FOCOPS Review

Aadil Ahamed
UC San Diego
`aahamed@ucsd.edu`

Abstract

This report is a review of the paper FOCOPS (First Order Constrained Optimization in Policy Space) by Zhang et al. [1]. We begin by presenting a brief summary of the motivation, related work and methods of FOCOPS followed by a critical review of the results and experiments described in the paper. Finally, we present a few additional experiments to gain some more insight into the details of the FOCOPS algorithm. The code for the experiments are avaialbe at https://github.com/aahamed/focops-review.

1 Introduction

Reinforcement Learning (RL) has enjoyed great success at a wide variety of tasks, such as video games and robotics, where the goal of the agent is to maximize the accumulated reward through free exploration of the state space. However, in many real world scenarios, it may not be practical to allow the agent to explore the state space without any constraints. For example, when training an autonomous vehicle, we would like to ensure that the vehicle never crashes during training since it would be extremely expensive to replace the vehicle every time it crashed. Hence, safe exploration has become an increasingly important area of research in RL.

Many works in this area model safety through the use of a Constrained Markov Decision Process (CMDP). A CMDP is similar to a MDP but it has an additional set of cost constraints that the optimal policy must satisfy. These constraints often conflict with the reward function which makes the problem more challenging than a traditional MDP. While methods to solve CMDPs with known models have been around for a long time, solutions for CMDPs with large model free environments has only gained traction in recent years. Constrained Policy Optimization (CPO) is one such work that is most relevant to the methods presented in this paper. CPO approximates the CMDP using Taylor expansions and then uses a backtracking line search to account for approximation errors. CPO has the advantage that it ensures intermediate policies are safe during training and that it can be integrated with parametric policies, such as neural networks, which are essential to tackling problems with high dimensional state and action spaces. However, CPO does use second order methods which can make the algorithm computationally expensive and its use of approximations can lead to suboptimal performance.

First Order Constrained Optimization in Policy Space (FOCOPS) addresses the issues present in CPO by taking a two step approach. First it solves the CMDP in the non-parametric policy space and then projects the solution into the parameter space by minimizing the KL divergence. This algorithm only uses first order gradients and removes some of the approximation errors incurred from using taylor expansions. The authors of FOCOPS present a set of experiments that provide empirical data proving that the FOCOPS algorithm outperforms CPO. Despite the improvement in performance, FOCOPS does have some drawbacks. When compared to CPO, FOCOPS is more likely to violate cost constraints during training and it also introduces two more hyperparameters that need to be tuned in order to obtain optimal performance.

2 Related Work

The concept of safety, in RL literature, is modeled in many different ways. Moldovan et al propose an algorithm to enforce ergodicity, which encourages the agent to only visit states from which it can get back to the original state [2]. The idea here is that agents should only visit states that it can recover from. In a similar vein, Lipton et al [3] introduce an intrinsic fear heuristic which shapes the reward function so that the agent never goes near catastrophic states. This formulation may not always make sense, since in some domains, such as video games, obtaining the higher reward signals may require entering dangerous states. But in such domains safety might be less of a concern. Other definitions of safety include maximizing the worst case expected return or reducing the variance of the return [4].

But most often, safe exploration is modeled as a CMDP where additional constraints must be satisfied while trying to maximize the reward function. There has been a fairly large body of research in CMDPs. For small CMDPs, linear programming can be used to learn the optimal policy that satisfy constraints [5]. For CMDPs with large state-action spaces, primal-dual methods can be used to achieve optimal policies that satisfy constraints once training completes [6]. However, these methods do not provide any guarantees of constraint satisfaction during training. [7] proposes incorporating the constraints into the objective with hyperparameters to control how much importance is given to the constraints. This approach has the drawback of requiring significant hyperparameter tuning in order to achieve good performance. CPO, combines TRPO with line search to ensure cost constraint satisfaction during training [8]. However, this approach requires second order methods and can be computationally expensive. Some approaches, such as [9], use Lyapunov functions, from control theory, to solve the CMDP, but they do not apply their method to policy gradient algorithms. PCPO [10] applies a two step approach where the policy is first updated using a TRPO step and then projected back into the set of feasible policies that satisfy all constraints. This method also has the disadvantage of requiring the use of second order methods during policy iteration.

FOCOPS differs from all these works because it first solves for the optimal policy update in the non-parametric policy space and then projects it back into the parameter space using only first order methods. While the idea of solving for policies in the non-parametric space and projecting it back to parameter space is not novel [11] it is the first time this approach has been applied to the CMDP framework.

3 Technical Results

3.1 Preliminaries

3.1.1 MDP

A MDP (Markov Decision Process) can be expressed as a 5-tuple $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P}, \mu)$ where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $\mathcal{R}$ is the reward function, $\mathcal{P}$ describes the state transition probabilities and μ is the initial state distribution. We define a policy π as a mapping from state to actions: $\pi = \{\pi(a|s) : s \in \mathcal{S}, a \in \mathcal{A}\}$ and the set of all such stationary policies as Π. Solving the MDP requires finding the policy π that maximizes the expected discounted return $J(\pi)$:

$$\max_{\pi \in \Pi} J(\pi) = \max_{\pi \in \Pi} \mathbb{E}_{\tau \sim \pi} \left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \middle| s_0 = s \right]$$

Here τ is a trajectory denoting a sequence of states and actions and $\tau \sim \pi$ means that the trajectories are drawn according to the policy π. The variable γ represents the discount factor. For a policy π, the value function is defined as $V^{\pi}(s) = \mathbb{E}_{\tau \sim \pi} \left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) | s_0 = s \right]$, the action-value function as $Q^{\pi}(s, a) = \mathbb{E}_{\tau \sim \pi} \left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) | s_0 = s, a_0 = a \right]$ and the advantage function as $A^{\pi}(s, a) = Q^{\pi}(s, a) - V^{\pi}(s, a)$. Finally the future state visitation distribution is defined as $d^{\pi}(s) = (1 - \gamma) \sum_{t=0}^{\infty} \gamma^t P(s_t = s|\pi)$.

3.1.2 CMDP

A CMDP (Constrained Markov Decision Process) is the exact same as a MDP except it contains an additional set of constraints $\mathcal{C}$. Each constraint $\mathcal{C}_i$ maps state-action pairs to a real number denoting the cost (similar to reward functions): $\mathcal{C}_i : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. The expected discounted cost return is then defined as $J_{C_i}(\pi) = \mathbb{E}_{\tau\sim\pi}\left[\sum_{t=0}^{\infty}\gamma_c^t \mathcal{C}_i(s_t,a_t)|s_0=s\right]$. The set of all feasible policies can now be described as:

$$\Pi_C : \{\pi \in \Pi \text{ s.t. } J_{C_i}(\pi) < b_i \quad \forall i\}$$

Here b_i is the cost threshold for the i^{th} constraint. Solving the CMDP then requires finding the optimal policy π that maximizes the expected discounted reward while satisfying all the constraints:

$$\max_{\pi\in\Pi_C} J(\pi) = \max_{\pi\in\Pi_C} \mathbb{E}_{\tau\sim\pi}\left[\sum_{t=0}^{\infty}\gamma^t R(s_t,a_t)\middle| s_0=s\right]$$

Note here that the only difference between this objective and the previous one is that $\pi \in \Pi_C$ instead of Π. The value V_C^π, action-value Q_C^π and advantage A_C^π functions for the cost are defined in the same way as before except the reward $R(s,a)$ is replaced with the cost $C(s,a)$. Note that FOCOPS paper and this review assumes there is only one constraint.

3.1.3 CPO

FOCOPS primarily builds upon the ideas of CPO (Constrained Policy Optimization) by Achiam et al. [8]. Since the CPO algorithm is designed to tackle CMDPs with high dimensional state and action spaces, it only considers parameterized policies $\Pi_\theta = \{\pi_\theta : \theta \in \Theta\}$. Here Θ is the set of all possible weights for a neural network with a given architecture and θ is a specific instance of such weights. CPO then reformulates the CMDP objective as:

$$\max_{\pi_\theta\in\Pi_\theta} \quad \mathop{\mathbb{E}}_{\substack{s\sim d^{\pi_{\theta_k}} \\ a\sim\pi_\theta}}\left[A^{\pi_{\theta_k}}(s,a)\right] \tag{1}$$

$$\text{subject to} \quad J_C(\pi_{\theta_k}) + \frac{1}{1-\gamma}\mathop{\mathbb{E}}_{\substack{s\sim d^{\pi_{\theta_k}} \\ a\sim\pi_\theta}}\left[A_C^{\pi_{\theta_k}}(s,a)\right] \le b \tag{2}$$

$$\bar{D}_{\mathrm{KL}}\left(\pi_\theta \| \pi_{\theta_k}\right) \le \delta \tag{3}$$

The objective in eq(1) is equivalent to maximizing the expected discounted return $J(\pi_\theta)$. The constraint in eq(2) is equivalent to the cost constraint $J_C(\pi_\theta) \le b$ but since it is too expensive to evaluate $J_C(\pi_\theta)$, CPO uses the left hand side of the inequality in eq(2) as a surrogate cost function. Lastly, eq(3) is a trust region constraint where $\bar{D}_{\mathrm{KL}}\left(\pi_\theta\|\pi_{\theta_k}\right)$ is the KL divergence between π_θ and π_{θ_k}. The trust region constraint helps prevent the updated policy from collapsing. Since it is not feasible to solve eqs(1-3) directly, CPO applies first and second order taylor expansions in order to simplify the problem. The resulting problem is convex and CPO uses standard convex optimization techniques to find the solution.

3.2 FOCOPS

FOCOPS differs from CPO by replacing the first and second order taylor expansions in the policy update step with the following method:

- First it solves for the optimal update policy π^* that satisfies eq(1-3) in the non-parametric policy space.
- Next, it projects π^* back into the parameter policy space by minimizing the KL divergence.

3.2.1 Optimal Update Policy

Since FOCOPS searches in the non-parametric policy space, eq(1-3) can be transformed to:

$$\max_{\pi\in\Pi} \quad \mathop{\mathbb{E}}_{\substack{s\sim d^{\pi_{\theta_k}} \\ a\sim\pi}}\left[A^{\pi_{\theta_k}}(s,a)\right] \tag{4}$$

$$\text{subject to} \quad J_C(\pi_{\theta_k}) + \frac{1}{1-\gamma}\mathop{\mathbb{E}}_{\substack{s\sim d^{\pi_{\theta_k}} \\ a\sim\pi}}\left[A_C^{\pi_{\theta_k}}(s,a)\right] \le b \tag{5}$$

$$\bar{D}_{\mathrm{KL}}\left(\pi \| \pi_{\theta_k}\right) \le \delta \tag{6}$$

Notice that the only difference between eq(4-6) and eq(1-3) is that $\pi \in \Pi$ instead of Π_Θ. Assuming the current policy π_{θ_k} is feasible, the authors of FOCOPS prove that the optimal update policy π^* that satisfies eq(4-6) takes the form:

$$\pi^*(a|s) = \frac{\pi_{\theta_k}(a|s)}{Z_{\lambda,\nu}(s)} \exp\left[\frac{1}{\lambda}\left(A^{\pi_{\theta_k}}(s,a) - \nu A_C^{\pi_{\theta_k}}(s,a)\right)\right] \tag{7}$$

Here $Z_{\lambda,\nu}(s)$ is a partition function that ensures π^* is a valid probability distribution that sums to 1. The parameters λ and ν are solutions to the following constrained optimization problem:

$$\min_{\lambda,\nu\geq 0} \lambda\delta + \nu\widetilde{b} + \lambda \underset{\substack{s\sim d^{\pi_{\theta_k}} \\ a\sim\pi}}{\mathbb{E}} [\log Z_{\lambda,\nu}(s)] \tag{8}$$

Here $\widetilde{b} = (1-\gamma)(b - J_C(\pi_{\theta_k}))$. For the purposes of this review we can ignore eq(8) since it is too expensive to obtain the partition function $Z_{\lambda,\nu}$ for high dimensional state-action spaces. Instead we will focus on eq(7) and provide some high level intuition of the result. The result shows that $\pi^*(a|s)$ is large for state action pairs with high reward ($A^{\pi_{\theta_k}}(s,a)$) and low cost ($A_C^{\pi_{\theta_k}}(s,a)$). This seems reasonable since we would like the policy to sample actions that maximize the reward without incurring significant cost. The parameter λ acts like a temperature, where setting $\lambda = 0$ causes the policy π^* to mimic π_{θ_k} and as λ increases, π^* becomes more exploratory. The parameter ν determines how much to penalize state-action pairs with high cost in order to ensure constraint satisfaction.

3.2.2 Projecting Back to Parametrized Policy Space

The optimal update policy π^* described in the previous section is not in Π_Θ so FOCOPS projects π^* back to Π_Θ by minimizing the KL divergence between π^* and π_θ:

$$\min_\theta \mathcal{L}(\theta) = \min_\theta \mathbb{E}_{s\sim d^{\pi_{\theta_k}}}\left[D_{\mathrm{KL}}\left(\pi_\theta \| \pi^*\right)[s]\right] \tag{9}$$

The paramters θ that minimize eq(9) can be obtained via gradient descent:

$$\nabla_\theta \mathcal{L}(\theta) = \mathbb{E}_{s\sim d^{\pi_{\theta_k}}}\left[\nabla_\theta D_{\mathrm{KL}}\left(\pi_\theta \| \pi^*\right)[s]\right] \tag{10}$$

where

$$\nabla_\theta D_{\mathrm{KL}}\left(\pi_\theta\|\pi^*\right)[s] = \nabla_\theta D_{\mathrm{KL}}\left(\pi_\theta\|\pi_{\theta_k}\right)[s] - \frac{1}{\lambda} \underset{a\sim\pi_{\theta_k}}{\mathbb{E}}\left[\frac{\nabla_\theta \pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}(A^{\pi_{\theta_k}}(s,a) - \nu A_C^{\pi_{\theta_k}}(s,a))\right] \tag{11}$$

The main takeaway from eq(10-11) is that it does not depend on the partition function $Z_{\lambda,\nu}(s)$ and that the rest of the terms, except for λ and ν, can be estimated by sampling trajectories using the current policy π_{θ_k}.

The only part that remains to be covered is how to set the parameters λ and ν. As stated previously, it is not possible to solve eq(8) directly and hence FOCOPS relies on heuristics to set these parameters. The authors report that keeping the parameter λ fixed during training provides good results. The exact value of λ is determined empirically through hyperparameter tuning. However, ν does need to be updated at every iteration in order to satisfy cost constraints during training. The authors show that updating ν according to the following policy works well in practice:

$$\nu \leftarrow \mathrm{proj}_\nu\left[\nu - \alpha(b - J_C(\pi_{\theta_k}))\right] \tag{12}$$

Here the proj_ν operator maps $\nu \in [0, \nu_{max}]$. The value of ν_{max} is also chosen through hyperparameter tuning. Qualitatively, this update policy increases ν when $J_C(\pi_{\theta_k}) > b$ and decreases it otherwise. This is intuitive since ν acts as a penalty term and if the current cost return exceeds the threshold then we want to increase the penalty and if it doesn't we want to decrease it.

An outline of the full FOCOPS algorithm is presented in algorithm 1.

Algorithm 1: FOCOPS algorithm

```
Initialize policy $\pi_\theta$ ;
for $k = 1$ to MAX_ITER do
    Generate trajectories according to the current policy $\pi_\theta$ and store each state transition in a
      memory buffer ;
    Estimate $J_C(\pi_\theta)$, $A^{\pi_\theta}$ and $A_C^{\pi_\theta}$ ;
    Update $\nu$ according to eq(12) ;
    Store current policy as $\pi_{\theta_{old}} = \pi_\theta$ ;
    for epoch $= 1$ to NUM_EPOCHS do
        for minibatch $M$ in memory buffer do
            Update policy network $\pi_\theta$ according to eq(10). This will make $\pi_\theta$ closer to $\pi^*$ ;
        end
        if $D_{\mathrm{KL}}(\pi_\theta \| \pi_{\theta_{old}}) > \delta$ then
            Do early stopping ;
        end
    end
end
```

4 Review

The main contribution of FOCOPS is that it provides a new approach to solve CMDPs. The results, specifically equations 7 and 10, provide a novel way to optimize the policy while still approximately satisfying cost constraints during training. Moreover, they can be obtained using only first order methods whereas much of the related work in this area relies on second order methods. In practice, this makes the algorithm simpler to implement and computationally more efficient. However, despite its strengths, FOCOPS does have some weaknesses that were not fully addressed in the paper.

A central claim of this paper is that FOCOPS outperforms CPO in all of the reported experiments. While this is true for the experiment settings described in the paper it may not be a fair comparison. The expected cost plots presented in Figure 1 show that FOCOPS is much more likely to violate cost constraints during training compared to CPO. Constraint violation is actually baked into the design, since the parameter ν will only increase when the cost constraint is violated resulting in the expected cost return to oscillate around the threshold in a damped fashion. In contrast, CPO is more conservative and uses principled methods to ensure constraints are never violated during training. Hence, in practical applications, it would be reasonable to give CPO a higher cost threshold compared to FOCOPS and this could in turn lead to better overall performance.

Another disadvantage of FOCOPS over CPO is that it relies on heuristics to set the parameters λ and ν. As a result two more hyperparameters λ and ν_{max} are introduced which adds which increases the amount of hyperparameter tuning required. However, to their benefit, the authors do show that FOCOPS is fairly robust to the choice of λ and ν_{max} by showing that the average performance degradation from the optimal λ is 7.4% while the degradation from the optimal ν_{max} is 0.3%.

FOCOPs also doesn't mention some inherent limitations that arise from using the CMDP framework. For the Robots with Speed Limit experiment, the authors imply that setting the a cost threshold is equivalent to setting a speed limit on the robot. However, because of the definition of CMDP, the cost threshold is actually a distance limit and the robot is free to move at any velocity it wants as long as it doesn't travel past the specified distance. Additional details regarding this particular limitation will be provided in the Experiments section. The takeaway here is that not all constraints are easy to express in the CMDP framework and that CMDPs only ensure that the policy is safe on average.

The authors of FOCOPS only apply their algorithm to environments with continuous state and action spaces. As a result we do not get an idea of how well FOCOPS might perform in domains with discrete state or action spaces. To remedy this, we adapt the FOCOPS implementation to use a discrete policy, and run it on two new environments with discrete action spaces.

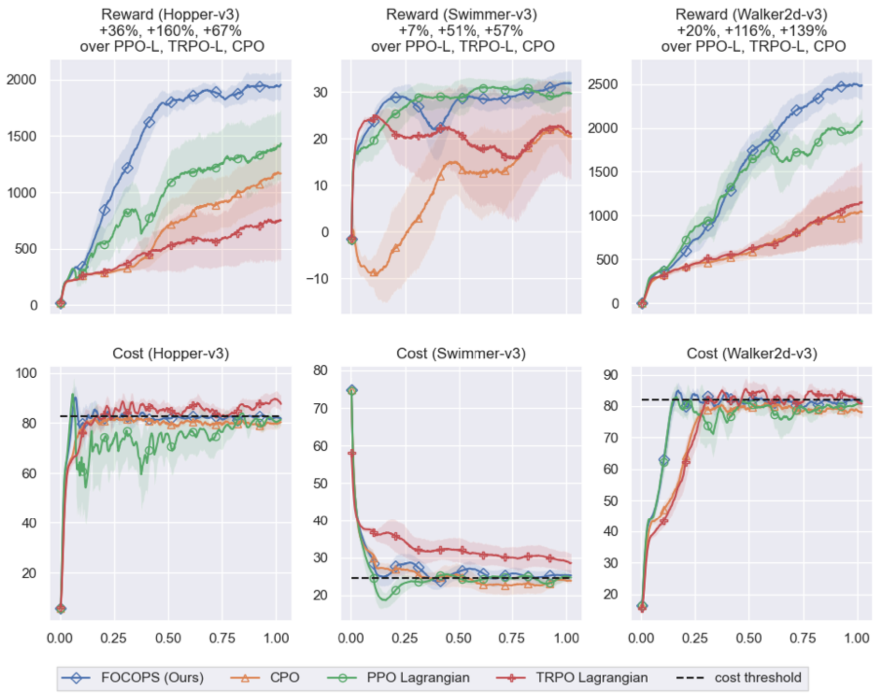

Figure 1: The top row shows the expected reward vs. millions of timesteps and the bottom row shows the expected cost return vs. millions of timesteps from the Robot Speed Limit experiment in the FOCOPS paper [1]. In all of the cost return plots, given a feasible policy, CPO (orange) always satisfies cost constraints whereas FOCOPS (blue) oscillates around the cost threshold.

5 Experiments

In this section we will present some of the additional experiments that we conducted in order to better understand the FOCOPS algorithm. The code for the experiments are avaialbe on github.

5.1 Robots with Speed Limit

In our first experiment, we will look at the Robots with Speed Limit experiment described in the FOCOPS paper. In the original experiment, the authors design a cost function which is equal to the linear velocity of the robot:

$$C(s) = v_x$$

They then apply a cost threshold that is 25%, 50% and 75% of the expected cost return of an unconstrained PPO agent trained for a million timesteps on different mujoco [12] robot environments. Finally they run FOCOPS on these environments and imply that because the algorithms satisfy the cost constraints the robots must be observing the speed limit. However, if we dig a little deeper we can see that this is not actually the case. We repeated the experiment on the Hopper-v3 environment and additionally collected the speed vs. timestep at different iterations and plotted them in Figure 2. The plots show that the robot moves at a speed well over the average speed limit in the first 300 timesteps and is stationary for the remainder of the episode. Even though the speed of the robot, averaged across the entire 1000 timestep episode, is under the speed limit, it clearly does not observe any traditional definition of a speed limit. Moreover, the average reward vs. iteration plot shows the expected reward flatlining after iteration 100, indicating that the learned behavior does not improve with further training. This issue is not due to the algorithm itself, but a limitation of the CMDP framework, which only requires constraints to be satisified in expectation across the duration of the episode. Still, this limitation is not mentioned in the FOCOPS paper and illustrates that even simple constraints, such as a speed limit, can be hard to express in the CMDP framework.

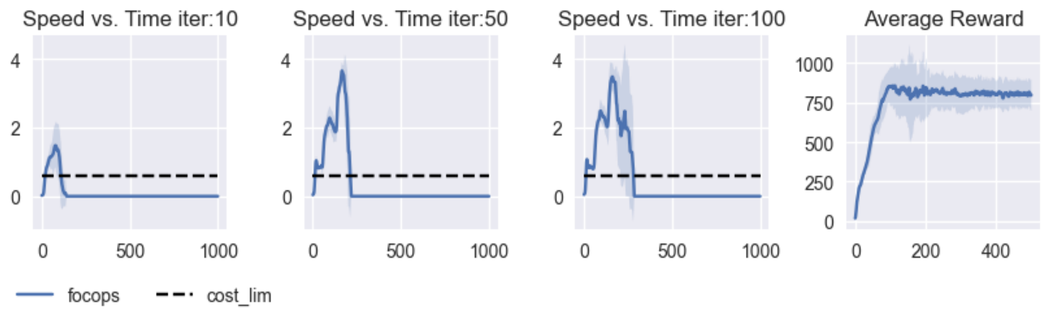

Figure 2: The first three plots from the left show the instantaneous speed (averaged across a 100 episodes) vs timestep at policy iterations 10, 50 and 100 during training. The dashed line indicates the speed limit. The final plot on the right shows the expected reward per policy iteration. These plots prove that the learned behavior can deviate significantly from the intended behavior of the cost constraints.

5.2 CartSafe

In this next experiment we see if the FOCOPS algorithm can generalize to a new environment called CartSafe [13]. This environment is similar to the CartPole [14] environment except here the pendulum starts hanging down and needs to be swung into the upright position while keeping the cart inside a safe region. The initial state of this environment is illustrated in Figure 3. The cost function used in this environment penalizes the agent whenever the cart moves outside the safe region:

$$C(s) = \begin{cases} 1 & x_{cart} > |x_{lim}| \\ 0 & \text{o.w.} \end{cases}$$

Here x_{cart} is the position of the cart's center and $x_{lim} = 1\text{cm}$ is the distance from the center of the track. CartSafe uses a discrete action space, but the FOCOPS implementation only supports continuous action spaces. So we added a discrete policy and modified the FOCOPS algorithm to handle discrete action spaces in order to test it on this new environment. The results of the experiment, with a cost threshold of 10, are presented in Figure 3.

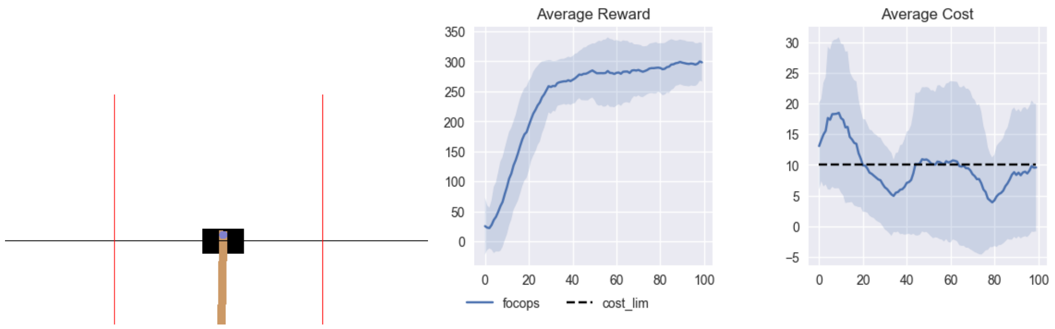

Figure 3: The image on the left shows the initial state of the CartSafe environment with the pendulum hanging upside down and the red vertical bars indicating the safe region. The image in the center is a plot of average reward vs. policy iterations. The image on the right is a plot of average cost vs. policy iterations. The dashed line represents the cost constraint and the highlighted regions represent the standard deviation.

The results show that FOCOPS was able to generalize to a new environment with a discrete action space since the expected reward increases monotonically during training. Though the initial policy is infeasible, FOCOPS is able to figure out how to drive down the expected cost return in subsequent iterations and approximately satisfy the cost constraint for the remainder of the training.

5.3 Grid Navigation with Obstacles

In our final experiment we run FOCOPS on the 2D motion planning task introduced in [9] and implemented in [13]. In this problem, the robot starts in the bottom right of a 25×25 grid and must navigate to the goal which is randomly placed at some location in the first row as illustrated in Figure 4.

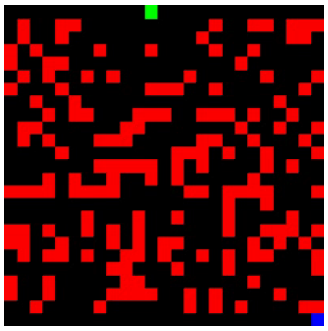

Figure 4: Illustration of the grid navigation task with obstacles. The blue square represents the robot, the green square represents the goal and the red squares represent obstacles.

The robot can specify a move in any of the four cardinal directions but the environment may move the robot in a random direction with some small probability. Additionally, there are obstacles dispersed throughout the grid that the robot should try to avoid. The robot incurs a cost of 1 whenever it lands on a cell containing an obstacle. It also loses a reward of 1 for every timestep it spends exploring and receives a reward of 1000 when it reaches the goal. The task of the robot is to navigate to the goal in the shortest number of timesteps possible while avoiding the obstacles. Since this reward function is very sparse (-1 everywhere except at the goal) we modified the reward function to increase as the robot gets closer to the goal:

$$R(s) = -1 - \frac{d}{d_{max}}$$

Here d is the euclidean distance between the robot's position and the goal and d_{max} is the distance between two opposite corners of the grid. The plots in Figure 5 show the performance of FOCOPS on this environment. When the cost threshold is 15, FOCOPS is eventually able to learn a feasible policy that reliably reaches the goal since it achieves an expected return of around 900. However, when the cost threshold is reduced to 10, FOCOPS is no longer able to reliably achieve the goal state which is why the expected reward lingers around -300 for the entirety of training. The poor performance here is not really due to the contributions of FOCOPS but a general limitation of on-policy algorithms in environments with sparse reward signals. This experiment does, however, provide motivation to extend the FOCOPS algorithm to use off-policy data as it would expand the scope of applications in which FOCOPS can be effective.

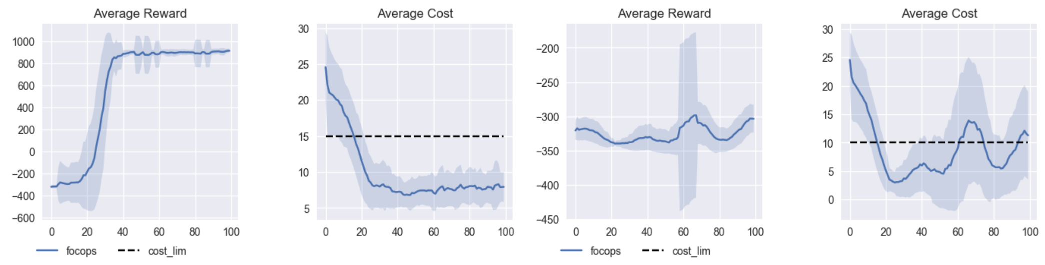

Figure 5: The left two plots show the average reward and cost returns vs. policy iterations for a cost threshold of 15. The right two plots show the average reward and cost returns vs policy iterations for a cost threshold of 10. The highlighted regions represent the standard deviation.

References

[1] Zhang, Y., Q. Vuong, K. W. Ross. First order constrained optimization in policy space, 2020.

[2] Moldovan, T. M., P. Abbeel. Safe exploration in markov decision processes, 2012.

[3] Lipton, Z. C., K. Azizzadenesheli, A. Kumar, et al. Combating reinforcement learning's sisyphean curse with intrinsic fear, 2018.

[4] García, J., Fern, o Fernández. A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research*, 16(42):1437–1480, 2015.

[5] Altman, E. *Constrained Markov Decision Processes*. Chapman and Hall, 1999.

[6] Chow, Y., M. Ghavamzadeh, L. Janson, et al. Risk-constrained reinforcement learning with percentile risk criteria, 2017.

[7] Tessler, C., D. J. Mankowitz, S. Mannor. Reward constrained policy optimization, 2018.

[8] Achiam, J., D. Held, A. Tamar, et al. Constrained policy optimization, 2017.

[9] Chow, Y., O. Nachum, E. Duenez-Guzman, et al. A lyapunov-based approach to safe reinforcement learning, 2018.

[10] Yang, T.-Y., J. Rosca, K. Narasimhan, et al. Projection-based constrained policy optimization, 2020.

[11] Abdolmaleki, A., J. T. Springenberg, Y. Tassa, et al. Maximum a posteriori policy optimisation, 2018.

[12] Todorov, E., T. Erez, Y. Tassa. Mujoco: A physics engine for model-based control. In *IROS*, pages 5026–5033. IEEE, 2012.

[13] Jemaw. Gym-safety. `https://github.com/jemaw/gym-safety`, 2019.

[14] Brockman, G., V. Cheung, L. Pettersson, et al. Openai gym, 2016.