO	Proximal Policy Optimization
TRPO	Trust Region Policy Optimization
RN	Risk-Neutral
RA	Risk-Averse

SUMMARY

While the standard Reinforcement Learning [1] objective is to minimize (maximize) the expected cumulative cost (reward) over the horizon, finite or infinite, the Risk Averse RL approach [2] tries to optimize not only the common objective function but involves also the minimization of what is defined as Risk. There are several ways to express the Risk of the given random cost (reward) received by the environment and the main ones are the utility functions and the proper risk measures. However, the issue in dealing with the Risk is that its minimization is not as simple as the standard optimization problem and sometimes it could become also NP-hard affecting the optimality of the found solution. The common approach historically used by the literature to find the optimal solution of this problem is based on the idea of creating new ad hoc algorithms capable to optimize the new objective function instead of the cumulative cost only. This, of course, makes the algorithm not general and specific to that kind of Risk measure or Utility function. More other, there are more standard algorithms respect to the specific ones and they are usually more studied and optimized because they are more frequently used. This is why some researchers came up with another alternative approach based on the transformation of the Markov Decision Process itself. Transforming the MDP means modifying the characteristics of the optimization problem's formalization in terms of state space, transition kernel or Reward function to incorporate the Risk Aversion in the structure of the problem. In this way, it is possible to apply the standard Reinforcement Learning algorithms as if they are optimizing the usual objective function, while instead, they

are taking into account the Risk too. The issue with this method is that we should consider how much the transformation costs in terms of computational power (and so time). Furthermore, there could be also an increment of the cost for the optimization itself after the transformation of the Markov Decision Process. In both these cases, the approach does not worth it and it would be better to adopt an ad hoc algorithm following the basic strategy for Risk-Averse optimization. My thesis deals with these problems with Risk Averse Reinforcement Learning. The idea is to modify the Markov Decision Process via a state-space augmentation that gives a piece of partial information about the history of the current chosen policy and then find the optimal solution of the modified problem through a standard Reinforcement Learning algorithm. After a proper background to present the basics needed to understand the research work, we will describe some of the interesting papers we studied about Risk Aversion and MDP transformation that gave me the idea of the state of the art in this field. Then we will explain the transformation that we decided to adopt and finally how we applied the standard RL algorithm to find the optimal Risk Sensitive policy. Finally, we will give some conclusions talking about the practical results and some possible future works in this field.

CHAPTER 1

INTRODUCTION

Since the modern computer's invention in the twentieth century and the consequent birth of the concept of nowadays Computer Science, a huge question came out for the human being: is it possible to replicate and reproduce the human mind? If yes, how? This interrogative is destined to become one of the most important ones more and more in the future, following the increasing power and precision of technologies during the last years.

At the moment the attempt to answer this question is brought ahead by what is called Artificial Intelligence and in particular by an application of it which is Machine Learning. The latter is based around the idea that we should just be able to give machines access to data and let them learn for themselves to make them carry out some tasks in a way we can consider somehow "smart".

Indeed, the core concept which is most recently studied by researchers is this enigmatic word: Learning. Considering our human experience, we can describe our learning process as based on the interaction with the environment, the external Reality we daily meet. When we are kids, we learn to perform actions like walking or looking around by continuously trying to do them and seeing the effect of our actions within the environment. The more we try, the more we enrich our experients and so the more we learn the rules under which the reality falls and we improve our actions to reach our small or big daily goals.

These interactions are the fundamental part to increase our knowledge about the external re-

ality, the environment and ourselves. Whenever we learn something, that could be running, studying or handling conversations we are deeply focused on how the environment responds to our actions and we try to change the external events with our behavior. Studying this interaction between the system and the acting subject is the base of all theories which aim to replicate learning and intelligence.

Within Machine Learning, the principal computational approach devoted to learning from interaction is Reinforcement Learning [1]. Indeed, it is a goal-directed learning approach from an interaction where an agent, the subject of the algorithm, aims at finding the best way to maximize a certainly given reward function. It is in this specific Machine Learning sector that my Thesis is set up trying to figure out a suitable MDP transformation to modify the objective of the agent is not only maximizing the standard reward but also minimizing a defined function describing the concept of Risk, as we will explain in the following sections.

CHAPTER 2

BACKGROUND

In this chapter, we will present the main basic concepts of Reinforcement Learning theory useful to deeply understand our solution and the methods we adopted to find it. We will go through the definitions of Markov Decision Process, Dynamic Programming for policy optimization, basic Policy Gradient algorithms and some general way to formulate and approach the Risk Aversion within Reinforcement Learning settings.

2.1 Markov Decision Process

The key concept on which Reinforcement Learning [1] is built is surely the Markov Decision Processb[1]. To understand it we should firstly focus on the general area in which the MDP is applied, that is the sequential decision problem. The sequential decision problem, as suggested by the name, is when we have an agent, the proper subject of RL algorithms, that interacts with the environment performing a series of actions devoted to achieving a determined goal. The environment shows himself to the agent in the form of States. On the other hand, the goal is represented through Rewards, numeric values that the environment returns to the agent according to the state in which it is and the action it performs. Of course, the objective of the agent (also called decision-maker or controller) is to maximize these Rewards following the best trajectory, which consists of the series of States encountered by the agents and the Actions

performed by it. According to the convention, the Reward could be replaced by the Cost that the agent wants to minimize instead. How the decision-maker chooses the action according to the State given by the environment (or System) is known as Policy. The environment evolves in a stochastic way and it could be also influenced by the agent's actions during this interaction. The problem that seems simple in appearance hides a lot of complexities. The first is the fact that the agent, to maximize his total Reward, needs to make a decision balancing the maximum reward that could take immediately and the one that could cumulate in the future, maybe following another trajectory. The total cumulative Rewards are defined as the Return. There are several examples of applications: one of the most common ones is the managing of a Portfolio, in which the objective is the maximization of the total earnings. The other main setting is the so-called Grid-World in which the agent wants to reach a terminal state from a starting position, choosing the shortest path but avoiding the obstacles that it encounters.

Usually, when we talk about the sequential decision problem beneath uncertainty, it is visualized as in the figure Figure 1. Every time t, the agent observes the state s_t given by the environment and chooses to perform an action a_t. As a result of performing that specific action a_t in that specific state s_t, the agent obtains an immediate reward r_{t+1} (or cost) and it will move to a new state s_{t+1} defined by the environment with a given probability. The latter is taken from a suitable distribution which depends on the action performed by the decision-maker. Then, the same process is done for the time t+1 and so on. The interaction described above can be modeled rigorously by the so-called Markov Decision Process [1]. As in [3],

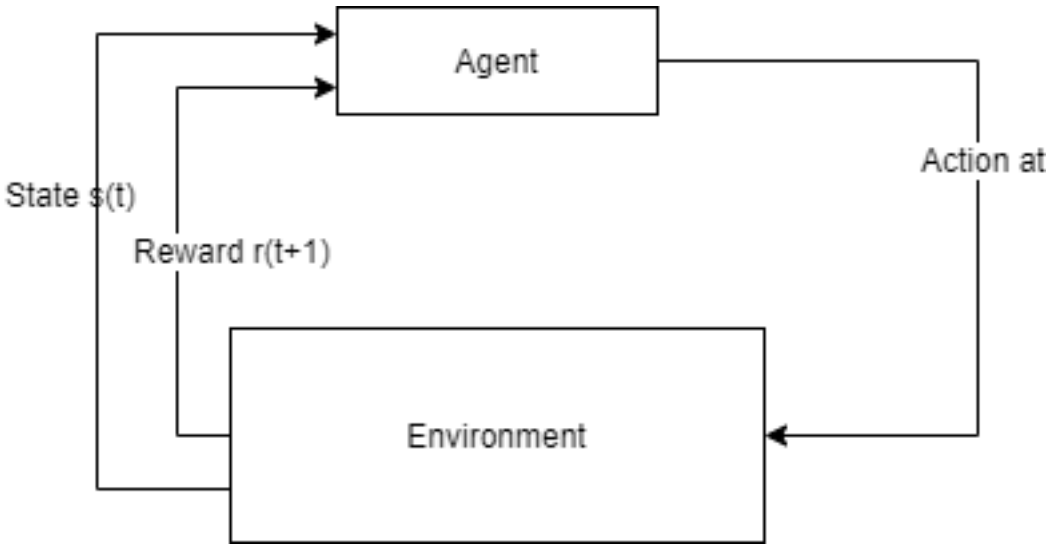

Figure 1: MDP AGENT-ENVIRONMENT INTERACTIONS

Markov Decision Processes are stochastic dynamic systems defined by the tuple <S,A,P,R,γ>, in particular:

1. The state space S which is measurable
2. The action space A, which is measurable
3. The Markov transition kernel P: $\mathbb{S} \times \mathrm{A} \times \mathrm{S} \to \mathbb{R}$
4. The Reward Function $\mathrm{R} : \mathbb{S} \times \mathrm{A} \to \mathbb{R}$
5. The discount factor $\gamma \in (0,1)$

Particular attention should be paid on the transition kernel P, which gives the probability, given the state s_t and the action a_t performed in that state, to arrive in another state s_{t+1}. It is possible to notice that it does not depend on the history of the environment, but only on the

previous state and action. This characteristic is defined as Markvovian property [1] and, as it is easy to understand from the name, is fundamental for Markov Decision Processes [1].

$$P(s, a, B) = \mathbb{P}(S_{t+1} \in B | S_t = s, A_t = a) \tag{2.1}$$

Another thing to notice is that the Reward function is defined as the function which given the state s_t and the action a_t, returns the expected reward collected under the probability defined by the kernel function.

$$R(s, a) = \mathbb{E}(R_{t+1} | S_t = s, A_t = a) \tag{2.2}$$

2.2 The Policy

As we mentioned before, another basic concept to keep in mind dealing with Markov Decision Processes is the Policy that defines how the decision-maker chooses the action to make. The policy is, of course, delineates the probability to take that certain actions in the given state. There are also deterministic policies that return directly the action to perform given the state but we are not interested in them for what we do. As we highlighted before, even here the fact that the policy depends only on the state and not on history follows the important Markovian property [1], however, some policies can depend on history. As history we mean all the infor-

mation of the followed trajectory and so the sequence of states, actions reward met by the agent.

2.3 The Return

Now having the idea of the MDP and so on the formalization of the problem we can focus on the objective function, that is what the agent wants to maximize (or minimize if we have cost).

Of course, we start from the standard Risk Neutral optimization problem in which we have only the concept of Return. The Total Return is defined as the discounted cumulative costs incurred by the decision-maker starting from the time t.

$$G_t = \sum_{t=0}^{\infty} \gamma^t R_{t+k+1} \tag{2.3}$$

As we said this is the real objective of the agent, that has to focus on the total future performance instead of looking simply at the immediate reward greedily encountered. The discount factor is a parameter that follows this idea and allows us to change the importance that we are giving to the Rewards (or cost) in the further future. If $\lambda \to 0$, the agent will focus on the immediate reward while with $\lambda \to 1$ it will concentrate on the future. There are other interpretations of the discount factor: the first one is that it helps convergence avoiding that the objective sums to infinite, in case of large rewards or long-horizon term. The second one is strictly connected to the agent orientation to the future but from the environment point of view. Indeed, it could be seen as the representation of the uncertainty of the future, defining

from which time on the agent does not know which reward can take.

2.4 Value functions

Given the definition of Return, we can define functions. The first one is the State-Value Function and it is the expected discounted cumulative reward (alias the return) over the policy starting from a given state. Thus it is what the agent expects to gain following a certain policy π. Of course, it is interpreted as the goodness of the policy starting from the state s [3].

$$V_\pi(s) = \mathbb{E}[G_t|S_t = s] \tag{2.4}$$

In the same way, we can define the State-Action Value Function as the expected discounted cumulative reward over the policy starting from the state s and performing the action a [3].

$$Q(s, a) = \mathbb{E}[G_t|S_t = s, A_t = a] \tag{2.5}$$

It follows that the relationship between the two value functions is that the State one is the sum of the State Action Value Function of all possible performed actions in that state, weighted by the policy [3].

$$V_\pi(s) = \int_{\mathbb{A}} \pi(s, a)Q_\pi(s, a)da \tag{2.6}$$

These two functions are fundamental because these are the real objective functions that the Reinforcement Learning algorithms aim at optimizing. Usually, they are represented with their Bellman Equations [1] as below [3].

$$V_\pi(s) = R_\pi + \gamma T_\pi V_\pi(s) \tag{2.7}$$

$$Q_\pi(s,a) = R(s,a) + \gamma T_a V_\pi(s) \tag{2.8}$$

Where T_a is the transition operator taking the action a, while T_π is the one following the policy π [3]:

$$\begin{aligned} T_a F(s) &= \mathbb{E}[F(S_{t+1})|S_t = s, A_{t]=a}] = \int_{\mathbb{S}} P(s,a,s')F(s')ds'da \\ T_\pi F(s) &= \mathbb{E}[F(S_{t+1})|S_t = s] = \int_{\mathbb{A}\pi(s,a)} \int_{\mathbb{S}} P(s,a,s')F(s')ds'da \end{aligned} \tag{2.9}$$

Finally, we define the Bellman expectation operator [1] as [3]

$$B_\pi V_\pi(s) = R_\pi + \gamma T_\pi V_\pi(s) \tag{2.10}$$

Hence, the equation Equation 2.7 can be compacted in [3]

$$V_\pi(s) = B_\pi V_\pi(s) \tag{2.11}$$

If some simple assumptions about the reward function are satisfied, the Bellman Equation leads to a unique solution. Thus the final goal of the agent is to find the so-called Optimal Policy, which is the policy that maximizes the value function of every state of the environment. That is why we need the definitions of optimal State-Value and State-Action-Value Function. The first one is the maximum expected reward possibly gained, starting from the specific state s [3]

$$V_*(s) = \sup_\pi V_\pi(s) \tag{2.12}$$

The second one is the maximum expected reward possibly gained, starting from the specif state s and taking the action a [3]

$$Q_*(s,a) = \sup_\pi Q_\pi(s,a) \tag{2.13}$$

The Bellman Optimality Equations are [3]

$$V_*(s) = \sup_a Q_*(s,a) = \sup_a \{R(s,a) + \gamma T_a V_*(s =\} \tag{2.14}$$

$$\begin{aligned} Q_*(s,a) &= R(s,a) + \gamma T_a V_*(s) \\ &= R(s,a) + \gamma \int_{\mathbb{S}} P(s,a,s') \sup_{a'} Q_*(s',a') ds' \end{aligned} \tag{2.15}$$

Therefore, the Optimal Policy is simply the policy that has the highest state value function for each state for all the other possible policies [3]:

$$\pi \geq \pi' \Leftrightarrow V_\pi(s) \geq V_{\pi'}(s) \forall s \in \mathbb{S} \tag{2.16}$$

For any Markov Decision process, it exists an optimal policy $\pi_* \geq \pi, \forall \pi$, and its optimal value functions are equal to the General Optimal Value functions of the MDP.
These formulas defined properly the optimization problem in the finite time horizon. Whereas, for the infinite horizon setting they cannot work because there are no absorbing states and we cannot compute the real value functions. In this case, the average expected reward is used but we are not interested in it because our implemented algorithm, as we will see, works with finite trajectories to be performed.

2.5 Risk-Aversion in Reinforcement Learning

As we said before, the most used Risk Neutral problem which maximizes the merely expected value of the reward is not always enough. Indeed, this assumption in many cases is not realistic. Our daily life actions, for instance, are always based on some concept of risk that we want to minimize while we are doing our tasks even if sometimes taking that risks can give us more in terms of "reward". The same case is for the agent in the Reinforcement Learning problem in which we are trying to replicate somehow the learning method of a human being. The concept of Risk in the Markov Decision process can be derived from two different kinds of uncertainties (indeed the risk always deals with not being sure about something): the inherent uncertainty

and the model uncertainty. The former is again divided into static and dynamic risk and induces the transformation of the objective function. As the name suggests, it is linked to the stochasticity proper of the environment as we described above with the transition kernel and the consequent random reward.

While the model uncertainty (also called parametric uncertainty) is about robust MDPs and is related to the not precise knowledge of the parameters of the model. So in the Risk Averse optimization problem, the standard value function is combined with a risk measure of the total return. There are different ways to take into account the risk parameter within the solution. The utility approach, in which the standard value function is replaced by the expected value of a certain utility function of the reward. The risk measures strategy, which consists in adding a non-linear term that can be the variance or a class of functions that describe somehow the risk, like the Conditional Value at Risk. Finally, we have the robust MDPs in which there are no risk terms added but it is considered the worst-case scenario in terms of model parameters.

2.5.1 Utility functions

The utility-based one consists in transforming the cumulative reward by appropriate functions that corresponds to our new way to define how much a state is valuable or not. The risk sensitivity is based on the convexity of the new value functions. Given the fact that a risk-averse decision-maker is usually predisposed to the diversification.

$$\rho(\alpha X + (1-\alpha)Y)) > \alpha\rho(X) + (1-\alpha)\rho(Y) \tag{2.17}$$

Where X and Y are two random variables representing two choices and α is the probability to take the first one. Thus a risk-seeking function should be convex while a risk-averse one should be concave.

One of the most used utility functions is the exponential ones ($u(x) = e^{\lambda}x$) applied to the return. It is possible to notice that the second-order approximation of this function, known also as an entropic map, it is equal to the common mean-variance optimization problem.

$$U(T, \pi) = \mathbb{E}_{\pi}[R] + \lambda Var_{\pi}[R] + \mathcal{O}(\lambda^2) \tag{2.18}$$

2.5.2 Risk Measures

The second approach includes a function that explicitly describes the risk. Indeed a risk measure is a defined as a mapping $\rho : Z \to \mathbb{R}$ from a random variable Z to Real values. One of the most important and also the easiest one to visualize is the Variance of the return, which is how much the evaluated function is distant from its expected value.

$$Var(X) = \mathbb{E}[(X - \mu)^2] = \mathbb{E}[X^2] - \mathbb{E}[X]^2 \tag{2.19}$$

From the Variance formulation, a new objective function could be derived, like Mean-Variance [4], which tries to maximize the expected value with a constraint on the Variance , or Sharpe ratio, in which the expected value is multiplied by the inverse of the square root of the Variance, balancing the maximization of the former with the minimization of the latter.

$$MVar = \mathbb{E}_\pi - \beta Var_\pi[R] \tag{2.20}$$

$$Sh = \mathbb{E}_\pi / \sqrt{Var_\pi[R]} \tag{2.21}$$

Other interesting functions for the representation of the risk are the so-called Coherent Risk Measures. They are functions with some suitable properties that are helpful for the convex optimization problems [5].For all $X, Y \in X$[2]:

1. Convexity: $\forall \alpha \in [0,1], \rho(\alpha X + (1-\alpha)Y \leq \alpha\rho(X) + (1-\lambda)\rho(Y)$
2. Monotonicity: if $X \leq Y$, then $\rho(X) \leq \rho(Y)$
3. Translation invariance: $\forall b \in \mathbb{R}, \rho(X+b) = \rho(X) + b$
4. Positive homogeneity: if $\alpha \geq 0$, then $\rho(\alpha X) = \alpha\rho(X)$

These conditions ensure the "rationality" of single-period risk assessments. Indeed, 1. establishes that diversifying the outcomes by a factor α reduces the risk. Thinking about a portfolio environment, if we diversify the investment spending only the portion α on a first stock that gives X as an outcome and the remaining amount $(1-\alpha)$ on a second stock leading to Y outcome, the result is $\alpha X + (1-\alpha)Y$. Thus applying the risk measure to this combination returns a value less or equal than the combination of the risks with the same parameter α. 2. shows that the higher the cost, the higher the risk associated with it. 3. ensures that the constant part of the cost does not influence the risk. 4. finally, assures that multiplying an outcome by a

factor and computing the risk associated with it is equal to the risk of the outcome multiplied by the same factor.

2.5.3 Robust MDP

As we said before, the Robust Markov Decision Process is related to the model-uncertainty, alias when the transition kernel probabilities are not precisely known. Thus, the solution is to take the worst condition possible for the agent considering the model parameters,

$$V^{\pi} = \inf_{P \in \mathbb{P}} V^{\pi, P} \tag{2.22}$$

Hence, even there is not an explicit formulation of the Risk, optimizing a worst-case scenario implies optimizing also some for of Risk.

2.6 Dynamic Programming

Before talking about the proper RL gradient-based algorithms and the mathematical basis on top of that they work, we should first approach the analytical way to compute the optimal policy, i.e. the Dynamic Programming [1]. Indeed, having the precise solution of the problem, allows us to properly analyze the performance and the precision of the used algorithm.
DP, in general, is a method for solving complex problems by breaking them down into several subproblems, whose combined solutions give the exact total solution of the starting problem. The two main properties on which it is based are the so-called optimal substructure and the overlapping subproblems. The first one ensures that the optimal solutions of the various sub-

problems in which we fragmented the overall one when are combined, result in the optimal solution of the global problem. The second one tells us that the subproblems that are resolved once, will be resolved several times.

The good thing is that Markov Decision Processes satisfy these two properties we defined above. The Bellman Equation is de facto the recursive decomposition of the DP following the optimal substructure. Indeed, the equation has always the optimal part of the one-step followed by the optimal solution of all the following steps. The second property is satisfied by the Value function that stores the information about the optimal solutions we found out before, propagating them to the step before. Being an exact analytical solution, surely the Dynamic Programming needs the full knowledge of the Markov Decision Process.

2.6.1 Policy Evaluation

Keeping in mind our general for MDPs, which is finding an optimal policy that maximizes the discounted cumulative reward, the first thing we need is the evaluation of the policy. The evaluation of the policy means that we have to evaluate every state of the environment based on the actions selected by the policy we are currently using.

Given the Bellman equation, what we apply is an iterative method. We start with an initial approximation of $v(s)$ (usually setting all the values of the states to 0) and then we use the Bellman equation substituting the value of the approximation v_0 to compute the approximation v_1 of the previous state. Then we iterate the same process for every $k+1$ approximation computed through the approximation k until the values converge. Practically speaking, we can compute the delta between the approximation $k+1$ and the approximation k, taking the

maximum difference in the update among all the states. When it becomes sufficiently small, we can stop the Policy Evaluation.

$$
\begin{aligned}
v_{k+1}(s) &= \sum_{a \in A} \pi(a|s)(R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_k(s')) \\
\mathbf{v_{k+1}} &= \mathbf{R}^\pi + \gamma \mathbf{P}^\pi \mathbf{v_k}
\end{aligned}
\tag{2.23}
$$

2.7 Policy Improvement

After the evaluation of the policy, we need to know how to make it better respect to its value. This part is called Policy Improvement in Dynamic Programming. What we have to do is checking if, for a given state, it is better to follow the currently adopted policy or to select a different action. How we evaluate the goodness of a certain action in a certain state is through State-Action Value Function and its Bellman equation [1]. So what we are comparing the evaluation of the action taken by the current policy and all the other possible actions.

$$
\begin{aligned}
q_\pi(s,a) &= \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s, A_t = a] \\
&= \sum_{s',r} p(s', r|s, a)[r + \gamma v_\pi(s')]
\end{aligned}
\tag{2.24}
$$

If the evaluation of any of the alternative actions is greater than the one chosen by the policy we change it with the one having the maximum value, greedily improving the policy itself.

$$q_\pi(s, \pi'(s)) = \max_{a \in A} q_\pi(s, a) = q_\pi(s, \pi(s)) = v_\pi \tag{2.25}$$

Hence the Bellman equation [1] for the optimality is satisfied

$$v_\pi(s) = \max_{a \in A} q_\pi(s, a) \tag{2.26}$$

Finally, we put together these two algorithms to find the best policy. We start with random policy, possibly initialized with probabilities of all actions set to the same value for every state of the system. Then we evaluate it through Policy Evaluation until the value converges as we explained before. Having the value of the policy, we check if we can improve it greedily, evaluating all the actions in every state respect to the one chosen by the policy. If every action is always better than the others for every state, we have the optimal policy. Otherwise, we change the policy with the better action discovered and we evaluate the new policy. Thus, the algorithm stops as soon as the policy is stable which means that chooses always the best action. This algorithm is known as Policy Iteration in the DP literature.

2.7.1 Value Iteration

However, this algorithm has a computational issue: it continuously computes the evaluation of the policy every time the policy improvement part changes at least one action of the current policy. So the question is, do we need to have the exact values for every state to evaluate the policy? The answer is no and so we can truncate the convergence of the policy evaluation algorithm to a few steps. In particular one of the best ways to truncate the policy evaluation

step is when we stop it after just one update for each state, with the so-called Value Iteration algorithm. Indeed, it simply turns the Bellman optimality equation of the value function onto an update rule. The update is almost identical to the one done with the standard policy evaluation except for the fact that we are not taking the expected value over the actions but the maximum. Now we have to make converge this algorithm with the usual delta method and once we have our value function for every state the optimal policy can be found simply taking the action that maximizes the expected value of the next state, i.e. the best action for every state.

$$\begin{aligned} v_{k+1}(s) &= \max_{a \in A}(R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_k(s')) \\ \mathbf{v_{k+1}} &= \max_{\mathbf{a} \in \mathbf{A}} \mathbf{R^a} + \gamma \mathbf{P^a v^k} \end{aligned} \tag{2.27}$$

All these algorithms converge to an optimal policy for discounted finite MDPs and as we said we will use them to evaluate our RL algorithms results.

2.8 Reinforcement Learning

Reinforcement Learning as we said is a set of Machine Learning algorithms that aim at solving the optimization problem of the cumulative discounted outcome, as we presented above. They can be interpreted as computational methods for solving MDPs by interacting directly with the system for which we may or may not know the structure of the model. Reinforcement Learning is something in what is known as Supervised and Unsupervised Learning in the ML

field. Indeed in the former, the algorithm has to learn the mapping between some input variables x and an output variable y, both in Discrete (Classification) and Continuous (Regression) case. The learner is provided with some examples of the expected behavior and the decision-maker has to understand how to replicate it. In the latter, the goal is to find the underlying structure or distribution of the data, deeply characterizing them, without any information from an external supervisor. The most common example is indeed Clustering. On the other hand, in RL we have information only in terms of Rewards, which are partial feedback of the goodness of how the agent is performing with its actions. One of the main problems that Reinforcement Learning algorithms have to face is the trade-off between exploration and exploitation. Indeed, the agent, that tries to maximize the cumulative reward function, tends to take those actions it a have already performed and have found out they gave it high rewards. This is exploitation. On the other hand, the decision-maker should also try other actions to understand if they are better or not, even if in this way they could perform poorly. And this is, of course, the exploration strategy. The objective is to find a good balance between them

2.8.1 Model-Free Reinforcement Learning algorithms

There are two main classes of Reinforcement Learning algorithms: the Model-Based and the Model-Free. The formers, as the name suggests, need to have all the information about the underlying model. The best example is Dynamic Programming, that as we explained has the complete knowledge of the Markov Decision Process. On the other hand, Model-Free methods have to sample the MDP to have enough statistical information about the model. Anyhow, the optimal state-action value function Q^* remains the same

$$\int_{\mathbb{A}} \pi^*(s,a)Q^*(s,a)da = \sup_a Q^*(s,a) \tag{2.28}$$

However, there are three different statistics of the problem that can be approximated without the knowledge of the Markov Decision Process:

1. Model-approximation: approximation of the MDP and then the estimation of the optimal policy through Dynamic Programming
2. Value-approximation: approximation of $V*$ and $Q*$ by sampling and then greedy computation of the optimal policy π^*
3. Policy-approximation: direct estimation of the policy

There exist also algorithms that combine these approaches.

Dealing with the first solution, Learning the model could be complex but anyway easier than learning the policy's value or the policy itself. However, usually, it is not possible to derive the value functions through Dynamic Programming and to find greedily an optimal policy. We can sample different trajectories to determine the value functions and the policy. But in this way, the accuracy of the value function and the policy can be as good as the approximated model so it is usually preferable to adopt the other approaches.

The second technique is one of the most used in Reinforcement Learning and it is split into two different kinds of algorithms depending on when they perform the update of the value function, that could be online or offline [1]. The on-policy ones approximate the value function through samples of the same policy that is being evaluated. Whereas, the off-policy algorithms

approximate the value of a policy which is not the one used in sampling.

Finally, the policy-approximation approach consists in storing policy and updating it to maximize a defined measure of performance and thus converge to the optimal policy. If we only store the policy the algorithm is called actor-only, while if we store also the value function it is known as the actor-critic algorithm.

2.9 Policy Gradient Algorithms

The policy gradient algorithms use the policy-approximation approach and consist in storing and greedily approximating the optimal policy, without the total knowledge of the MDP in which the optimization problem is defined. There two main approaches to approximate the policy: the first one is the actor-only, in which the value function instead is not approximated (indeed the actor step is referred to the computation of the policy), the second one is based on the exploitation of the approximation of the value function and is divided into two phases, this is why it is called actor-critic algorithm. The fact of storing the policy estimation derives from the fact that, as we said before, the optimal policy can be found acting greedily, but in large or continuous space the complexity of the optimization problem can become too high and could too much computationally expensive to solve it. Having a policy from which we can select the actions it is beneficial in this sense. Other advantages other value-based techniques are the fact that the policy representation is simpler and more compact than the value one and is that the gradient-based approach can learn also stochastic policies useful for instance in partially observable systems. Being an approximated approach, Policy Gradient algorithms usually lead

to a suboptimal solution to the problem. However, this is considered enough in the majority of cases for practical reasons.

2.9.1 General idea of Policy Gradients algorithms

The Policy Gradient Methods [1] give an approximation of the optimal policy through the adoption of a parameterized policy $\pi : S \times A \times \Theta \to R$, with parameter vector $\theta \in \Theta \subseteq \mathbb{R}^{\mathbb{D}_\theta}$, that returns the probability of acting a being in the state s. So the objective of the algorithm is to optimize the policy parameters θ maximizing what we defined as the objective function that we can call J [3].

$$\theta^* = \arg\max *_{\theta \in \Theta} J(\Theta) \tag{2.29}$$

The Policy Gradient approach, which as we said is a model-free set of algorithms, is based on the idea of updating the policy parameters step by step, according to the well-known gradient ascent method (or descent if the objective is minimizing). It consists of adding to the previous value of θ, of the gradient of the objective function, controlling the size of the step multiplying by the gradient a time-dependent factor called learning rate [3].

$$\theta_{k+1} = \theta_k + \alpha_k \nabla_\theta(\theta_k) \tag{2.30}$$

Anyway, if the gradient estimate is unbiased and the learning rate satisfies the Robbins-Monro conditions the algorithm is proved to converge at least to a local optimum of the function we want to optimize [3].

$$\sum_{k=0}^{\infty} \alpha_k = \infty$$
$$\sum_{k=0}^{\infty} \alpha_k^2 < \infty \tag{2.31}$$

2.9.2 Objective Functions

Recalling the definition of the value function, we can choose as the objective function to be optimized the starting Value, i.e. the value of the starting state [3].

$$J_{start}(\theta) = V_{\pi_\theta}(s_0) = \mathbb{E}_{\pi_\theta}[G_0|S_0 = s_0] \tag{2.32}$$

This, as we mentioned above, works in the discrete case but not in the continuous one. If we want to treat the second problem we should use the average value or the average reward per time step [3].

$$J_{avV}(\theta) = \mathbb{E}_{S\sim d^\theta}[V_{\pi_\theta}(S)] = \int_{\mathbb{S}} d^\theta(s) V_{\pi_\theta}(s) ds \tag{2.33}$$

$$J_{avR}(\theta) = \rho(\theta) = \mathbb{E}_{S\sim d^\theta, A\sim\pi_\theta}[R(S,A)] = \int_{\mathbb{S}} d^\theta(s) \int_{\mathbb{A}} \pi_\theta(s,a) R(s,a) da ds \tag{2.34}$$

2.9.3 Likelihood Ratio Approach

The most common way to estimate the gradient is the Likelihood Ratio approach [1]. Having Z as Random variable, p_θ his probability density that is known, explicitly computable and differentiable respect to the parameters θ. The gradient computed with likelihood ratio is [3]

$$\nabla_\theta \mathbb{E}_{Z\sim p_\theta}[f(Z)] = \nabla_\theta \int p_\theta(z) f(z) dz \tag{2.35}$$

If some regularized assumptions are satisfied we can rewrite it as [3]

$$\nabla_\theta \mathbb{E}_{Z\in p_\theta} = \mathbb{E}_{Z\sim p_\theta}[\nabla_\theta \log p_\theta(Z) f(Z)] \tag{2.36}$$

In this way is easy to approximate the gradient through the straightforward estimation [3]

$$\nabla_\theta \mathbb{E}_{Z\in p_\theta} \approx \frac{1}{M} \sum_{m=1}^{M} \nabla_\theta \log p_\theta(Z_m) f(Z_m) \tag{2.37}$$

2.9.4 Monte Carlo methods

The main policy gradient method is the Monte Carlo one, based on the estimation of the gradient through a set of trajectories of the MDP denoted as hst, at, with probability p_θ(h) to perform this trajectory following the policy π_θ. G(h) is the expected return gained by the trajectory h [3].

$$G(h) = \mathbb{E}[G_0|H = h] = \sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \tag{2.38}$$

Considering the start value as the objective function, being in the discrete case we can we rewrite it as an expectation over all the possible sampled trajectories [3]

$$J_{start}(\theta) = \mathbb{E}_{H\sim p_\theta}[G(H)] \tag{2.39}$$

Through the adoption of the likelihood ratio approach, we derive [3]

$$\nabla_\theta J(\theta) = \mathbb{E}_{H\sim p_\theta}[\nabla_\theta \log p_\theta(H) G(H)] \tag{2.40}$$

The most important point is that we can compute the gradient of the logarithm without knowing the transition kernel P. Actually, only the policy depends on the parameters θ [3]

$$\nabla_\theta \log p_\theta(H) = \sum_{t=0}^{\infty} \nabla_\theta \log \pi_\theta(s_t, a_t) \tag{2.41}$$

Usually, given the fact that the expected values of the gradient are proved to be 0, a constant known as a baseline can be subtracted without modifying the real value of the gradient [3].

$$\nabla_\theta J(\theta) = \mathbb{E}_{H\sim p_\theta}[\nabla_\theta \log p_\theta(H)(G(H) - b)] \tag{2.42}$$

This constant can be interpreted as the expected return under the policy, in this way higher average returns turn out in positive gradient and vice versa the lower ones. Another advantage of adding the baseline in the formula is that if suitably computed it can leverage the variance of the estimation of the gradient. Thus, the final estimation of the gradient with H sampled trajectories, following the policy $\pi\theta$ with the Monte Carlo approximation is shown below [3]

$$\hat{g}_{RF} = \frac{1}{M} \sum_{m=1}^{M} [\sum_{i=0}^{T^{(m)}} \nabla_\theta \log \pi_\theta(s_i^{(m)}, a_i^{(m)})][\sum_{j=0}^{T^{(m)}} \gamma^j r_{j+1}^{(m)} - b] \tag{2.43}$$

This algorithm is known in Reinforcement Learning field as REINFORCE algorithm and the convergence is guaranteed.

A good way to compute the baseline is when it minimizes the Variance of the Return. Here below is shown the computation used by us too [3]

$$b_k^* = \frac{\sum_{m=1}^{M}[\sum_{i=0}^{T^{(m)}} \partial_{\theta_k} \log \pi_\theta(s_i^{(m)}, a_i^{(m)}]^2 \sum_{j=0}^{T^{(m)}} \gamma^j r_{j+1}^{(m)}}{\sum_{m=1}^{M}[\sum_{i=0}^{T^{(m)}} \partial_{\theta_k} \log \pi_\theta(s_i^{(m)}, a_i^{(m)}]^2} \tag{2.44}$$

2.10 GPOMDP

However, the Monte Carlo approach leads to a high variance in the estimation, making the computation usually too slow. Observing that future actions and past rewards are independent unless the policy has been changed, we can derive that [3]

$$\mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(S_t, A_t) R(S_s, A_s)] = 0, \forall t > s \tag{2.45}$$

This simplifies the above formula in this way, generating the so-called Gradient of the average reward in Partially Observable Markov Decision Processes (GPOMDP), which one of the fundamental algorithms in RL and it is used a lot (indeed we will use it in our experiments) [3].

$$\hat{g}_{GPOMDP} = \frac{1}{M} \sum_{m=1}^{M} \sum_{i=0}^{T^{(m)}} \nabla_\theta \log \pi_\theta(s_i^{(m)}, a_i^{(m)}) (\sum_{j=0}^{T^{(m)}} \gamma^j r_{j+1}^{(m)} - b) \tag{2.46}$$

2.11 Stochastic Policies

The likelihood ratio approach we have just seen can be adopted only in the case of stochastic policies. Usually, this is not a huge problem, given the fact that stochastic policies allow the agent to explore the state-action space, besides, to exploit what he learned and this is required in the majority of RL settings. The most used stochastic policy (which we used too) is the Boltzmann exploration policy, called also softmax. The formulation of the policy is the following [3]

$$\pi_\theta(s,a) = \frac{e^{\theta^T \phi(s,a)}}{\sum_{b \in A} e^{\theta^T \phi(s,b)}} \tag{2.47}$$

where $\phi(s,a)$ is the feature vector which specific to the state s and action a considered. Then the likelihood gradient of the policy can be expressed as [3]

$$\nabla_\theta \log \pi_\theta(s,a) = \phi(s,a) - \sum_{b \in A} \pi_\theta(s,b)\phi(s,b) \tag{2.48}$$

CHAPTER 3

RELATED WORKS

As we said, our field of application is the Risk-Averse Reinforcement Learning, in which we want to minimize some sort of Risk instead of the usual cumulative cost. Specifically, we will take into account the Conditional-Value-at-Risk, an important coherent risk measure that, as we saw above, is the expected value of the cumulative costs greater or equal to the Value-at-Risk under τ, which is the minimum value z such that the probability of the cost being less or equal to it, is greater or equal to τ.

There are several related works we studied within the literature that helped us the derivation of our algorithm.

To properly judge the goodness of our approach, we first need to take into account the main CVaR optimization strategies within the literature. After reading some papers, it turned out that on the most considered ones is the Gradient Conditional-Value-at-Risk minimization stated by Tamar [6]. Now we will shortly explain the fundamental steps of it, giving us the possibility to compare this solution to our approach. Then we will consider other smaller related works about CVaR optimization.

3.1 Optimizing the CVaR via Sampling

The solution proposed by Tamar [6] goes exactly in the opposite direction of our research intentions. The whole idea behind the paper is based on the derivation of the specific gradient for this coherent risk measure.

Thus, let's start from the well known Conditional-VAlue-at-Risk formula [6]

$$\rho_\tau(Z) = \mathbb{E}[Z|Z \geq v_\tau(Z)] \tag{3.1}$$

where C is the standard cumulative cost and v_τ is the VaR under τ.

To compute the gradient we need first of all three assumptions [6]:

1. C is a continuous bounded random variable for all θ
2. The gradients of CVaR, $\frac{\partial \rho_\tau(Z;\theta)}{\partial \theta_i}$, and VaR, $\frac{\partial v_\tau(Z;\theta)}{\partial \theta i}$, exist and are bounded for every θ_i associated to the policy
3. The gradient of the policy $\frac{\partial p_Z(Z;\theta)}{\partial \theta_i}$ exists and is bounded for all parameters θ_i and costs z

Then, the gradient of the coherent risk measure is defined as [6]

$$\frac{\partial \rho_\tau(Z;\theta)}{\partial \theta_i} = \mathbb{E}_{\pi_\theta}[\frac{\partial \log p_\theta(Z)}{\partial \theta}(Z - v_\tau(Z;\theta)|Z \geq v_\tau(Z;\tau)] \tag{3.2}$$

Notice that $p_Z(Z;\theta)$ is the parametric probability density of the random variable Z, with θ that are the policy parameters.

The proof is quite straightforward: Defining L_θ as the set-level $\{z \in [-d,d] : z \geq v_\tau(Z;\theta)\}$,in which $[-d,d]$ defined the bounding values of Z, we have [6]

$$\int_{z \in L_\theta} p_Z(z;\theta)dz = 1 - \tau \tag{3.3}$$

Through the derivative and Leibniz rule, we lead to [6]

$$\begin{aligned} 0 =& \frac{d}{d\theta_i} \int_{-d}^{v_\tau(Z;\theta)} p_Z(z;\theta) \\ & \int_{-d}^{v_\tau(Z;\theta)} \frac{\partial p_Z(z;\theta)}{\partial \theta_i} dz + \frac{\partial v_\tau(Z;\theta)}{\partial \theta_i} p_Z(v_\tau(Z;\theta);\theta) \end{aligned} \tag{3.4}$$

Then, expressing Equation 3.1 with the integrals, we obtain [6]

$$\begin{aligned} \rho_\tau(Z) =& \int_{z \in L\theta} \frac{p_Z(z;\theta)z}{1-\tau} dz = \\ & = \frac{1}{1-\tau} \int_{-d}^{v_\tau(Z;\theta)} p_Z(z;\theta) z dz \end{aligned} \tag{3.5}$$

Taking again the derivative with the Leibniz rule [6]

$$\frac{\partial \rho_\tau(Z;\theta)}{\partial \theta_i} = \frac{1}{1-\tau} \int_{-d}^{v_\tau(Z;\theta)} \frac{\partial p_Z(z;\theta)}{\partial \theta_i} z dz + \frac{1}{1-\tau} \frac{\partial v\tau(Z;\theta)}{\partial \theta_i} p_Z(v_\tau(Z;\theta);\theta) v_\tau(Z;\theta) \tag{3.6}$$

Substituting Equation 3.4 in Equation 3.5 we derive [6]

$$\frac{\partial \rho_\tau(Z;\theta)}{\partial \theta_i} = \frac{1}{1-\tau}\int_{-d}^{v_\tau(Z;\theta)} \frac{\partial p_Z(z;\theta)}{\partial \theta_i}(z - v_\tau(Z;\theta))dz \tag{3.7}$$

Thanks to the likelihood ratio usual transformation consisting of the multiplication and division by $p_Z(z;\theta)$, allowed by Assumption 3, we prove the equality with initially expected value expression.

The estimation of this gradient could be done via sampling of N trajectories and it is given by [6]

$$\hat{\rho}_{j;N} = \frac{1}{1-\tau}\frac{1}{N}\sum_{i=1}^{N} \frac{d \log p_\theta(h^{(m)})}{d\theta}(C_i - \hat{v})\mathbf{1}_{C_i \geq \hat{v}} \tag{3.8}$$

where $p_\theta(h^{(m)}$ is the usual probability of obtaining the trajectory h following the policy π_θ. Notice that we have also the estimation of the VaR, hatv, equal to $c_{\lceil \tau N \rceil}$, having the sorted list of the cumulative costs.

3.1.1 Considerations about the paper

The application to the Reinforcement Learning setting can be easily done, having the standard cost function C(s,a) and using a stochastic policy for the derivation of the gradient, like the softmax (or Boltzmann), to compute the probability $p_a(a;\theta)$ of taking the action a following the policy π_θ

Besides, Tamar [6] affirms two interesting things about the algorithm he derived, called Gradient Conditional-Value-at-Risk (GCVaR), that will be useful for our final considerations and

especially for the juxtaposition between out two solutions. The first one is the fact that without a state-space augmentation, the policy found by the algorithm is not guaranteed to be optimal, and in some cases, it will be necessary to adopt it. The second is a consideration of the CVaR estimation having a high variance, given the fact that the averaging is effectively computed merely over $(1 - \tau)N$ samples. This is a typical problem within techniques to estimate VaR and CVaR through sampling. It is usually alleviated by approaches reducing the variance, like for instance Importance Sampling. The technique is based on the idea of using samples from different distributions and properly modifying to estimators to not make them biased. However, this solution is not completely trivial and is highly domain-dependent.

Tamar extends also his method to all the set of coherent risk measures through a general formulation of the optimization problem based on the formulation of the coherent risk measures as a minimization over a worst-case scenario under the Envelope of the risk measure itself [5]. We will not analyze this method because it departs from our approach from the specific case of the CVaR minimization problem considered in this thesis.

3.2 MDP tranformation

Our main idea is to find a suitable transformation of the Markov Decision Process to allow us the application of a standard Reinforcement Learning algorithm and at the same time to minimize not the standard objective function of the cumulative rewards, but certain Risk-Averse objective functions. We deeply studied the state of the art in the literature, studying a different kind of transformations for different settings and Risk Measure. Finally, we figured out that

two main possible MDP transformations are mostly used: the Robust MDPs and the state space augmentation. The first is well presented by Osogami in his paper about Robustness and risk-sensitivity in Markov decision processes [7]. The second one is presented by Bauerle in Markov Decision Processes with Average-Value-at-Risk [8] criteria and more in general in More risk-sensitive Markov Decision Processes [9]. The basic concept behind these transformations is the fact that giving a new formulation of the problem we are moving the optimization problem towards the model uncertainty (in the case of robust MDPs) or adding some other important information to the typical state to act. In the following section, we will present these types of transformations, in particular, the second one, which is the one we have chosen to adopt, that will be deeply analyzed in the Methods and Theoretical Results.

3.2.1 Robust Markov Decision Processes transformation

As we said one of the typical transformations applied to the MDP is the one related to the robustness. As we said in the background chapter, a robust MDP is a Markov Decision Process that takes the worst case of what is known as model-uncertainty. This kind of uncertainty models the risk considering the precision in which the parameters of the model can be approximated. Thus, it consists of optimizing the possible with the worst possible realization of the parameters (the worst situation in which the environment can be presented to the agent). The robust MDP minimization problem can be written as [7]:

$$\min \pi \max_{p \in U_p, f \in U_f} \mathbb{E}_{p,f}[C^T(\pi)] \tag{3.9}$$

where p and f are the model parameters respectively for the transition and the cost function. Osogami in [7] shows the relation between the optimization of the expected exponential utility function, which is taken to minimize the Risk and a robust MDP in which we are minimizing the worst-case expectation having a penalty parameter multiplied by the divergence of the uncertain parameters from their nominal values. Indeed, dealing with the minimization of $\mathbb{E}[exp(\gamma C^T(\pi))]$ for $\gamma > 0$, that is known to be equal to the minimization of the Entropic Risk Measure (ERM): $ERM_\gamma[C^T(\pi)] = \frac{1}{\gamma} \ln \mathbb{E}[exp(\gamma C^T(\pi))]$, we have for [7] that Equation 3.9 is equal to the minimization of [7]

$$ERM_\gamma[Y] = \max_{q \in P(q0)} \{E_{q0}[Y] - \gamma KL(q||q0)\} \tag{3.10}$$

wherein this case, KL is the Kullback-Leibler divergence.

Furthermore, always [7] shows that the Risk-Averse optimization of an Iterated Risk Measure in which the considered risk measure is a coherent risk measure (like the Conditional-Value-at-Risk) is equal to a robust MDP in which we minimize the worst-case expectation in which we use a concave function to describe the divergence of the uncertain parameters from their nominal values.

Also in this case, Equation 3.9 is equal to [7]

$$\min_\pi ICTE_\alpha^T[D^T(\pi)] \tag{3.11}$$

where α is the uncertainty parameter and D^T the cumulative value of D(s, a).

3.2.2 State-Space Augmentation

The second way in which a Markov Decision Process could be transformed to embrace the Risk-Sensitivity in the optimization problem is the State-Space Augmentation. Bauerle in his papers presents its solution showing the possibility to apply it to CVaR optimization problem [8] and in general the utility function optimization problem. For the sake of completeness, we show here the bivariate MDP formulation for the Utility function case, while we will go through the proof for the Conditional-Value-at-Risk (useful for the derivation of our solution) in Methods and Theoretical Solution chapter [8]

$$\begin{aligned} V_{t\tilde{\pi}(x,s)} &= \mathbb{E}_x^{\tilde{\pi}}[U(U^T + s)], x \in X, s \in \mathbb{R}_+, \tilde{\pi} \in \tilde{\Pi} \\ V_{t(x,s)} &= \inf_{\tilde{\pi} \in \tilde{\Pi}} V_{t\tilde{\pi}(x,s)}, x \in X, s \in \mathbb{R}_+ \end{aligned} \tag{3.12}$$

in which s stores the cumulated cost along the performed trajectory. Indeed, the main idea that underlines the solution of Bauerle is to augment the state space with a variable that could describe useful information about history. With this trick, we are capable to resolve non-Markovian problems finding policies that are some way can still be considered Markovian, in the sense that they don't depend on the history but only on a statistic of it. As we will see, this transformation leads to the opportunity to resolve the optimization problem with Dynamic Programming, exploiting the equations derived in [8]. Then, varying the various initial values of this variable s for the starting state and keeping the value of τ, which controls the Risk-

Aversion of the agent, fixed, we can easily minimize the Conditional-Value-at-Risk finding the optimal s^*.

3.2.3 Different State-Space Augmentation for CVaR

In our research phase within the literature, we found also another interesting approach that considers the state-space augmentation to minimize the Conditional-Value-at-Risk within finite horizon settings. That is the case of Carpin's paper [10]. Even in this approach, the idea is to store some information about the history that in this case are the cumulated running cost y and the current step z. Then, the rest is based on the computation of the occupancy distribution and finally using a bilinear program to practically resolve the optimization problem. The formulation is the following: the minimization of CVaR can be written as [10]

$$\min_{p,\theta} \min_{s\in[0,Kd]} s + \frac{1}{1-\tau}\sum_{y\in N}(y-s)^{+}\theta(y) \tag{3.13}$$

where $s \in \mathbb{R}$ and τ is the Risk-Aversion parameter. More other, [10]

$$\theta(k) = \sum \mathbf{1}(y = k \ \ \& \ \ z = d)p(x, y, z, a), k \in [0, N] \tag{3.14}$$

and the occupancy measure has the following constraints [10]

$$\sum_{x',y',z'} \in X'^{T} \sum_{a\in A(x',y',z')} p(x', y', z', a)[\delta_(x, y, z)(x', y', z') - P'((x', y', z')|(x, y, z), a)] = \beta(x, y, z) \forall (x, y, z) \in X \tag{3.15}$$

The equation states that ρ is a vector with $|K'|$ nonnegative components and these legitimate vectors are constrained by the initial distribution β and determined by the policy π.
Hence, using unconstrained stochastic control approaches, the optimal policies can be found with different Risk-Aversion degrees, given by much the value of τ is high.

CHAPTER 4

THEORETICAL SOLUTION AND METHODS

In this chapter, we will present the methods and the main theoretical contributes to our thesis work. First of all, we will describe the State Space Augmentation derived by Bauerle [8] that we adopted for our solution. Then we will go into details of the structure of the algorithm we adopted for the minimization of the Conditional Value at Risk with the mentioned state-space augmentation [8]. The idea is to optimize the risk measure without changing the RL algorithm structure: indeed the State Space Augmentation, as we saw in the previous chapter, induces also a modification of the Transition function and the Cost function. The latter returns 0 for every 2-Dimensional state, apart from the terminal one in which it is returned the variable s in which the cumulated cost history is stored clipped to 0 in case the cumulated cost did not exceed the initial value of s in the starting state. The only real addition will be the optimization of the initial value of s that in any case can be separated from the algorithm for the optimization of the policy parameters. Indeed, Bauerle and Ott [8] divides the solution for CVaR into an inner and an outer optimization problem. To be more clear, we will call our application of Reinforcement Learning algorithms to the augmented space MDP problem as the Augmented Policy Gradient algorithm (APG). Then, we will describe the main differences in terms of structure between our solution and the one gave by Tamar in [6] with his Gradient Conditional Value at Risk algorithm, based on the derivation of the CVaR specific gradient as we saw.

4.1 Bauerle in Markov Decision Processes with Average-Value-at-Risk criteria

To present the state space augmentation of Bauerle [8] we need first to define the Conditional-Value-at-Risk (or Average-Value-at-Risk), a particular coherent risk measure commonly used in Risk-Sensitive Reinforcement Learning optimization problems. Then we will deeply analyze the Dynamic Programming rules stated in the paper that we will implement allowing us to compare the performances of our solution to the exact analytical one derived by DP.

4.1.1 The Conditional Value at Risk

The Conditional-Value-at-Risk is in turn based on the concept of Value at Risk as the name itself suggests:

Let Z $\in$ L(Ω, F, $\mathbb{P}$) defined as random variable with real values and let $\tau \in (0,1]$.

1. The Value-at-Risk of Z at level τ is represented as [8]

$$VaR_\tau(Z) = inf\{z \in \mathbb{R} : \mathbb{P}(Z \leq z) \geq \tau\} \tag{4.1}$$

2. The Conditional Value at Risk of Z at level τ, is computed as

$$CVaR_\tau(Z) = \frac{1}{1-\tau}\int_\tau^1 VaR_\tau(Z)dt \tag{4.2}$$

With a continuous distribution of Z, the Conditional-Value-at-Risk can be rewritten as [8]

$$CVaR_\tau = \mathbb{E}[Z|Z \geq VaR_\tau(Z)] \tag{4.3}$$

Thus, we want to fix the τ taking a value in $(0, 1]$ and then optimize the risk measure. The finite horizon version (which is the case of our setting) is as follows [8]

$$\inf_{\pi \in \Pi} CVaR_{\tau}^{\pi}(C^N | Z_0 = z) \tag{4.4}$$

in which C^N is the cumulative cost over N steps and $CVaR_{\tau}^{\pi}$ is the Conditional-Value-at-Risk taken under the policy π. The optimization problem for the finite horizon setting becomes [8]

$$\inf_{\pi \in \Pi} CVaR_{\tau}^{\pi}(C^N | Z_0 = z) = CVaR_{\tau *}^{\pi}(C^N | Z_0 = z) \tag{4.5}$$

The interesting thing of this optimization is that in this way is not a standard MDP anymore, given the fact that the Conditional-Value-at-Risk is a convex risk measure. Anyway, we can note that if $\tau \to 0$ the formulation is simplified to the common optimization problem over the expectation of the cumulated costs [8].

$$\lim_{\tau \to 0} CVaR_{\tau}^{\pi}(C^N | Z_0 = z) = CVaR_{\tau *}^{\pi}(C^N | Z_0 = z) \tag{4.6}$$

Whereas, with $\tau \to 1$ we derive the Worst-Case risk measure as shown below [8]

$$WC(C^N) := \sup_{\omega} C^N(\omega) \tag{4.7}$$

4.1.2 MDP Transformation for finite horizon

The first thing we have to do is defining the Conditional-Value-at-Risk as a solution to a convex optimization problem. In particular [8]

$$CVaR_\tau(Z) = \min_{s\in\mathbb{R}}\{s + \frac{1}{1-\tau}\mathbb{E}[(Z-s)^+]\} \tag{4.8}$$

and the minimum point s^* is actually the Value at Risk $VaR_\tau(X)$

Ergo, we derive the following formulation of the optimization problem

$$\begin{aligned}
&\inf_{\pi\in\Pi} CVaR_\tau^\pi(C^N|Z_0 = z) = \inf_{\pi\in\Pi}\inf_{s\in\mathbb{R}}\{s + \frac{1}{1-\tau}\mathbb{E}[(Z-s)^+]\} \\
&= \inf_{s\in\mathbb{R}}\inf_{\pi\in\Pi}\{s + \frac{1}{1-\tau}\mathbb{E}[(Z-s)^+]\} \\
&= \inf_{s\in\mathbb{R}}\{s + \frac{1}{1-\tau}\inf_{\pi\in\Pi}\mathbb{E}[(Z-s)^+]\}
\end{aligned} \tag{4.9}$$

Hence we have a double optimization problem, an inner one and an external one. The inner one is the most interesting because it consists of finding an optimal policy for this kind of modified expectation over the cumulative cost. Then we will explain also the solution to the external problem that is less complex to solve in practice but is very interesting theoretically speaking.

4.1.3 Inner Optimization Problem

As we said this part considers the optimization under the policy of the Expected Value of the Cumulative Costs subtracted by "s" (alias the Value at Risk) clipping the difference to 0

in case it is negative. Let's say that we are concentrating our attention on that Cumulative costs that exceed the VaR, being coherent with the well known standard definition of CVaR. The optimization problem can be divided in this way: for any time step t=0,1,..., T (with T as finite horizon upper bound) [8]

$$\begin{aligned} w_t\pi(x,s) &:= \mathbb{E}[(C^t - s)^+],\ x \in X, s \in \mathbb{R}, \pi \in \Pi \\ w_t(x,s) &:= \inf_{\pi \in \Pi} w_t\pi(x,s),\ x \in X, s \in \mathbb{R}, \pi \in \Pi \end{aligned} \tag{4.10}$$

To optimize this new objective function, we need to define a different Markov Decision Process which differs from the most common one by a state-space augmentation. Indeed, we have this 2-dimensional state $\tilde{X} := X \times \mathbb{R}$, where X is the standard State Space to which we add a Real-Valued number, while the action space $\tilde{A}$ is equal to the standard action space A. So, the new single state will be described by the couple $(x, s) \in \tilde{X}$, in which the interpretation of s will be presented better later but it is needed to store the important information about the history of the trajectory. In fact, the agent has to decide which action to perform based on the disturbance variable $Z_t = (Z_t^1, Z_t^2) = (X_t, C_t - 1)$ with values in $X \times \mathbb{R}_+$ (costs should be only positive), that is to say the choice of the decision-maker depends on the state in which and on the past cumulative cost encountered so far. Thus, the transition function is redefined as $P : \tilde{X} \times A \times X \times \mathbb{R}_+ \to \tilde{X}$, specifically [8]:

$$\tilde{P}((x,s), a, (z1, z2)) = (z1, \frac{s - z2}{\beta}) \tag{4.11}$$

The functions could seem more complex than what it is, indeed it is simply the classical transition function used to retrieve the next state position given the previous state and the action performed, to which it is added the update of the history to be stored in the variable s of the augmented state. Here we show the relationship within the two, underlining the effect of the augmentation to the transition function [8]

$$\tilde{P}((x,s),a,(x',s')) = P((x,a,x')\mathbf{1}\{s' = \frac{s-c}{\beta}\} \tag{4.12}$$

where c is the cost incurred in that transition. The huge difference is that this cost is not returned to the agent during the run, but it is simply used to update the history variable s of the state. The cost function $\tilde{C} : \tilde{X} \to \mathbb{R}^+$ returns 0 for every augmented state apart from the terminal one, which is [8]

$$\tilde{C}_{-1\pi}(x,s) := \tilde{C}_{-1}(x,s) = s^- \tag{4.13}$$

where $s^- = -min\{0,s\} = max\{0,-s\}$

Now we should define the new value functions of the modified MDP to allow the analytical solution through Dynamic Programming, as in the standard case. So, having the state value function $\tilde{V} : \tilde{X} \to \mathbb{R}^+$ and the state-action value function $\tilde{Q} : \tilde{X} \times \tilde{A} \to \mathbb{R}^+$ we define the operator T^* as [8]

$$T * \tilde{Q}((x,s),a) = \gamma \sum_{x' \in X} P(x'|x,a)\tilde{V}(x', \frac{s-c}{\gamma}) \tag{4.14}$$

and consequently [8]

$$\tilde{V} = \min_{a \in \tilde{A}} \tilde{Q}((x,s),a) \tag{4.15}$$

The optimal policy is easily derived as [8]

$$\tilde{\pi}^*((x,s)) = \arg\min_{a \in \tilde{A}} \tilde{Q}((x,s),a) \tag{4.16}$$

Pay attention to the notation $\tilde{\pi}$ that indicates the policies within the set Π^S that depends also on the history but weakly, that is only on the variable s of the state, not on the entire information about the history.

Thus, to resolve the Dynamic Program of the transformed Markov Decision Process, the following update equations are adopted [8]

$$\tilde{V}_{-1}((x,s)) = \tilde{C}_{-1}((x,s)),\ \forall x \forall s \tag{4.17}$$

$$\tilde{V} = \min_{a \in \tilde{A}} \tilde{Q}_t(x,s),a) \tag{4.18}$$

$$\tilde{Q}_t((x,s),a) = T * \tilde{Q}_{t-1}((x,s),a) \tag{4.19}$$

As in the typical DP, if there exist minimizers of V on all steps n, then optimization problem has the Markov policy π^* as the optimal policy.

Here is the proof by induction for the optimality condition thanks to the adoption of the modified MDP. For $t = 0$ [8],

$$\begin{aligned} V_{0\tilde{\pi}}(x,s) &= TV_{-1}(x,s) \\ &= \beta \int V_{-1}(x', \frac{s-c}{\beta} P(x'|x,a) \\ &= \beta \int (\frac{s-c}{\beta})^{-} P(x'|x,a) \\ &= \int (c-s)^{+} P(x'|x,a) \\ &= \mathbb{E}_x^{\pi}[(C^0 - s)^{+}] = \omega_{0\tilde{\pi}} \end{aligned} \tag{4.20}$$

Then assuming the statement true for t, the case $t+1$ will be [8]

$$\begin{aligned} V_{t+1\tilde{\pi}}(x,s) &= TV_{t\tilde{\pi}}(x,s) \\ &= \beta \int V_{t\tilde{\pi}}(x', \frac{s-c}{\beta} P(x'|x,a) \\ &= \beta \int \mathbb{E}_{x'} \tilde{\pi}(C^t + \frac{s-c}{\beta})^{+} P(x'|x,a) \\ &= \int \mathbb{E}_{x'} \tilde{\pi}(\beta C^t + c - s)^{+} P(x'|x,a) \\ &= \mathbb{E}_x^{\pi}[(C^{t+1} - s)^{+}] = \omega_{t+1\tilde{\pi}} \end{aligned} \tag{4.21}$$

Finally, thanks to the theorem 2.2.3 in [11] we can prove that taking Σ is the non-Markovian policies (totally dependent on the history) [8]

$$\inf_{\tilde{\pi}\in\tilde{\Pi}} V_{t\tilde{\pi}(x,s)} = \inf_{\sigma\in\tilde{\Pi}} V_{t\tilde{\pi}(x,s)} \tag{4.22}$$

thanks to which we can derive [8]

$$\inf_{\tilde{\pi}\in\tilde{\Pi}^M} \omega_{t\tilde{\pi}} \geq \inf_{\pi\in\Pi} \omega_{t\pi} \geq \inf_{\sigma\in\Sigma} V_{t\sigma} = \inf_{\tilde{\pi}\in\tilde{\Pi}^M} V_{t\tilde{\pi}} = \inf_{\tilde{\pi}\in\tilde{\Pi}^M} \omega_{t\tilde{\pi}} \tag{4.23}$$

Under these three assumptions and the theorem 2.4.6 of [11], than [8] shows that this optimal policy exists:

1. D(x) is compact for all $x \in X$
2. x $\rightarrow$ D(x) is upper semicontinuous
3. $(x, a) \rightarrow \int C(x', \frac{s-c}{\beta} P(x'|x, a)$ is lower semicontinuous for all lower semicontinuous functions $vgeq0$

4.1.4 Outer optimization problem

The second part of [8] proofs the optimality fro the outer problem, but we are not interested in the details of it because we are focusing on the policy optimization of the inner one in our research work. Anyway, the result is fundamental for ur solution and it is the fact that given the outer problem formulation [8]

$$\inf_{s\in\{\mathbb{R}} (s + \frac{1}{1-\tau}\omega_T(x, s)\} \tag{4.24}$$

It exists a solution s^* to this problem and the optimal policy found by the minimization of the inner problem Equation 4.10 with starting state $(x0, s^*)$ it is the policy that optimizes the Conditional-Value-at-Risk.

4.2 Augmented Policy Gradient algorithm

As we mentioned above, our solution is based on the State Space augmentation derived by Bauerle and that we have already explained before. So, let's focus on the objective function of the optimization problem that we have:

$$\min_{s\in\mathbb{R},\theta\in\Theta}\{f_\tau(\theta,s)\} = \min_{s\in\mathbb{R},\theta\in\Theta}\{s + \frac{1}{1-\tau}\mathbb{E}_{\pi_\theta}[(C^T - s)^+]\} \tag{4.25}$$

where T is the length of the finite horizon.

Hence, the optimization for a RL algorithm, as in the case of the Dynamic Programming, should split the solution in two optimization problems.

4.2.1 Inner Optimization of the policy

Dealing with the inner optimization problem, we notice that in case of a State Space Augmented MDP we can apply a standard Reinforcement Learning Gradient based algorithm. Actually:

$$\nabla_\theta f_\tau = \frac{1}{1-\tau}\nabla_\theta\mathbb{E}_{\pi_\theta}[(C^T - s)^+] \tag{4.26}$$

Form this formulation it seems we should adjust the computation of the gradient, because of the modification of the Expected value's argument where, instead of the standard discounted cumulative cost, we have this difference between the cumulative cost itself and the s variable of the starting state, clipped to 0 if the value of the operation is negative (i.e. the cumulative cost does not exceed the value of s). But, taking a deeper look to the argument, we can notice that this is exactly the definition of the new cost function of the modified MDP

$$(C^T - s)^+ = s_T^- = \tilde{C}_{-1}(x_T, s_T) \tag{4.27}$$

in which s is the starting cumulated cost, while s_T is the value of the cumulated cost variable at time step T (terminal state).

Thus we have to compute the gradient $\nabla \mathbb{E}_{\pi_\theta}[\tilde{C}_{-1}]$ that can be resolved by any RL algorithms based on the gradient approach, such as the common REINFORCE or GPOMDP (the one we have chosen to use for our experimental setting).

4.2.2 Outer Optimization of variable s

The outer optimization problem instead does not depend on the policy as we mentioned before. For this reason, can be resolved apart without the modification of the real Reinforcement Learning algorithm that optimizes the parameters θ.

At this point we performed two steps to reach the results we defined to achieve: firstly, we computed the RL solution following the same approach of the Dynamic Program presented in the Bauerle paper [8], resolving the outer problem in a simple discretized way.

We resolved the inner optimization problem for different values of s, computing the approxima-

tions of the various objective functions and finally, varying the value of τ, we took the minimum of the function

$$f_\tau(\theta, s) = s + \frac{1}{1-\tau} + \mathbb{E}_{\pi_\theta}[(C^T - s)^+] \tag{4.28}$$

We said discretized approach because, of course, it is practically impossible to try every Real value of s. That is why we took different suitable values of s to substitute and then we minimized the discrete functions we derived by changing the values of τ, finding the optimization of the Conditional Value at Risk.

The results, as we will see, are pretty similar to the DP ones, with some predictable approximation error given by the gradient method against the exact analytical solution.

4.2.3 Double parameters optimization

In the second step, we concentrated our attention on the estimation of the value of s to move from the Discrete to the Continuous optimization of the function f_τ.

We took into account two techniques to estimate the initial cumulated cost variable: the gradient

estimation and the closed-form one. The first one is based on the derivation of the function f_τ with respect to s:

$$\begin{aligned}
\frac{d}{ds} f_\tau(\theta, s) &= 1 + \frac{1}{1-\tau} \frac{d}{ds} \mathbb{E}_{\pi_\theta}[(C^T - s)^+] \\
&= 1 + \frac{1}{1-\tau} \mathbb{E}_{\pi_\theta}[\frac{d}{ds} \max\{(C^T - s), 0\}] \\
&= 1 + \frac{1}{1-\tau} \mathbb{E}_{\pi_\theta}[(-1)\mathbf{1}\{C^T - s > 0\} \\
&= 1 + \frac{1}{1-\tau} (-1) \mathbb{P}_{\pi_\theta}\{C^T - s > 0\} \\
&= 1 - \frac{1}{1-\tau} \mathbb{P}_{\pi_\theta}\{C^T - s > 0\}
\end{aligned} \tag{4.29}$$

At convergence, i.e. with the gradient close to 0, the value of $s$tends to the Value at Risk at level tan

$$\begin{aligned}
& 1 - \frac{1}{1-\tau} \mathbb{P}_{\pi_\theta}\{C^T - s > 0\} = 0 \\
\implies & \mathbb{P}_{\pi_\theta}\{C^T > s\} = 1 - \tau \\
\implies & \mathbb{P}_{\pi_\theta}\{C^T \leq s\} = \tau \\
\implies & s = VaR_\tau(C^T)
\end{aligned} \tag{4.30}$$

Giving these premises, one of the simplest way to estimate this kind of gradient is the following

$$\hat{\frac{d}{ds} f_\tau}(\theta, s) \cong 1 - \frac{1}{1-\tau} \frac{1}{N} \sum_{i=1}^{N} \mathbf{1}\{c_i > s\} \tag{4.31}$$

TABLE I: AUGMENTED POLICY GRADIENT

Algorithm APG $(\eta_\theta, \eta_s, \theta_0, s_0)$:
Initial Sampling N_0 trajectories
for k=0,1,...
$\theta_{k+1} = \theta_k - \eta_\theta \hat{\nabla}_\theta f(\theta, s)$
Sampling N_{k+1} trjectories
$s_{k+1} = s_k - \eta_s \frac{\hat{d}}{ds} f(\theta, s)$
until convergence or max K iterations

return π_θ^* optimal poliy

with N trajectories sampled from π_θ.

As we said, this is a simple estimator of the gradient but allows us to define the following general structure of our Augmented Policy Gradient algorithm.

In table 1 it is shown the pseudocode of the Augmented Policy Gradient algorithm. The body of the computation is pretty straightforward: it consists of a continuous update of firstly the policy parameters θ and then the value s of the initial cumulated cost. Notice that the sampling should be done every time the policy parameters are changed, but it is not needed after the update of s not loosing in this way in terms of time speed performance. Furthermore, the speed of the update can also be set differently for the two equations, so we can have all these cases:

1. One (or more) update(s) of θ and one (or more) update(s) of s
2. Convergence of θ and one (or more) update(s) of s

3. One (or more) update(s) of θ and convergence of s

4.2.4 Closed-form aprroximation of s

Hence, there are a lot of different solutions to be explored through the adoption of Augmented Policy Gradient, with different possible performances in terms of precision and speed in finding the optimal solution. In our research work, we applied the 3^{rd} case, with one update for θ and then making s to converge. But we will describe the details of the implementation later.

With the fact that we are considering the convergence of s, we can use a better estimator for the s variable. Indeed, the gradient estimator we presented before is quite weak and with low precision, being one of the simplest among quantiles' estimators. Anyway, as we stated before, s converges to the Value-at-Risk at level τ so we can apply directly the closed-form expression of VaR to compute the final value of s for the update.

These are the steps to compute the closed-form approximation of the starting value of s:

1. Collect N trajectories from π_θ
2. Define $\tilde{f}_N = s + \frac{1}{1-\tau}\frac{1}{N}\sum_{i=1}^{N}(C_i^T - s)^+$
3. Compute the sub-gradient: $\frac{d\tilde{f}_N(s)}{ds} = 1 - \frac{1}{1-\tau}\frac{1}{N}\sum_{i=1}^{N}\mathbf{1}\{Ci > s\}$
4. Ideally, we would need to find s such that: $\frac{d\tilde{f}_N(s)}{ds} = 0 \implies \frac{1}{N}\sum_{i=1}^{N}\mathbf{1}\{Ci \leq s\} = \tau$ But this might be impossible, or at strongly inaccurate due to the finite samples. Thus, we can find the s that manes $\frac{1}{N}\sum_{i=1}^{N}\mathbf{1}\{Ci \leq s\}$ closest to τ. If we choose the minimum s with this property, we can find it as:

(a) Sort the costs C_i^T in ascending order: $c_1 \leq c2 \leq ...c_N$

(b) Take the first s such that $\frac{1}{N}\sum_{i=1}^{N} \mathbf{1}\{Ci \leq s\} \geq \tau$

$\implies$ This is equivalent to taking the $\lceil \tau_N \rceil$ element of the sorted list ($\lceil \cdot \rceil$ denotes the ceiling operator

$\implies$ We get the optimum s*=$C_{\lceil \tau_N \rceil}$

$\implies$ This is the empirical quantile/VaR estimator used by Tamar []

This estimator is a way better estimator than the previous one in terms of precision and it is really easy and fast to compute. The performance of the entire implemented algorithm will be considered in the next chapter also compared to one implemented by Tamar.

4.3 Structural Comparison between APG and GCVaR

In this section we compare the skeleton of the two algorithms, highlighting the differences and the similarities between the two. From a high-level perspective, they seem almost the same: in fact, both compute the gradient of the policy and then estimate the Value-at-Risk of the cumulative costs of the N sampled trajectories. The adopted estimator is the same and it is based on the closed-form expression of VaR (taking the $C_{\lceil \tau_N \rceil}$ element of the sorted cumulative costs list). However, while Tamar uses this quantile just to build the baseline and the clipping threshold for the cumulative returns in the gradient computation, APG exploits this value as the starting cumulated cost variable which augments the state space. To understand better the difference, let's analyze the two formulations of the gradient of the objective function. In Gradient Conditional-Value-at-Risk it is written as

$$\begin{aligned}\nabla_\theta \rho_\tau &= \mathbb{E}_{\pi_\theta}[\frac{d\log p_\theta(H)}{d\theta}(C - v_\tau(C;\theta)|C \geq v_\tau(C;\tau)] \\ &\approx \frac{1}{1-\tau}\frac{1}{N}\sum_{i=1}^{N}\frac{d\log p_\theta(H)}{d\theta}(C_i - \hat{v})\mathbf{1}_{C_i \geq \hat{v}}\end{aligned} \tag{4.32}$$

with $\hat{v}$ which is the empirical (1-τ)-quantile. While for Augmented Policy Gradient is

$$\begin{aligned}\nabla_\theta f_\tau =& \frac{1}{1-\tau}\nabla_\theta \mathbb{E}_{\pi_\theta}[(C^T - s)^+] \\ &= \frac{1}{1-\tau}\mathbb{E}_{\pi_\theta}[\frac{d\log p_\theta(H)}{d\theta}(\tilde{C} - b)] \\ &\approx \frac{1}{1-\tau}\frac{1}{N}\sum_{i=1}^{N}\frac{d\log p_\theta(H)}{d\theta}(\tilde{C}_i - b)\end{aligned} \tag{4.33}$$

recalling that $\tilde{C}_{-1} = s_T^-$, while the other values are equal to 0, and b is the common baseline. Both s and $\hat{v}$ are interpreted as the Value-at-Risk and are computed with the same approach we presented before (closed-form expression).

We can notice that GCVaR derives the real gradient of the Conditional-Value-at-Risk, finding out that it is equal to the expectation under the policy of the product between the standard gradient of the policy multiplied by the difference between the cumulative costs and the quantile (instead of the common baseline). The whole thing, considering only the cumulative costs that exceed the quantile, the rest costs are clipped to 0 for the multiplication. Conceptually speaking, also the APG does the same thing but instead of taking the standard cumulative costs, it takes one of the modified MDP, which returns s_T^- in case the standard cumulative cost

exceeds the initial value of s, 0 otherwise. Furthermore, the standard baseline is still subtracted in the computation. This difference seems small, but it has a huge consequence theoretically speaking. Hence, on one hand the Tamar approach needs the RL algorithm to be adapted to compute, practically changing the code for the computation of the gradient (even if it has only to turn to 0 all the cumulative costs on which the optimization problem is not focusing on and it has to substitute the standard baseline with the VaR). Whereas, our approach just needs the augmentation of the state space which induces itself the modification of the transition function and especially of the cost function that in this way is already properly adjusted by the MDP and the RL algorithm can optimize as if it was the typical case.

Of course, the augmentation of the space leads to a consequent augmentation of the feature of the policy for the computation of the gradient. This augmentation makes the policy to consider during the selection of the action another important set of information that describes the history of the trajectory. Tamar himself recognized that in some cases this state-space augmentation is needed to achieve the optimality of the policy. However, he did not realize that this augmentation leads also to the possibility of application of standard RL algorithms, without having to derive an ad hoc one.

Last but not least there is also an issue present in the Tamar solution that is not met by the Augmented Policy Gradient algorithm. The estimation of CVaR done through GCVaR requires the application of Importance Sampling to reduce the variance of the estimation. This approach is not trivial to apply and highly domain-dependent. While, for our proposed solution, thanks to the state space augmentation, the IS is not required at least for the estimation of the gradient

of the policy (the VaR estimator instead, that it is used to find the initial cumulated cost s, could have the same problem being the same used by Tamar).

CHAPTER 5

IMPLEMENTATION AND EXPERIMENT RESULTS

In this section we are going in-depth into the implementation of our Algorithm to judge the practical performances, comparing them, especially to the CVaR solution. Firstly, we will describe the domain in which the algorithms are run and the adopted environment. Then we will show the specific implementation of the various solutions and finally, we will compare and evaluate the performances taking into account different aspects of the optimization problem.

5.1 Domain description

The domain for a Reinforcement Learning algorithm is strictly correlated to the selection of the environment in which the algorithms are run. Indeed, within it, the whole Markov Decision Process' structure is defined. First of all the state, the action and the cost spaces characteristic of the environment, in addition to the length of the horizon, in case we are in a finite setting. Then the transition kernel P and the Cost function C are both determined. The library used to implement all this information is the gym package in python, that contains also a set of commonly used prefabbed environments. Now let's focus on the peculiarities of the one we have chosen to adopt.

5.1.1 Grid-World Map

The environment we selected is the so-called Grid-World. It is one of the most simple and easily readable environments available in the gym library and indeed is used a lot in

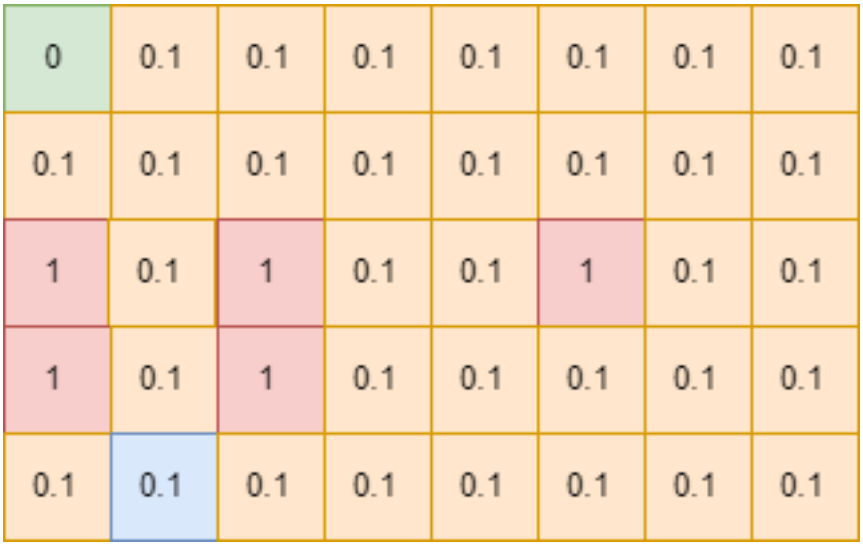

Figure 2: GRID-WORLD MAP

the literature experimental implementations (the other principal environment adopted by the researchers is the Portfolio one). The goal of the agent is pretty easy: he has to reach a certain terminal state, starting from a certain starting position that for simplicity we will keep constant. More other, there will be some obstacles around the map, in the forms of holes in our case, with a high cost associated. The objective, of course, will be to reach the goal state using the shortest path avoiding the obstacles. In order to diversify the policies of the agent in case of Risk-Neutrality or Risk-Sensitiveness, we should add some stochasticity to the environment, that we implemented inserting what we can call a random move: whenever the agent performs a certain movement from a given state, it has a small probability to move in a different direction to the chosen one. As follows, the Risk-Averse agent is supposed to go further from the obstacles than the Risk-Neutral one.

5.1.2 Markov Decision Process construction

The gym library makes you easily define the Markov Decision Process of the system. As a first thing, we declare the state space (5x8) and the action space (4 actions divided as UP=0, RIGHT=1, DOWN=2, LEFT=3). The time horizon is 30 steps long (enough to reach the goal even trying several non-optimal actions, allowing the agent to learn the optimal one through a proper exploration.

Then, we defined the transition kernel to designate the next state and the incurred cost associated with a certain probability. The probability is influenced by the so-called random moved in which the agent has $1 - ((Total - actions) - 1) * 0.0375 = 0.8875$ of going towards the selected direction and 0.0375 per each for all the others. It should be too high, otherwise, the agent would not be able to properly learn. The actual transition is defined by the step function, that performs it following the probability of the transition kernel and it returns the next state and the effective cost incurred. The latter is valued as 0.1 if it is a simple step, 1 if the agent follows in a hole (without anyway stopping his trajectory), 0 in case he reaches the goal. It is possible also to set the discount factor, changing how the agent focuses on the costs farther in the future concerning the immediate ones. For our experiments we set it to 1, defining a not discounted setting.

Another important thing to highlight is that whenever the decision-maker ends up in the goal position (the number 0 in the grill), it stays in that state until the end of the finite horizon time with probability 1. Finally, the reset function puts the agent in a constant starting position which is the number 33 in the grill.

5.1.3 Modified Markov Decision Process construction

In our experimental setting, we are applying the RL gradient algorithm which works with the state space augmentation defined by Bauerle, the Augmented Policy Gradient. For this reason, we had, of course, to modify some of the data about the environment construction, referring to it as Augmented-Grid-World. Specifically, the step function that incorporates the cost function, now returns to the agent a running cost of 0, apart from the terminal time step. On the other hand, now the state is not only represented by the number for the grill position, but also by a real value that describes the history of the trajectory. It is the initial cumulated cost variable s, updated for every step as $s' = \frac{s-c}{\gamma}$ where c is the actual cost incurred and gamma is the discount factor. Of course, also the reset function needs to give the information about the initial cumulated cost to the agent who is starting the trajectory. The kernel function and all the other settings will be the same.

5.2 Algorithms Implementation

In this section, we will go through the actual implementation of the various algorithms we tried within the Grid-World to learn the optimal policy with different Risk-Aversion degrees. We tried different approaches to initially understand how the environment works and how it is resolved by the different algorithms known in the literature. We applied especially at the beginning REINFORCE and Mean-Variance REINFORCE to see the difference between Risk-Neutral and Risk-Averse solutions and how the performances change. However, we skip in the results all these preliminaries experiments to focus on the implementations aimed at answering the question we posed at the beginning: is it possible to apply a standard Reinforcement

algorithm that minimizes some notion of risk thanks to a suitable MDP transformation? We answered this question in theory, now we want to compare the performances of our deduced Augmented Policy Gradient algorithm (with state-space augmentation) and the ad hoc algorithm derived by Tamar [6] to minimize the Conditional-Value-at-Risk. The steps through which we developed the practical experiments is the following:

1. Implemented the Dynamic Programming to optimize the CVaR as presented by Bauerle
2. Implemented the Reinforcement Learning Gradient algorithm for the Discrete solution of the optimization problem resolved by the DP
3. Implemented the Reinforcement Gradient algorithm which optimizes the policy and the value of s, referring to it as Augmented Policy Gradient Algorithm (APG)
4. Implemented the Gradient Conditional-Value-at-Risk algorithm (GCVaR) of Tamar [6] for the comparison with our solution

Now we will go through the different implementations, showing the results to make some analysis and considerations.

5.2.1 Augmented Dynamic Programming

The Dynamic Program was implemented to resolve the inner optimization problem of the CVaR formulation presented by Bauerle. The implementation is the well known Value Iteration algorithm to find the analytical of the optimal policy. The main difference is the augmentation of the state space of the environment that leads to a modification in the evaluation of the Value function. Indeed, now it depends not only on the state position but also on the value of the s

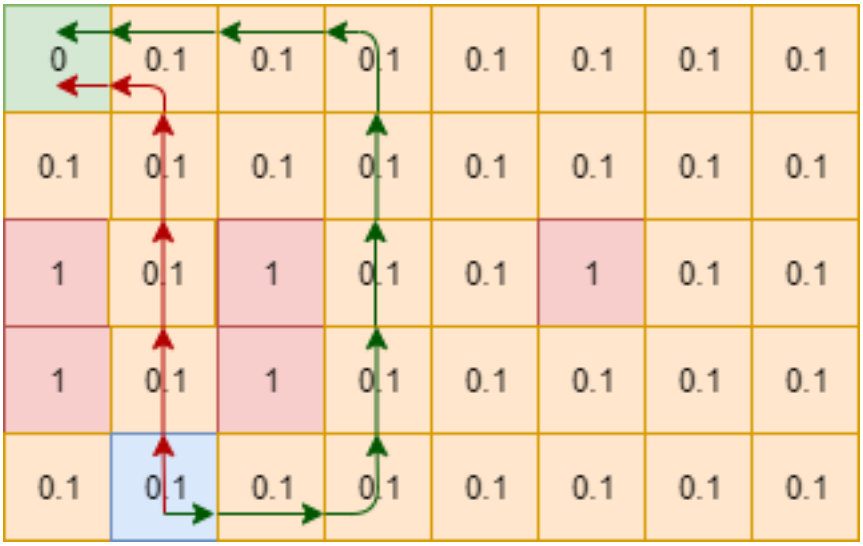

Figure 3: RISK NEUTRAL vs RISK AVERSE OPTIMAL POLICY

variable, which describes the cumulated cost of the trajectory. The issue is that s is real-valued, so it is practically impossible to compute the complete value function with the given domain. Hence, we computed the value function, following the new operators stated in Bauerle, with different suitable values of s. In particular, we took values in (0, T * holeCost)=(0, 30 * 1)=(0, 30) given the fact that the maximum possible cumulative cost in which the agent can incur is obtained only if the agent passes through the holes for every time step of the finite horizon. Of course, we should discretize the set and we did by a factor equal to the cost a single step, i.e. 0.1. We tried 300 different values of the initial value of s to compute this huge Value function which depends on the state position, on the cumulated cost s and on the time step in which we are, given the fact that only the terminal values should return s^{-}, 0 for the others.

After evaluating the value function, we used it to compute the expected values useful for the

outer optimization problem. Indeed, substituting the starting state-space position, we can obtain the 300 expected values based on the variation of the value s. To find the solution of the final CVaR problem, we took the minimum of the function $s + \frac{1}{1-\tau}\mathbb{E}[(C^T - s)^+]$, computed as we said varying the value of s withing the bounded discrete space (0, 30, 0.1).

We resolved this discrete minimization with different values of τ to see the differences among the various degrees of Risk-Aversion.

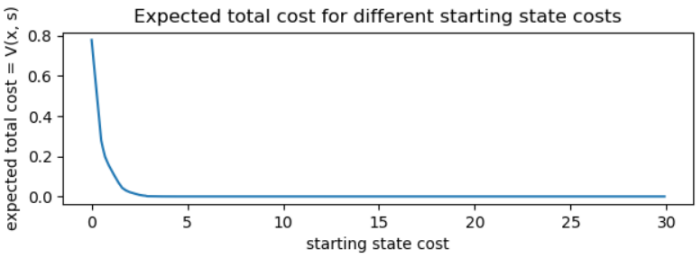

Figure 4: DP EXTERNAL OPTIMIZING FUNCTION

5.2.2 Augmented Discrete Reinforcement Learning solution

The discrete solution to the double optimization for CVaR minimization is the same approach we used in the Dynamic Programming case. The difference is that we used a Reinforcement Learning algorithm instead of the DP to find the optimal solution to the inner problem. The algorithm we adopted is the typical GPOMDP with sampling from a Boltzmann (softmax) stochastic policy. The augmentation of the state space is the one we have already presented

and induces, of course, a modification of the policy features used in the computation of the gradient. Indeed, usually, the selected feature for the update of θ is the one-hot encoding with dimension state space $\times$ action space. That is, the feature is a vector of dimension state space $\times$ action space (in our case 40 * 4 = 160) with a one in the position state * action space + action, and 0 in the other places. This makes the computation focus on only that particular state, giving an important gain in terms of gradient value to the action performed respect to the others.

To take into account the information given by the real-valued s variable, we cannot apply one-hot encoding because it increases too much the computational complexity. That is why we chose to represent it simply adding 1 cell to the feature vector's size in which we write the value of s itself. Of course, this is a weaker approximation of the policy feature space rather than using the one-hot encoding technique, but it allows to lose almost nothing in terms of time complexity of the algorithm. Furthermore, it revealed to be good enough for our case leading the algorithm to good performances.

To handle also the outer optimization problem, we had to resolve the policy optimization with different initial values of s. Hence, we used a discrete interval similar to the one proposed in the DP solution but reducing the size to leverage the running time. The interval we adopted is (0,3,0.1) which leads to run the algorithm 30 times with the specs we will describe.

Finally, we minimized the discrete function derived from these 30 approximations of the expected value, applying several τ values to the formula, that correspond to different levels of Risk-Aversion.

Dealing with the tuning of the algorithm we run it for 50 iterations and sampling of 750 trajectories with length equal to 30 (finite horizon dimension). The learning rate of the gradient update is 10.

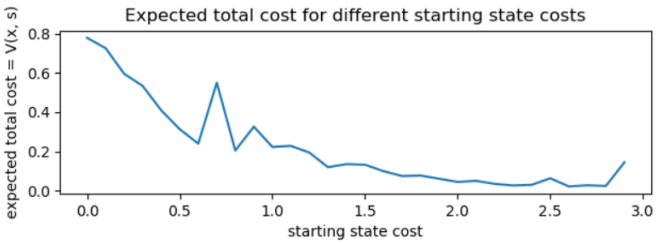

Figure 5: DISCRETE RL EXTERNAL OPTIMIZING FUNCTION

5.2.3 DP vs Discrete RL solution results

τ	Optimal s (DP)	Policy	Optimal s (Discrete RL)	Policy
0	0	RN	0	RN
0.1	0.5	RN	0.5	RN
0.3	0.5	RN	0.6	RN
0.5	0.5	RN	0.6	RN
0.7	0.7	RN	0.6	RN
0.9	1.6	RN	2	RA
0.98	2.5	RA	2.3	RA

TABLE II: OPTIMAL S VALUES FOUND BY DP AND DISCRETE RL.

Both the results are coherent and easy to read. Indeed, as we can see from the two plots of the computed expected values, the higher the value of s, the lower the value of the expectation $\mathbb{E}[(C^T - s)^+]$. We can notice of course some oscillations in the graph of the RL solution, because of the lack of precision of the approximation (with only 50 iterations for the gradient convergence). Indeed, here we were only interested in the general correctness of the optimization, not trying to perfectly approximate the result of the DP. In the images below, we can notice the

comparison between the results of the outer optimization problem, with different values of τ

As we can see, changing the risk aversion through τ, we change also the optimal value of s found as the outer solution. Recall that with $\tau \to 0$ the agent would act in a Risk Neutral manner, while with $\tau \to 1$ we obtain Risk-Averse policies of the decision-maker. Indeed, the bigger the τ, the bigger the optimal value of s. This can be interpreted as making the minimization problem concentrate only on the higher cumulative costs, the one that exceeds the value of the found optimal s that as we said we are increasing with bigger values of τ.

More other, in this case, the two results are comparable with some differences induced by the approximation error that we noticed in the Expected Value approximation of the RL solution. Anyway, the results are similar and on both sides, we see the diverse Risk-Aversion degrees.

As we can see from table Table II, the Risk-AVersion is achieved by the RL approach with $\tau = 0.9$ while with DP we need to have at least $\tau = 0.98$ (we plotted the corresponding function to be minimized in figue Figure 7

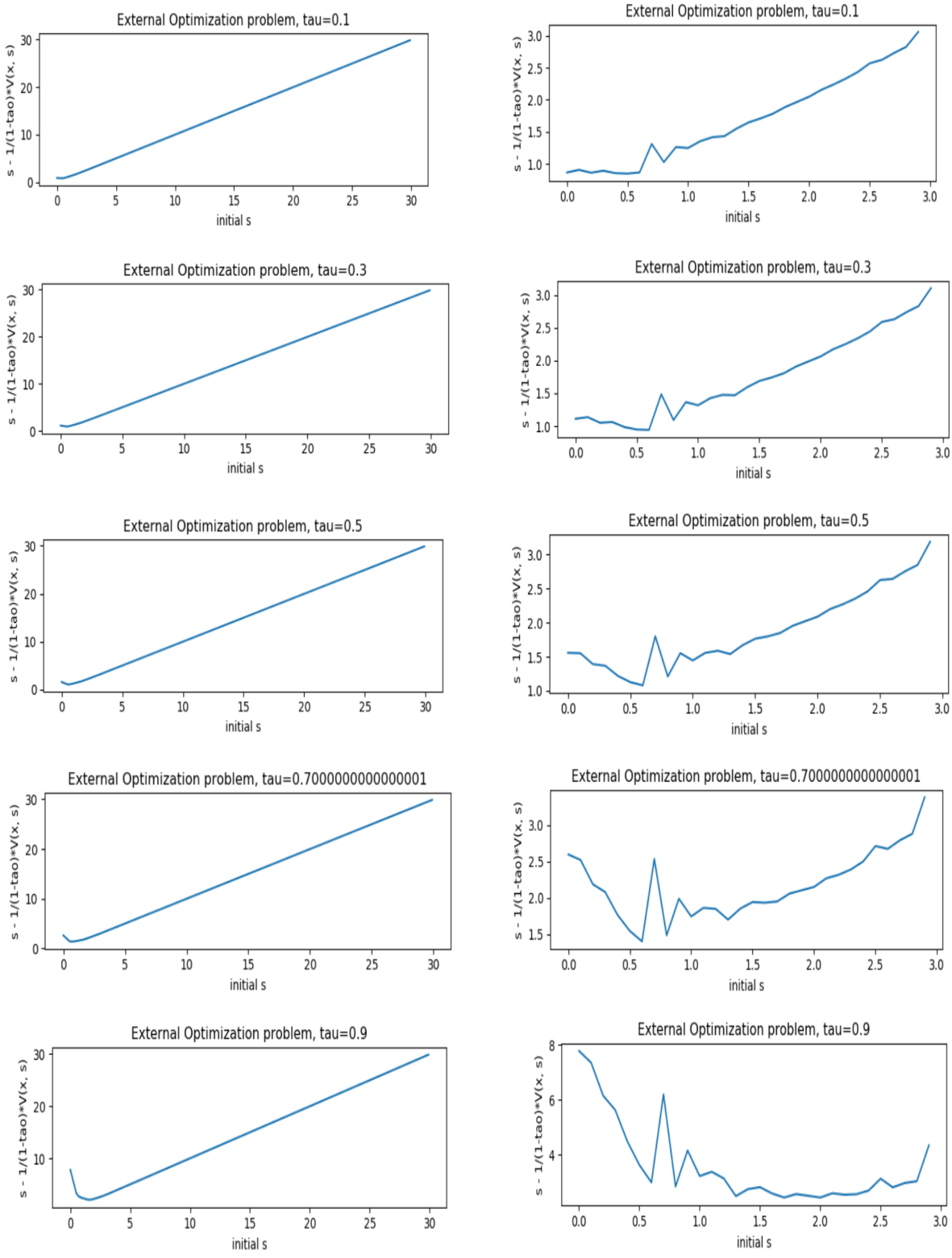

Figure 6: COMPARISON BETWEEN DP AND DISCRETE RL

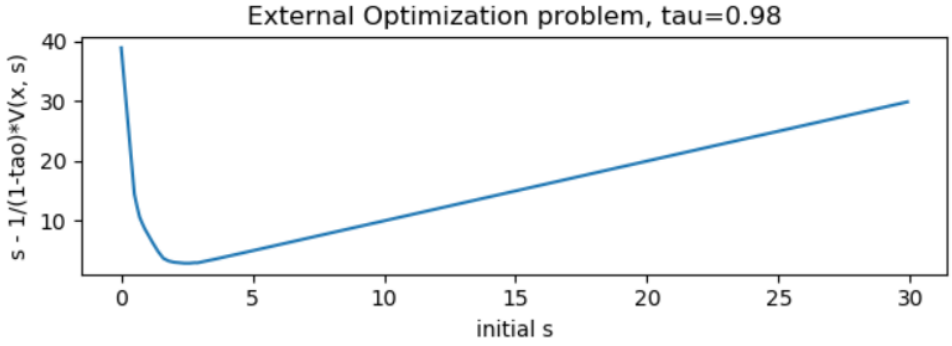

Figure 7: $\tau = 0.98$ LEADING TO RISK-AVERSE OPTIMAL POLICY

5.2.4 Augmented Policy Gradient algorithm

We have already presented the skeleton of the new algorithm we are trying to apply through the pseudocode in the table[1]. We used, as in the previous experiment, the same GPOMDP with Boltzmann stochastic policy for the sampling We implemented the version of it with different speed of updates for the two parameters to be optimized (s and θ). For every iteration the APG does one step for the optimization of θ, then samples N trajectories with the updated θ and then finds the optimal value of s for that specific policy π_θ, computing it through the closed-form expression. Hence, it is like doing a step of the θ optimization and then find the value of s at convergence.

Dealing with the parameters for the runs, we chose to take 1000 iterations, sampling of 750 trajectories with length equal to 30. The initial parameter of s is chosen randomly in a suitable set (looking at the DP solution, the optimal "s" takes values in (0,3)).

As always, we have run the algorithm with the usual set of τ in (0,1) to analyze the different Risk Sensitive behaviors and we plot the performances of the Mean Retain and the Conditional-Value-at-Risk per iterations.

5.2.5 GPOMDP with State-Space Augmentation

In this subsection, we will describe with more details the actual effect of the State Space augmentation on the common GPOMDP. In the Background chapter, we have already described the typical GPOMDP Reinforcement Learning algorithm, used to find the optimal policy through a gradient-based method. The core part of the algorithm is the computation of the likelihood score of the stochastic policy, which in our case is the Boltzmann one. Of course, the augmentation of the state does not change any part of the algorithm but simply requires an expansion of the feature vector space of the policy. Indeed, usually, the policy feature consists of a one-hot encoding for every state-action pair. In particular, we will have a vector with one corresponding to the state-action pair we are focusing on, 0 otherwise. This is a really precise way to represent the features because de facto, gives weight to every single state-action of the MDP. However, applying the one-hot encoding also for every possible augmented state would be impossible, given the fact that the variable s is a Real value. Thus, in our implementation we have chosen to express the feature space as the common one-hot encoding for every position-action pair, plus the exact value of the cumulated cost variable s, giving one single weight to the entire space of the augmenting variable

5.2.6 Gradient Conditional-Value-at-Risk algorithm

The last algorithm we implemented was the one proposed by Tamar [6], based on the derivation of the specific gradient of the considered risk measure. The algorithms are pretty similar in his structure to APG apart from the user environment (the standard Grid-World instead of the Augmented one) and of course the consequent dimension of the policy feature space. The RL algorithm for the computation of the gradient and the sampling technique is the same (GPOMDP with Boltzmann policy). The computation of the quantile is the same that we used for the computation of the optimal s value through the closed-form ($c_{\lceil \tau N \rceil}$. The difference is in the fact that our optimal$s$ value influences the policy update but does not changes the actual computation of the gradient. On the other hand, GCVaR [6] has to modify the computation of the expected cumulative returns with the clipped expectation $\mathbb{E}[(C^T - s)^+]$ that the augmentation we adopted had already incorporated modifying the cost function (s^- only for the terminal state, 0 otherwise). More other, Tamar uses the quantile substituting it to the typical baseline used in the gradient computation to reduce the variance, while our algorithm still uses it as if we were in the standard Risk Neutral optimization problem.

Consider also that we did not implement the Importance Sampling, required to reduce the Variance in the Conditional-Value-at-Risk gradient approximation, to see how this could influence the algorithm performances and also to underline the fact that our solution does not need it.

As before, we run the algorithm for different values of τ in (0,1) plotting the Mean Return and the CVaR at the variation of the iterations.

Talking about the specific specs of the runs, we chose the same parameters of Augmented Policy

Gradient to compare them properly in terms of performances. Ergo, we set 1000 iterations with a sampling of 750 trajectories with length 30.

5.2.7 APG vs GCVaR results

Looking at the plots we can easily see the differences in terms of performances considering the Mean Return and the Conditional-Value-at-Risk per iteration. As we can see, with lower τ the Mean Return and the CVaR have close values (indeed we are considering almost all the cumulative cost for the expectation $\mathbb{E}[C|C \geq VaR_\tau]$ which turns to be close to the Neutral case $\mathbb{E}[C]$). Both the algorithms learn easily a Risk Neutral policy with $\tau = 0.1$.

However, increasing the value of Risk-Aversion, our algorithm still performs well while the one implemented by Tamar [6] does not find an optimal policy. As we can see from the graphs, the APG Mean Return with lower τ is always around 0.8, which can be interpreted as the approximation of the cost of the Risk Neutral path (ideally 0.5). While with big values of τ, starting from 0.9 for APG, the Mean Return becomes a bit bigger than 1 showing that the found optimal policy is the Risk-Averse one (with the exact cost of 0.7). The differences between the real and the approximated value of the policies can be comprehended considering the stochasticity of the environment with the random move and also the stochasticity of the Boltzmann policy. Indeed, the algorithm after 1000 iterations can still perform a lot of exploration. These two factors induce of course a small error in the computation. Anyway, to confirm our hypothesis, we printed also a trajectory done performing the best action according to the learned policy, for every met state.

On the contrary, the Mean Return of the GCVaR [6] starts to be higher than 3 as soon as we

increase the values of τ, which means that he did not find the goal state (given the fact that the finite time orizon is 30 steps and so not finding the goal will result in a mean return at least of 0.1*30=3). This can due to the fact that the Tamar [6] approach without the state space augmentation loses the optimality, being a Markovian solution to a well known non-Markovian problem (the minimization of CVaR) that needs some information about the history. Thus we can conclude that in this specific setting with a Grid-World environment, the state space augmentation is needed and for this reason APG performs better than GCVaR [6].

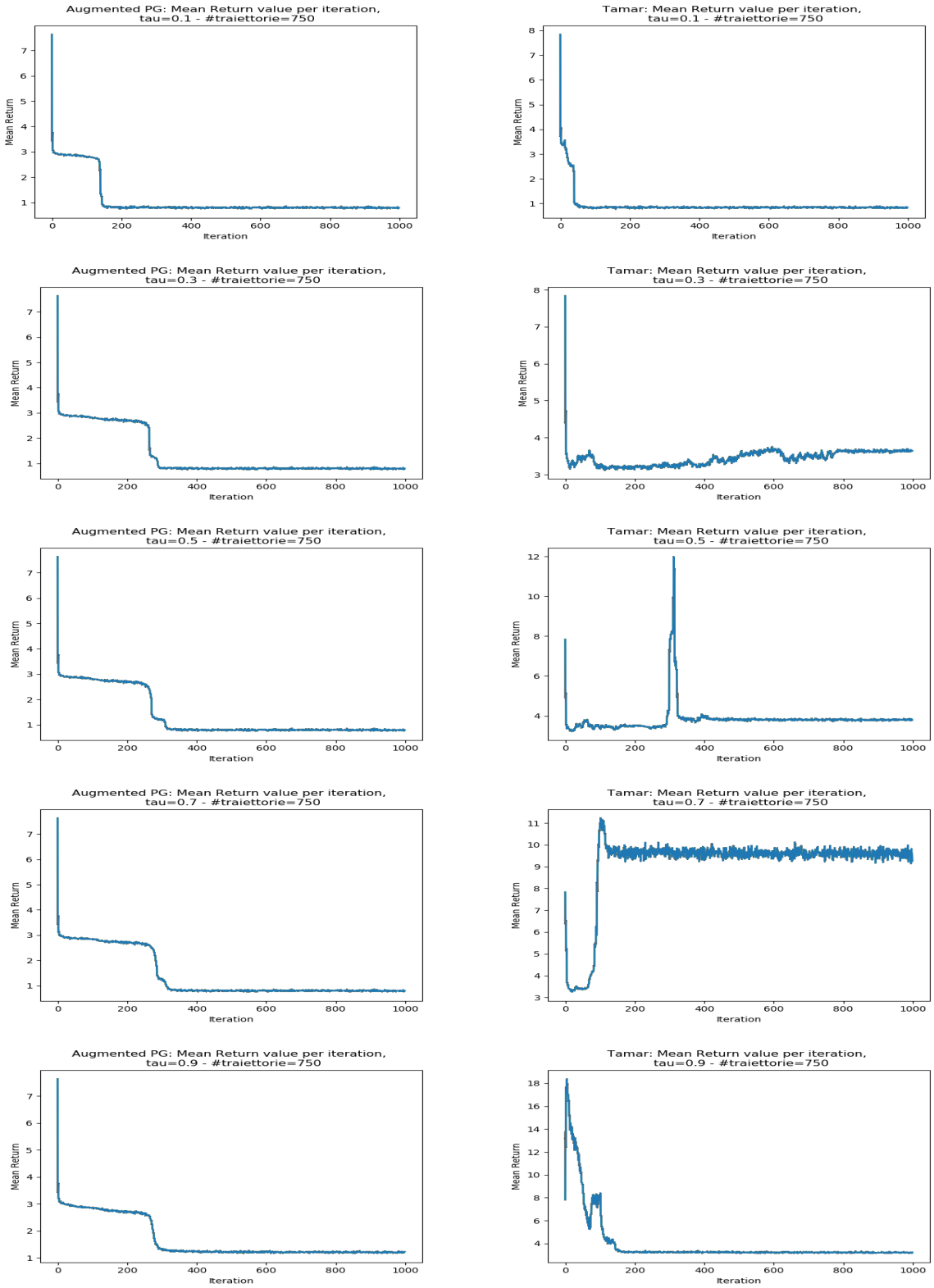

Figure 8: EXPECTED MEAN RETURN COMPARISON APG vs GCVaR

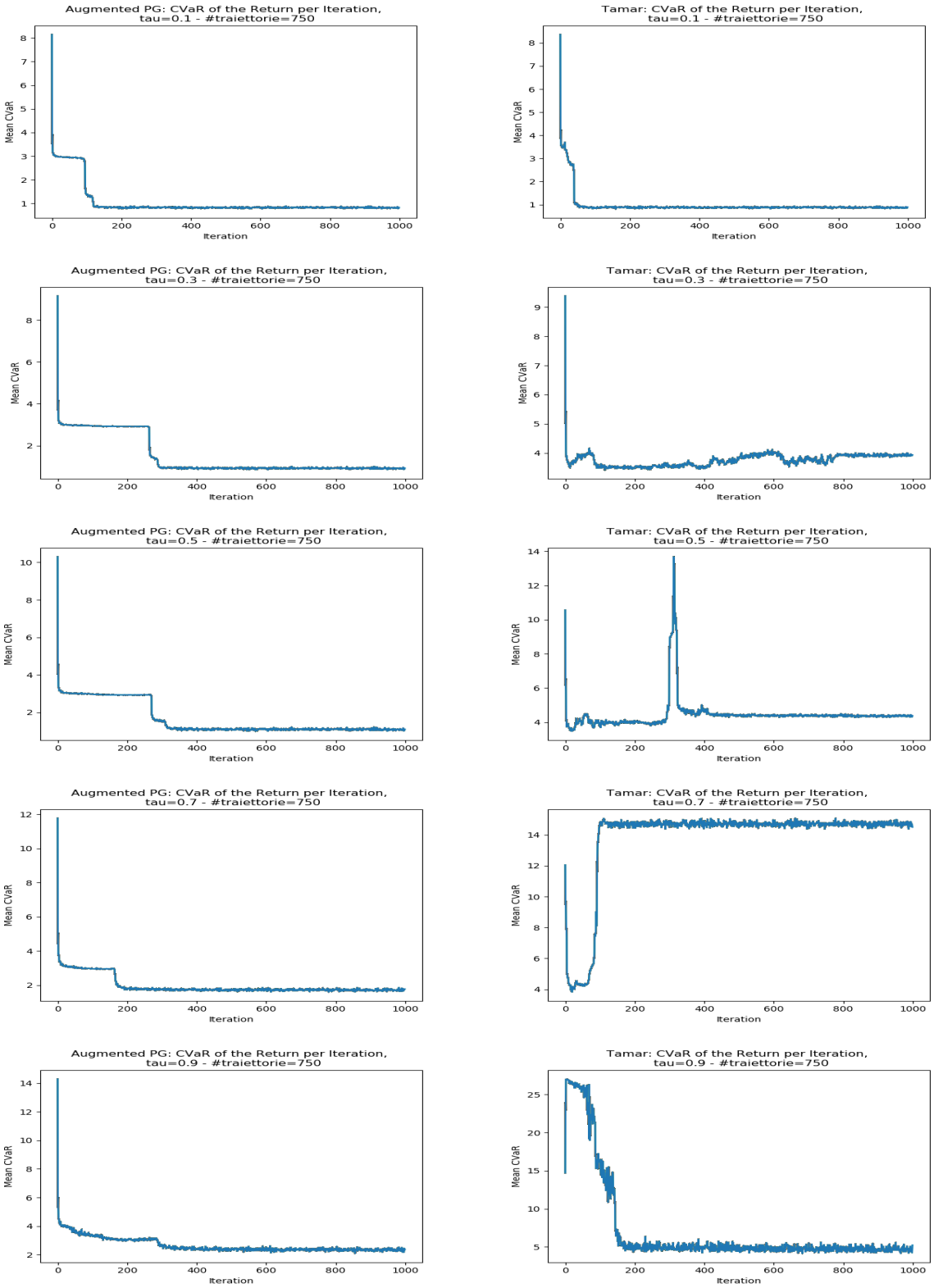

Figure 9: CVaR COMPARISO APG vs GCVaR

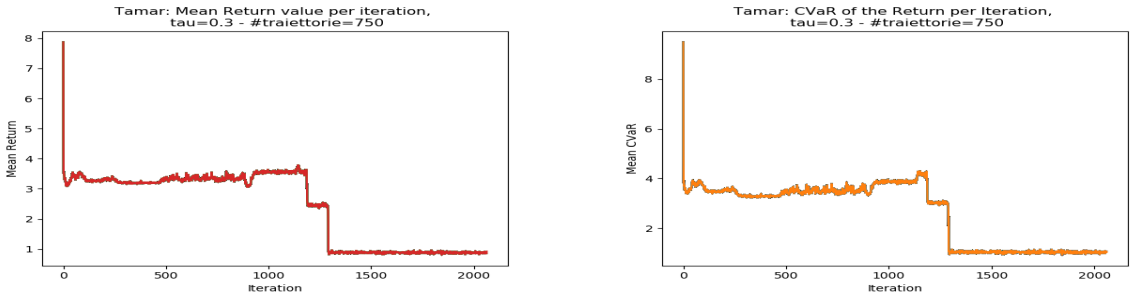

Figure 10: CONVERGENCE OF GCVaR WITH LONGER RUN AND $\tau = 0.3$

In this picture we notice another thing about the different performances of the two algorithms. We have seen that with $\tau = 0.3$ GCVar [6] already performs badly, not finding the optimal policy. However, increasing the number of iterations to 2000, we discovered that it was just taking longer in finding a good solution, still demonstrating the difficulties met in optimizing the return without the state space augmentation. This is an additional proof of APG being better than Tamar's [6] algorithm in this specific setting.

Below we can also analyze the optimization of the s parameter in Augmented Policy Gradient. This is important to visualize given the fact that not only it is interpreted as the Value-at-Risk of the cumulative cost, but it is only the augmenting variable of the state that directly influences the agent's action selection.

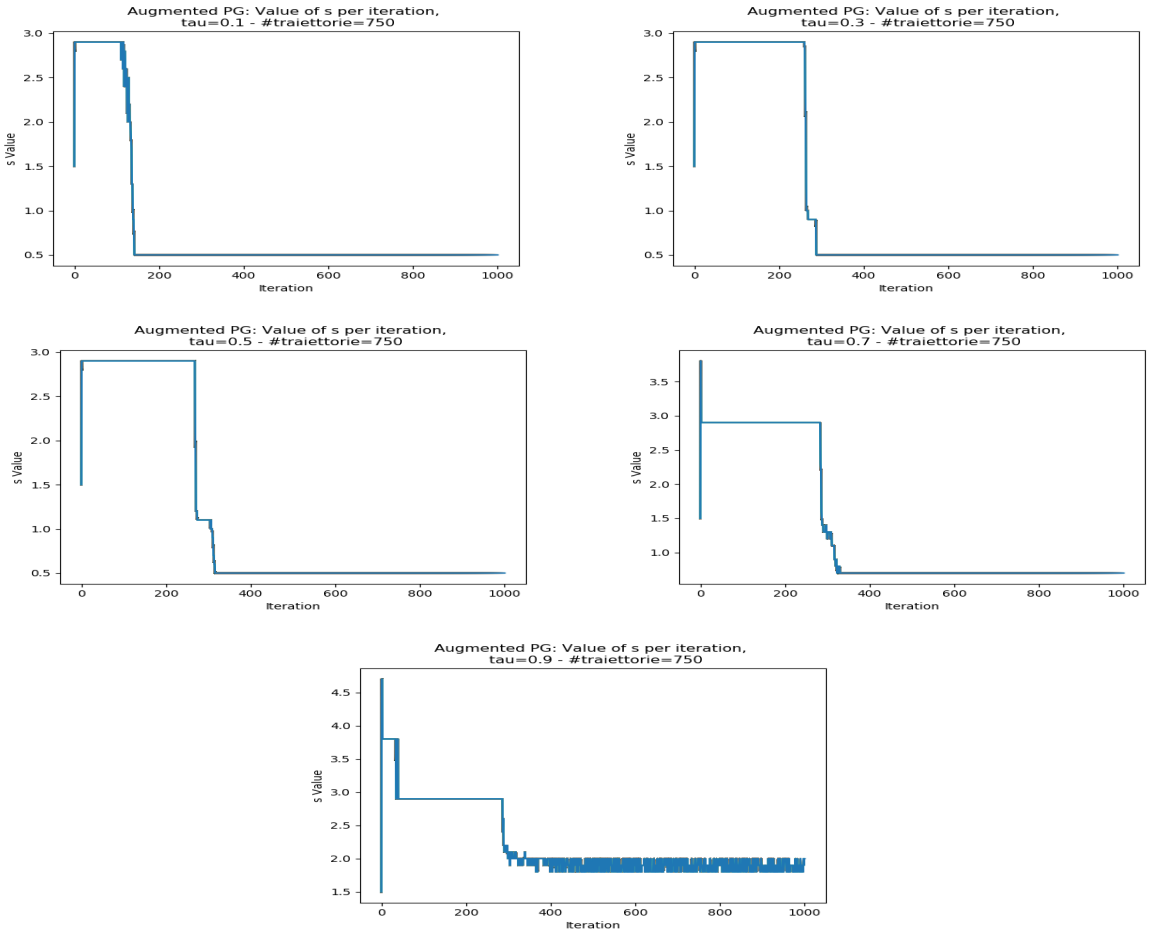

Figure 11: VARIATION OF S WITH DIFFERENT τ

CHAPTER 6

CONCLUSION

So after all these experimental results, it is time to make some conclusions about what we did. To do this, we need to restart from our initial interrogatives. The question we wondered at the beginning of our research work was: is it possible to take into account some notion of Risk with some suitable modification of the Markov Decision Process, without the need to derive a new ad hoc Reinforcement Learning algorithm? If yes, is it worth it? Is the modification easy enough without any big decrements in the time performances?

According to what we found out in theory and seen in practice, the answer seems affirmative. Indeed, the State Space Augmentation we took from Bauerle's idea is a simple and straightforward modification of the MDP. The only thing that is done, is adding the s variable to the state which makes the agent policy depend on some information about the history, that is its cumulated cost incurred so far.

For the rest, the transformation of the cost function $\tilde{C}$ (which returns 0 to the agent, until it reaches the terminal state of the finite horizon receiving s^-) has two fundamental consequences. On one hand, it allows the optimization of the Conditional-Value-at-Risk, which consists in minimizing the function $s + \frac{1}{1-\tau}\mathbb{E}_{\pi_\theta}[\tilde{C}^N]$. On the other hand, it permits the optimization of the inner problem through a standard RL algorithm. Actually, the latter will minimize the expected value $\mathbb{E}_{\pi_\theta}[\tilde{C}^N]$ as if it were the typical $\mathbb{E}_{\pi_\theta}[C^N]$. However, the real function that it is been optimized is $\mathbb{E}_{\pi_\theta}[(C^N - s)^+]$, the one GCVaR [6] it is adjusted for.

This explains the structural differences between the two algorithms: the GCVaR modifies the computation of the gradient(in particular, the part of the gradient related to the expectation of the cumulative cost that we have just shown) to optimize the Conditional-Value-at-Risk instead of the usual expected Return. While, Augmented Policy Gradient exploits the State Space Augmentation to do that, without modifying the RL algorithm.

This causes an important effect theoretically speaking. The Tamar algorithm [6], as it is proposed, requires the modification of the specific algorithm we are using to optimize the policy and in this sense, it could not be considered a general solution to the problem. Whereas, the proposed solution with augmentation can use whatever RL algorithm we desire (it could be GPOMD, REINFORCE, PPO, TRPO, etc.) just through the one shot of the modification of the MDP (i.e. of the gym environment specs).

Furthermore, it is proven (as Bauerle states) that this modification of the underlying Markov Decision Process can be applied to different risk measures, like exponential risk measures or other coherent risk measures rather than CVaR. Hence, also from this perspective, our approach is more general.

Dealing with the results that we have analyzed in the last chapter, we can also notice that Gradient Conditional-Value-at-Risk without State Space Augmentation does not find the optimal solution as we increase the value of τ. Indeed, as Tamar already recognized in his paper [6], the minimization of CVaR is a non-Markovian problem (that needs information about the history) and trying to resolve it with a Markovian solution is not always successful (as it was on the contrary, in its experimental results).

Furthermore, we run the Tamar solution [6] without the use of Importance Sampling, that according to him is necessary to reduce the approximation's variance of the CVaR gradient. Indeed, our algorithm does not need it (given the fact that we are not estimating the specific gradient of the considered risk measure) and we wanted to show the possible problems induced by the lack of this technique. Adding it would make Gradient Conditional-Value-at-Risk even more complex than our solution.

In conclusion, Augmented Policy Gradient outperformed GCVaR [6] in the Grid-World setting we have chosen and most importantly we have proven that is a way more generalized solution in terms of risk measures optimizations and Reinforcement Learning algorithms to adopt. Future works can be targeted on one side to the improvement of APG itself, considering, for instance, other and more precise estimations of the value of s or tuning the different speeds in the update of θ and s parameters. On a second side, it would be interesting to derive the same solution for alternative risk measures, like the exponential ones, exploiting the generalization of the State Space Augmentation presented by Bauerle. But after all, the most important thing to say is yes, a suitable modification of the MDP can lead to the optimization of the measured Risk without the derivation of specific Reinforcement algorithms.

CITED LITERATURE

1. Sutton, R. S. and Barto, A. G.: Reinforcement learning: An introduction. MIT press, 2018.

2. Garcıa, J. and Fernández, F.: A comprehensive survey on safe reinforcement learning. Journal of Machine Learning Research, 16(1):1437–1480, 2015.

3. NECCHI, P. G.: Policy gradient algorithms for the asset allocation problem. 2016.

4. Tamar, A. and Mannor, S.: Variance adjusted actor critic algorithms. arXiv preprint arXiv:1310.3697, 2013.

5. Tamar, A., Chow, Y., Ghavamzadeh, M., and Mannor, S.: Policy gradient for coherent risk measures. In Advances in Neural Information Processing Systems, pages 1468–1476, 2015.

6. Tamar, A., Glassner, Y., and Mannor, S.: Optimizing the cvar via sampling. In Twenty-Ninth AAAI Conference on Artificial Intelligence, 2015.

7. Osogami, T.: Robustness and risk-sensitivity in markov decision processes. In Advances in Neural Information Processing Systems, pages 233–241, 2012.

8. Bäuerle, N. and Ott, J.: Markov decision processes with average-value-at-risk criteria. Mathematical Methods of Operations Research, 74(3):361–379, 2011.

9. Bäuerle, N. and Rieder, U.: More risk-sensitive markov decision processes. Mathematics of Operations Research, 39(1):105–120, 2013.

10. Carpin, S., Chow, Y.-L., and Pavone, M.: Risk aversion in finite markov decision processes using total cost criteria and average value at risk. In 2016 IEEE International Conference on Robotics and Automation (ICRA), pages 335–342. IEEE, 2016.

11. Bäuerle, N. and Rieder, U.: Markov decision processes with applications to finance. Springer Science & Business Media, 2011.

VITA

NAME	your name
EDUCATION	
	Master of Science in Computer Science, University of Illinois at Chicago, December 2019, USA
	Master of Science in Computer Science and Engineering, Politecnico di Milano, April 2019, Italy
	Bachelor's Degree in Computer Science and Engineering, Politecnico di Milano, July 2017, Italy -
LANGUAGE SKILLS	
Italian	Native speaker
English	Full working proficiency
	2017 - TOEFL examination (88/120)
	2013 - First Certificate in English
SCHOLARSHIPS	
Spring 2018	Exemption from tuition fees for high academic performance
Spring 2017	Exemption from tuition fees for high academic performance
Spring 2016	Exemption from tuition fees for high academic performance
Spring 2015	Best freshmen of Politecnico di Milano for high academic performance
Spring 2015	Exemption from tuition fees for high academic performance
TECHNICAL SKILLS	
Basic level	Database
Average level	Mobile-Applications
Advanced level	Machine Learning and Deep Learning, Reinforcement Learning, Software Engineer

VITA (continued)

WORK EXPERIENCE AND PROJECTS

fall 2019	Research Assistant
	Design an Arduino device to help rehabilitation for stroke survivors with Unity games
Spring 2019	Research Assistant
	Proving database implementation through Coq
Fall 2017 - Spring 2018	IT Assistant in Apple School