Review paper: A Lyapunov-based Approach to Safe Reinforcement Learning

Tobias Kastenholz
RWTH Aachen University
tobias.kastenholz@rwth-aachen.de

Abstract—**In this report, a new approach to Safe Reinforcement Learning is reviewed. Safe Reinforcement Learning is particularly important in real-world scenarios, where an agent is given a number of constraints. This is, e.g., done to guarantee safety where the agent could otherwise harm itself or the environment. The presented approach was first introduced in [1] and is based on Lyapunov functions, which so far have been mainly used in control theory. The Lyapunov method is put into the general context of Reinforcement Learning and also compared to other approaches that incorporate safety into real-world scenarios. The results show that the proposed Lyapunov-based algorithms manage very well to balance constraint violations with good performance.**

I. Introduction

Reinforcement Learning (RL) is a method of Machine Learning where an agent learns through interaction with an environment. The agent perceives the current state from the environment. Based on the current state, the agent takes an action and in return receives a reward and the next state. This cyclic process is shown in Fig. 1.

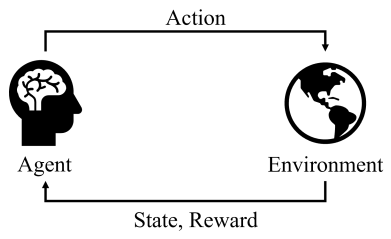

Fig. 1. Reinforcement Learning cycle

The reward can either be positive or negative. The agent's goal is to maximise the accumulated reward over time. To achieve this goal, the agent needs to find a trade-off between immediate and future rewards. The agent's policy $\pi(a|x)$ is a map that contains the probabilities of taking a particular action a when in a particular state x. Technically, RL problems can be formalised as a Markov decision process (MDP).

RL applications work well when there is a single main goal. In real-world problems, however, the agent often needs to consider additional constraints that can conflict with the main goal. One crucial real-world constraint is safety during training and deployment. Some actions could harm the agent's hardware or the environment and should therefore better be avoided. An example for this kind of safety violation is a robot that is hitting a wall. The constraint could be defined as the maximum number of hits that the robot can take or the maximum energy-level of a single hit. Another typical safety constraint is the battery consumption during a navigation process, which conflicts with the goal of reaching the destination as quickly as possible. The research area dealing with the conflict of goals and constraints is called Safe Reinforcement Learning (Safe RL).

Constrained Markov decision processes (CMDP) [2] are used to expand MDPs to restrict the available policies to safe ones and to solve problems with conflicting objectives. Several methods of modeling the constraints in a CMDP exist. In this review paper, the main focus is on using Lyapunov functions, which are commonly used in control theory for stability analysis. A set of safe policies is constructed using these Lyapunov functions, which work as an upper bound to the constraint violations. Furthermore, we show that the set of safe policies can be constructed in a way that ensures it also contains an optimal policy. Four different algorithms that use the Lyapunov approach are presented and evaluated using a motion planning example. In addition, this report aims to compare the presented work to other Safe RL approaches.

II. Problem Statement

An MDP is a tuple $(\mathcal{X}, \mathcal{A}, c, P, x_0)$, where $\mathcal{X}$ is the state space with initial state x_0, $\mathcal{A}$ is the action space and c is the immediate cost function. P is the transition probability distribution that holds the probabilities of ending in a specific state after taking an action. Moreover, $\mathcal{X}' \in \mathcal{X}$ is the set of all transient (i.e., non-terminal) states. A CMDP is a tuple $(\mathcal{X}, \mathcal{A}, c, d, P, x_0, d_0)$ with two added components compared to the unconstrained MDP: d is the immediate constraint cost and d_0 is the constraint threshold, i.e., an upper bound for the expected cumulative constraint cost.

In a CMDP, there are two different return functions. The standard cost or reward function is already known from unconstrained MDPs and defined as the expected cumulative cost

$$\mathcal{C}_\pi(x_0) := \mathbb{E}\left[\sum_{t=0}^{T^*-1} c(x_t, a_t)|x_0, \pi\right]. \tag{1}$$

Here, T^* refers to the stopping time of the MDP, i.e., the time until a terminal state $x_{term} \in \mathcal{X} \setminus \mathcal{X}'$ is reached.

Additionally, the safety constraint function is defined as

$$\mathcal{D}_\pi(x_0) := \mathbb{E}\left[\sum_{t=0}^{T^*-1} d(x_t, a_t)|x_0, \pi\right], \tag{2}$$

which is the expected cumulative constraint cost and could, e.g., be the number of obstacle hits during one episode. This leads to the overall problem to solve in a CMDP: Given an initial state x_0 and a constraint threshold d_0, find the optimal policy π^*, that results in the lowest cost $\mathcal{C}_\pi(x_0)$ while keeping $\mathcal{D}_\pi(x_0) \leq d_0$.

III. Background

In this section, we provide the necessary background on CMDPs and Lyapunov functions.

A. Constrained Markov Decision Processes

To apply the return functions $\mathcal{C}_\pi$ and $\mathcal{D}_\pi$ in RL algorithms, the terms state value and action value need to be defined. In general RL, starting from from state x, the value $V(x)$ of a state is defined as the expected total return [3]

$$V(x) = \mathbb{E}\left[R_{t+1} + V(X_{t+1}) | X_t = x\right], \tag{3}$$

where R is the return function. With $\mathbb{E}[R_{t+1}|X_t = x] = R_x$ and $X_{t+1} = x'$, (3) can be rewritten as

$$V(x) = R_x + \sum_{x' \in \mathcal{X}'} P^\pi(x'|x,a)V(x'). \tag{4}$$

Now, for any arbitrary return function $h(x,a)$, where $R_x = \sum_a \pi(a|x)h(x,a)$, the state-value function can be described as

$$V(x) = \sum_a \pi(a|x)\left[h(x,a) + \sum_{x' \in \mathcal{X}'} P(x'|x,a)V(x')\right]. \tag{5}$$

Finally, for later usage, (5) is rewritten as the expected Bellman operator

$$T_{\pi,h}[V](x) = \sum_a \pi(a|x)\left[h(x,a) + \sum_{x' \in \mathcal{X}'} P(x'|x,a)V(x')\right]. \tag{6}$$

Equation (6) can be seen as an iterative function that converges to a unique fixed point.

Similar to the state value function, the action value function $Q(x,a)$ is defined as the expected total return starting in state x and executing action a. With $V(x) = \sum_a \pi(a|x)Q(x,a)$, the action value function becomes

$$Q(x,a) = h(x,a) + \sum_{x' \in \mathcal{X}'} P(x'|x,a)V(x'). \tag{7}$$

B. Lyapunov functions

Lyapunov functions are scalar functions that have been mainly used in control theory for stability analysis. The equilibrium point at $x = 0$ of a system $x_{k+1} = f(x_k)$ is considered as globally asymptotically stable, if there exists a Lyapunov function $L(x)$ with the following three properties:

- $x = 0 \Rightarrow L(x) = 0$
- $x \neq 0 \Rightarrow L(x) > 0$
- $x \neq 0 \Rightarrow L(x_{k+1}) - L(x_k) < 0$

A simple example for the usage of Lyapunov functions is the damped pendulum with state vector $x = (\theta, \dot{\theta})^T$ shown in Fig. 2.

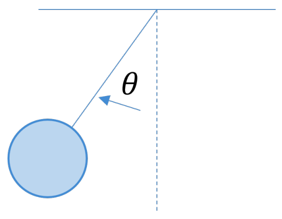

Fig. 2. Example of Lyapunov functions: damped pendulum

A possible Lyapunov function for this system is the total mechanic energy. This function satisfies the three Lyapunov conditions and, clearly, the equilibrium point is stable if the mechanic energy dissipates over time. This shows that the Lyapunov function often relates to the energy of a system.

IV. Related Work

In this section, we present previous work on Safe RL and the first approaches of using Lyapunov functions in general RL tasks.

A. Safe RL and CMDPs

Safe RL has been studied before in different works. A straightforward approach is to define safety over particular state action pairs. In [4], reward functions are defined imprecisely to improve real-world adaptability of MDP solutions. The constraints are handled by customising the immediate cost function. Another approach is presented in [5], where an aerial robot should track a ground vehicle by predicting its dynamics. Here, a safe region of the state space is defined to keep the ground vehicle in the field of view.

In contrast to these approaches, other works showed that defining safety over the whole trajectory, i.e., a whole sequence of states and actions, achieves better results. In [6], safe exploration is approached by formulating safety through ergodicity. Ergodicity in an MDP means that any state is reachable from any other state. This would mean that any mistake (i.e. reaching an unsafe state) is revocable, which is often not the case in real-world scenarios. The idea in [6] is to restrict the policy to a subset of policies that enable ergodicty with a probability δ. However, this only ensures safety with this probabiltiy and never completely. In [7], an algorithm for joint chance-constrained problems is presented. This approach deals with the problem that some constraints are not additive, i.e., the safety constraint function cannot be modeled as in (2). Instead, these constraints are reformulated using indicator random variables, which are variables that are equal to one, if the current state is infeasible, and zero otherwise. This approach also guarantees that the probability of constraint violation is within a specified risk bound.

CMDPs also allow to model safety constraints over the whole sequence of states and actions. As safety is modeled here using the expected cost (see (2)), which is a probabilistic value, safety is also only expected and not guaranteed. There exist some approaches to solving CMDPs. An often used approach is the Lagrangian method, where a penalty on constraint violation is introduced using a Lagrange multiplier λ. The Lagrange multiplier can be fixed or learnable. Then, the starting point of a solution is the following problem:

$$\min_{\pi \in \Delta} \max_{\lambda \geq 0} \mathcal{C}_\pi(x_0) + \lambda(\mathcal{D}_\pi(x_0) - d_0). \tag{8}$$

Unfortunately, this method may lead to unsafe policies during training, which is undesirable in real-world applications. Moreover, it may lead to numerical stability issues.

Another approach to solving CMDPs is the use of surrogate algorithms. These approaches transform the constrained problem into one that is easier to solve. Two examples are algorithms that use a stepwise constraint function [8] or a super-martingale constraint function [9]. However, these surrogate algorithms suffer from suboptimal results due to their conservativeness. The transformed constraints are often much stricter compared to the original constraints.

A recent solution of CMDPs is Constrained Policy Optimisation (CPO) [10]. This algorithm is based on the idea that the difference of expected returns of two policies is bounded. This way, the policy can be guaranteed to be safe throughout the training. CPO, however, is hard to apply to other RL algorithms and therefore not very versatile.

B. Lyapunov functions in RL

Besides their application in control theory, Lyapunov functions are also used to represent more abstract quantities. More specifically, they are already used in RL applications. In [11], e.g., Lyapunov functions are used to design controllers for a given task. By using Lyapunov functions, it is ensured that all of these controllers are safe. The agent then learns to switch between the controllers in order to achieve optimal performance. Here, the Lyapunov functions are not applied to the learning process directly, but instead used in their original purpose of demonstrating closed-loop stability.

Another Lyapunov approach is presented in [12]. Here, an inital safe policy and an initial safe attraction region, i.e., a subset of the state space that is never left, are defined before the start. Then, the policy is safely improved by collecting data and expanding the attraction region using Lyapunov functions. The authors show how they apply this approach to the inverted pendulum task and manage to improve the policy without the pendulum ever falling down. However, similar to [11], the dynamics of the model need to be known beforehand in order to construct the Lyapunov functions. In [1], Lyapunov functions are used to explicitly model the constraints in a CMDP for the first time. A major advantage compared to previous Lyapunov applications is that the Lyapunov functions in this approach are constructed by an algorithm and not by hand. In the following section, we will look at the contributions of [1] in detail.

V. Solving CMDPs with Lyapunov functions

To automatically construct Lyapunov functions, first, a set of Lyapunov functions is defined as (9). Here, x_0 is the initial state and d_0 is the constraint threshold. π_B is a baseline policy that can be assumed to be accessible. A possible baseline policy, e.g., is a policy that minimises the safety constraint function $\mathcal{D}_\pi$. It is easy to see that all three Lyapunov conditions (see III-B) reappear in the set definition, where the Bellman operator is used to ensure the negative gradient. This can be seen by comparing this term to the third Lyapunov condition, i.e. $L(x_{k+1}) < L(x_k), \forall x \neq 0$. Additionally, the term $L(x_0) \leq d_0$ ensures safety for the initial state and therefore the whole trajectory. For any function $L \in \mathcal{L}_{\pi_B}$, the set of new policies that satisfy the Lyapunov conditions is defined as

$$\mathcal{F}_L(x) = \{\pi(\cdot|x) \in \Delta : T_{\pi,d}[L](x) \leq L(x), \forall x \in \mathcal{X}'\}. \tag{10}$$

All of these policies have the property

$$\mathcal{D}_\pi(x) = \lim_{k\to\infty} T^k_{\pi,d}[L](x) \leq L(x), \forall x \in \mathcal{X}'. \tag{11}$$

Equation (11) states that due to the contraction mapping theorem, the safety constraint function $\mathcal{D}_\pi$ is the fixed point of the Bellman operator $T^k_{\pi,d}[L](x)$. Therefore, the cumulative constraint cost is smaller than or equal to the Lyapunov function, from which it follows that the policy is safe.

Now that the safety of all policies in $\mathcal{F}_L(x)$ is proven, the second step is to also guarantee $\mathcal{F}_L(x)$ is containing an optimal policy. To do this, an auxiliary constraint cost ϵ is introduced, which can be used to transform the safety constraint function $\mathcal{D}_\pi(x)$ into a Lyapunov function $L_\epsilon(x) \in \mathcal{L}_{\pi_B}(x_0, d_0)$:

$$L_\epsilon(x) = \mathbb{E}\left[\sum_{t=0}^{T^*-1} d(x_t) + \epsilon(x_t)|\pi_B, x\right]. \tag{12}$$

For an optimal policy π^*, $L_\epsilon(x)$ is equal to $\mathcal{D}_{\pi^*}(x)$. It follows, therefore, that ϵ serves as a buffer for available constraint violations. Because ϵ is hard to construct without knowing an optimal policy beforehand, its upper bound ϵ^* is used to propose a Lyapunov function candidate $L_{\epsilon^*}(x)$, which again serves as an upper bound to $\mathcal{D}_{\pi^*}(x)$. The following theorem is one of the main results in [1] and shows that policies constructed with $L_{\epsilon^*}(x)$ are indeed safe and contain an optimal policy.

Theorem 1. *If the baseline policy π_B is sufficiently close to an optimal policy π^*, then $L_{\epsilon^*}(x)$ is in the set of feasible Lyapunov functions, i.e. $L_{\epsilon^*}(x) \in \mathcal{L}_{\pi_B}(x_0, d_0)$, and $\mathcal{F}_{L_{\epsilon^*}}(x)$ contains an optimal policy.*

Different state-of-the-art algorithms can easily be transformed into their safe counterparts using the presented approach. In the following, four example algorithms are described in detail.

$$\mathcal{L}_{\pi_B}(x_0, d_0) = \{L : \mathcal{X} \to \mathbb{R}_{\geq 0} : T_{\pi_B,d}[L](x) \leq L(x), \forall x \in \mathcal{X}'; L(x) = 0, \forall x \in \mathcal{X} \setminus \mathcal{X}'; L(x_0) \leq d_0\} \tag{9}$$

$$\pi'(\cdot|x) \in \arg\min_{\pi\in\Delta}\{\pi(\cdot|x)^T Q(x,\cdot) : \pi(\cdot|x)^T Q_L(x,\cdot) \le \tilde{\epsilon}' + \pi_B(\cdot|x)^T Q_L(x,\cdot)\} \tag{13}$$

A. *Safe Policy Iteration*

Safe Policy Iteration (SPI) is based on its unsafe counterpart Policy Iteration, which is a model-based algorithm that evaluates and improves the policy in every step. This is repeated until convergence.

The upper bound ϵ^* of the auxiliary constraint cost can be estimated by

$$\tilde{\epsilon}(x) = (d_0 - \mathcal{D}_{\pi_B}(x_0))/\bar{T}. \tag{14}$$

Here, $\bar{T}$ refers to the upper bound of the estimated stopping time of the CMDP, i.e., the estimated time left until a terminal state is reached. Then, starting with a feasible policy π_0, two steps are performed in each training episode. Firstly, the current policy π_k is used to evaluate the Lyapunov function $L_{\tilde{\epsilon}}$. Secondly, the next policy is computed by finding the policy π_{k+1} that minimises (6), where V is the cost function $\mathcal{C}_{\pi_k}$ and $\pi_{k+1} \in \mathcal{F}_{L_{\tilde{\epsilon}}}$. This last condition ensures that only policies induced by the Lyapunov function, i.e., safe policies, are taken into account for the policy update.

B. *Safe Value Iteration*

Value Iteration is another model-based algorithm, where instead of updating the policy in every episode, the optimal policy is only once extracted directly from the converged optimal value function. Its safe counterpart Safe Value Iteration (SVI), however, still needs access to the current policy to calculate the Lyapunov function in every step. Therefore, SVI starts with an initial random Q-function (see (7)) and an initial $\epsilon = 0$. Then, in each training episode, the first step is to compute the Q-function update using (7) with $V(x') = \min_{\pi\in\mathcal{F}_{L_{\tilde{\epsilon}}}} \pi(\cdot|x')^T Q_k(x',\cdot)$. In the second step, $\tilde{\epsilon}$ is also calculated using (14) and used to construct the Lyapunov function update.

We can see that the key difference between SPI and SVI is that in SPI, first the Lyapunov function is constructed and then used to update the policy w.r.t. the cost function. In SVI, on the other hand, first the Q-function is updated to generate a new policy, which is then used to update the Lyapunov function.

C. *Safe Q-Learning*

Q-Learning is a model-free algorithm; therefore, unlike in SPI and SVI, the transition probabilites are unknown. In Q-Learning, a Q-table is used to store the values of every action in a particular state. In every step, the Q-value for the taken action is updated based on the Bellman equation. A huge disadvantage of Q-Learning is that the Q-table is not continuous and gets huge very fast. Therefore, it is not suited for large state spaces. This problem is solved by introducing Deep Q-Learning (DQN) [13], in which the Q-table is replaced by a neural network that serves as a function approximator. This allows a continuous state space, and consequently, the solution of much more complex problems. In every step, the weights θ of the Q-network are updated by minimising a loss function using the target value

$$y = h(x) + \gamma \min_{a'}(Q(x', a'; \theta)), \tag{15}$$

where γ is a discount factor which sets the importance of immediate and future costs. If the next state is a terminal state, the target value is set to the cost value $h(x)$ of the current state.

Safe Q-Learning (SDQN) extends DQN to guarantee safety. As will be shown, only few changes are necessary to transform DQN into SDQN. Equation (7), where the state value function now is a Lyapunov function, is approximated using two neural networks: $Q_D(x, a, \theta_D)$, which estimates the safety constraint function and $Q_T(x, a, \theta_T)$ which estimates the stopping time of the system. The action value function then is

$$Q_L(x, a, \theta_D, \theta_T) = Q_D(x, a, \theta_D) + \tilde{\epsilon} \cdot Q_T(x, a, \theta_T). \tag{16}$$

Using these function approximators, (14) also changes to

$$\tilde{\epsilon} = \frac{d_0 - \pi_k(\cdot|x_0)^T Q_D(x_0, \cdot, \theta_D)}{\pi_k(\cdot|x_0)^T Q_T(x_0, \cdot, \theta_T)}. \tag{17}$$

The current policy π_k is needed in this expression to transform the function approximators, which are action value functions, into state value functions (see III-A). Then, using (16) and (17), in each iteration the policy is updated using (13). Here, the next policy is the policy $\pi'(\cdot|x)$ that minimises the state cost $\pi(\cdot|x)^T Q(x,\cdot)$ under the condition that the constraint cost under the policy $\pi(\cdot|x)^T Q_L(x,\cdot)$ is smaller than or equal to $\tilde{\epsilon}$ plus the constraint cost under the baseline policy $\pi_B(\cdot|x)^T Q_L(x,\cdot)$. In practice, this policy update can be done using the concept of policy distillation [14], where the policy is also approximated using a neural network. Fig. 3 shows a summary of the SDQN algorithm.

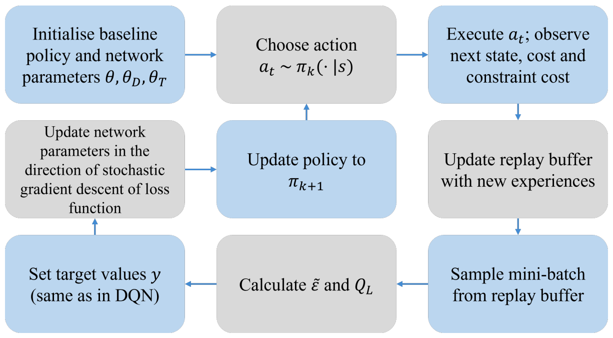
Initialise baseline policy and network parameters $\theta, \theta_D, \theta_T$
Choose action $a_t \sim \pi_k(\cdot|s)$
Execute a_t; observe next state, cost and constraint cost
Update network parameters in the direction of stochastic gradient descent of loss function
Update policy to π_{k+1}
Update replay buffer with new experiences
Set target values y (same as in DQN)
Calculate $\tilde{\epsilon}$ and Q_L
Sample mini-batch from replay buffer

Fig. 3. Safe Q-Learning algorithm

Note that three separate neural networks are initialised, compared to only one in DQN. In contrast to DQN, where the next action is chosen using the Q-network, in SDQN the next action is chosen by the current policy. This is necessary because the policy considers both the cost network and the constraint network. The next three steps are exactly the same as in DQN. Then, Q_L and $\tilde{\epsilon}$ are calculated using (16) and (17) and used to calculate the greedy action probabilities by solving (13). The target values y are set identical to DQN and then used to update the network parameters. Finally, the policy is updated to the next iteration.

D. Safe Policy Improvement

The concept of Safe Policy Improvement (SDPI) is similar to SDQN, as it also uses the three neural networks Q, Q_D and Q_T as function approximators and its algorithm structure is similar to the one presented in Fig. 3. The difference to SDQN is that the target values are set using the current policy instead of greedy actions. SDPI also uses (16) and (17) to calculate the Lyapunov function. Then, a policy improvement step is performed again using (13) and policy distillation.

VI. Experiments and Results

In this section, we present the motion planning environment that is used to evaluate the developed algorithms. Furthermore, we present the experimental results for all algorithms.

A. Environment description

The presented algorithms are validated using the grid-world motion planning environment that is shown in Fig. 4.

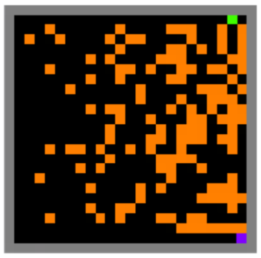

Fig. 4. 2D Grid-world environment

In this environment, the agent (purple) has the objective to move to a known destination (green). At the same time, it should avoid the obstacles (orange). The agent starts in a safe spot, i.e., a spot without an obstacle. It can move to any of the four neighbouring states (up, down, left, right). A probability parameter δ, however, defines the chance of moving to a random state instead of the one selected by the policy. This error probability is known to the algorithms SPI and SVI, but not to SDQN and SDPI, since these are model-free algorithms.

The cost for each move is 1 and the reward (i.e. negative cost) for reaching the destination is 1000. Additionally, the safety constraint function is defined with a constraint cost of 1 for moving to a field with an obstacle. Thus, the overall goal of the agent is the goal that is already known from CMDPs: Reach the destination in the shortest possible number of moves while passing through obstacles at most d_0 times or less. A small parameter d_0 makes the problem harder to solve while a very large parameter $d_0 \rightarrow \infty$ results in a problem corresponding to the unconstrained one. The density of obstacles is adjusted using the parameter ρ, that ranges from 0 to 0.5. The higher ρ, the harder to solve the problem gets.

B. Dynamic Programming algorithms

Fig. 5 shows the results of SPI and SVI compared to three baseline algorithms: step-wise surrogate, super-martingale surrogate and Lagrangian (see IV). Additionally, for reference, the results of a feasible and conservative baseline as well as the results of a policy iteration algorithm without constraints are shown. The results are shown for different obstacle densities ρ that are plotted on the x-axes. The y-axes represent the return and the constraint violations. Each algorithm starts with a feasible baseline policy that minimises the constraint cost.

We can see that the surrogate algorithms do not improve the return compared to the baseline policy. SPI and SVI, however, return feasible solutions with good performance. Only for a high obstacle density ρ the performance of SVI degrades due to numerical instabilities. The Lagrangian method returns promising performance but violates constraints with higher obstacle density.

C. Reinforcement Learning algorithms

The results of SDQN and SDPI compared to their unsafe counterparts DQN and DPI as well as the Lagrangian approach with a learnable multiplier λ are shown in Fig. 6. The results

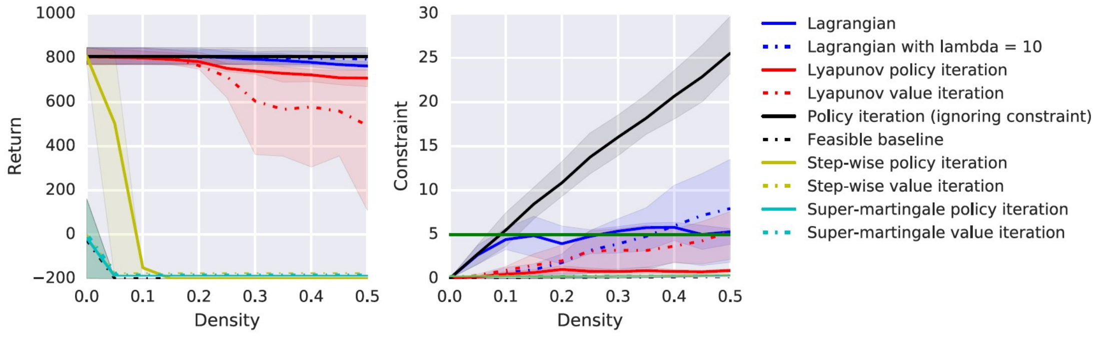

Fig. 5. Results of SPI and SVI compared to different baseline algorithms

are shown for two different constraint thresholds: $d_0 = 5$ and $d_0 = 1$.

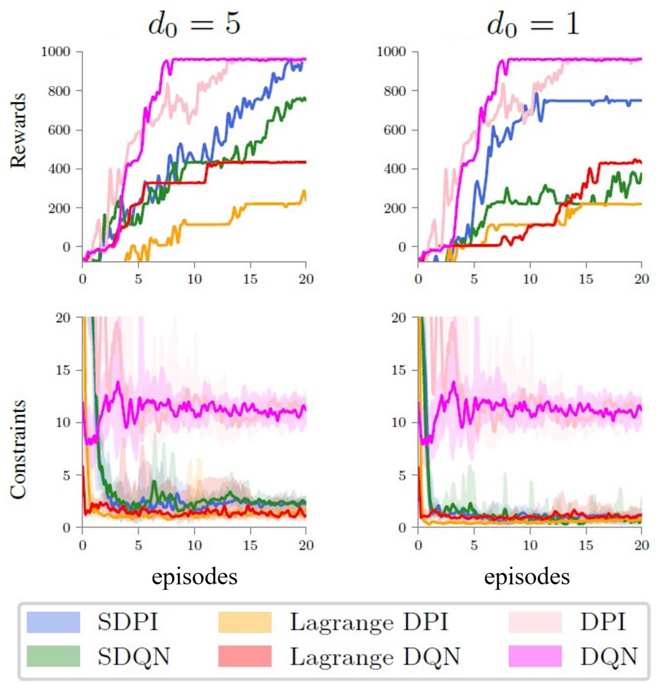

Fig. 6. Results of SDQN and SDPI compared to different baseline algorithms

The x-axes represent the number of thousands of episodes and the y-axes again represent the return and constraint violations. Here, the observations are 2D image representations of the grid-world environment. The image observations are fed through a convolutional neural network with three convolutional and two fully connected layers. SDQN and SDPI are able to solve the tasks with good performance while guaranteeing safety. For the lower constraint threshold $d_0 = 1$, the return of the SQDN approach decreases compared to SDPI. It is, however, still on a similar level compared to the Lagrangian approaches.

VII. Discussion

In this review paper, we presented the high level idea of Safe RL and concentrated on a Lyapunov-based approach to solve these problems. This section serves as a discussion of the results and embeds them into the context of previous work. The presented work is the first method to explicitly model the constraints in a CMDP using Lyapunov functions. The results show a high potential of the presented Lyapunov-based RL approaches for safe and robust learning in real-world scenarios. The algorithms guarantee safety and return good performance. In Fig. 6, it looks like the returns are sub-optimal compared to the unconstrained algorithms. However, the unconstrained algorithms allow the agent to move to the destination on a direct path, as there is no cost of moving onto an obstacle. One could implement this cost using standard MDP approaches like DQN and achieve high returns without constraint violations. However, it would not be possible to allow a maximum number of constraint violations during one iteration, like it is possible with CMDPs. Imagine a scenario where the path to the destination is blocked by a whole row of obstacles. Standard MDP algorithms would never reach the destination, as the episode immediately ends as soon as one obstacle is hit. This is a major advantage of the CMDP approach.

Compared to the other CMDP approaches, the Lagrangian method and the surrogate algorithms, the Lyapunov algorithms achieve similar or higher returns while better complying with the constraints. Another major advantage of the Lyapunov algorithms is the computation time, which is a drawback for the Lagrangian approach. One disadvantage of the presented Lyapunov algorithms is that it is still hard to estimate whether the baseline policy is sufficiently close to an optimal policy. In practical implementations, however, this assumption is often valid and can therefore be simplified.

In [15], SDQN has already been applied in a scenario, where it is used to maximise energy-efficiency in live video streaming from unmanned aerial vehicles while limiting transmission failures. In that method, the transmission failures are modeled as the constraints in a CMDP. The simulated results show excellent performance compared to other baseline approaches. However, the other three presented algorithms (SPI, SVI and SDPI) are yet to be validated in more benchmarks or real-world scenarios, since the authors only constructed a small 2D grid-world domain. In [10], for example, CPO (see IV) has already been validated in much more complex problems like walking humanoids or ants as well as collecting items.

In a following work, the team of [1] extends their Lyapunov approach to policy gradient algorithms to allow continuous RL problems [16]. Here, a policy gradient method is transformed into a safe algorithm. The result is validated using several MuJoCo [17] benchmarks and even an on-robot real-world navigation scenario. Future work focuses on a few directions. Training stability and safety can still be improved by exploring more Lyapunov function properties. Also, the constraints can be used more efficiently and the Lyapunov approach can be extended to other model-based algorithms.

References

[1] Y. Chow, O. Nachum, E. A. Duéñez-Guzmán, and M. Ghavamzadeh, “A lyapunov-based approach to safe reinforcement learning,” *CoRR*, vol. abs/1805.07708, 2018.

[2] E. Altman, *Constrained Markov decision processes*, vol. 7. CRC Press, 1999.

[3] D. Silver, “Lectures on reinforcement learning.” URL: https://www.davidsilver.uk/teaching/, 2015.

[4] K. Regan and C. Boutilier, “Regret-based reward elicitation for markov decision processes,” *CoRR*, vol. abs/1205.2619, 2012.

[5] J. H. Gillulay and C. J. Tomlin, “Guaranteed safe online learning of a bounded system,” in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2979–2984, 2011.

[6] T. M. Moldovan and P. Abbeel, “Safe exploration in markov decision processes,” *CoRR*, vol. abs/1205.4810, 2012.

[7] M. Ono, M. Pavone, Y. Kuwata, and J. Balaram, “Chance-constrained dynamic programming with application to risk-aware robotic space exploration,” *Autonomous Robots*, vol. 39, 12 2015.

[8] M. El Chamie, Y. Yu, and B. Açıkmeşe, “Convex synthesis of randomized policies for controlled markov chains with density safety upper bound constraints,” in *2016 American Control Conference (ACC)*, pp. 6290–6295, 2016.

[9] Z. Gábor, Z. Kalmár, and C. Szepesvári, “Multi-criteria reinforcement learning.,” pp. 197–205, 01 1998.

[10] J. Achiam, D. Held, A. Tamar, and P. Abbeel, “Constrained policy optimization,” *CoRR*, vol. abs/1705.10528, 2017.

[11] T. J. Perkins and A. G. Barto, “Lyapunov design for safe reinforcement learning,” *Journal of Machine Learning Research*, vol. 3, no. Dec, pp. 803–832, 2002.

[12] F. Berkenkamp, M. Turchetta, A. P. Schoellig, and A. Krause, “Safe model-based reinforcement learning with stability guarantees,” 2017.

[13] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller, “Playing atari with deep reinforcement learning,” *CoRR*, vol. abs/1312.5602, 2013.

[14] A. A. Rusu, S. G. Colmenarejo, C. Gulcehre, G. Desjardins, J. Kirkpatrick, R. Pascanu, V. Mnih, K. Kavukcuoglu, and R. Hadsell, “Policy distillation,” 2016.

[15] Q. Zhang, J. Miao, Z. Zhang, F. R. Yu, F. Fu, and T. Wu, “Energy-efficient video streaming in uav-enabled wireless networks: A safe-dqn approach,” in *GLOBECOM 2020 - 2020 IEEE Global Communications Conference*, pp. 1–7, 2020.

[16] Y. Chow, O. Nachum, A. Faust, E. Duenez-Guzman, and M. Ghavamzadeh, “Lyapunov-based safe policy optimization for continuous control,” 2019.

[17] E. Todorov, T. Erez, and Y. Tassa, “Mujoco: A physics engine for model-based control,” in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033, IEEE, 2012.