Monte-Carlo Tree Search for Constrained MDPs

Jongmin Lee[1], **Geon-Hyeong Kim**[1], **Pascal Poupart**[2], **Kee-Eung Kim**[1,3]
[1] School of Computing, KAIST, Republic of Korea
[2] David R. Cheriton School of Computer Science, University of Waterloo, Canada
[3] PROWLER.io
{jmlee,ghkim}@ai.kaist.ac.kr, ppoupart@uwaterloo.ca, kekim@cs.kaist.ac.kr

Abstract

Monte-Carlo Tree Search (MCTS) is the state-of-the-art online planning algorithm for very large MDPs. However, many real-world problems inherently have multiple goals, where multi-objective sequential decision models are more natural. The constrained MDP (CMDP) is such a model that maximizes the reward while constraining the cost. The common solution method for CMDPs is linear programming (LP), which is hardly applicable to large real-world problems. In this paper, we present CCUCT (Cost-Constrained UCT), an online planning algorithm for large constrained MDPs (CMDPs) that leverages the optimization of LP-induced parameters. We show that CCUCT converges to the optimal stochastic action selection in CMDPs and it is able to solve very large CMDPs through experiments on the multi-objective version of an Atari 2600 arcade game.

1 Introduction

Monte-Carlo Tree Search (MCTS) [Kocsis and Szepesvári, 2006; Coulom, 2006; Browne *et al.*, 2012] is a generic online planning algorithm that effectively combines random sampling and tree search, and has shown great success in many areas such as online Bayesian reinforcement learning [Guez *et al.*, 2013] and computer Go [Gelly and Silver, 2011; Silver *et al.*, 2016]. MCTS efficiently explores the search space by investing more search effort in promising states and actions while balancing exploration and exploitation in the direction of maximizing the cumulative (scalar) rewards. Due to its outstanding performance without prior domain knowledge or heuristic function, MCTS has become the de-facto standard method for solving very large MDPs.

However in many situations, reward maximization alone is insufficient. For example, consider the budget allocation problem of an online advertiser [Archak *et al.*, 2012]. In this problem, the advertiser should determine how to advertise optimally while constraining the total cost within the budget. The constrained MDP (CMDP) [Altman, 1999] is an appealing framework for dealing with this kind of multi-objective sequential decision making problems. The model assumes that the action incurs not only rewards, but also K different types of costs, and the goal is to find an optimal policy that maximizes the expected cumulative rewards while bounding each of K expected cumulative costs to certain levels.

Although the CMDP is an attractive framework, the common solution method still remains to be linear programming [Altman, 1999; Zadorojniy *et al.*, 2009]. This is in contrast to MDPs, where dynamic programming and MCTS are readily applicable to large problems. One exception is the seminal work on dynamic programming for CMDPs [Piunovskiy and Mao, 2000], but the method is intractable even for small state spaces, sharing a lot of similarity with partially observable MDPs. As such, the CMDP has not been a practical model for very large state spaces.

In this paper, we present MCTS for CMDPs, which precisely addresses the scalability issue. To the best of our knowledge, extending MCTS to CMDPs has remained mostly unexplored since it is not straightforward to handle the constrained optimization in CMDPs. This challenge is compounded by the fact that optimal policies can be stochastic.

There exists some prior work on MCTS for multi-objective problems, e.g. [Wang and Sebag, 2012; Liebana *et al.*, 2015], which focused on computing Pareto-optimal policies with multiple reward functions. The main idea was to scalarize the rewards using the hyper-volume indicator [Zitzler and Thiele, 1998] and performing vanilla UCT on the resulting single-objective problem. In contrast, we focus on the problem of finding a *single* optimal policy satisfying the cost constraints, yielding a solution that would be more useful in practice.

In order to develop MCTS for CMDPs, we first show that solving CMDPs is essentially equivalent to jointly solving an unconstrained MDP while optimizing its LP-induced parameters that control the trade-off between the reward and the costs. From this result, we describe our algorithm, Cost-Constrained UCT (CCUCT), for solving large CMDPs that combine traditional MCTS with LP-induced parameter optimization. In the experiments, we show that CCUCT converges to the optimal stochastic actions using a synthetic domain, and also show that it scales to very large CMDP problems using a multi-objective version of an Atari 2600 arcade game.

2 Background

Constrained Markov decision processes (CMDPs) [Altman, 1999] provide a framework for modeling sequential decision making problems with cost-constrained situations. It is formally defined by tuple $\langle S, A, T, R, \mathbf{C}, \hat{\mathbf{c}}, \gamma, s_0\rangle$, where S is the set of states s, A is the set of actions a, $T(s'|s,a) = \Pr(s'|s,a)$ is the transition probability of reaching s' when the agent takes action a in state s, $R(s,a) \in \mathbb{R}^+$ is the reward function that returns the immediate reward incurred by taking action a in state s, $\mathbf{C} = \{C_k\}_{1..K}$ is the set of K cost functions with individual thresholds $\hat{\mathbf{c}} = \{\hat{c}_k\}_{1..K}$, $\gamma \in [0,1)$ is the discount factor, and $s_0 \in S$ is the initial state. We assume that cost functions are non-negative, i.e. $C_k(s,a) \in [0, +\infty)$.

The goal is to compute an optimal policy that maximizes the expected cumulative reward while bounding the expected cumulative costs:

$$\max_{\pi} V_R^{\pi}(s_0) = \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t,a_t)|s_0\right]$$
$$s.t.\ V_{C_k}^{\pi}(s_0) = \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t C_k(s_t,a_t)|s_0\right] \le \hat{c}_k \ \ \forall k$$

The optimal policy of the CMDP is generally stochastic and can be obtained by solving the following linear program (LP) [Altman, 1999]:

$$\max_{\{y(s,a)\}\forall s,a} \sum_{s,a} R(s,a) y(s,a) \tag{1}$$
$$s.t.\ \sum_{a'} y(s',a') = \delta(s_0,s') + \gamma \sum_{s,a} T(s'|s,a) y(s,a) \ \ \forall s'$$
$$\sum_{s,a} C_k(s,a) y(s,a) \le \hat{c}_k \ \ \forall k$$
$$y(s,a) \ge 0 \ \ \forall s,a$$

where $y(s,a)$ can be interpreted as a discounted occupancy measure of (s,a), and $\delta(x,y)$ is a Dirac delta function that has a value of 1 if $x = y$ and 0 otherwise. Once the optimal solution $y^*(s,a)$ is obtained, an optimal stochastic policy and the corresponding optimal value are computed as $\pi^*(a|s) = \Pr(a|s) = y^*(s,a)/\sum_{a'} y^*(s,a')$ and $V_R^*(s_0;\hat{\mathbf{c}}) = \sum_{s,a} R(s,a) y^*(s,a)$ respectively.

UCT [Kocsis and Szepesvári, 2006; Kocsis *et al.*, 2006] is a Monte-Carlo tree search (MCTS) algorithm for (unconstrained) MDPs that adopts UCB1 [Auer *et al.*, 2002] as the action selection rule in the intermediate nodes of the search tree:

$$\arg\max_a \left[Q_R(s,a) + \kappa \sqrt{\frac{\log N(s)}{N(s,a)}}\right] \tag{2}$$

where $Q_R(s,a)$ is the average of the sampled rewards, $N(s)$ is the number of simulations performed through s, $N(s,a)$ is the number of times action a is selected in s, and κ is the exploration constant to adjust the exploration-exploitation trade-off. UCT expands the search tree non-uniformly, focusing more search efforts into promising nodes. It can be formally shown that $Q_R(s,a)$ asymptotically converges to the optimal value $Q_R^*(s,a)$ in MDPs with some proper constant κ.

Unfortunately, it is not straightforward to use UCT for constrained MDPs since the original UCB1 action selection rule does not have any notion of cost constraints. If we follow reward-maximizing vanilla UCT without any modification, we may obtain cost-violating action sequences. Even though we could retain the average of sampled cumulative costs Q_C, it is not straightforward how to use them. If we simply prevent action branches that violate the cost constraint $Q_C(s,a) \le \hat{c}$, we obtain policies that are too conservative and thus sub-optimal: a feasible policy can be rejected from the search if the Monte-Carlo estimate violates the cost constraint.

3 Solving CMDP via an MDP Solver

The derivation of our algorithm starts from the dual of Eq. (1):

$$\min_{\substack{\{\mathcal{V}(s)\}\forall s\\ \{\lambda_k\}\forall k}} \sum_s \delta(s_0,s)\mathcal{V}(s) + \sum_k \hat{c}_k \lambda_k \tag{3}$$
$$s.t.\quad \mathcal{V}(s) \ge R(s,a) - \sum_k C_k(s,a)\lambda_k + \gamma \sum_{s'} T(s'|s,a)\mathcal{V}(s') \ \ \forall s,a$$
$$\lambda_k \ge 0 \ \ \forall k$$

Observe that if we treat $\boldsymbol{\lambda} = [\lambda_1, \ldots, \lambda_K]$ as a constant, the problem becomes an *unconstrained* MDP with the scalarized reward function $R(s,a) - \boldsymbol{\lambda}^\top \mathbf{C}(s,a)$. Let $\mathcal{V}_{\boldsymbol{\lambda}}^*$ be the optimal value function of this unconstrained MDP. Then, for any $\boldsymbol{\lambda}$, there exists the corresponding unique $\mathcal{V}_{\boldsymbol{\lambda}}^*$, and we can compute $\mathcal{V}_{\boldsymbol{\lambda}}^*$ with an MDP solver. Thus, (3) reduces to:

$$\min_{\boldsymbol{\lambda} \ge \mathbf{0}} \left[\mathcal{V}_{\boldsymbol{\lambda}}^*(s_0) + \boldsymbol{\lambda}^\top \hat{\mathbf{c}}\right] \tag{4}$$

Moreover, if there is an optimal solution y^* to the primal LP in Eq. (1), then there exists the corresponding dual optimal solution $\mathcal{V}^*$ and $\boldsymbol{\lambda}^*$, and the duality gap is zero, i.e.

$$V_R^*(s_0;\hat{\mathbf{c}}) = \sum_{s,a} R(s,a) y^*(s,a) = \mathcal{V}_{\boldsymbol{\lambda}^*}^*(s_0) + \boldsymbol{\lambda}^{*\top}\hat{\mathbf{c}}$$

by the strong duality theorem.

To compute the optimal $\boldsymbol{\lambda}$ in Eq. (4), we have to consider the trade-off between the first term and the second term according to the given cost constraint $\hat{\mathbf{c}}$. For example, if the cost constraint $\hat{\mathbf{c}}$ is very large, the optimal solution $\boldsymbol{\lambda}^*$ tends to be close to zero since the objective function would be mostly affected by the second term $\boldsymbol{\lambda}^\top \hat{\mathbf{c}}$. On the other hand, if $\hat{\mathbf{c}}$ is sufficiently small, the first term will be dominant and the optimal solution $\boldsymbol{\lambda}^*$ gets larger in order to have a negative impact on the reward $R(s,a) - \boldsymbol{\lambda}^\top \mathbf{C}(s,a)$. Thus, it may seem that Eq. (4) is a complex optimization problem. However, as we will see in the following proposition, the objective function in Eq.(4) is actually piecewise-linear and convex over $\boldsymbol{\lambda}$, as depicted in Figure 1.

Proposition 1. *Let $\mathcal{V}_{\boldsymbol{\lambda}}^*$ be the optimal value function of the MDP with the scalarized reward function $R(s,a) - \boldsymbol{\lambda}^\top \mathbf{C}(s,a)$. Then, $\mathcal{V}_{\boldsymbol{\lambda}}^*(s_0) + \boldsymbol{\lambda}^\top \hat{\mathbf{c}}$ is a piecewise-linear and convex (PWLC) function over $\boldsymbol{\lambda}$.*

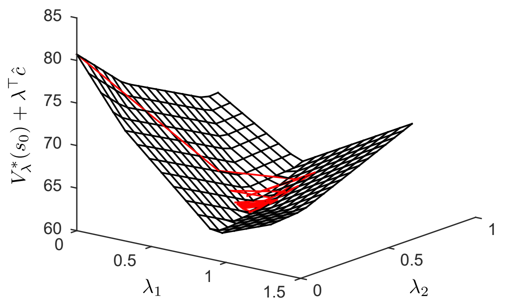


Figure 1: $\mathcal{V}^*_{\boldsymbol{\lambda}}(s_0) + \boldsymbol{\lambda}^\top \hat{\mathbf{c}}$ for a simple CMDP, which is piecewise-linear and convex. Red line represents the trajectory of $\boldsymbol{\lambda}$ starting from $\boldsymbol{\lambda} = [0,0]^\top$ and sequentially updated by $\boldsymbol{\lambda} \leftarrow \boldsymbol{\lambda} + \alpha_i(V^{\pi^*_{\boldsymbol{\lambda}}}_{\mathbf{C}}(s_0) - \hat{\mathbf{c}})$ as described in Algorithm 1.

Proof. We give a proof by induction. For all s,

$$\mathcal{V}^{(0)}_{\boldsymbol{\lambda}}(s) = \max_a \Big[R(s,a) - \boldsymbol{\lambda}^\top \mathbf{C}(s,a)\Big]$$

is a piecewise-linear and convex function over $\boldsymbol{\lambda}$ since the max of linear functions is piecewise linear and convex. Now, assume the following induction hypothesis: $\mathcal{V}^{(k)}_{\boldsymbol{\lambda}}(s)$ is piecewise-linear and convex function over $\boldsymbol{\lambda}$ for all s. Then,

$$\mathcal{V}^{(k+1)}_{\boldsymbol{\lambda}}(s) = \max_a \Big[\underbrace{R(s,a) - \boldsymbol{\lambda}^\top \mathbf{C}(s,a)}_{\text{linear in } \boldsymbol{\lambda}} + \gamma \sum_{s'} \underbrace{T(s'|s,a)\mathcal{V}^{(k)}_{\boldsymbol{\lambda}}(s')}_{\text{PWLC in } \boldsymbol{\lambda}}\Big]$$

is also PWLC since the summation of PWLC functions is PWLC and max over PWLC functions is again PWLC. As a consequence, $\mathcal{V}^*_{\boldsymbol{\lambda}}(s_0)$ is PWLC over $\boldsymbol{\lambda}$ and so is $\mathcal{V}^*_{\boldsymbol{\lambda}}(s_0) + \boldsymbol{\lambda}^\top \hat{\mathbf{c}}$. □

In addition, we can show that the optimal solution $\boldsymbol{\lambda}^*$ is bounded [Lee *et al.*, 2017].

Proposition 2 (Lemma 4 in [Lee *et al.*, 2017])**.** *Suppose that the reward function is bounded in $[0, R_{\max}]$ and there exists $\tau > 0$ and a (feasible) policy π such that $V^\pi_{\mathbf{C}}(s_0) + \tau \mathbf{1} \leq \hat{\mathbf{c}}$. Then, the following inequality holds:*

$$\|\boldsymbol{\lambda}^*\|_1 \leq \frac{R_{\max}}{\tau(1-\gamma)}$$

Thus, from Propositions 1 and 2, we now know that optimal $\boldsymbol{\lambda}^*$ can be obtained by greedily optimizing Eq. (4) with λ_k in the range $[0, \frac{R_{\max}}{\tau(1-\gamma)}]$. The remaining question is how to compute the direction for updating $\boldsymbol{\lambda}$. We start with the following lemma to answer this question.

Lemma 1. *Let $\mathcal{M}_1 = \langle S, A, T, R_1, \gamma\rangle$ and $\mathcal{M}_2 = \langle S, A, T, R_2, \gamma\rangle$ be two MDPs and V^π_1 and V^π_2 be the value functions of $\mathcal{M}_1$ and $\mathcal{M}_2$ with a fixed policy π. Then, the value function of the new MDP $\mathcal{M} = \langle S, A, T, pR_1 + qR_2, \gamma\rangle$ with the policy π is $V^\pi(s) = pV^\pi_1(s) + qV^\pi_2(s)$ for all $s \in S$.*

Proof. We give a proof by induction. For all s,

$$\begin{aligned} V^{(0)}(s) &= \sum_a \pi(a|s)\left[pR_1(s,a) + qR_2(s,a)\right] \\ &= p\sum_a \pi(a|s)R_1(s,a) + q\sum_a \pi(a|s)R_2(s,a) \\ &= pV^{(0)}_1(s) + qV^{(0)}_2(s) \end{aligned}$$

Then, assume the induction hypothesis $V^{(k)}(s) = pV^{(k)}_1(s) + qV^{(k)}_2(s)$. For all s,

$$\begin{aligned} &V^{(k+1)}(s) \\ &= \sum_a \pi(a|s)\Big[pR_1(s,a) + qR_2(s,a) + \gamma \sum_{s'} T(s'|s,a)V^{(k)}(s')\Big] \\ &= p\sum_a \pi(a|s)\Big[R_1(s,a) + \gamma\sum_{s'} T(s'|s,a)V^{(k)}_1(s')\Big] \\ &\quad + q\sum_a \pi(a|s)\Big[R_2(s,a) + \gamma\sum_{s'} T(s'|s,a)V^{(k)}_2(s')\Big] \\ &= pV^{(k+1)}_1(s) + qV^{(k+1)}_2(s) \end{aligned}$$ □

Lemma 1 implies that $\mathcal{V}^*_{\boldsymbol{\lambda}}$ can be decomposed into $\mathcal{V}^*_{\boldsymbol{\lambda}}(s_0) = V^{\pi^*_{\boldsymbol{\lambda}}}_R(s_0) - \boldsymbol{\lambda}^\top V^{\pi^*_{\boldsymbol{\lambda}}}_{\mathbf{C}}(s_0)$ where $\pi^*_{\boldsymbol{\lambda}}$ is the optimal policy with respect to the scalarized reward function $R(s,a) - \boldsymbol{\lambda}^\top \mathbf{C}(s,a)$, and thus Eq. (4) becomes:

$$\min_{\boldsymbol{\lambda} \geq \mathbf{0}} \Big[V^{\pi^*_{\boldsymbol{\lambda}}}_R(s_0) - \boldsymbol{\lambda}^\top V^{\pi^*_{\boldsymbol{\lambda}}}_{\mathbf{C}}(s_0) + \boldsymbol{\lambda}^\top \hat{\mathbf{c}}\Big] \tag{5}$$

A possible way to compute the descent direction for $\boldsymbol{\lambda}$ would be by taking the derivative of Eq. (5) with respect to $\boldsymbol{\lambda}$ while holding $\pi^*_{\boldsymbol{\lambda}}$ constant so that we use the direction

$$V^{\pi^*_{\boldsymbol{\lambda}}}_{\mathbf{C}}(s_0) - \hat{\mathbf{c}}. \tag{6}$$

The following theorem shows that this indeed is a valid direction.

Theorem 1. *For any $\boldsymbol{\lambda}$, $V^{\pi^*_{\boldsymbol{\lambda}}}_{\mathbf{C}}(s_0) - \hat{\mathbf{c}}$ is a negative subgradient that decreases the objective in Eq. (4), where $\pi^*_{\boldsymbol{\lambda}}$ is the optimal policy with respect to the scalarized reward function $R(s,a) - \boldsymbol{\lambda}^\top \mathbf{C}(s,a)$. Also, if $V^{\pi^*_{\boldsymbol{\lambda}}}_{\mathbf{C}}(s_0) - \hat{\mathbf{c}} = \mathbf{0}$ then $\boldsymbol{\lambda}$ is the optimal solution of Eq. (4).*

Proof. For any $\boldsymbol{\lambda}_0$ and $\boldsymbol{\lambda}_1$,

$$\begin{aligned} &(\mathcal{V}^*_{\boldsymbol{\lambda}_1}(s_0) + \boldsymbol{\lambda}_1^\top \hat{\mathbf{c}}) - (\mathcal{V}^*_{\boldsymbol{\lambda}_0}(s_0) + \boldsymbol{\lambda}_0^\top \hat{\mathbf{c}}) \\ =&(V^{\pi^*_{\boldsymbol{\lambda}_1}}_R(s_0) - \boldsymbol{\lambda}_1^\top V^{\pi^*_{\boldsymbol{\lambda}_1}}_{\mathbf{C}}(s_0)) - (V^{\pi^*_{\boldsymbol{\lambda}_0}}_R(s_0) - \boldsymbol{\lambda}_0^\top V^{\pi^*_{\boldsymbol{\lambda}_0}}_{\mathbf{C}}(s_0)) \\ &+ (\boldsymbol{\lambda}_1 - \boldsymbol{\lambda}_0)^\top \hat{\mathbf{c}} \qquad \text{(Lemma 1)} \\ \geq&(V^{\pi^*_{\boldsymbol{\lambda}_0}}_R(s_0) - \boldsymbol{\lambda}_1^\top V^{\pi^*_{\boldsymbol{\lambda}_0}}_{\mathbf{C}}(s_0)) - (V^{\pi^*_{\boldsymbol{\lambda}_0}}_R(s_0) - \boldsymbol{\lambda}_0^\top V^{\pi^*_{\boldsymbol{\lambda}_0}}_{\mathbf{C}}(s_0)) \\ &+ (\boldsymbol{\lambda}_1 - \boldsymbol{\lambda}_0)^\top \hat{\mathbf{c}} \qquad (\because \pi^*_{\boldsymbol{\lambda}_1} \text{ is optimal w.r.t. } R - \boldsymbol{\lambda}_1^\top \mathbf{C}) \\ =&(\boldsymbol{\lambda}_1 - \boldsymbol{\lambda}_0)^\top(\hat{\mathbf{c}} - V^{\pi^*_{\boldsymbol{\lambda}_0}}_{\mathbf{C}}(s_0)) \end{aligned} \tag{7}$$

Therefore, $\hat{\mathbf{c}} - V^{\pi^*_{\boldsymbol{\lambda}_0}}_{\mathbf{C}}(s_0)$ is a *subgradient* of $\mathcal{V}^*_{\boldsymbol{\lambda}}(s_0) + \boldsymbol{\lambda}^\top \hat{\mathbf{c}}$ at point $\boldsymbol{\lambda} = \boldsymbol{\lambda}_0$, which concludes that $V^{\pi^*_{\boldsymbol{\lambda}_0}}_{\mathbf{C}}(s_0) - \hat{\mathbf{c}}$ is a *negative subgradient* at point $\boldsymbol{\lambda} = \boldsymbol{\lambda}_0$.

Algorithm 1 CMDP dual solver via MDP solver

```
Input: CMDP $\mathcal{M}_C = \langle S, A, T, R, \mathbf{C}, \hat{\mathbf{c}}, \gamma, s_0 \rangle$
1: $\boldsymbol{\lambda}$ is initialized randomly.
2: $\pi$ is initialized randomly.
3: for $i = 1, 2, ...$ and $\boldsymbol{\lambda}$ has not converged do
4:     $\pi \leftarrow$ SolveMDP($\langle S, A, T, R - \boldsymbol{\lambda}^\top \mathbf{C}, \gamma \rangle$)
5:     $V_{\mathbf{C}} \leftarrow$ PolicyEvaluation($\langle S, A, T, \mathbf{C}, \gamma \rangle, \pi$)
6:     $\boldsymbol{\lambda} \leftarrow \boldsymbol{\lambda} + \alpha_i (V_{\mathbf{C}}(s_0) - \hat{\mathbf{c}})$
7:     Clip $\lambda_k$ to range $[0, \frac{R_{\max}}{\tau(1-\gamma)}]$ $\forall k = \{1, 2, ...K\}$
8: end for
Output: $\boldsymbol{\lambda}$: optimal solution of Eq. (4)
```

In addition, suppose $\boldsymbol{\lambda}_1 = \boldsymbol{\lambda}_0 + \alpha(V_{\mathbf{C}}^{\pi^*_{\boldsymbol{\lambda}_0}}(s_0) - \hat{\mathbf{c}})$. Then, for sufficiently small $\alpha > 0$, the new reward function $R - \boldsymbol{\lambda}_1^\top \mathbf{C}$ which is slightly changed from the old reward $R - \boldsymbol{\lambda}_0^\top \mathbf{C}$ still satisfies the *reward optimality condition* with respect to policy $\pi^*_{\boldsymbol{\lambda}_0}$ [Oh and Kim, 2011]. In this situation, $\pi^*_{\boldsymbol{\lambda}_0} = \pi^*_{\boldsymbol{\lambda}_1}$ and we obtain:

$$\begin{aligned}
&(\mathcal{V}^*_{\boldsymbol{\lambda}_0}(s_0) + \boldsymbol{\lambda}_0^\top \hat{\mathbf{c}}) - (\mathcal{V}^*_{\boldsymbol{\lambda}_1}(s_0) + \boldsymbol{\lambda}_1^\top \hat{\mathbf{c}}) \\
&\geq (\boldsymbol{\lambda}_0 - \boldsymbol{\lambda}_1)^\top (\hat{\mathbf{c}} - V_{\mathbf{C}}^{\pi^*_{\boldsymbol{\lambda}_1}}(s_0)) \qquad \big(\text{by result of (7)}\big) \\
&= -\alpha (V_{\mathbf{C}}^{\pi^*_{\boldsymbol{\lambda}_0}}(s_0) - \hat{\mathbf{c}})^\top (\hat{\mathbf{c}} - V_{\mathbf{C}}^{\pi^*_{\boldsymbol{\lambda}_1}}(s_0)) \\
&= \alpha \|\hat{\mathbf{c}} - V_{\mathbf{C}}^{\pi^*_{\boldsymbol{\lambda}_1}}(s_0))\|_2^2 \\
&\geq 0
\end{aligned}$$

Also, if $V_{\mathbf{C}}^{\pi^*_{\boldsymbol{\lambda}}}(s_0) - \hat{\mathbf{c}} = \mathbf{0}$, then the dual objective in Eq. (5) becomes $V_R^{\pi^*_{\boldsymbol{\lambda}}}(s_0)$. By the weak duality theorem, $V_R^{\pi^*_{\boldsymbol{\lambda}}}(s_0) \geq \sum_{s,a} R(s,a) y^*(s,a) = V_R^*(s_0; \hat{\mathbf{c}})$ holds where $y^*(s,a)$ is the optimal solution of the primal LP (1). Assuming that $V_{\mathbf{C}}^{\pi^*_{\boldsymbol{\lambda}}}(s_0) = \hat{\mathbf{c}}$, $\pi^*_{\boldsymbol{\lambda}}$ is a feasible policy that satisfies the cost constraints, and $V_R^{\pi^*_{\boldsymbol{\lambda}}}(s_0)$ cannot be larger than $V_R^*(s_0; \hat{\mathbf{c}})$. That is, $V_R^{\pi^*_{\boldsymbol{\lambda}}}(s_0) = V_R^*(s_0; \hat{\mathbf{c}})$ and the duality gap is zero, which means that $\boldsymbol{\lambda}$ is the optimal solution. □

We can interpret the direction $V_{\mathbf{C}}^{\pi^*_{\boldsymbol{\lambda}}}(s_0) - \hat{\mathbf{c}}$ as follows: if the current policy violates the k-th cost constraint (i.e. $V_{C_k}^{\pi^*_{\boldsymbol{\lambda}}} > \hat{c}_k$), λ_k increases so that incurring the cost is penalized more in the scalarized reward function $R(s,a) - \boldsymbol{\lambda}^\top \mathbf{C}(s,a)$. On the other hand, if the current policy is too conservative in the k-th cost constraint (i.e. $V_{C_k}^{\pi^*_{\boldsymbol{\lambda}}} < \hat{c}_k$), λ_k decreases so that the cost is less penalized.

Algorithm 1 summarizes our idea, which solves the dual of LP for CMDPs, iteratively solving MDPs with the scalarized reward $R - \boldsymbol{\lambda}^\top \mathbf{C}$ using any unconstrained MDP solver and updating $\boldsymbol{\lambda}$ to minimize the objective (4). By Theorem 1, the algorithm is a subgradient method, guaranteed to converge to the optimal solution by using a step-size sequence α_i such that $\sum_i \alpha_i = \infty$ and $\sum_i \alpha_i^2 < \infty$ as depicted in Figure 1.

4 Cost-Constrained UCT

Although Algorithm 1 avoids using an LP solver, it still relies on exactly solving MDPs via *SolveMDP* in each iteration, which is not practical for large CMDPs. Fortunately, we can further show that even when *SolveMDP* does not solve MDPs exactly, $\boldsymbol{\lambda}$ still converges to the optimal $\boldsymbol{\lambda}^*$ as long as it monotonically improves the policy. For example, suppose that we substitute Lines 4-5 in Algorithm 1 with

$$\begin{aligned}
[Q_R, Q_{\mathbf{C}}] &\leftarrow \text{PolicyEvaluation}(\langle S, A, T, [R, \mathbf{C}], \gamma \rangle, \pi) \\
\pi(s) &\leftarrow \arg\max_a \Big[Q_R(s,a) - \boldsymbol{\lambda}^\top Q_{\mathbf{C}}(s,a) \Big] \quad \forall s \\
V_{\mathbf{C}}(s_0) &\leftarrow Q_{\mathbf{C}}(s_0, \pi(s_0))
\end{aligned} \tag{8}$$

which performs a single policy improvement step with the current $\boldsymbol{\lambda}$. The following theorem supports that the optimal $\boldsymbol{\lambda}^*$ still can be obtained:

Theorem 2. *For any $\boldsymbol{\lambda}_0$, there exists $\epsilon > 0$ such that if $\|\boldsymbol{\lambda}_1 - \boldsymbol{\lambda}_0\|_1 < \epsilon$ then $\forall s$, $\mathcal{V}^{\pi_1}_{\boldsymbol{\lambda}_0}(s) > \mathcal{V}^{\pi_0}_{\boldsymbol{\lambda}_0}(s) \Rightarrow \forall s$, $\mathcal{V}^{\pi_1}_{\boldsymbol{\lambda}_1}(s) > \mathcal{V}^{\pi_0}_{\boldsymbol{\lambda}_1}(s)$. In other words, for a sufficiently small change of $\boldsymbol{\lambda}$, the ordering of the policies is preserved.*

Proof. Let $\Delta\boldsymbol{\lambda} = \boldsymbol{\lambda}_1 - \boldsymbol{\lambda}_0$. Then, for any s and π,

$$\begin{aligned}
\mathcal{V}^{\pi}_{\boldsymbol{\lambda}_1}(s) &= V_R^\pi(s) - \boldsymbol{\lambda}_1^\top V_{\mathbf{C}}^\pi(s) \qquad \text{(Lemma 1)} \\
&= V_R^\pi(s) - (\boldsymbol{\lambda}_0 + \Delta\boldsymbol{\lambda})^\top V_{\mathbf{C}}^\pi(s) \\
&= \mathcal{V}^{\pi}_{\boldsymbol{\lambda}_0}(s) - \Delta\boldsymbol{\lambda}^\top V_{\mathbf{C}}^\pi(s)
\end{aligned}$$

Now assume $\forall s$, $\mathcal{V}^{\pi_1}_{\boldsymbol{\lambda}_0}(s) > \mathcal{V}^{\pi_0}_{\boldsymbol{\lambda}_0}(s)$, let $\epsilon = \min_s \Big[\frac{1-\gamma}{C_{\max}}(\mathcal{V}^{\pi_1}_{\boldsymbol{\lambda}_0}(s) - \mathcal{V}^{\pi_0}_{\boldsymbol{\lambda}_0}(s))\Big] > 0$, and suppose $\|\Delta\boldsymbol{\lambda}\|_1 = \|\boldsymbol{\lambda}_1 - \boldsymbol{\lambda}_0\|_1 < \epsilon$. Then, for any s,

$$\begin{aligned}
&\mathcal{V}^{\pi_1}_{\boldsymbol{\lambda}_1}(s) - \mathcal{V}^{\pi_0}_{\boldsymbol{\lambda}_1}(s) \\
&= \mathcal{V}^{\pi_1}_{\boldsymbol{\lambda}_0}(s) - \mathcal{V}^{\pi_0}_{\boldsymbol{\lambda}_0}(s) - \Delta\boldsymbol{\lambda}^\top (V_{\mathbf{C}}^{\pi_1}(s) - V_{\mathbf{C}}^{\pi_0}(s)) \\
&\geq \mathcal{V}^{\pi_1}_{\boldsymbol{\lambda}_0}(s) - \mathcal{V}^{\pi_0}_{\boldsymbol{\lambda}_0}(s) - \|\Delta\boldsymbol{\lambda}\|_1 \|V_{\mathbf{C}}^{\pi_1}(s) - V_{\mathbf{C}}^{\pi_0}(s)\|_\infty \\
&> \mathcal{V}^{\pi_1}_{\boldsymbol{\lambda}_0}(s) - \mathcal{V}^{\pi_0}_{\boldsymbol{\lambda}_0}(s) - \epsilon \frac{C_{\max}}{1-\gamma} \\
&= \mathcal{V}^{\pi_1}_{\boldsymbol{\lambda}_0}(s) - \mathcal{V}^{\pi_0}_{\boldsymbol{\lambda}_0}(s) - \min_{s'} \big[(\mathcal{V}^{\pi_1}_{\boldsymbol{\lambda}_0}(s') - \mathcal{V}^{\pi_0}_{\boldsymbol{\lambda}_0}(s')) \big] \geq 0 \; \square
\end{aligned}$$

Based on Theorem 2, we can observe that even when Eq. (8) is used instead of Lines 4-5 in Algorithm 1, the algorithm still converges to the optimal solution according to the following argument: since the step-size α_i is strictly decreasing, we can eventually meet the condition $\|\Delta\boldsymbol{\lambda}\|_1 < \epsilon$, and the policy ordering will be preserved in the local region of the $\boldsymbol{\lambda}$. Thus, in that local region, the policy converges to the optimal policy $\pi^*_{\boldsymbol{\lambda}}$ via finite policy improvement steps.

We are now ready to present our online algorithm for large CMDPs, named Cost-Constrained UCT (CCUCT), shown in Algorithm 2. CCUCT is an extension of UCT with cost constraints and can be seen as an anytime approximation of Algorithm 1 using Eq. (8): the policy is sequentially evaluated via the averaged Monte-Carlo return

$$Q_R(s,a) \leftarrow Q_R(s,a) + \frac{R - Q_R(s,a)}{N(s,a)}$$

$$Q_{\mathbf{C}}(s,a) \leftarrow Q_{\mathbf{C}}(s,a) + \frac{\mathbf{C} - Q_{\mathbf{C}}(s,a)}{N(s,a)}$$

Algorithm 2 Cost-Constrained UCT (CCUCT)

```
function SEARCH($s_0$)
    $\lambda$ is randomly initialized.
    repeat
        SIMULATE($s_0$, 0)
        $\pi \leftarrow$ GREEDYPOLICY$(s_0, 0)$
        $a \sim \pi(\cdot|s_0)$
        $\lambda \leftarrow \lambda + \alpha_t \left[Q_C(s_0, a) - \hat{c}\right]$
        Clip $\lambda_k$ to range $[0, \frac{R_{\max}}{\tau(1-\gamma)}]$  $\forall k = \{1, 2, ...K\}$
    until TIMEOUT()
    return GREEDYPOLICY$(s_0, 0)$
end function

function ROLLOUT($s$, $d$)
    if $d =$ (maximum-depth) then
        return $[0, 0]$
    end if
    $a \sim \pi_{rollout}(\cdot|s)$ and $(s', r, c) \sim \mathcal{G}(s, a)$
    return $[r, c] + \gamma\cdot$ ROLLOUT($s'$, $d + 1$)
end function

function GREEDYPOLICY($s$, $\kappa$)
    Let $\mathcal{Q}^{+}_{\lambda}(s, a) = Q_R(s, a) - \lambda^{\top} Q_C(s, a) + \kappa\sqrt{\frac{\log N(s)}{N(s,a)}}$
    $A^* \leftarrow \{a^*_i \mid \mathcal{Q}^{+}_{\lambda}(s, a^*_i) \simeq \max_{a^*} \mathcal{Q}^{+}_{\lambda}(s, a^*)\}$
    Compute a policy $\pi(a^*_i|s) = w_i$ such that
    $\min_{w_i} \sum_{k=1}^{K} \lambda_k \left( \sum_{i: a^*_i \in A^*} w_i Q_{C_k}(s, a^*_i) - \hat{c}_k \right)^2$
    $s.t. \sum_{i: a^*_i \in A^*} w_i = 1$ and $w_i \geq 0$
    return $\pi$
end function

function SIMULATE($s$, $d$)
    if $d =$ (maximum-depth) then
        return $[0, 0]$
    end if
    if $(s, d) \notin T$ then
        $T(s, d) \leftarrow (N_{init}, V_{C,init})$
        $T(s, d, a) \leftarrow (N_{init}, Q_{R,init}, Q_{C,init})\ \forall a \in A$
        return ROLLOUT($s$, $d$)
    end if
    $\pi \leftarrow$ GREEDYPOLICY$(s, \kappa)$
    $a \sim \pi(\cdot|s)$ and $(s', r, c) \sim \mathcal{G}(s, a)$
    $[R, C] \leftarrow [r, c] + \gamma\cdot$ SIMULATE($s'$, $d + 1$)
    $N(s) \leftarrow N(s) + 1$
    $N(s, a) \leftarrow N(s, a) + 1$
    $V_C(s) \leftarrow V_C(s) + \frac{C - V_C(s)}{N(s)}$
    $Q_R(s, a) \leftarrow Q_R(s, a) + \frac{R - Q_R(s,a)}{N(s,a)}$
    $Q_C(s, a) \leftarrow Q_C(s, a) + \frac{C - Q_C(s,a)}{N(s,a)}$
    return $[R, C]$
end function

function MAINLOOP()
    $s \leftarrow$ (initial state)
    $\hat{c} \leftarrow$ (cost constraint)
    while $s$ is not terminal do
        $\pi \leftarrow$ SEARCH($s$)
        $a \sim \pi(\cdot|s)$ and $(s', r, c) \sim \mathcal{G}(s, a)$
        $\hat{c} \leftarrow V_C(s')$
        $s \leftarrow s'$
    end while
end function
```

and the policy is implicitly improved by the UCB1 action selection rule, which increases the scalarized value $Q_R(s, a) - \boldsymbol{\lambda}^{\top} Q_{\mathbf{C}}(s, a)$:

$$\arg\max_a \left[Q_R(s, a) - \boldsymbol{\lambda}^{\top} Q_{\mathbf{C}}(s, a) + \kappa\sqrt{\frac{\log N(s)}{N(s, a)}} \right] \quad (9)$$

Finally, $\boldsymbol{\lambda}$ is updated simultaneously using the current estimate of $V_{\mathbf{C}}(s_0) - \hat{\mathbf{c}}$, which is the descent direction of the convex objective function:

$$\boldsymbol{\lambda} \leftarrow \boldsymbol{\lambda} + \alpha_t(Q_{\mathbf{C}}(s_0, a) - \hat{\mathbf{c}}) \text{ where } a \sim \pi(\cdot|s_0) \quad (10)$$

The following theorem states that using this update with CCUCT is guaranteed to improve $\boldsymbol{\lambda}$, under some mild assumptions:

Theorem 3. *Under the following assumptions, the direction $V_C(s_0) - \hat{c}$ estimated by CCUCT becomes a descent direction of the objective in Eq.* (4) *within finitely large number of simulations T, i.e. $(V_{\mathbf{C}}(s_0) - \hat{\mathbf{c}})^{\top}(V_{\mathbf{C}}^{\pi^*_{\boldsymbol{\lambda}}}(s_0) - \hat{\mathbf{c}}) > 0$:*

1. *$\|\frac{\mathbf{C}_{\max}}{1-\gamma} + \hat{\mathbf{c}}\|_1 \sum_{t=t_0}^{t_0+T} \alpha_t \leq \epsilon$ where ϵ is defined in Theorem 2 with respect to the current $\boldsymbol{\lambda}$, t_0 is the current time step.*

2. $\left| \left(V_R^{\pi^*_{\boldsymbol{\lambda}}}(s_0) - \boldsymbol{\lambda}^{\top} V_{\mathbf{C}}^{\pi^*_{\boldsymbol{\lambda}}}(s_0)\right) - \left(V_R(s_0) - \boldsymbol{\lambda}^{\top} V_{\mathbf{C}}(s_0)\right) \right| = O\left(\frac{\log N(s)}{N(s)}\right)$ *(Asymptotic bias of UCT holds.)*

Proof. Without loss of generality, there exists a policy $\pi^*_{\boldsymbol{\lambda}}$ such that CCUCT's policy asymptotically approaches to that $\pi^*_{\boldsymbol{\lambda}}$ as long as Assumption 1 holds (i.e. the policy ordering is preserved). Since V_R and $\{V_{C_k}\}_{k=1}^{K}$ are evaluated by Monte-Carlo return of the identical CCUCT's policy, asymptotic convergence rate of $V_R(s_0)$ and $\{V_{C_k}(s_0)\}_{k=1}^{K}$ can be considered as same:

$$O\left(|V_R^{\pi^*_{\boldsymbol{\lambda}}}(s_0) - V_R(s_0)|\right) = O\left(|V_{C_k}^{\pi^*_{\boldsymbol{\lambda}}}(s_0) - V_{C_k}(s_0)|\right) \ \forall k$$

From this, Assumption 2 of

$$\begin{aligned} &\left| \left(V_R^{\pi^*_{\boldsymbol{\lambda}}}(s_0) - \boldsymbol{\lambda}^{\top} V_{\mathbf{C}}^{\pi^*_{\boldsymbol{\lambda}}}(s_0)\right) - \left(V_R(s_0) - \boldsymbol{\lambda}^{\top} V_{\mathbf{C}}(s_0)\right) \right| \\ &= \left| \left(V_R^{\pi^*_{\boldsymbol{\lambda}}}(s_0) - V_R(s_0)\right) - \boldsymbol{\lambda}^{\top}\left(V_{\mathbf{C}}^{\pi^*_{\boldsymbol{\lambda}}}(s_0) - V_{\mathbf{C}}(s_0)\right) \right| \\ &= O\left(\frac{\log N(s)}{N(s)}\right) \end{aligned}$$

implies that

$$|V_R^{\pi_{\boldsymbol{\lambda}}^*}(s_0) - V_R(s_0)| = O\left(\frac{\log N(s)}{N(s)}\right) \text{ and}$$

$$|V_{C_k}^{\pi_{\boldsymbol{\lambda}}^*}(s_0) - V_{C_k}(s_0)| = O\left(\frac{\log N(s)}{N(s)}\right) \;\; \forall k$$

, which means $\exists n_0 > 0,\ \exists M > 0$ such that $\forall N(s_0) \geq n_0$,

$$\|V_{\mathbf{C}}^{\pi_{\boldsymbol{\lambda}}^*}(s_0) - V_{\mathbf{C}}(s_0)\|_2 \leq M\frac{\log N(s_0)}{N(s_0)} \tag{11}$$

Let $T = N(s_0) - t_0$ be a large enough number so that satisfies:

$$\frac{\log(t_0+T)}{t_0+T} = \frac{\log N(s_0)}{N(s_0)} < \frac{\|V_{\mathbf{C}}^{\pi_{\boldsymbol{\lambda}}^*}(s_0) - \hat{\mathbf{c}}\|_2}{M} \tag{12}$$

Then, after the simulation time steps more than T, we get the result:

$$\begin{aligned}
&(V_{\mathbf{C}}(s_0) - \hat{\mathbf{c}})^\top (V_{\mathbf{C}}^{\pi_{\boldsymbol{\lambda}}^*}(s_0) - \hat{\mathbf{c}}) \\
&= (V_{\mathbf{C}}^{\pi_{\boldsymbol{\lambda}}^*}(s_0) - \hat{\mathbf{c}} + V_{\mathbf{C}}(s_0) - V_{\mathbf{C}}^{\pi_{\boldsymbol{\lambda}}^*}(s_0))^\top (V_{\mathbf{C}}^{\pi_{\boldsymbol{\lambda}}^*}(s_0) - \hat{\mathbf{c}}) \\
&= (V_{\mathbf{C}}^{\pi_{\boldsymbol{\lambda}}^*}(s_0) - \hat{\mathbf{c}})^\top (V_{\mathbf{C}}^{\pi_{\boldsymbol{\lambda}}^*}(s_0) - \hat{\mathbf{c}}) \\
&\quad + \left(V_{\mathbf{C}}(s_0) - V_{\mathbf{C}}^{\pi_{\boldsymbol{\lambda}}^*}(s_0)\right)^\top \left(V_{\mathbf{C}}^{\pi_{\boldsymbol{\lambda}}^*}(s_0) - \hat{\mathbf{c}}\right) \\
&\geq \|V_{\mathbf{C}}^{\pi_{\boldsymbol{\lambda}}^*}(s_0) - \hat{\mathbf{c}}\|_2^2 - \|V_{\mathbf{C}}(s_0) - V_{\mathbf{C}}^{\pi_{\boldsymbol{\lambda}}^*}(s_0)\|_2 \|V_{\mathbf{C}}^{\pi_{\boldsymbol{\lambda}}^*}(s_0) - \hat{\mathbf{c}}\|_2 \\
&\hfill (\because -\|a\|_2\|b\|_2 \leq a^\top b \leq \|a\|_2\|b\|_2) \\
&\geq \|V_{\mathbf{C}}^{\pi_{\boldsymbol{\lambda}}^*}(s_0) - \hat{\mathbf{c}}\|_2^2 - M\frac{\log N(s_0)}{N(s_0)} \|V_{\mathbf{C}}^{\pi_{\boldsymbol{\lambda}}^*}(s_0) - \hat{\mathbf{c}}\|_2 \\
&\hfill \big(\text{by } (11)\big) \\
&> \|V_{\mathbf{C}}^{\pi_{\boldsymbol{\lambda}}^*}(s_0) - \hat{\mathbf{c}}\|_2^2 - M\frac{\|V_{\mathbf{C}}^{\pi_{\boldsymbol{\lambda}}^*}(s_0) - \hat{\mathbf{c}}\|_2}{M} \|V_{\mathbf{C}}^{\pi_{\boldsymbol{\lambda}}^*}(s_0) - \hat{\mathbf{c}}\|_2 \\
&\hfill \big(\text{by } (12)\big) \\
&= 0.
\end{aligned}$$

□

Note that CCUCT does not require a model of the environment to solve the CMDP: all we need is a black-box simulator $\mathcal{G}$ of the CMDP, which generates a sample $(s', r, c) \sim \mathcal{G}(s, a)$ of the next state s', reward r, and cost c, given the current state s and action a. Thus CCUCT is scalable to very large CMDPs, which we demonstrate through experiments in the next section.

4.1 Admissible Cost

After the planner takes an action, the cost constraint threshold $\hat{c}$ must be updated at the next time step. We reformulate the notion of *admissible cost* [Piunovskiy and Mao, 2000], originally formulated for dynamic programming, for the update.

The admissible cost $\hat{\mathbf{c}}_{t+1}$ at time step $t+1$ denotes the expected total cost allowed to be incurred in future time steps $\{t+1, t+2, ...\}$ without violating the cost constraints. Under dynamic programming, the update is given by [Piunovskiy and Mao, 2000]:

$$\hat{\mathbf{c}}_{t+1} = \frac{\hat{\mathbf{c}}_t - \mathbb{E}[\mathbf{C}(s_t, a_t)|s_0, \pi]}{\gamma}$$

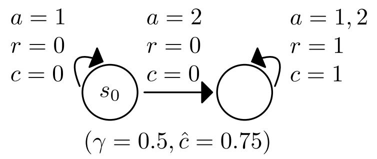


Figure 2: Constrained MDP synthetic domain. In this problem, the optimal policy is stochastic $\pi^*(a_1|s_0) = 0.4$ and $\pi^*(a_2|s_0) = 0.6$, and the optimal value is $V_R^*(s_0; \hat{c}) = V_C^*(s_0; \hat{c}) = 0.75$.

where evaluating $\mathbb{E}[\mathbf{C}(s_t, a_t)|s_0, \pi]$ requires the probability of reaching (s_t, a_t) at time step t, which in turn requires marginalizing out the past history $(a_0, s_1, a_2, ..., s_{t-1})$. This is intractable for large state spaces.

On the other hand, under forward search, the admissible cost at the next time step $t+1$ is simply:

$$\hat{\mathbf{c}}_{t+1} = V_{\mathbf{C}}^{\pi^*}(s_{t+1})$$

We can access $V_{\mathbf{C}}^{\pi^*}(s_{t+1})$ by starting from the root node of the search tree s_t, and sequentially following the action branch a_t and the next state branch s_{t+1}. Here we are assuming that the exact optimal $V_{\mathbf{C}}^{\pi^*}$ is obtained, which is certainly achievable after infinitely many simulations of CCUCT. Note also that even though $\hat{\mathbf{c}}_t > V_{\mathbf{C}}^{\pi^*}(s_t)$ is possible in general, assuming $\hat{\mathbf{c}}_t = V_{\mathbf{C}}^{\pi^*}(s_t)$ does not change the problem fundamentally. If $\hat{\mathbf{c}}_t < V_{\mathbf{C}}^{\pi^*}(s_t)$ then this means that no feasible policy exists.

4.2 Filling the Gap: Stochastic vs Deterministic Policies

Our approach works on the MDP with scalarized rewards, but care must be taken as the optimal policy of a CMDP is generally stochastic: given optimal $\boldsymbol{\lambda}^*$, let $\pi_{\boldsymbol{\lambda}}^*$ be the *deterministic* optimal policy for the MDP with the scalarized reward function $R - \boldsymbol{\lambda}^{*\top}\mathbf{C}$. Then, by the duality between the primal (1) and the dual (3),

$$\begin{aligned}
V_R^*(s_0; \hat{\mathbf{c}}) &= \mathcal{V}_\lambda^*(s_0) + \boldsymbol{\lambda}^{*\top}\hat{\mathbf{c}} \\
&= V_R^{\pi_{\boldsymbol{\lambda}}^*}(s_0) - \boldsymbol{\lambda}^{*\top}(V_{\mathbf{C}}^{\pi_{\boldsymbol{\lambda}}^*}(s_0) - \hat{\mathbf{c}})
\end{aligned}$$

This implies that if $\lambda_k^* > 0$ and $V_{C_k}^{\pi_{\boldsymbol{\lambda}}^*}(s_0) \neq c_k$ for some k then $\pi_{\boldsymbol{\lambda}}^*$ is not optimal for the original CMDP. This is exactly the situation where the optimal policy is stochastic. In order to make the policy computed by our algorithm stochastic, we make sure that the following *optimality condition* is satisfied, derived from $V_R^*(s_0; \hat{\mathbf{c}}) = V_R^{\pi_{\boldsymbol{\lambda}}^*}(s_0)$:

$$\begin{aligned}
&\sum_{k=1}^{K} \lambda_k^*(V_{C_k}^{\pi_{\boldsymbol{\lambda}}^*}(s_0) - \hat{c}_k) \\
&= \sum_{k=1}^{K} \lambda_k^* \left(\sum_a \pi_{\boldsymbol{\lambda}}^*(a|s_0) Q_{C_k}^{\pi_{\boldsymbol{\lambda}}^*}(s_0, a) - \hat{c}_k\right) = 0
\end{aligned} \tag{13}$$

That is, actions a_i^* with equally maximal scalarized action values $\mathcal{Q}_{\boldsymbol{\lambda}}(s, a_i^*) = Q_R(s, a_i^*) - \boldsymbol{\lambda}^\top Q_{\mathbf{C}}(s, a_i^*)$ participate as the support of the stochastic policy, and are selected with probability $\pi(a_i^*|s)$ that satisfies

$$\forall k : \lambda_k^* > 0, \sum_{a_i^*} \pi(a_i^*|s) Q_{C_k}(s, a_i^*) = \hat{c}_k$$

For example, consider the synthetic domain in Figure 2. In this domain, the optimal λ^* is 1, and $\mathcal{Q}_\lambda^*(s_0, a_1) = \mathcal{Q}_\lambda^*(s_0, a_2) = 0$. Since the two scalarized action values are maximal, they are the support of the optimal policy which is stochastic. The probability of executing these two actions can be obtained from Q_C, which amount to $Q_C^*(s_0, a_1) = 0.375$ and $Q_C^*(s_0, a_2) = 1$. Thus, action 1 is chosen with probability 0.4 and action 2 with probability 0.6 to meet the condition Eq. (13): $\sum_a \pi(a|s_0) Q_C^*(s_0, a) = \hat{c} = 0.75$.

GreedyPolicy in Algorithm 2 computes the stochastic policy according to the above principle. In practice, due to the randomness in Monte-Carlo sampling, action values in Eq. (13) are always subject to estimation error, so it is reformulated as a convex optimization problem:

$$\min_{w_i \geq 0} \sum_{k=1}^{K} \lambda_k \left(\sum_{i:a_i^* \in A^*} w_i Q_{C_k}(s, a_i^*) - \hat{c}_k \right)^2$$
$$s.t. \sum_{i:a_i^* \in A^*} w_i = 1$$

where $A^* = \{a_i^* \mid \mathcal{Q}_\lambda(s, a_i^*) \simeq \max_{a^*} \mathcal{Q}_\lambda(s, a^*)\}$ and the solutions are $w_i = \pi(a_i^*|s)$.

5 Experiment

In this section, we will empirically demonstrate that CCUCT converges to the optimal stochastic actions through the synthetic domain, and also show that it scales to a very large CMDP problem with 2^{1024} states.

Baseline Planner

To the best of our knowledge, this work is the first attempt to solve constrained MDP using Monte-Carlo Tree Search. Since there is no algorithm for direct performance comparison, we implemented a simple baseline planner. This planner works as outlined in section 2: it chooses an action via reward-maximizing UCT while preventing action branches that violate the cost constraint $Q_C(s, a) \leq \hat{c}$. If all action branches violate the cost constraints, the planner chooses an action uniformly at random.

5.1 Synthetic Domain

The synthetic domain described in Figure 2 is intended to demonstrate the convergence to the stochastic optimal actions. It consists of two states and two actions, where the initial state is denoted as s_0. If the action 1 is taken in the state s_0, the agent stays in the same state and receives reward 0 and cost 0. Otherwise, if action 2 is taken, the agent moves to the next state and starts to receive reward 1 and cost 1 continuously from the next time step. The discount factor and the cost constraint are set to $\gamma = 0.5$ and $\hat{c} = 0.75$ respectively.

In this domain, no deterministic policy can be optimal: if $\pi(s_0) = a_1$ then it receives no reward $V_R^\pi(s_0) = 0$ thus suboptimal. Otherwise ($\pi(s_0) = a_2$), the reward is maximized but the cost constraint is violated $V_C^\pi(s_0) = 1 > \hat{c}$. Thus the policy should be *stochastic*, and the optimal solution is $\pi^*(a_1|s_0) = 0.4$ and $\pi^*(s_2|s_0) = 0.6$. The corresponding optimal value functions are given as $V_R^{\pi^*}(s_0) = V_C^{\pi^*}(s_0) = 0.75$, $Q_R^{\pi^*}(s_0, a_1) = Q_C^{\pi^*}(s_0, a_1) = 0.375$, and $Q_R^{\pi^*}(s_0, a_2) = Q_C^{\pi^*}(s_0, a_2) = 1$ respectively. In addition, the optimal solution of the dual in Eq. (4) is $\lambda^* = 1$.

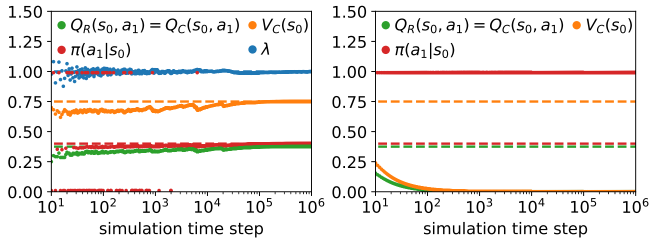


Figure 3: The result of synthetic domain (Left: CCUCT. Right: Baseline). The scattered dots represent the estimates of λ, $\pi(a_1|s_0)$, $V_C(s_0)$, and $Q_C(s_0, a_1)$ at each simulation time step, and the dotted horizontal lines are the optimal values for these estimates. As for the colors, blue corresponds to λ, red to stochastic/deterministic policy $\pi(a_1|s_0)$ returned by GreedyPolicy in Algorithm 2, orange to $V_C(s_0)$, and green to $Q_R(s_0, a_1) = Q_C(s_0, a_1)$ respectively.

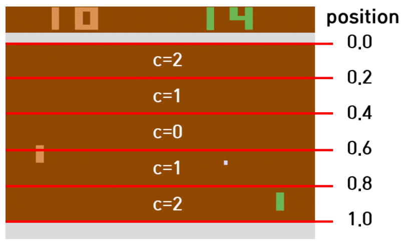


Figure 4: Multi-objective version of Atari 2600 Pong. In this modified domain, the cost is additionally incurred as well as the reward according to the position of the planner's paddle.

We use the following parameters for CCUCT: the exploration constant in Eq. (9) $\kappa = 1$, clipping constant $\tau = 0.75$, which is the same as $\hat{c}$, and the step-size sequence $\alpha_t = 10/t$. We run CCUCT with a root node s_0 for one million simulation iterations.

Figure 3 presents how the estimates computed by algorithms change over the simulations. At the early stage of the simulations of CCUCT, λ differs a lot from the optimum $\lambda^* = 1$, which makes the scalarized action values $\mathcal{Q}_\lambda(s_0, a) = Q_R(s_0, a) - \lambda Q_C(s_0, a)$ be very different for each action. As a consequence, GreedyPolicy returns a deterministic policy and $\pi(a_1|s_0)$ oscillates between 0 and 1. However, given enough time, we can see that all the estimates computed by CCUCT converge to their optima (i.e. $\lambda \to 1$, $V_C(s_0) \to 0.75$, and $Q_C(s_0, a_1) \to 0.375$), and so does the stochastic policy $\pi(a_1|s_0) \to 0.4$. Whereas the baseline planner always rejects cost-violating action a_2, thus the estimates computed by the planner converges to suboptimal values.

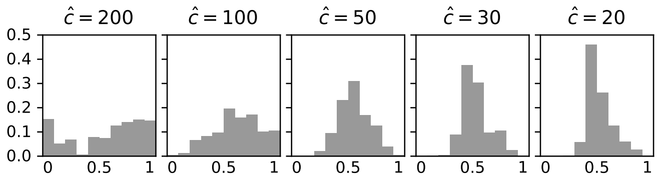
5.2 Multi-Objective Pong

We also conducted experiments on a multi-objective version of PONG, an arcade game running on the Arcade Learning Environment (ALE) [Bellemare *et al.*, 2013], depicted in Figure 4. In this domain, the left paddle is handled by the default computer opponent and the right paddle is controlled by our planner. We use the RAM state feature, i.e. the states are binary strings of length 1024 which results in $|S| = 2^{1024}$. The action space is $\{$up, down, stay$\}$. The agent receives a reward of $\{1, -1\}$ for each round depending on win/lose. The episode terminates if the accumulated reward is $\{21, -21\}$. We assigned cost 0 to the center area ($position \in [0.4, 0.6]$), 1 to the neighboring area ($position \in [0.2, 0.4] \cup [0.6, 0.8]$), and 2 to the area farthest away from the center ($position \in [0.0, 0.2] \cup [0.8, 1.0]$). This cost function can be seen as encouraging the agent to stay in the center. The discount factor is set to $\gamma = 0.99$. We tested with various cost constraint thresholds $\hat{c} \in \{200, 100, 50, 30, 20\}$ ranging from the unconstrained case $\hat{c} = 200$ ($\because \frac{C_{\max}}{1-\gamma} = 200$) to the tightly constrained case $\hat{c} = 20$.

We can see that the agent has two conflicting objectives: in order to achieve a high reward, it sometimes needs to move the paddle to positions far away from the center, but if this happens too often, the cost constraint will be violated. Thus, it needs to trade off between reward and cost properly depending on the cost constraint threshold $\hat{c}$.

We use the following parameters: maximum-depth 100, the number of simulations 1000, exploration constant $\kappa = 0.1$, τ equal to the cost constraint $\hat{c}$, and the step-size $\alpha_t = 1/t$. The results are averaged over 40 trials.

Figure 5 summarizes the experimental results for CCUCT and baseline algorithms with varying $\hat{c}$. When $\hat{c} = 200$ (unconstrained case), both algorithms always win the game 21 by 0. As we lower $\hat{c}$, CCUCT tends to stay in the center in order to make a trade off between reward and cost, as shown in the histograms of Figure 5. We can also see that the agent gradually performs worse in terms of scores as $\hat{c}$ decreases. This is a natural result since it is forced to stay in the center and thus sacrifice the game score. Overall, CCUCT computes a good policy while generally respecting the cost constraint. On the other hand, the baseline fails to show a meaningful policy except when $\hat{c} = 200$ since the Monte-Carlo cost returns at early stage mostly violates cost constraints, resulting in a random behavior.



$\hat{c}$	ALGO	avg cumulative rewards	avg discounted cumulative costs	avg score FOE vs ALGO
200	CCUCT	21.00±0.00	133.00±4.97	0.0 vs 21.0
	Baseline	21.00±0.00	136.66±4.45	0.0 vs 21.0
100	CCUCT	19.27±1.63	99.88±0.13	1.4 vs 20.7
	Baseline	−15.05±3.83	110.88±3.86	18.9 vs 3.9
50	CCUCT	17.88±1.79	49.95±0.07	2.8 vs 20.7
	Baseline	−20.45±0.26	130.40±4.99	21.0 vs 0.6
30	CCUCT	−0.07±5.23	30.40±0.46	13.2 vs 13.2
	Baseline	−20.48±0.30	131.37±5.08	21.0 vs 0.5
20	CCUCT	−17.80±2.91	25.36±1.25	20.1 vs 2.2
	Baseline	−20.48±0.30	131.37±5.08	21.0 vs 0.5

Figure 5: Results of the constrained PONG. Above: histogram of the CCUCT planner's position, where the horizontal axis denotes the position of the planner (0: topmost, 1: bottommost) and the vertical axis denotes the relative discounted visitation rate for each bin.

6 Conclusion and Future Work

We presented CCUCT, an online planning algorithm for large CMDPs. We showed that solving the dual LP of CMDPs was equivalent to jointly solving an unconstrained MDP and optimizing its LP-induced parameters $\boldsymbol{\lambda}$, and provided theoretical findings that gave insight into the properties of $\boldsymbol{\lambda}$ and how to optimize them. We then extended UCT to maximize the scalarized value while simultaneously updating $\boldsymbol{\lambda}$ using the current action-value estimates $Q_{\mathbf{C}}$. We also empirically confirmed that CCUCT converges to the optimal stochastic actions on a synthetic domain and easily scales to very large CMDPs through a multi-objective version of PONG.

As for future work, two research directions would be promising. First, a more formal regret analysis of CCUCT would strengthen the theoretical aspect of our work. Formulated as a bandit problem, CCUCT faces a non-stationary reward since the LP-induced parameter $\boldsymbol{\lambda}$ changes over time. However, we believe that CCUCT can be formally proven to converge to the optimal stochastic actions asymptotically, with the help of the convex and bounded properties of the objective function and the local preservation of the policy ordering. Second, extending CCUCT to *constrained* partially observable MDPs and *constrained* Bayesian reinforcement learning settings. In the similar manner UCT is used for partially observable MDPs [Silver and Veness, 2010] and Bayes-adaptive MDPs [Guez *et al.*, 2013], we believe that CCUCT can be used for the constrained versions of the corresponding problems.

Acknowledgments

This research was supported by the MSIT (Ministry of Science and ICT), Korea, under the ITRC (Information Technology Research Center) support program (IITP-2018-2016-0-00464) supervised by the IITP (Institute for Information & communications Technology Promotion)

References

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

[Archak *et al.*, 2012] Nikolay Archak, Vahab Mirrokni, and S. Muthukrishnan. Budget optimization for online campaigns with positive carryover effects. In *Proceedings of the 8th International Conference on Internet and Network Economics*, pages 86–99, 2012.

[Auer *et al.*, 2002] Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2):235–256, 2002.

[Bellemare *et al.*, 2013] Marc G. Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

[Browne *et al.*, 2012] Cameron Browne, Edward Powley, Daniel Whitehouse, Simon Lucas, Peter I. Cowling, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of Monte Carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4:1–49, 2012.

[Coulom, 2006] Rémi Coulom. Efficient selectivity and backup operators in Monte-Carlo tree search. In *Proceedings of the 5th International Conference on Computers and Games*, pages 72–83, 2006.

[Gelly and Silver, 2011] Sylvain Gelly and David Silver. Monte-Carlo tree search and rapid action value estimation in computer Go. *Artif. Intell.*, 175(11):1856–1875, 2011.

[Guez *et al.*, 2013] Arthur Guez, David Silver, and Peter Dayan. Scalable and efficient Bayes-adaptive reinforcement learning based on Monte-carlo tree search. *Journal of Artificial Intelligence Research*, pages 841–883, 2013.

[Kocsis and Szepesvári, 2006] Levente Kocsis and Csaba Szepesvári. Bandit based Monte-Carlo planning. In *Proceedings of the Seventeenth European Conference on Machine Learning (ECML 2006)*, pages 282–293, 2006.

[Kocsis *et al.*, 2006] Levente Kocsis, Csaba Szepesvári, and Jan Willemson. Improved Monte-Carlo Search. Technical Report 1, Univ. Tartu, Estonia, 2006.

[Lee *et al.*, 2017] Jongmin Lee, Youngsoo Jang, Pascal Poupart, and Kee-Eung Kim. Constrained Bayesian reinforcement learning via approximate linear programming. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence (IJCAI-17)*, pages 2088–2095, 2017.

[Liebana *et al.*, 2015] Diego Perez Liebana, Sanaz Mostaghim, Spyridon Samothrakis, and Simon M. Lucas. Multiobjective Monte Carlo tree search for real-time games. *IEEE Transactions on Computational Intelligence and AI in Games*, 7(4):347–360, 2015.

[Oh and Kim, 2011] Eunsoo Oh and Kee-Eung Kim. A geometric traversal algorithm for reward-uncertain MDPs. In *Proceedings of the Twenty-Seventh Conference on Uncertainty in Artificial Intelligence (UAI-11)*, pages 565–572, 2011.

[Piunovskiy and Mao, 2000] Alexei B Piunovskiy and Xuerong Mao. Constrained Markovian decision processes: the dynamic programming approach. *Operations research letters*, 27(3):119–126, 2000.

[Silver and Veness, 2010] David Silver and Joel Veness. Monte-Carlo planning in large POMDPs. In *Advances in Neural Information Processing Systems 23*, pages 2164–2172. 2010.

[Silver *et al.*, 2016] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, pages 484–489, 2016.

[Wang and Sebag, 2012] Weijia Wang and Michèle Sebag. Multi-objective Monte-Carlo tree search. In *Asian Conference on Machine Learning*, volume 25 of *JMLR: Workshop and Conference Proceedings*, pages 507–522, 2012.

[Zadorojniy *et al.*, 2009] Alexander Zadorojniy, Guy Even, and Adam Shwartz. A strongly polynomial algorithm for controlled queues. *Math. Oper. Res.*, 34(4):992–1007, 2009.

[Zitzler and Thiele, 1998] Eckart Zitzler and Lothar Thiele. Multiobjective optimization using evolutionary algorithms - a comparative case study. In *Proceedings of the 5th International Conference on Parallel Problem Solving from Nature*, pages 292–304, 1998.