Threshold UCT: Cost-Constrained Monte Carlo Tree Search with Pareto Curves

Martin Kurečka[1], **Václav Nevyhoštěný**[1], **Petr Novotný**[1], **Vít Unčovský**[1]

[1]Faculty of Informatics, Masaryk University
Botanická 68a, 612 00 Brno, Czech Republic
petr.novotny@fi.muni.cz

Abstract

Constrained Markov decision processes (CMDPs), in which the agent optimizes expected payoffs while keeping the expected cost below a given threshold, are the leading framework for safe sequential decision making under stochastic uncertainty. Among algorithms for planning and learning in CMDPs, methods based on Monte Carlo tree search (MCTS) have particular importance due to their efficiency and extendibility to more complex frameworks (such as partially observable settings and games). However, current MCTS-based methods for CMDPs either struggle with finding safe (i.e., constraint-satisfying) policies, or are too conservative and do not find valuable policies. We introduce Threshold UCT (T-UCT), an online MCTS-based algorithm for CMDP planning. Unlike previous MCTS-based CMDP planners, T-UCT explicitly estimates Pareto curves of cost-utility trade-offs throughout the search tree, using these together with a novel action selection and threshold update rules to seek safe and valuable policies. Our experiments demonstrate that our approach significantly outperforms state-of-the-art methods from the literature.

1 Introduction

Safe Decision Making and MCTS Monte-Carlo tree search (MCTS) has emerged as the de-facto method for solving large sequential decision making problems under uncertainty (Browne et al. 2012). It combines the scalability of sampling-based methods with the robustness of heuristic tree search, the latter feature making it easily extendable to settings with partial observability (Silver and Veness 2010), multiple agents (Silver et al. 2018), or settings with history-dependent optimal decisions (Chatterjee et al. 2018). While MCTS-based methods demonstrated remarkable efficiency in optimizing the agent's performance across diverse domains, the deployment of autonomous agents in real-world domains necessitates balancing the agent performance with the *safety* of their behavior. In AI planning and reinforcement learning, the standard way of modeling safety issues is via the *constrained decision-making* framework. Here, apart from the usual reward signals, the agents are also collecting *penalties* (or *costs*), and the objective is to maximize the expected accumulated reward under the constraint that the expected accumulated cost is below a given threshold Δ. Compared with ad-hoc reward shaping, the constrained approach provides explicit and domain-independent way of controlling agent safety. Hence, safe and efficient probabilistic planning (and indeed, also model-free reinforcement learning, where algorithms such as MuZero (Schrittwieser et al. 2020) are built on top of efficient MCTS planners) necessitates the development of stable and sample-efficient cost-constrained MCTS algorithms.

Key Components of Constrained MCTS An efficient constrained MCTS-based algorithm must be able to identify *safe* and *valuable* policies.

Finding safe policies (i.e., those that do not exceed the cost threshold) requires identifying "dangerous" (in terms of future cost) decisions and keeping track of *cost risk* accumulated in stochastic decisions: a $50/50$ gamble which incurs cost C if lost contributes at least $C/2$ towards the expected cost of the policy irrespective of the gamble's outcome.

An agent which never moves might be safe but never does anything useful. To identify reward-valuable policies among the safe ones, the algorithm must not be constrained beyond the requirements given by the threshold Δ and hence it must be able to reason about the *trade-off* between rewards and costs during both tree search and actual action selection.

Limitations of previous approaches Two prominent examples of constrained MCTS-based algorithms are CC-POMCP (Lee et al. 2018) and RAMCP (Chatterjee et al. 2018). While these algorithms represented significant steps towards practical constrained decision making, they exhibit fundamental limitations in identifying both safe and valuable policies.

Safety limitations: A usual way of tracking the cost risk is updating the current threshold Δ appropriately after each decision. As we discuss in Section 3, both CC-POMCP and RAMCP perform this update in an unsound manner and might thus produce policies violating the cost constraint even if a safe policy exists within the explored part of the tree.

Value limitations: Both CC-POMCP and RAMCP compute randomized policies, which are necessary for optimality in constrained decision-making scenarios (Altman 1999). However, their reasoning about the reward-cost payoff is incomplete. RAMCP does not use the cost information during

the tree search phase (which mimics the standard UCT (Kocsis and Szepesvári 2006)) at all: costs are only considered in the actual action selection phase, when a linear program (LP) encoding the constraints is solved over the constructed tree. Although the LP drives the agent to satisfy the constraints, the data used to construct the LP are sampled using a cost-agnostic search procedure which might lead to sub-optimal decisions. CC-POMCP, on the other hand, is a Lagrangian dual method in which the Lagrangian multiplier λ represents a concrete reward-cost tradeoff to be used in both tree search and action selection. The key limitation of the Lagrangian approach is the scalar nature of the tradeoff estimate λ: unless λ quickly converges to the optimal tradeoff, the algorithm will collect data according to either overly conservative or overly risky policies, yielding instability that hampers convergence to a valuable policy. This behavior was witnessed in multiple of our experiments (Section 4).

Our contributions We introduce *Threshold UCT (T-UCT)*, an online MCTS-based constrained decision-making algorithm designed so as to seek policies that are both safe and valuable. T-UCT achieves this by estimating the *Pareto curves* of the cost-payoff tradeoff in an online fashion. In every step, T-UCT uses these Pareto estimates to play a randomized action mixture optimal w.r.t. the current cost threshold. This is done both during the actual agent's action selection and during tree search. The latter phase resolves the exploration/exploitation tradeoff through a variant of the UCT (*upper confidence bound on trees* (Kocsis and Szepesvári 2006)) approach, adapted to the constrained setting. In particular, T-UCT's exploration is cost-sensitive, and the algorithm comes with a new threshold update rule, which ensures that the agent is not incorrectly driven into excessive risk. We evaluate T-UCT on benchmarks from the literature, including a model of an autonomous vehicle navigating the streets of Manhattan. Our experiments show that T-UCT significantly outperforms state-of-the-art algorithms, demonstrating notable sample efficiency and stable results across different environments.

Further related work The problem of constrained decision making under uncertainty, formalized via the notion of constrained Markov decision processes (Altman 1999) has received lot of attention in recent years, with approaches based on linear programming (Altman 1999; Poupart et al. 2015; Lee et al. 2017), heuristic search (Undurti and How 2010; Santana, Thiébaux, and Williams 2016), primal-dual optimization (Chow et al. 2017; Tessler, Mankowitz, and Mannor 2019), local policy search (Achiam et al. 2017), backward value functions (Satija, Amortila, and Pineau 2020), or Lyapunov functions (Chow et al. 2018). Unlike these works, our paper focuses on the MCTS-based approach to the problem, due to its scalability and extendibility to more complex domains.

The RAMCP algorithm has been extended into an AlphaZero-like MCTS-based learning algorithm RAlph in (Brázdil et al. 2020). In this paper, we do not consider function approximators and instead focus on the correctness and efficiency of the underlying tree search method. The recent learning algorithm ConstrainedZero (Moss et al. 2024) computes deterministic policies, considers chance constraints, and does not track the cost risk of stochastic decision, thus solving a problem different from ours.

Our work is also related to *multi-objective (MO)* planning (Barrett and Narayanan 2008; Moffaert and Nowé 2014). There, the task is to estimate tradeoffs among multiple payoff functions w.r.t. various solution concepts (including Pareto optimality). T-UCTs approach of performing full Bellman updates of Pareto curves during backpropagation is similar in spirit to *convex hull MCTS* (Painter, Lacerda, and Hawes 2020). However, MO approaches do not consider constrained optimization and thresholds; the main novelty of T-UCT is using the Pareto curves to guide the tree search towards valuable constraint-satisfying parts via threshold-based action selection and sound threshold updates.

Constrained decision making is also related to *risk-sensitive* planning and learning (e.g., (Chow and Ghavamzadeh 2014; Chow et al. 2015; L.A. and Fu 2022; Hou, Yeoh, and Varakantham 2016; Ayton and Williams 2018; Křetínský and Meggendorfer 2018)), where safety is enforced by putting a constraint on some *risk-measure* of the underlying policy. Some risk measures, such as *chance constraints,* can be expressed in our framework by encoding accumulated payoffs into states.

2 Preliminaries

We denote by $\mathcal{D}(X)$ the set of all probability distributions over a finite support X. We formalize the constrained decision making problem via the standard notion of constrained Markov decision processes (CMDPs).

Definition 1. *A* constrained Markov decision process (CMDP) *is a tuple* $\mathcal{C} = (\mathcal{S}, \mathcal{A}, \delta, r, c, s_0)$ *where:*

- $\mathcal{S}$ *is a finite set of* states,
- $\mathcal{A}$ *is a finite set of* actions,
- $\delta \colon \mathcal{S} \times \mathcal{A} \to \mathcal{D}(\mathcal{S})$ *is a probabilistic* transition function; *we abbreviate* $\delta(s,a)(t)$ *to* $\delta(t \mid s,a)$*,*
- $r \colon \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ *is a* reward function,
- $c \colon \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ *is a* cost function, *and*
- $s_0 \in \mathcal{S}$ *is the initial state*

CMDP dynamics CMDPs evolve identically to standard MDPs. A *history* is an element of $(\mathcal{S}\mathcal{A})^*\mathcal{S}$, i.e., a finite alternating sequence of states and actions starting and ending in a state. A *policy* is a function assigning to each history a distribution over actions.

Under a given policy π, a CMDP evolves as follows: we start in the initial state; i.e., the initial history is $h_0 = s_0$. Then, for every timestep $i \in \{0, 1, 2, \ldots\}$, given the current history $h_i = s_0 a_0 s_1 a_1 \ldots s_{i-1} a_{i-1} s_i$, the next action a_i is sampled from π: $a_i \sim \pi(h_i)$. The next state s_{i+1} is sampled according to the transition function, i.e, $s_{i+1} \sim \delta(s_i, a_i)$. Then, the agent obtains the reward $r(s_i, a_i, s_{i+1})$ and incurs the cost $c(s_i, a_i, s_{i+1})$. The current history is updated to $h_{i+1} = s_0 a_0 s_1 a_1 \ldots s_{i-1} a_{i-1} s_i a_i s_{i+1}$ and the process continues in the same fashion *ad infinitum.*

We denote by $\mathbb{P}^\pi(E)$ the probability of an event E under policy π, and by $\mathbb{E}^\pi[X]$ the expected value of a random

variable X under π. We denote by $|h|$ the *length* of history h, putting $|s_0a_0 \dots s_{i-1}a_{i-1}s_i| = i$.

We will sometimes abuse notation and use a history in a context where a state is expected - in such a case, the notation refers to the last state of a history. E.g. $\delta(- \mid h, a)$ denotes a transition probability distribution from the last state of h under action a.

Problem statement Under a fixed policy π, the agent accumulates (with possible discounting) both the rewards and costs over a finite *decision horizon* T:

$$Payoff_\pi = \mathbb{E}^\pi[\sum_{i=0}^{T-1} \gamma_r^i \cdot r(s_i, a_i, s_{i+1})],$$
$$Cost_\pi = \mathbb{E}^\pi[\sum_{i=0}^{T-1} \gamma_c^i \cdot c(s_i, a_i, s_{i+1})],$$

where $\gamma_r, \gamma_c \in (0, 1]$ are reward and cost *discount factors.*

Our goal is to maximize the accumulated payoff while keeping the accumulated cost below a given threshold. Formally, given a CMDP, the horizon $T \in \mathbb{N}$, discount factors $\gamma_r, \gamma_c \in (0, 1]$, and a *cost threshold* $\Delta \in \mathbb{R}_{\geq 0}$, our task is to solve the following constrained optimization problem:

$$\max_\pi Payoff_\pi$$
$$\text{subject to } Cost_\pi \leq \Delta.$$

Our algorithm tackles the above problem in an online fashion, producing a local approximation of the optimal constrained policy.

3 Threshold UCT

We propose a new algorithm for CMDP planning, *Threshold UCT (T-UCT)*. Like many other MCTS-based algorithms, T-UCT only requires access to a generative simulator of the underlying CMDP, i.e., an algorithm allowing for an efficient sampling from $\delta(-|s, a)$, given (s, a); and providing $r(s, a, s')$ and $c(s, a, s')$ for given (s, a, s').

History-action values, feasibility We consider payoffs achievable by a policy π after witnessing a history h and possibly also playing action a:

$$Payoff_\pi(h) = \mathbb{E}^\pi[\sum_{i=|h|}^{T-1} \gamma_r^{i-|h|} \cdot r(s_i, a_i, s_{i+1}) \mid h],$$
$$Payoff_\pi(h, a) = \mathbb{E}^\pi[\sum_{i=|h|}^{T-1} \gamma_r^{i-|h|} \cdot r(s_i, a_i, s_{i+1}) \mid h, a],$$

where $(\cdot|h)$ is a condition of producing history h in the first $|h|$ steps and $(\cdot|h, a)$ is a condition that a is played immediately after witnessing h. The quantities $Cost_\pi(h)$ and $Cost_\pi(h, a)$ are defined analogously. We say that π is Δ-feasible from h if $Cost_\pi(h) \leq \Delta$.

Achievable vectors A vector $(c, r) \in \mathbb{R} \times \mathbb{R}$ is *achievable* from history h if there exists a policy π such that $Cost_\pi(h) \leq c$ and $Payoff_\pi(h) \geq r$. Similarly, we say that (c, r) is achievable from (h, a) if $Cost_\pi(h, a) \leq c$ and $Payoff_\pi(h, a) \geq r$ for some π.

We write $(c', r') \preceq (c, r)$ if $c' \geq c$ and $r' \leq r$. A $\preceq$-closure of a set $X \subseteq \mathbb{R} \times \mathbb{R}$ is the set of all vectors $(c', r') \in \mathbb{R} \times \mathbb{R}$ s.t. $(c', r') \preceq (c, r)$ for some $(c, r) \in X$.

Pareto sets A *Pareto set* of history h is the set of all vectors achievable from h, while the Pareto set of (h, a) is the set of all vectors achievable from (h, a).

It is known (Chatterjee, Forejt, and Wojtczak 2013; Barrett and Narayanan 2008) that the Pareto sets are (i) convex (since we allow randomized policies), and (ii) $\preceq$-closed (i.e., if (c, r) belongs to the set and $(c', r') \preceq (c, r)$, then (c', r') also belongs to the set). From (ii) it follows that a Pareto set is wholly determined by its *Pareto curve*, i.e. the set of all points maximal w.r.t. the $\preceq$-ordering. Furthermore, in finite MDPs, the Pareto curve is piecewise linear, with finitely many pieces. The whole Pareto set can then be represented by a finite set of *vertices,* i.e., points in which the piecewise-linear curve changes its slope; indeed, the Pareto set is the $\preceq$-closure of the convex hull of the set of vertices. In what follows, we denote by $P(h)$ and $P(h, a)$ these finite representations of the Pareto sets of h and (h, a), respectively.

Bellman equations for Pareto sets Pareto sets in CMDPs obey local optimality equations akin to classical unconstrained MDPs. To formalize these, we need additional notation. The sum $X + Y$ is the standard Minkowski sum of the sets of vectors. For a vector (a, b) we define $X \cdot (a, b) = \{(x \cdot a, y \cdot b) \mid (x, y) \in X\}$, with $X \cdot a$ as a shorthand for $X \cdot (a, a)$. It is known (Barrett and Narayanan 2008; Chen et al. 2013) that for the finite-vertex representation of Pareto sets it holds:

$$P(h) = prune\Big(\bigcup_{a \in \mathcal{A}} P(h, a)\Big) \tag{1}$$
$$P(h, a) = prune\Big(\sum_{t \in \mathcal{S}} \delta(t|h, a)\big(P(hat) \cdot (\gamma_c, \gamma_r) + \{(c(h, a, t), r(h, a, t))\}\big)\Big), \tag{2}$$

where the $prune$ operator removes all points that are $\preceq$-dominated by a convex combination of some other points in the respective set.

T-UCT: Overall structure T-UCT is presented in Algorithm 1. It follows the standard Monte Carlo tree search (MCTS) framework, with blue lines highlighting parts that conceptually differ from the setting with unconstrained payoff optimization (the whole procedure `GetActionDist` is constraint-specific, and hence we omit its coloring). The algorithm iteratively builds a *search tree* whose nodes represent histories of the CMDP, with child nodes of h representing one-step extensions of h. Each node stores additional information, in particular the estimates of $P(h)$ and $P(h, a)$, denoted by $\mathcal{P}(h)$ and $\mathcal{P}(h, a)$ in the pseudocode. The tree structure is global to the whole algorithm and not explicitly pictured in the pseudocode.

The algorithm uses transition probabilities δ of the CMDP. If these are not available (e.g., when using a generative model of the CMDP), we replace δ with a sample

Algorithm 1: Threshold UCT

1 **Procedure** `ThresholdUCT`(T, Δ)
2 $h \leftarrow s_0$;
3 **while** $T > 0$ **do**
4 **repeat**
5 $leaf \leftarrow$ `GetLeaf`(h, Δ);
6 $newleaf \leftarrow$ `Expand`$(leaf)$;
7 `Propagate` $(newleaf, h)$;
8 **until** *timeout*;
9 $\sigma \leftarrow$ `GetActionDist`$(h, \Delta, 0)$;
10 $a \sim \sigma$; play a and observe new state t ;
11 $\Delta \leftarrow$ `UpdateThr`(Δ, σ, a, t);
12 $h \leftarrow hat$; $T \leftarrow T - 1$;
13 **Function** `GetLeaf`$(h, \tilde{\Delta})$
14 **while** h *is not a leaf* **do**
15 $\sigma \leftarrow$ `GetActionDist`$(h, \tilde{\Delta}, 1)$;
16 $a \sim \sigma$; $t \sim \hat{\delta}(- \mid h, a)$;
17 $\tilde{\Delta} \leftarrow$ `UpdateThr`$(\tilde{\Delta}, \sigma, a, t)$;
18 $h \leftarrow hat$;
19 **return** h
20 **Procedure** `Propagate`$(h, root)$
21 $(c, r) \leftarrow$ `Rollout`(h);
22 $\mathcal{P}(h) \leftarrow \{(c, r), (0, 0)\}$;
23 **while** $h \neq root$ **do**
24 write h as $h'as$; $h \leftarrow h'$;
25 update $\mathcal{P}(h, a)$ via equation (2);
26 update $\mathcal{P}(h)$ via equation (1);
27 **Function** `GetActionDist`(h, Δ, e)
28 $\widetilde{\mathcal{P}} \leftarrow$
$prune\big(\bigcup_{a' \in \mathcal{A}} \mathcal{P}(h, a') + \{expl(h, a') \cdot (-e, e)\}\big)$;
29 $\widetilde{C}^- \leftarrow \{c \mid (c, r) \in \widetilde{\mathcal{P}} \wedge c \leq \Delta\}$;
30 $\widetilde{C}^+ \leftarrow \{c \mid (c, r) \in \widetilde{\mathcal{P}} \wedge c \geq \Delta\}$;
31 **if** $\widetilde{C}^- = \emptyset$ **then**
32 $a \leftarrow \arg\min_{a'} \min\{c \mid (c, r) \in \widetilde{\mathcal{P}}(h, a')\}$;
33 **return** $det\_distr(a)$;
34 **else if** $\widetilde{C}^+ = \emptyset$ **then**
35 $a \leftarrow \arg\max_{a'} \max\{r \mid (c, r) \in \widetilde{\mathcal{P}}(h, a')\}$;
36 **return** $det\_distr(a)$;
37 **else**
38 $c_l \leftarrow \max \widetilde{C}^-$;
39 $a_l \leftarrow$ action realising c_l;
40 $c_h \leftarrow \min \widetilde{C}^+$;
41 $a_h \leftarrow$ action realising c_h;
42 $\sigma_h \leftarrow \frac{\Delta - c_l}{c_h - c_l}$; $\sigma_l \leftarrow 1 - \sigma_h$;
43 **return** $mix\_distr(\sigma_l, a_l, \sigma_h, a_h)$;

estimate based on the visit count of transitions during the tree search. The estimates are updated globally with every sample from the simulator (omitted in the pseudocode). In what follows, we denote by $\hat{\delta}$ either the real transition probabilities or, if these are unavailable, their sample estimates.

Initially, the tree contains a single node representing the history $h_0 = s_0$. In each decision step $1, 2, \ldots, T$, T-UCT iterates, from the current root h, the standard four stages of MCTS until the expiry of a given timeout. The stages are: (i) *search,* where the tree is traversed from top to bottom using information from previous iterations to navigate towards the most promising leaf (function `GetLeaf`); (ii) *leaf expansion,* where a successor of the selected leaf is added to the search tree (line 6); followed by (iii) *rollout:* where the Pareto curve of the new leaf is estimated, e.g., via Monte Carlo simulation following some default policy (lines 21–22), or possibly via a pre-trained predictor. Note that we also add a tuple $(0, 0)$ to the curve to make the exploration "cost-optimistic", even if the rollout policy is unable to find safe paths. Finally, T-UCT performs (iv) *backpropagation,* of the newly obtained information (particularly, the Pareto curve estimates) from the leaf back to the root (the rest of procedure `Propagate`). We provide more details on the individual stages below.

After this, T-UCT computes and outputs an action distribution σ from which action a to be played is sampled (lines 9–10). The action is performed, and a new state t of the environment (10) and the immediate reward and cost (omitted from pseudocode) are incurred. The cost threshold Δ is then updated (line 11, details below) to account for the cost incurred, discounting, and cost risk of the stochastic decision. The node hat becomes the new root of the tree and the process is repeated until a decision horizon is reached.

The key components distinguishing T-UCT from standard UCT are the backpropagation, action selection, and threshold update. In the following, we present these components in more detail.

Backpropagation T-UCT's backpropagation is similar to *convex-hull MCTS* (Painter, Lacerda, and Hawes 2020): performing full Bellman updates of the Pareto set estimates according to equations (1) and (2) (using estimates $\hat{\delta}$ in lieu of δ when necessary).

Action selection The function `GetActionDist` computes an action distribution based on the current estimates of Pareto curves. In the search phase of the algorithm (where e is set to 1), we encourage playing under-explored actions with an exploration bonus as depicted on line 28:

$$expl(h, a) = C \cdot \alpha(h) \cdot \sqrt{\frac{\log N(h)}{N(h, a) + 1}},$$

where $N(h)$ and $N(h, a)$ are the visit counts of node h and action a in h, respectively, C is a fixed static exploration constant, and $\alpha(h)$ is a dynamic adjustment of the exploration constant equal to the difference between the maximum and minimum achievable value (payoff or cost) in h. Note that for each $a \in \mathcal{A}$, the bonus is applied to all vertices in $\mathcal{P}(h, a)$; thereafter the exploration-augmented estimate $\widetilde{\mathcal{P}}$

of $P(h)$ is computed via (1). When playing the actual action, the exploration bonus is disabled by setting $e = 0$.

If there is no feasible policy based on $\widetilde{\mathcal{P}}$ (i.e., policy satisfying $Cost_\pi(h) \leq \Delta$), we return a distribution which deterministically selects the action minimizing the future expected cost (lines 31–33). Conversely, if the expected future cost can be kept below Δ no matter which action is played, we deterministically select the action with the highest future Δ-constrained payoff (lines 34–36).

Otherwise, we find vertices $v_l = (c_l, r_l)$ and $v_h = (c_h, r_h)$ of $\widetilde{\mathcal{P}}$ whose future cost estimates are the nearest to Δ from below and above, respectively. Due to pruning, necessarily $r_l \leq r_h$. We identify actions a_l and a_h that realize the cost-reward tradeoff represented by these vertices, and mix them in such a proportion that the resulting convex combination of c_l and c_h equals Δ, so that the "cost budget" is maxed out (lines 37–43). The resulting mixture is returned to the agent or the tree search procedure.

Threshold update After an action a is played and its outcome t observed, the cost threshold Δ must be updated to account for (a) the immediate cost incurred, (b) the discount factor γ_c, and (c) the predicted contribution of outcomes other than t to the overall *expected* cost achieved by the policy from h. The update is performed by the `UpdateThr` function, described below.

If the transition hat has not been expanded yet, the update involves only the subtraction of the immediate cost and the division by γ_c. Otherwise $\texttt{UpdateThr}(\Delta, \sigma, a, t)$ first computes the intermediate threshold Δ_{act}: If σ is deterministic, we set Δ_{act} to Δ; if σ is a stochastic mix of two different actions, we check if the action a sampled from σ is the a_l or a_h from line 43; accordingly, we set Δ_{act} to either c_l or c_h.

Based on Δ_{act}, the value of Δ is updated to a new value Δ' in a way depending on "how much" Δ_{act} is feasible from ha. In the following, we use $conv$ to denote the convex hull operator. There are three cases to consider, similar to those in `GetActionDist`:

Case "mixing": In the first case, there exists a maximal $\rho \in \mathbb{R}$ such that $(\Delta_{act}, \rho) \in conv(\mathcal{P}(h, a))$; i.e., by (2) ρ is the maximum real satisfying

$$(\Delta_{act}, \rho) = \sum_{s \in \mathcal{S}} \hat{\delta}(s \mid h, a) \cdot \big[(c_s \cdot \gamma_c, r_s \cdot \gamma_r) + (c(h, a, s), r(h, a, s))\big], \tag{3}$$

for some points (c_s, r_s) such that $(c_s, r_s) \in conv(\mathcal{P}(has))$ for each $s \in \mathcal{S}$. $\texttt{UpdateThr}(\Delta, \sigma, a, t)$ then updates Δ to

$$\Delta' \leftarrow c_t. \tag{4}$$

This ensures that no matter which t is sampled, the overall expected cost in ha is bounded by Δ_{act}, provided that the points in the Pareto curve estimates are achievable.

Case "surplus": The second case is when there is a surplus in the cost budget, i.e., $\Delta_{act} > c_{\max} = \max\{c \mid (c, r) \in \mathcal{P}(h, a)\}$. A naive update would be to ignore the surplus and continue as if Δ_{act} was $c_{\max}$. Per the Pareto curve estimates, there would be no decrease in payoff since indeed, under the estimated dynamic $\hat{\delta}$, the optimal policy does not exceed the cost $c_{\max}$. However, $\hat{\delta}$ can be incorrect and too cost-optimistic. Hence, ignoring the surplus $\Delta_{act} - c_{\max}$ could over-restrict the search in future steps. Instead, we distribute the surplus over all possible outcomes of a proportionally to the outcomes' predicted cost. Formally, $\texttt{UpdateThr}(\Delta, \sigma, a, t)$ identifies a vertex $(c_{\max}, \rho) \in \mathcal{P}(h, a)$ and computes vectors (c_s, r_s) satisfying (3) with left-hand side set to $(c_{\max}, \rho)$. Then, it computes

$$\Delta' \leftarrow c_t + (\Delta_{act} - c_{\max}) \frac{B - c_t}{\bar{c}(h, a) + \gamma_c B - c_{\max}}, \tag{5}$$

where $\bar{c}(h, a) = \sum_s \hat{\delta}(s \mid h, a) \cdot c(h, a, s)$ is the expected immediate cost, and $B = T \cdot \max_{(s,a,t)} c(s, a, t)$ is an upper bound on the accumulated cost of any trajectory.

Case "unfeasible": Finally, if Δ_{act} is unfeasible according to $\mathcal{P}(h, a)$, i.e., when $\Delta_{act} < c_{\min} = \min\{c \mid (c, r) \in \mathcal{P}(h, a)\}$, $\texttt{UpdateThr}(\Delta, \sigma, a, t)$ distributes all the missing cost to the current outcome t (unlike in the previous case). Formally, the update is the following:

$$\Delta' \leftarrow c_t - \frac{c_{\min} - \Delta_{act}}{\hat{\delta}(t \mid h, a)\gamma_c}, \tag{6}$$

where the c_s-values are, again, computed by applying (3) to a vector $(c_{\min}, \rho) \in \mathcal{P}(h, a)$.

Theoretical analysis The threshold update function `UpdateThr` enjoys two notable properties that are important for the algorithm's correctness: First a), the update never increases the estimated expected threshold, i.e.,

$$\Delta \geq \mathbb{E}\left[c(h, a, t) + \gamma_c \cdot \Delta'\right] [1], \tag{7}$$

and b), the updated value is feasible according to the estimates, i.e.,

$$\Delta' \geq c_t, \tag{8}$$

whenever Δ is feasible according to the estimates.

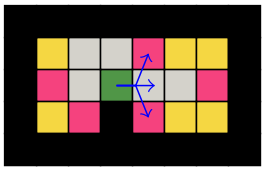
The properties are important for two reasons. First, they ensure the asymptotical convergence of the algorithm, and second, they prevent the algorithm from an excessive increase of the threshold under limited exploration.

Concerning the asymptotical convergence, we prove the ε-soundness of T-UCT in the sense formalized by the following theorem, proved in the extended version (Kurečka et al. 2024).

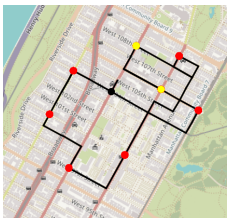
Theorem 1. *Let $\mathcal{C}$ be a CMDP and Δ a threshold such that there exists a Δ-feasible policy. Then for every $\varepsilon \in \mathbb{R}^+$, there exists n such that T-UCT with n MCTS iterations per action selection is $(\Delta + \varepsilon)$-feasible.*

We explore the importance of the properties (7) and (8) in further sections. Our experiments reveal that in certain cases, RAMCP violates the inequality (7). This leads to an excessive increase of the threshold and results in RAMCP ignoring the cost constraint. In the extended version (Kurečka et al. 2024), we further show that CC-POMCP can violate (8) by not taking the action outcomes into account during threshold updates, yielding a threshold-violating policy.

[1] The expectation is taken under the distribution $\hat{\delta}$.



(a) A Gridworld map with the initial tile (green), golds (yellow), and traps (red). The possible outcomes of moving right are depicted by blue arrows.



(b) Manhattan map depicting the initial junction and the agent's trajectory (black), maintenance points (red) and accepted requests (yellow).

Figure 1: The Gridworld and Manhattan environments.

4 Experiments

Baseline Algorithms We compare T-UCT to two state-of-the-art methods for solving CMDPs: *CC-POMCP* (Lee et al. 2018) and *RAMCP* (Chatterjee et al. 2018).

CC-POMCP is a dual method based on solving the Lagrangian relaxation of the CMDP objective:

$$\min_{\lambda \geq 0} \max_{\pi} Payoff_{\pi} + \lambda \cdot (\Delta - Cost_{\pi}). \tag{9}$$

CC-POMCP performs stochastic gradient descent on λ while continuously evaluating (9) via the standard UCT search. For a fixed λ, the maximization of (9) yields a point on the cost-payoff Pareto frontier, where a larger value of λ induces more cost-averse behavior. A caveat of the method is its sample inefficiency when λ converges slowly.

RAMCP is a primal method combining MCTS with linear programming. The search phase of RAMCP greedily optimizes the payoff in a standard single-valued UCT fashion, completely ignoring the constraint. Consequently, the cost is considered only in the second part of the action selection phase, where the algorithm solves a linear program to find a valid probability flow through the sampled tree (which serves as a local approximation of the original CMDP) such that the expected cost under the probability flow satisfies the constraint and the payoff is maximized.

Benchmarks We evaluated the algorithms on three tasks: two of them are variants of the established *Gridworld* benchmark (Brázdil et al. 2020; Undurti and How 2010); the third is based on the *Manhattan* environment (Blahoudek et al. 2020), where the agent navigates through mid-town Manhattan, avoiding traffic jams captured by a stochastic model.

Task: Avoid The setup (see Figure 1a) involves navigating the agent (robot) through a grid-based maze with four permissible actions: moving left, right, down, or up. The agent's movements are subject to stochastic perturbations: it can be displaced in a direction perpendicular to its intended movement with probability p_{slide}. The agent's objective is to visit special "gold" tiles to receive a reward while avoiding "trap" tiles, which, with probability p_{trap}, incur a cost of 1 and terminate the environment (the agent suffers fatal damage).

The task is evaluated on a set of 128 small maps (6×6 grid, 5 gold, approx. 10^3 states) and 64 large maps (25×25 grid, 50 gold, approx. 10^{17} states). All maps are sampled from our map generator, which guarantees a varying distribution of traps, walls, and gold. We provide the generator and the resulting map datasets *GridworldSmall* and *GridworldLarge* in the the project repository.

Task: SoftAvoid The setup is similar to *Avoid* (including the same generated maps) except that fixed amount of the cost p_{trap} is deterministically incurred upon stepping on a trap and the environment does not terminate in that case. Thus the goal is to collect as much gold as possible while keeping the number of visited trap tiles low.

Task: Manhattan The setup involves navigation in the eponymous *Manhattan* environment implemented in (Blahoudek et al. 2020). The agent moves through mid-town Manhattan (250 junctions, 8 maintenance points, approx. 10^{21} states) while the time required to pass each street is sampled from a distribution estimated from real-life traffic data. Periodically, selected points on the map request maintenance. If the request is accepted, the agent receives a reward of 1 when reaching the point of the request; however, if it does not deliver the order in the given time limit, it incurs a cost of 0.1.

Task	Reward r	Cost c
Avoid	1	1
SoftAvoid	1	p_{trap}
Manhattan	1	0.1

Table 1: Reward and cost summary. In all environments, the agent receives rewards deterministically either for collecting gold or finishing the order. In *SoftAvoid* and *Manhattan*, the agent incurs the cost deterministically on the trigger (stepping on a trap or not delivering the order in time), while in *Avoid*, the cost is incurred with probability p_{trap}.

Implementation We implemented T-UCT, RAMCP, and CC-POMCP algorithms within a common C++ based MCTS framework, so as to maximize the amount of shared code among the algorithms, reducing the influence of implementation details on the experiments. The *Gridworld* environments are implemented as part of our C++ codebase, while the *Manhattan* environment is built on top of the Python implementation provided by (Blahoudek et al. 2021). The code is available at https://github.com/kurecka/rats/tree/AAAI25.

Experimental setup Each task was evaluated with various settings of parameters such as the risk threshold Δ, map, or slide probability. We evaluated *GridworldSmall* tasks (both *Avoid* and *Soft-Avoid*) on 1144, *GridworldLarge* on 2304, and *Manhattan* on 600 configurations. The full description of configurations and of the hardware setup is in the extended version (Kurečka et al. 2024).

We performed 300 independent runs on each configuration. The algorithms were given a fixed time per decision summarized in Figure 2; note the time limit on *Manhattan*

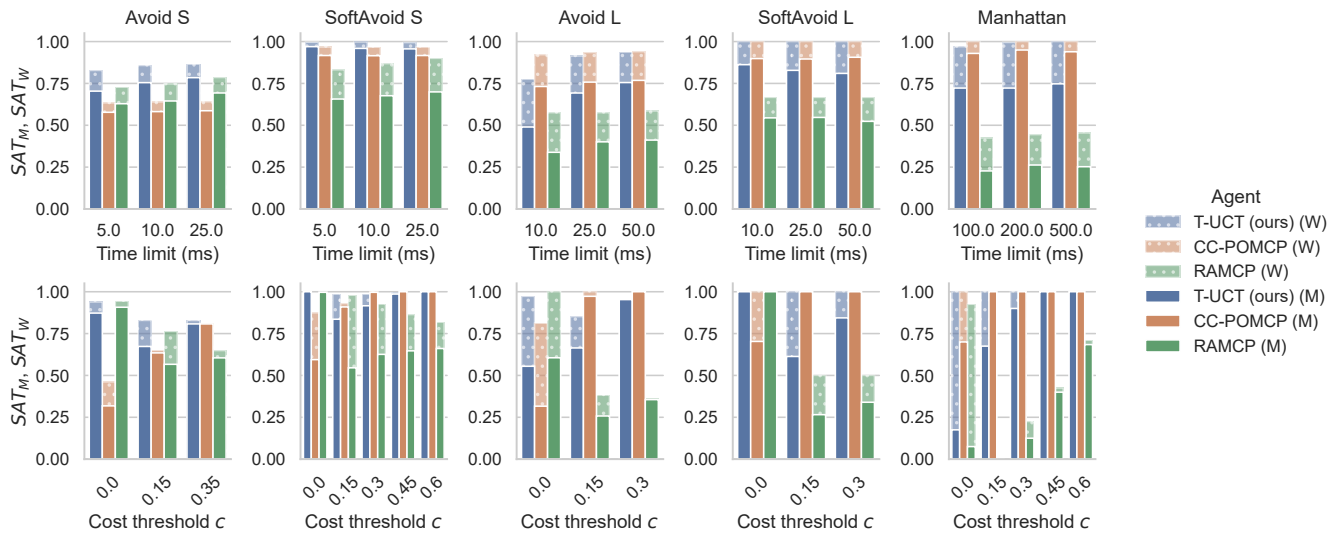


Figure 2: The fractions of satisfied configurations in the *mean* (plain) and the *weak* (dotted) sense. Upper row: The overall fraction of satisfied instances across varied time limits. Lower row: Breakdown of the fractions of satisfied instances across varied thresholds with the time limit set to the maximum value considered (25, 50, or 500 ms, respectively).

is looser so as to compensate for the slower Python implementation of the environment. The runs on *GridworldSmall*-based tasks had $T = 100$ steps, while the runs on *GridworldLarge* and *Manhattan* maps had $T = 200$ steps.

Metrics For each experiment configuration, we computed the mean collected payoff $\hat{r}$, mean collected cost $\hat{c}$, and its standard deviation. Based on these, we report two statistics detailed below: the fraction of satisfied instances SAT and the mean payoff achieved on satisfied instances.

Since the empirical cost $\hat{c}$ suffers from stochastic inaccuracy, we define two levels of satisfaction. The *mean satisfaction* metric SAT_M is simply the fraction of instances where the empirical cost satisfies the constraint, i.e., $\hat{c} \leq \Delta$. For the *weak satisfaction* SAT_W, we weaken the constraint, which allows us to give it a statistical significance; it is the fraction of instances where the t-test ($\alpha = 0.05$) rejects the hypothesis that the real expected cost of the algorithm is more than $\Delta + 0.05$. Since it is meaningless to compare the payoffs of the algorithms on instances where they violate the constraint, for each baseline algorithm, we further identify the instances where the algorithm and T-UCT both satisfy the constraint (in the weak sense) and compute the average payoff on these instances.

Results The safety results are shown in Figure 2. The first row shows the overall safety on individual environments. Each sub-plot displays the fraction of instances satisfied under the given time limits. Both T-UCT and CC-POMCP consistently solve a substantial proportion of instances, irrespective of the environment, while RAMCP struggles to find feasible solutions, especially in the *Manhattan* environment.

The second row of Figure 2 contains a breakdown of solved instances across different thresholds. All algorithms have similar satisfaction ratios when no cost is permitted (threshold 0); However, RAMCP quickly deteriorates with the increasing threshold as it is not able to estimate the expected cost accurately (only its positivity). T-UCT and CC-POMCP keep the number of satisfied instances or even improve it with more risk permitted.

Figure 3 compares the *payoffs* collected by T-UCT to other two algorithms, and only considers the instances satisfied by both of the compared algorithms. From the first row, it is clear that CC-POMCP is overly pessimistic in its actions, resulting in overconservative behaviour. On the other hand, due to its reward-oriented exploration, RAMCP often finds valuable strategies once it finds feasible ones. Nevertheless, T-UCT is still able to achieve similar or higher payoffs than RAMCP on all environments.

At the same time, T-UCT satisfied many more instances than RAMCP. The difference is especially notable on the large environments with non-zero cost thresholds where RAMCP satisfies at most half of the instances while T-UCT steadily achieves between 0.8 and 1.0 satisfaction rate. Regarding their performance in terms of payoffs, there is little statistically significant difference, although T-UCT achieved notably higher payoffs on large *SoftAvoid* instances.

On the small benchmarks, CC-POMCP both solved fewer instances and achieved lower payoffs than T-UCT. With larger environments, CC-POMCP's safety performance is on par with T-UCT or better. However, as we can see from the payoff values, this is mainly caused by the fact that CC-POMCP played extremely conservatively. The difference is most prominent in the *Manhattan* environment, where we observe up to a tenfold gap in payoff between CC-POMCP and T-UCT despite CC-POMCP achieving superior SAT values. This is caused by CC-POMCP refusing almost all of the maintenance requests, thus failing to find a valuable policy.

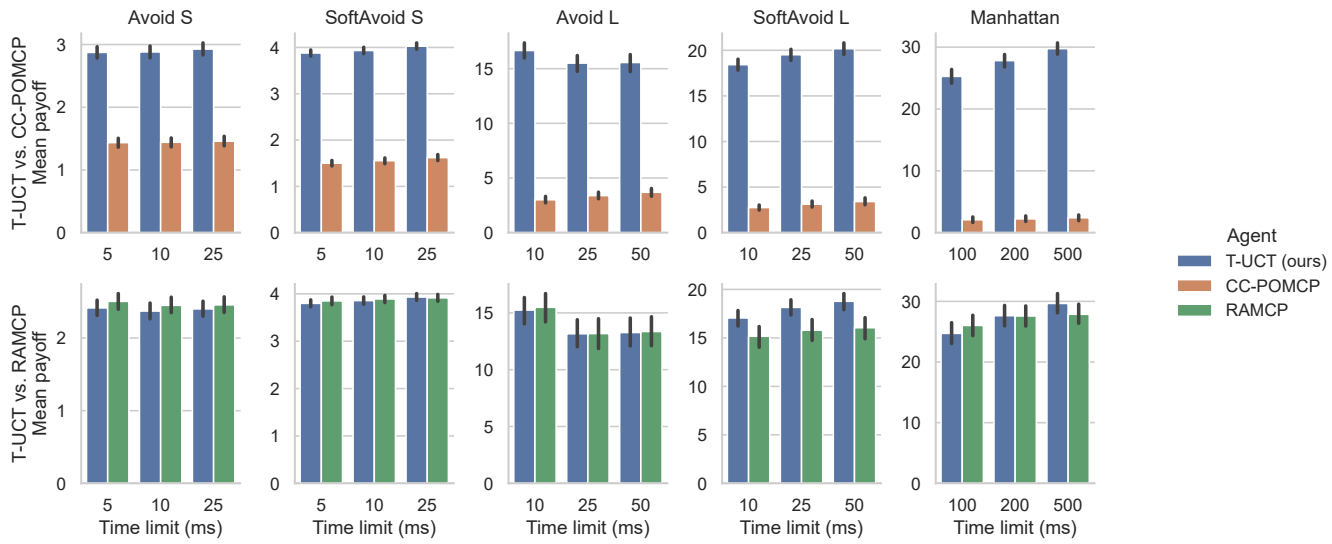


Figure 3: The mean payoff of T-UCT compared to each baseline. The average is calculated only over instances satisfied (in the weak sense) by both of the considered algorithms.

Notably, T-UCT was the only algorithm capable of solving a substantial number of *Manhattan* instances (approx. 74% SAT_M, 100% SAT_W) while achieving high payoffs. As all the algorithms performed at most approx. 500 MCTS samples per step (an order of magnitude less than in other benchmarks, see Table 2), this demonstrates T-UCT's data efficiency.

In summary, while CC-POMCP manages to find safe trajectories, it does so at the expense of the collected rewards. On the other hand, RAMCP generally accumulates a large reward, although it frequently disregards the safety of its behaviour in the process.

Environment	t	T-UCT (ours)	CC-POMCP	RAMCP
Avoid S	10	324	954	2580
SoftAvoid S	10	268	874	1920
Avoid L	50	1017	2634	1378
SoftAvoid L	50	1041	2587	1458
Manhattan	500	387	349	427

Table 2: Average number of simulations per step performed by each algorithm. The low number of simulations by T-UCT is due to its higher computational cost during backpropagation. The drop in RAMCP's simulations for larger environments is caused by its two-phase nature.

Discussion Per our observation, the suboptimal performance of CC-POMCP is caused by two factors: the unsound update rule, which occasionally sets Δ to a value that is not achievable; and the slow convergence of the Lagrange multiplier λ. For the latter reason, the algorithm often achieves low payoffs, even if the computed policies are safe.

RAMCP is often able to find relatively valuable strategies once it finds feasible ones, the latter task being its weak point. The problem of greedy exploration is pronounced in the *SoftAvoid* task, where rewards and costs are not directly correlated, and thus, the sampled tree does not provide a good approximation of the original CMDP. In *Manhattan,* where it is easy to underestimate the costs, the cost budget of RAMCP often explodes as described at the end of Section 3, which essentially leads to ignoring the constraint.

In summary, T-UCT strikes a good balance between both safety and payoff, a property not shared by either of the baseline algorithms. Although T-UCT performs significantly fewer simulations than the other algorithms, it provides stable performance across all environments. Using Pareto curves, T-UCT stores the collected information in a well-structured manner, allowing it to effectively reason about the cost-reward trade-offs even during exploration and thus find safe and valuable policies.

5 Conclusion and Future Work

We introduced Threshold UTC, a MCTS-based planner for constrained MDPs utilizing online estimates of reward-cost trade-offs in search for safe and valuable policies. We presented an experimental evaluation of our approach, showing that it outperforms competing cost-constrained tree-search algorithms. Our aim for future work is to augment T-UCT with function approximators of Pareto sets, thus obtaining a general MCTS-based *reinforcement learning* algorithm for cost-constrained optimization.

Acknowledgements

This research was supported by the Czech Science Foundation grant no. GA23-06963S.

References

Achiam, J.; Held, D.; Tamar, A.; and Abbeel, P. 2017. Constrained Policy Optimization. In Precup, D.; and Teh, Y. W., eds., *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, 22–31. PMLR.

Altman, E. 1999. *Constrained Markov Decision Processes*. Chapman&Hall/CRC. ISBN 9780849303821.

Ayton, B. J.; and Williams, B. C. 2018. Vulcan: A Monte Carlo Algorithm for Large Chance Constrained MDPs with Risk Bounding Functions. *CoRR*, abs/1809.01220.

Barrett, L.; and Narayanan, S. 2008. Learning All Optimal Policies with Multiple Criteria. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, 41–47. New York, NY, USA: Association for Computing Machinery. ISBN 9781605582054.

Blahoudek, F.; Brázdil, T.; Novotný, P.; Ornik, M.; Thangeda, P.; and Topcu, U. 2020. Qualitative Controller Synthesis for Consumption Markov Decision Processes.

Blahoudek, F.; Cubuktepe, M.; Novotný, P.; Ornik, M.; Thangeda, P.; and Topcu, U. 2021. Fuel in Markov Decision Processes (FiMDP): A Practical Approach to Consumption. In *Formal Methods: 24th International Symposium, FM 2021, November 20–26, 2021, Proceedings*, 640–656. Berlin, Heidelberg: Springer-Verlag. ISBN 978-3-030-90869-0.

Brázdil, T.; Chatterjee, K.; Novotný, P.; and Vahala, J. 2020. Reinforcement Learning of Risk-Constrained Policies in Markov Decision Processes. 9794–9801.

Browne, C. B.; Powley, E.; Whitehouse, D.; Lucas, S. M.; Cowling, P. I.; Rohlfshagen, P.; Tavener, S.; Perez, D.; Samothrakis, S.; and Colton, S. 2012. A Survey of Monte Carlo Tree Search Methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1): 1–43.

Chatterjee, K.; Elgyütt, A.; Novotný, P.; and Rouillé, O. 2018. Expectation Optimization with Probabilistic Guarantees in POMDPs with Discounted-Sum Objectives. 4692–4699.

Chatterjee, K.; Forejt, V.; and Wojtczak, D. 2013. Multi-objective Discounted Reward Verification in Graphs and MDPs. In McMillan, K.; Middeldorp, A.; and Voronkov, A., eds., *Logic for Programming, Artificial Intelligence, and Reasoning*, 228–242. Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN 978-3-642-45221-5.

Chen, T.; Forejt, V.; Kwiatkowska, M.; Simaitis, A.; and Wiltsche, C. 2013. On Stochastic Games with Multiple Objectives. In Chatterjee, K.; and Sgall, J., eds., *Mathematical Foundations of Computer Science 2013*, 266–277. Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN 978-3-642-40313-2.

Chow, Y.; and Ghavamzadeh, M. 2014. Algorithms for CVaR Optimization in MDPs. In Ghahramani, Z.; Welling, M.; Cortes, C.; Lawrence, N.; and Weinberger, K., eds., *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc.

Chow, Y.; Ghavamzadeh, M.; Janson, L.; and Pavone, M. 2017. Risk-Constrained Reinforcement Learning with Percentile Risk Criteria. *Journal of Machine Learning Research*, 18(1): 6070–6120.

Chow, Y.; Nachum, O.; Duenez-Guzman, E.; and Ghavamzadeh, M. 2018. A Lyapunov-based Approach to Safe Reinforcement Learning. In Bengio, S.; Wallach, H.; Larochelle, H.; Grauman, K.; Cesa-Bianchi, N.; and Garnett, R., eds., *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc.

Chow, Y.; Tamar, A.; Mannor, S.; and Pavone, M. 2015. Risk-Sensitive and Robust Decision-Making: a CVaR Optimization Approach. In Cortes, C.; Lawrence, N.; Lee, D.; Sugiyama, M.; and Garnett, R., eds., *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc.

Hou, P.; Yeoh, W.; and Varakantham, P. 2016. Solving Risk-Sensitive POMDPs With and Without Cost Observations. In *AAAI 2016*, 3138–3144. AAAI Press.

Kocsis, L.; and Szepesvári, C. 2006. Bandit Based Monte-Carlo Planning. In Fürnkranz, J.; Scheffer, T.; and Spiliopoulou, M., eds., *European Conference on Machine Learning (ECML 2006)*, volume 4212 of *LNCS*, 282–293. Springer.

Kurečka, M.; Nevyhoštěný, V.; Novotný, P.; and Unčovský, V. 2024. Threshold UCT: Cost-Constrained Monte Carlo Tree Search with Pareto Curves. arXiv:2412.13962.

Křetínský, J.; and Meggendorfer, T. 2018. Conditional Value-at-Risk for Reachability and Mean Payoff in Markov Decision Processes. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '18, 609–618. New York, NY, USA: Association for Computing Machinery. ISBN 9781450355834.

L.A., P.; and Fu, M. C. 2022. Risk-Sensitive Reinforcement Learning via Policy Gradient Search. *Foundations and Trends® in Machine Learning*, 15(5): 537–693.

Lee, J.; Jang, Y.; Poupart, P.; and Kim, K.-E. 2017. Constrained Bayesian Reinforcement Learning via Approximate Linear Programming. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*, 2088–2095.

Lee, J.; Kim, G.-h.; Poupart, P.; and Kim, K.-E. 2018. Monte-Carlo Tree Search for Constrained POMDPs. In Bengio, S.; Wallach, H.; Larochelle, H.; Grauman, K.; Cesa-Bianchi, N.; and Garnett, R., eds., *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc.

Moffaert, K. V.; and Nowé, A. 2014. Multi-Objective Reinforcement Learning using Sets of Pareto Dominating Policies. *Journal of Machine Learning Research*, 15(107): 3663–3692.

Moss, R. J.; Jamgochian, A.; Fischer, J.; Corso, A.; and Kochenderfer, M. J. 2024. ConstrainedZero: Chance-Constrained POMDP Planning Using Learned Probabilistic Failure Surrogates and Adaptive Safety Constraints. In *International Joint Conference on Artificial Intelligence (IJCAI)*.

Painter, M.; Lacerda, B.; and Hawes, N. 2020. Convex Hull Monte-Carlo Tree-Search. In *Proceedings of the 30th International Conference on Automated Planning and Scheduling (ICAPS)*, 217–225. AAAI Press.

Poupart, P.; Malhotra, A.; Pei, P.; Kim, K.; Goh, B.; and Bowling, M. 2015. Approximate Linear Programming for Constrained Partially Observable Markov Decision Processes. In *AAAI 2015.*, 3342–3348. AAAI Press.

Santana, P.; Thiébaux, S.; and Williams, B. C. 2016. RAO*: An Algorithm for Chance-Constrained POMDP's. In *AAAI Conference on Artificial Intelligence (AAAI 2016)*, 3308–3314. AAAI Press.

Satija, H.; Amortila, P.; and Pineau, J. 2020. Constrained Markov Decision Processes via Backward Value Functions. In III, H. D.; and Singh, A., eds., *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, 8502–8511. PMLR.

Schrittwieser, J.; Antonoglou, I.; Hubert, T.; Simonyan, K.; Sifre, L.; Schmitt, S.; Guez, A.; Lockhart, E.; Hassabis, D.; Graepel, T.; et al. 2020. Mastering Atari, Go, chess and Shogi by planning with a learned model. *Nature*, 588(7839): 604–609.

Silver, D.; Hubert, T.; Schrittwieser, J.; Antonoglou, I.; Lai, M.; Guez, A.; Lanctot, M.; Sifre, L.; Kumaran, D.; Graepel, T.; Lillicrap, T.; Simonyan, K.; and Hassabis, D. 2018. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science*, 362(6419): 1140–1144.

Silver, D.; and Veness, J. 2010. Monte-Carlo Planning in Large POMDPs. In *Neural Information Processing (NIPS 23)*, 2164–2172. Curran Associates, Inc.

Tessler, C.; Mankowitz, D. J.; and Mannor, S. 2019. Reward Constrained Policy Optimization. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.

Undurti, A.; and How, J. P. 2010. An Online Algorithm for Constrained POMDPs. In *International Conference on Robotics and Automation (ICRA'17)*, 3966–3973. IEEE.