2023 62nd IEEE Conference on Decision and Control (CDC)
December 13-15, 2023. Marina Bay Sands, Singapore

A Fully Polynomial Time Approximation Scheme for Constrained MDPs under Local Transitions

Majid Khonji[1]

***Abstract*— The fixed-horizon constrained Markov Decision Process (C-MDP) is a well-known model for planning in stochastic environments under operating constraints. Chance-constrained MDP (CC-MDP) is a variant that allows bounding the probability of constraint violation, which is desired in many safety-critical applications. CC-MDP can also model a class of MDPs, called Stochastic Shortest Path (SSP), under dead-ends, where there is a trade-off between the probability-to-goal and cost-to-goal. This work studies the structure of (C)C-MDP, particularly an important variant that involves local transition. In this variant, the state reachability exhibits a certain degree of locality and independence from the remaining states. More precisely, the number of states, at a given time, that share some reachable future states is always constant. (C)C-MDP under local transition is NP-Hard even for a planning horizon of two. In this work, we propose a fully polynomial-time approximation scheme for (C)C-MDP that computes (near) optimal deterministic policies. Such an algorithm is among the best approximation algorithms attainable in theory and gives insights into the approximability of constrained MDP and its variants.**

I. Introduction

The *Markov decision process* (MDP) [18] is a classical model for planning in uncertain environments. An MDP consists of states, actions, a stochastic transition function, a utility function, and an initial state. A solution of MDP is a policy that maps a state to an action that maximizes the global expected utility. The *stochastic shortest path* (SSP) [5] is an MDP with non-negative utility values and involves a set of absorbing goal states. The problem has an interesting structure and can be formulated with a dual linear programming (LP) formulation [10] that can be interpreted as a minimum cost flow problem. Moreover, MDPs admit many heuristics-based algorithms [6], [16] that utilize admissible heuristics to guide the search without exploring the whole state space.

Besides, constrained MDP (C-MDP) [2] provides the means to add mission-critical requirements while optimizing the objective function. Each requirement is formulated as a *budget constraint* imposed by a non-replenishable resource for which a bounded quantity is available during the entire plan execution. Resource consumption at each time step reduces the resource availability during subsequent time steps (see [9] for a detailed discussion). A stochastic policy of C-MDP is attainable using several efficient algorithms (e.g., [14]). A heuristics-based search approach in the dual LP can further improve the running time for large state spaces [25]. For deterministic policies, however, it is known that C-MDP is NP-Hard for the finite-horizon case [21] (even when the planning horizon is only 2). The problem is also NP-Hard for the discounted infinite-horizon case [13].

A special type of constraint occurs when we want to bound the probability of constraint violations by some threshold Δ, which is often called a *chance constrained* MDP (CC-MDP). To simplify the problem, [12] proposes approximating the constraint using Markov's inequality, which converts the problem to C-MDP. Another approach by [8] applies Hoeffding's inequality on the sum of independent random variables to improve the bound. Both methods provide conservative policies that respect safety thresholds at the expense of the objective value (which could be arbitrarily worse than optimal).

In the partially observable setting, the problem is called chance-constrained partially observable MDP (CC-POMDP). Several algorithms address CC-POMDP under risk constraints [24], [21]. However, due to partial observability, these methods require an enumeration of histories, making the solution space exponentially large with respect to the planning horizon. To speed up the computation, [17] provides an *anytime* algorithm using a Lagrangian relaxation method for CC-MDP and CC-POMDP that returns feasible sub-optimal solutions and gradually improves the solution's optimality when sufficient time is permitted. Unfortunately, the solution space is represented as an And-Or tree of all possible history trajectories, causing the algorithm to slow down as we increase the planning horizon.

The constrained MDP has a wide range of applications in AI and robotics. One application of CC-MDP is navigation in a discretized environment (e.g., space exploration using a rover [22]). Some states (or grid coordinates) cause the agent to fail, say a cliff. The goal is to maximize utility (or science discovery) while avoiding dangerous states with a probability of $1 - \Delta$. More applications for space landing and exploration are presented in [22]. Another application of CC-MDP is behavior planning for autonomous vehicles (AVs), which has been extensively studied in deterministic environments (see, e.g., [27]). One of the primary sources of uncertainty arises from drivers' intentions, i.e., potential maneuvers of agent vehicles [19]. An effective behavior planner should optimize the maneuvers, say, minimize total commute time while bounding collision probability below some threshold. The objective of CC-MDP is to minimize the expected compute time, and the chance constraint is to bound the probability of collision. See [3] for more empirical details. One application for C-MDP is a battery-operated

[1]Majid Khonji is with the Electrical Engineering and Computer Science Department, Khalifa University, Abu Dhabi, UAE. Email: majid.khonji@ku.ac.ae. This project was supported under award reference CIRA-2020-286 and RCI-2018-KUCARS.

unmanned aerial vehicle (UAV). The vehicle's goal is to maximize surveillance coverage, while the constraint is to keep energy consumption below battery capacity. See [2], [11], [1] for a list of constrained MDP applications[1].

In this work, we study a variant of (C)C-MDP in which the reachable set of states from a given state intersects with at most a constant number of reachable sets from any other states, denoted as (C)C-MDP under *local transitions*. This variant captures a class of MDP problems where state reachability exhibits a certain degree of *locality* such that only a constant number of states at a given time can share future states. The main contribution of this paper is a *fully polynomial time approximation scheme* (FPTAS) that computes (near) optimal deterministic policies for finite-horizon (C)C-MDP under local transitions in polynomial time. Since (C)C-MDP is shown to be NP-Hard (even under local transitions assumption [21]), our result is among the best possible approximation algorithms attainable in theory.

II. Problem Definition

In this section, we provide a formal problem definition and relevant background.

A. C-MDP.

A fixed-horizon constrained Markov decision process (C-MDP) is a tuple $M = \langle \mathcal{S}, \mathcal{A}, T, U, s_0, h, C, P \rangle$, where $\mathcal{S}$ and $\mathcal{A}$ are finite sets of discrete states and actions, respectively; $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is a probabilistic transition function between states, $T(s, a, s') = \Pr(s' \mid a, s)$, where $s, s' \in \mathcal{S}$ and $a \in \mathcal{A}$; $U : \mathcal{S} \times \mathcal{A} \to \mathbb{R}_+$ is a non-negative utility function; s_0 is an initial state; h is the planning horizon; $C : \mathcal{S} \times \mathcal{A} \to \mathbb{R}_+$ is a non-negative cost function; $P \in \mathbb{R}_+$ is a positive upper bound on the cost.

A *deterministic* policy $\pi(\cdot, \cdot)$ is a function that maps a state and time step into action, $\pi : \mathcal{S} \times \{0, 1, ..., h-1\} \to \mathcal{A}$. For simplicity, we write $\pi(s_k)$ to denote $\pi(s_k, k)$. A *run* is a sequence of random states $S_0, S_1, \ldots, S_{h-1}, S_h$ that result from executing a policy, where $S_0 = s_0$ is known. The objective is to compute a policy that maximizes (resp. minimizes) the expected utility (resp. cost) while satisfying the constraint. More formally,

$$\text{(C-MDP)} \qquad \max_{\pi} \mathbb{E}\Big[\sum_{k=0}^{h-1} U(S_k, \pi(S_k))\Big] \tag{1}$$

$$\text{Subject to} \quad \mathbb{E}\Big[\sum_{k=0}^{h-1} C(S_k, \pi(S_k)) \mid \pi\Big] \leq P.$$

The MDP problem and its constrained variants can be visualized by a direct acyclic And-Or graph (DAG) $\mathcal{G}$, where the vertices represent the states and actions. Thus, at depth k the set of state nodes are the states that are reachable from previous actions at depth $k-1$, denoted as $\mathcal{S}_k \subseteq \mathcal{S}$. At each depth, we have at most $|\mathcal{S}|$ states. Fig. 1 provides a pictorial illustration of the MDP And-Or (search) graph. Note that unlike And-Or search trees obtained by history enumeration algorithms (see, e.g., [17]), with such representation, a node may have multiple parents, leading to a significant reduction in the search space.

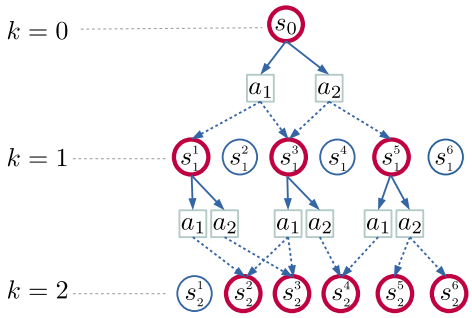

Fig. 1. MDP graph where circles are state nodes and squares are actions. Circles with thick borders represent reachable states at time k, denoted as $\mathcal{S}_k$.

The objective function and the constraint's left-hand side can be written recursively using the Bellman equation as

$$v_\pi(s_k) := \sum_{s_{k+1} \in \mathcal{S}_{k+1}} T(s_k, \pi(s_k), s_{k+1}) v_\pi(s_{k+1}) + U(s_k, \pi(s_k)),$$

$$c_\pi(s_k) := \sum_{s_{k+1} \in \mathcal{S}_{k+1}} T(s_k, \pi(s_k), s_{k+1}) c_\pi(s_{k+1}) + C(s_k, \pi(s_k)),$$

for $k = 0, ..., h-1$.

B. CC-MDP.

A fixed-horizon chance-constrained MDP (CC-MDP) problem is formally defined as a tuple $M = \langle \mathcal{S}, \mathcal{A}, T, U, s_0, h, r, \Delta \rangle$, where $\mathcal{S}, \mathcal{A}, T, U, s_0, h$ are defined as in C-MDP, and

- $r : \mathcal{S} \to [0, 1]$ is the probability of failure at a given state;
- Δ is the corresponding risk budget, a threshold on the probability of failure over the planning horizon.

Let $R(s)$ be a Bernoulli random variable that indicates failure at state s, such that $R(s) = 1$ if and only if s is a risky state and zero otherwise. For simplicity, we write $R(s)$ to denote $R(s) = 1$. The objective of CC-MDP is to compute a deterministic policy (or a conditional plan) π that maximizes (or minimizes) the cumulative expected utility (or cost) while bounding the probability of failure at *any* time step throughout the planning horizon. More precisely,

$$\text{(CC-MDP)} \qquad \max_{\pi} \mathbb{E}\Big[\sum_{k=0}^{h-1} U(S_k, \pi(S_k))\Big] \tag{2}$$

$$\text{Subject to} \quad \Pr\Big(\bigvee_{k=0}^{h} R(S_k) \mid \pi\Big) \leq \Delta. \tag{3}$$

To better understand Cons. (3), define the *execution risk* of a run at state s_k as

$$\text{ER}_\pi(s_k) := \Pr\Big(\bigvee_{k'=k}^{h} R(S_{k'}) \mid S_k = s_k\Big).$$

According to the definition, Cons. (3) is equivalent to $\text{ER}_\pi(s_0) \leq \Delta$. The lemma below shows that such constraint can be computed recursively.

Lemma II.1 ([3]). *The execution risk of policy π can be*

[1]most of the C-MDP applications require non-negative parameters (costs and reward).

written as

$$\textsc{Er}_\pi(s_k) = \begin{cases} r(s_k) + (1 - r(s_k)) \\ \qquad \sum_{s_{k+1} \in \mathcal{S}} \textsc{Er}_\pi(s_{k+1}) \pi(s_k) T(s_k, a, s_{k+1}), & \text{if } k = 0, ..., h-1, \\ r(s_h), & \text{if } k = h. \end{cases}$$

CC-MDP captures a class of MDPs called stochastic shortest path (SSP), where there is a set of absorbing goal states and dead-end states. A measure of policy feasibility under SSP is to have the probability-to-goal above some threshold ϵ [15]. This problem can be modeled as CC-MDP, where all non-goal states at horizon h are considered risky states. Hence, the probability of failure is set to $\Delta = 1 - \epsilon$. SSPs are often defined with infinite horizons. One can reduce SSP to fixed-horizon CC-MDP by successively incrementing the horizon h until a feasible policy is attainable.

C. Assumptions.

In this work, we study a variant of (C)C-MDP in which the number of state-action pairs that share subsequent states is bounded. Such extension is denoted as (C)C-MDP under *local transition*. More formally, define the set of potential next states after executing action a from state $s_k \in \mathcal{S}_k$ for $k = 0, ..., h-1$ by,

$$N_a(s_k) := \{s_{k+1} \mid T(s_k, a, s_{k+1}) > 0, s_{k+1} \in \mathcal{S}_{k+1}\},$$
$$N_a(s_h) := \varnothing.$$

Let $\textsc{Reach}(s_k)$ be a set of states in the MDP graph reachable from state s_k; more precisely,

$$\textsc{Reach}(s_k) := \begin{cases} \bigcup_{\substack{s_{k+1} \in N_a(s_k) \\ a \in \mathcal{A}}} \textsc{Reach}(s_{k+1}) & \text{if } k < h \\ s_k & \text{if } k = h. \end{cases}$$

Definition II.2 (Local Transition)**.** *An MDP graph $\mathcal{G}$ under local transition satisfies*

$$\left|\{s'_k \in \mathcal{G} \mid \textsc{Reach}(s_k) \cap \textsc{Reach}(s'_k) \neq \varnothing\}\right| \leq \psi,$$

for any $s_k \in \mathcal{G}$, where ψ is a constant[2].

When $\psi = 0$, we call our problem (C)C-MDP under *disjoint transition*. The And-Or graph under the disjoint transition assumption is, in fact, an And-Or tree. Such structure helps to easily obtain a dynamic programming structure that is exploited in our algorithms, shown in the next subsections.

We also assume that all parameters (utility and cost values) are non-negative. Without such an assumption, one can show as in [7] that the problem is inapproximable (i.e., no α-approximation algorithm exists unless P=NP).

D. Benchmark.

To analyze our algorithms, we rely on the notion of approximation algorithms. The subject of approximation algorithms is well-studied in the theoretical computer science community [26]. As follows, we define some standard terminology for approximation algorithms. Consider a maximization problem Π with non-negative objective function $f(\cdot)$; let F be a feasible solution to Π and $F^\star$ be an optimal solution to Π. $f(F)$ denotes the objective value of F. Let $\textsc{Opt} = f(F^\star)$ be the optimal objective value of $F^\star$. A common definition of approximate solutions is α-approximation, where α characterizes the approximation ratio between the approximate solution and an optimal solution.

Definition II.3 (Approximation Algorithm [26])**.** *For $\alpha \in [0, 1]$, an α-approximation to maximization problem Π is an algorithm that obtains a feasible solution F for any instance such that $f(F) \geq \alpha \cdot \textsc{Opt}$.*

In particular, *fully polynomial-time approximation scheme* (FPTAS) is a $(1-\epsilon)$-approximation algorithm to a maximization problem for any $\epsilon > 0$. The running time of an FPTAS is polynomial in the input size and for every fixed $\frac{1}{\epsilon}$. In other words, FPTAS allows the trade of the approximation ratio against the running time.

In the following, we first study the problem under disjoint transition and then extend the result to the local transition case.

III. Algorithm

For simplicity, we first study a special variant of (C)C-MDP in which actions stochastically lead to a small number of potential states, denoted as (C)C-MDP under *limited transition*.

Definition III.1 (Limited Transition)**.** *There exists a constant $\gamma \in \mathbb{N}_+$ such that $N_a(s) \leq \gamma$ for all $a \in \mathcal{A}, s \in \mathcal{S}$.*

In the next subsection, we study (C)C-MDP under limited and disjoint transition, whereas in the following subsection, we relax the limited transition assumption. In the last subsection, we present an FPTAS for (C)C-MDP under the local transition assumption, which generalizes the former cases.

A. FPTAS for (C)C-MDP under Limited and Disjoint Transition

Algorithm 1: `lim-DynMDP`$[M, \epsilon]$

Input : An instance of CC-MDP M; a parameter ϵ for the approximation guarantee
Output: A deterministic policy π

$\texttt{DP}_{\textsc{Er}}(s_k, \ell_k) \leftarrow \infty$; $\texttt{DP}_\pi(s_k, \ell_k) \leftarrow \varnothing$, $\texttt{DP}_{\bar{\boldsymbol{\ell}}}(s_k, \ell_k) \leftarrow \mathbf{0}$, for $k = 0, ..., h$;
$\texttt{DP}_{\textsc{Er}}(s_h, \ell_h) \leftarrow r(s_h)$, for all $s_h \in \mathcal{S}_h, \ell_h \in \mathcal{L}_k = \{0\}$
for $k = h-1, ..., 0; s_k \in \mathcal{S}_k; \ell_k \in \mathcal{L}_k$ **do**
  $\texttt{DP}_{\textsc{Er}}(s_k, \ell_k), \texttt{DP}_\pi(s_k, \ell_k), \texttt{DP}_{\bar{\boldsymbol{\ell}}}(s_k, \ell_k) \leftarrow$ `Update`$[s_k, \ell_k, \{\texttt{DP}_{\textsc{Er}}(s, \ell)\}_{s \in \mathcal{S}_{k+1}, \ell \in \mathcal{L}_{k+1}}]$
end for
$\pi \leftarrow$ `Fetch-Policy`$[\texttt{DP}]$
return π

The procedure involves constructing a 2-dimensional dynamic programming table, $\texttt{DP}(\cdot, \cdot)$, where each cell $\texttt{DP}(s_k, \ell_k)$ corresponds to state $s_k \in \mathcal{S}_k$, and a discrete utility value ℓ_k (which we will clarify next). Each cell contains three quantities, $\texttt{DP}_{\textsc{Er}}(s_k, \ell_k) \in \mathbb{R}_+$ which maintains the minimum execution risk from state s_k, executing an action that accrues a total value of at least ℓ_k; $\texttt{DP}_\pi(s_k, \ell_k) = a$, the corresponding policy action a; and $\texttt{DP}_{\bar{\boldsymbol{\ell}}}(s_k, \ell_k) \in \mathbb{R}_+^{|N_a(s_k)|}$, a value allocation for subsequent states as we see next. The main idea behind the algorithm lies in a utility discretization procedure that shrinks the set of possible values at a given state into a manageable number, exploiting the limited and disjoint transition assumptions. A detailed description is provided

[2]We mean by a constant that the number is relatively small and is not a function of MDP instance M.

Algorithm 2: $\texttt{Update}\big[s_k, \ell_k, \{\texttt{DP}(s,\ell)\}_{s\in\mathcal{S}_{k+1}}^{\ell\in\mathcal{L}_{k+1}}\big]$

1 $\text{ER}(s_k) \leftarrow \infty$
2 $\text{ACT} \leftarrow \varnothing$
3 $\text{ALLOC} \leftarrow \mathbf{0}$
4 **for** $a \in \mathcal{A}$ **do**
5 // Find an allocation $\overline{\boldsymbol{\ell}}_{k+1}$ that acheives the minimum execution risk for action a such that the total utility value is at least ℓ_k
6 **for** $\overline{\boldsymbol{\ell}}_{k+1} = (\overline{\ell}^1_{k+1}, \overline{\ell}^2_{k+1}, ...) \in \mathcal{L}_{k+1}^{|N_a(s_k)|}$ **do**
7 $\overline{v}_a(s_k, \overline{\boldsymbol{\ell}}_{k+1}) \leftarrow \Big\lfloor \frac{1}{L_k}\Big(\sum_{s^i_{k+1}\in N_a(s_k)} T(s_k,a,s^i_{k+1})\overline{\ell}^i_{k+1} + U(s_k,a)\Big)\Big\rfloor \cdot L_k$
8 **if** $\overline{v}_a(s_k, \overline{\boldsymbol{\ell}}_{k+1}) \geq \ell_k$ **then**
9 $\text{ER}_a(s_k, \overline{\boldsymbol{\ell}}_{k+1}) := r(s_k) + (1 - r(s_k)) \sum_{s^i_{k+1}\in N_a(s_k)} T(s_k,a,s^i_{k+1}) \cdot \texttt{DP}_{\text{ER}}(s^i_{k+1}, \overline{\boldsymbol{\ell}}^i_{k+1})$
10 **if** $\text{ER}_a(s_k, \overline{\boldsymbol{\ell}}_{k+1}) < \text{ER}(s_k)$ **then**
11 $\text{ER}(s_k) \leftarrow \text{ER}_a(s_k, \overline{\boldsymbol{\ell}}_{k+1})$
12 $\text{ACT} \leftarrow a$
13 $\text{ALLOC} \leftarrow \overline{\boldsymbol{\ell}}_{k+1}$
14 **end if**
15 **end if**
16 **end for**
17 **end for**
18 **return** $\text{ER}(s_k)$, ACT, ALLOC

Algorithm 3: $\texttt{Fetch-Policy}[\texttt{DP}]$

1 $\mathcal{C}_k \leftarrow \varnothing$ for $k = 1, ..., h-1$
2 $\ell_0 \leftarrow$ Find the maximum $\ell_0 \in \mathcal{L}_0$ such that $\texttt{DP}_{\text{ER}}(s_0, \ell_0) \leq \Delta$ and $\texttt{DP}_{\text{ER}}(s_0, \ell_0 + L_0) > \Delta$
3 $\mathcal{C}_0 \leftarrow \{(s_0, \ell_0)\}$
4 **for** $k = 0, ..., h-1$ **do**
5 **for** $(s^i_k, \overline{\ell}^i_k) \in \mathcal{C}_k$ **do**
6 $\pi(s^i_k) \leftarrow \texttt{DP}_\pi(s^i_k, \overline{\ell}^i_k)$
7 $\overline{\boldsymbol{\ell}}_{k+1} \leftarrow \texttt{DP}_{\overline{\boldsymbol{\ell}}}(s^i_k, \overline{\ell}^i_k)$
8 $\mathcal{C}_{k+1} \leftarrow \mathcal{C}_{k+1} \cup \{(s^i_{k+1}, \overline{\ell}^i_{k+1})\}_{s^i_{k+1}\in N_a(s^i_k)}$ where $a = \pi(s^i_k)$
9 **end for**
10 **end for**
11 **return** π

in Algorithm `lim-DynMDP` (Alg. 1). The algorithm relies on two subroutines, `Update` (Alg. 2) and `Fetch-Policy` (Alg. 3). The former computes a discretized version of the Bellman equation along with the corresponding execution risk, and the latter recursively extracts the corresponding policy. Line 7 of `Update` computes a discretized version of the Bellman equation under discretized future rewards, and Line 9 recursively computes the execution risk based on Lemma II.1. The pseudo-code provided herein is for CC-MDP; however, it also applies to C-MDP with minor modifications. Namely, Line 9 of `Update` should be replaced by

$$\text{ER}_a(s_k, \overline{\boldsymbol{\ell}}_{k+1}) \leftarrow \sum_{s^i_{k+1}\in N_a(s_k)} T(s_k,a,s^i_{k+1}) \cdot \texttt{DP}_{\text{ER}}(s^i_{k+1}, \overline{\boldsymbol{\ell}}^i_{k+1}) + C(s_k,a)$$

and Δ by P in Line 2 of `Fetch-Policy`. Thus, all results in this paper apply to C-MDP as well. (In the remaining text, the term *execution risk* in the context of C-MDP would refer to the *total cost* instead.) Let $U_{\max} := \max_{s\in\mathcal{S}, a\in\mathcal{A}} U(s,a)$ be the maximum utility of an action. Denote a discrete set of values $\mathcal{L}_k$ for each time step $k = 0, ..., h$ as

$$\mathcal{L}_k := \{0, L_k, 2L_k, ..., \lfloor \tfrac{U_{\max}\cdot(h-k)}{L_k}\rfloor L_k\}, \qquad \text{where}$$

$$L_k := \frac{\epsilon U_{\max}}{(h-k)(\ln h + 1)}. \tag{4}$$

Let π be a solution returned by `lim-DynMDP`, and $v_\pi(s_k) := \mathbb{E}[\sum_{k'=k}^{h-1} U(S_{k'}, \pi(S_{k'}))]$ be the corresponding value function at state s_k. Similarly, denote π^* to be an optimal solution, and $v_{\pi^*}(s_k)$ be the corresponding value function. Without loss of generality, assume that $v_{\pi^*}(s_0) \geq U_{\max}$ [3]. Define $\overline{v}_\pi(s_0)$ (resp., $\overline{v}_{\pi^*}(s_0)$) to be a discretized objective value computed recursively by,

$$\overline{v}_\pi(s_k) = \Big\lfloor \tfrac{1}{L_k}\Big(\sum_{s_{k+1}\in\mathcal{S}_{k+1}} T(s_k,\pi(s_k),s_{k+1})\cdot\overline{v}_\pi(s_{k+1}) + U(s_k,\pi(s_k))\Big)\Big\rfloor L_k. \tag{5}$$

The above equation corresponds to step 7 of `Update`.

Lemma III.2. *Let π be a policy obtained by* `lim-DynMDP` *and π^* be an optimal deterministic policy. The policy π is feasible and satisfies $\overline{v}_\pi(s_0) \geq \overline{v}_{\pi^*}(s_0)$.*

Proof: We show (by induction) that for some $\ell_k \in \mathcal{L}_k$ and $\overline{\boldsymbol{\ell}}_{k+1} \in \mathcal{L}_{k+1}^{|N_a(s_k)|}$, there exists an action a such that $\ell_k = \overline{v}_a(s_k, \overline{\boldsymbol{\ell}}_{k+1}) \geq \overline{v}_{\pi^*}(s_k)$, where $\overline{v}_a(\cdot,\cdot)$ is defined in Line 7 of `Update`. The algorithm enumerates all values of $\overline{\boldsymbol{\ell}}_{k+1}$ such that it attains the minimum execution risk for every $\ell_k \in \mathcal{L}_k$. Throughout recursion, the procedure ensures that a feasible solution π can be constructed such that $\ell_k = \overline{v}_a(s_k, \overline{\boldsymbol{\ell}}_{k+1}) \geq \overline{v}_{\pi^*}(s_k)$, as shown by `Fetch-Policy`.

We proceed with the induction proof; for the base case, $k = h-1$, we have $\overline{v}_a(s_{h-1}, \overline{\boldsymbol{\ell}}_h) = \lfloor U(s_{h-1},a)/L_{h-1}\rfloor L_{h-1}$. Clearly, there is an action that satisfies the claim. For the inductive step, suppose the claim holds at step k; we show that the claim also holds for step $k-1$. Note that algorithm `lim-DynMDP` enumerates all discretized allocations $\overline{\boldsymbol{\ell}}_k$ at step $k-1$ (as per Step 6 of `Update`). Also note that $\overline{v}_{\pi^*}(s_k) \in \mathcal{L}_k$ as the largest element in set $\mathcal{L}_k$, defined in Eq. (4), satisfies $\lfloor U_{\max}(h-k)/L_k\rfloor L_k \geq \overline{v}_{\pi^*}(s_k)$. Hence, there exists an allocation $\overline{\boldsymbol{\ell}}_k$ such that each i-th element $\overline{\ell}^i_k = \overline{v}_a(s^i_k, \overline{\boldsymbol{\ell}}_{k+1}) \geq \overline{v}_{\pi^*}(s^i_k)$ for some $\overline{\boldsymbol{\ell}}_{k+1}$ (inductive assumption). Hence, there exists an ℓ_{k-1} and an action a

[3] If $U_{\max} = U(s_k,a) > v_{\pi^*}(s_0)$, then any policy that outputs action a at state s_k must be infeasible. Thus, such an action can be deleted from the set of allowable actions at state s_k. Therefore, $U_{\max}$ can be taken as the second largest utility action and so on. The procedure can be performed in polynomial time as follows. Fix a policy $\pi(s_k) = a$, and set the rest $\pi(s_{k'}) = a'_{k'}$ such that action $a'_{k'}$ achieves the minimum execution risk for $k' = h-1, ..., 0$ (computed recursively using Lemma II.1). If the solution is infeasible, repeat the procedure at different k. If again infeasible, one can safely drop $U(s_k,a)$, consider the next largest utility, and then repeat the procedure.

such that

$$\begin{aligned}
\ell_{k-1} &= \overline{v}_a(s_{k-1}, \overline{\boldsymbol{\ell}}_k) \\
&= \left\lfloor \frac{1}{L_{k-1}} \Big(\sum_{s_k^i \in N_a(s_{k-1})} T(s_{k-1}, a, s_k^i) \overline{\ell}_k^i + U(s_{k-1}, a) \Big) \right\rfloor L_{k-1} \\
&\geq \left\lfloor \frac{1}{L_{k-1}} \Big(\sum_{s_k \in \mathcal{S}_k} T(s_{k-1}, a, s_k) \overline{v}_{\pi^*}(s_k) + U(s_{k-1}, a) \Big) \right\rfloor L_{k-1} \\
&= \overline{v}_{\pi^*}(s_{k-1}), \qquad (6)
\end{aligned}$$

where the inequality is followed by the inductive assumption.

It remains to show that such action a is feasible. Each cell $\texttt{DP}_\pi(\cdot,\cdot)$ corresponds to an action that achieves the minimum execution risk that accrues a total value of at least ℓ_k. Since the execution risk is a non-decreasing function (Line 9 of `Update`), $\texttt{DP}_{\text{ER}}(s_k, \ell_k) \leq \text{ER}_{\pi^*}(s_k) \leq \Delta$ for some $\ell_k = \overline{v}_{\pi^*}(s_k)$. By the disjoint transition assumption, there is a unique state s_{k-1} that involves the row $\texttt{DP}_{\text{ER}}(s_k, \cdot)$, $s_k \in N_a(s_{k-1})$, in computing $\texttt{DP}_{\text{ER}}(s_{k-1}, \ell_{k-1})$ (Line 9 of `Update`). Hence, only table cells related to state s_{k-1} sets the values of $\overline{\ell}_k^i$ of the subsequent states $s_k \in N_a(s_{k-1})$. As each cell $\texttt{DP}_\pi(s_k, \overline{\ell}_k^i)$ corresponds to a single action, no two s_{k-1}, s'_{k-1} share subsequent state s_k, and only one cell among row $\texttt{DP}_\pi(s_{k-1}, \cdot)$ is backtracked by `Fetch-Policy`, the policy remains consistent, i.e., it outputs a single action for each state. (Note that this is not the case if s_k has multiple parents in the And-Or graph, which is the case under the local transition assumption.) Such an action is backtracked by `Fetch-Policy`. Therefore, policy π is feasible.

Lemma III.3. *An optimal deterministic policy π^* satisfies $\overline{v}_{\pi^*}(s_k) \geq v_{\pi^*}(s_k) - \sum_{k'=k}^{h-1} L_{k'}$.*

Proof: We proceed with an inductive proof. For the base case, computing Eq. (5) for π^* at $k = h-1$, we have

$$\begin{aligned}
\overline{v}_{\pi^*}(s_{h-1}) &= \left\lfloor \frac{U(s_{h-1}, \pi^*(s_{h-1}))}{L_{h-1}} \right\rfloor \cdot L_{h-1} \\
&\geq U(s_{h-1}, \pi^*(s_{h-1})) - L_{h-1} = v_{\pi^*}(s_{h-1}) - L_{h-1}, \qquad (7)
\end{aligned}$$

which follows using the property $\lfloor \frac{x}{y} \rfloor y \geq x - y$ for $x, y \in \mathbb{R}_+$. For the inductive step, suppose that we have, $\overline{v}_{\pi^*}(s_k) \geq v_{\pi^*}(s_k) - \sum_{k'=k}^{h-1} L_{k'}$. We compute the corresponding inequality for s_{k-1} as follows,

$$\begin{aligned}
\overline{v}_{\pi^*}(s_{k-1}) &= \left\lfloor \frac{1}{L_{k-1}} \Big(\sum_{s_k \in \mathcal{S}_k} T(s_{k-1}, \pi^*(s_{k-1}), s_k) \cdot \overline{v}_{\pi^*}(s_k) \right. \\
&\quad \left. + U(s_{k-1}, \pi^*(s_{k-1})) \Big) \right\rfloor L_{k-1} \qquad (8) \\
&\geq \left\lfloor \frac{1}{L_{k-1}} \Big(\sum_{s_k \in \mathcal{S}} T(s_{k-1}, \pi^*(s_{k-1}), s_k) \cdot \big(v_{\pi^*}(s_k) \right. \\
&\quad \left. - \sum_{k'=k}^{h-1} L_{k'} \big) + U(s_{k-1}, \pi^*(s_{k-1})) \Big) \right\rfloor L_{k-1}, \qquad (9)
\end{aligned}$$

where Eq. (9) follows by the inductive assumption. Since $T(\cdot,\cdot,\cdot)$ is a probability function that adds up to one, and using the property $\lfloor \frac{x}{y} \rfloor y \geq x - y$ for $x, y \in \mathbb{R}_+$, we obtain,

$$\begin{aligned}
\overline{v}_{\pi^*}(s_{k-1}) &\geq \sum_{s_k \in \mathcal{S}} T(s_{k-1}, \pi^*(s_{k-1}), s_k) \cdot v_{\pi^*}(s_k) \\
&\quad + U(s_{k-1}, \pi^*(s_{k-1})) - \sum_{k'=k-1}^{h-1} L_{k'} \\
&= v_{\pi^*}(s_{k-1}) - \sum_{k'=k-1}^{h-1} L_{k'}, \qquad (10)
\end{aligned}$$

which completes the inductive proof.

Corollary III.4. `lim-DynMDP` *is an FPTAS for (C)C-MDP under limited and disjoint transition assumptions.*

Proof: First, observe that the algorithm maintains a dynamic programming table with minimum execution risk. Subroutine `Fetch-Policy` ensures that a feasible solution with such property is retrieved. The algorithm runs in $O\big(\big(\frac{h^2 \ln h + 1}{\epsilon}\big)^{\gamma+1} |\mathcal{A}||\mathcal{S}|\big)$. Note that by the limited transition assumption, γ is a constant; thus, the running time is polynomial. By Lemma III.3 and by the definition of L_k given in Eq. (4), we obtain

$$\begin{aligned}
\overline{v}_{\pi^*}(s_0) &\geq v_{\pi^*}(s_0) - \sum_{k=0}^{h-1} L_k \\
&= v_{\pi^*}(s_0) - \sum_{k=0}^{h-1} \frac{\epsilon U_{\max}}{(h-k)(\ln h + 1)} \\
&= v_{\pi^*}(s_0) - \frac{\epsilon U_{\max}}{\ln h + 1} \sum_{k=0}^{h-1} \frac{1}{(h-k)} \\
&\geq v_{\pi^*}(s_0) - \frac{\epsilon U_{\max}}{\ln h + 1} (\ln h + 1) \qquad (11) \\
&\geq (1 - \epsilon) \cdot v_{\pi^*}(s_0), \qquad (12)
\end{aligned}$$

where Eq. (11) follows by using an upper bound on the harmonic series, $\sum_{n=1}^{k} \frac{1}{n} \leq \ln k + 1$. By Lemma III.2 and Eq. (12), $v_\pi(s_0) \geq \overline{v}_\pi(s_0) \geq \overline{v}_{\pi^*}(s_0) \geq (1-\epsilon) v_{\pi^*}(s_0)$, which completes the proof.

B. FPTAS for (C)C-MDP under Disjoint Transition

In this section, we relax the limited transition assumption and show how to obtain an FPTAS for (C)C-MDP. In other words, we assume γ is a polynomial in definition III.1. The main idea behind our algorithm is to improve `Update` subroutine to avoid full enumeration of $\overline{\boldsymbol{\ell}}_{k+1}$, which is exponential in the number of subsequent states $|N_a(s_k)|$. Such enumeration could be feasible under the limited transition assumption, but not in general. We show here how the structure of this step could be exploited. Notably, finding an allocation that achieves the minimum execution risk such that the total utility value is at least ℓ_k is a slight generalization for a well-known problem called *minimum Knapsack* (MinKS) [23], [4]. More formally,

Definition III.5. *Multiple-choice minimum Knapsack problem (McMinKS) is defined as follows. Given a set of categories $\mathcal{N}$, and a set of allowable choices $\mathcal{M}_i$ per category $i \in \mathcal{N}$, an item (i,j) is defined by weight $w_{i,j} \in \mathbb{R}_+$ and value $v_{i,j} \in \mathbb{R}_+$ for $i \in \mathcal{N}$ and $j \in \mathcal{M}_i$. The goal is to*

select one item from the allowable choices $\mathcal{M}_i$ per category i (hence the name multiple-choice) such that the total weight is minimized and the total value is at least $D \in \mathbb{R}_+$.

The problem can be formally defined as an *integer linear program* (ILP) as follows.

$$\text{(McMinKS)} \quad \min_{x_{i,j} \in \{0,1\}} \sum_{i \in \mathcal{N}} \sum_{j \in \mathcal{M}_i} w_{i,j} x_{i,j},$$

$$\text{Subject to} \quad \sum_{i \in \mathcal{N}} \sum_{j \in \mathcal{M}_i} v_{i,j} x_{i,j} \geq D, \tag{13}$$

$$\sum_{j \in \mathcal{M}_i} x_{i,j} = 1, \quad \text{for all } i \in \mathcal{N}. \tag{14}$$

Algorithm 4, denoted by `KS-Update`, presents a reduction from the allocation subproblem to McMinKS in Lines 4-8. Indeed, finding an optimal solution for the corresponding McMinKS instance will obtain an FPTAS (Lemma III.2 holds and hence Corollary III.4 proof follows). However, MinKS is NP-Hard [23], [20]; therefore, our best bet is to find an approximate solution in polynomial time. Although there is an FPTAS for MinKS, the approximation guarantee is provided on the objective function, which in our case, following the reduction, is the constraint for the original (C)C-MDP problem. Thus, we need an algorithm that bounds the constraint violation of McMinKS (which is the objective of (C)C-MDP, following the reduction above). Some modifications are needed to the algorithm to obtain a bounded McMinKS constraint violation and handle the multiple-choice extension (as we will see next).

Algorithm 4: `KS-Update`$[s_k, \ell_k, \{\texttt{DP}(s,\ell)\}_{s \in \mathcal{S}_{k+1}}^{\ell \in \mathcal{L}_{k+1}}]$

1 $\text{ER}(s_k) \leftarrow \infty$; $\text{ACT} \leftarrow \varnothing$; $\text{ALLOC} \leftarrow \mathbf{0}$
2 **for** $a \in \mathcal{A}$ **do**
3 // Find an allocation $\overline{\boldsymbol{\ell}}_{k+1}$ that achieves the minimum execution risk for action a such that the total utility value is at least ℓ_k
4 Let $\mathcal{N} := \{1, ..., |N_a(s_k)|\}$
5 Let $\mathcal{M}_i := \mathcal{L}_{k+1}$ for $i \in \mathcal{N}$
6 Let $w_{i,j} := T(s_k, a, s^i_{k+1}) \cdot \texttt{DP}_{\text{ER}}(s^i_{k+1}, \overline{\ell}^i_{k+1})$ for all $j = \overline{\ell}^i_{k+1} \in \mathcal{L}_{k+1}$ and $s^i_{k+1} \in N_a(s_k)$
7 Let $v_{i,j} := T(s_k, a, s^i_{k+1}) \cdot \overline{\ell}^i_{k+1}$ for $j = \overline{\ell}^i_{k+1} \in \mathcal{M}_i$ and i such that $s^i_{k+1} \in N_a(s_k)$
8 Let $D := \ell_k - U(s_k, a)$
9 $\overline{\boldsymbol{\ell}}_{k+1} \leftarrow$ `Dyn-MinKS`$[(w_{i,j}, v_{i,j})_{i \in \mathcal{N}, j \in \mathcal{M}_i}, D]$
10 $\text{ER}_a(s_k, \overline{\boldsymbol{\ell}}_{k+1}) := r(s_k) + (1 - r(s_k)) \sum_{s^i_{k+1} \in N_a(s_k)} T(s_k, a, s^i_{k+1}) \cdot \texttt{DP}_{\text{ER}}(s^i_{k+1}, \overline{\ell}^i_{k+1})$
11 **if** $\text{ER}_a(s_k, \overline{\boldsymbol{\ell}}_{k+1}) < \text{ER}(s_k)$ **then**
12 $\text{ER}(s_k) \leftarrow \text{ER}_a(s_k, \overline{\boldsymbol{\ell}}_{k+1})$
13 $\text{ACT} \leftarrow a$
14 $\text{ALLOC} \leftarrow \overline{\boldsymbol{\ell}}_{k+1}$
15 **end if**
16 **end for**
17 **return** $\text{ER}(s_k)$, ACT, ALLOC

Algorithm 5, denoted as `Dyn-MinKS`, gives a dynamic programming procedure to solve McMinKS within a bounded constraint violation. The algorithm rounds the values into a discrete set of possible values that provably can have a bounded constraint violation (as per Lemma III.6 below). The set of possible *discretized* values $\mathcal{R}_k$ is defined

Algorithm 5: `Dyn-MinKS`$[(w_{i,j}, v_{i,j})_{i \in \mathcal{N}, j \in \mathcal{M}_i}, D]$

1 $\boldsymbol{\ell} = (\ell^i)_{i \in \mathcal{N}} \leftarrow \mathbf{0}$
2 $\texttt{TB}(i, \rho) \leftarrow \infty$ for all $i \in \mathcal{N}$ and $\rho \in \mathcal{R}$
3 $\texttt{TB}(0, \rho) \leftarrow 0$ for all ρ
4 Let $\overline{v}_{i,j} := \left\lfloor \frac{v_{i,j}}{R_k} \right\rfloor \cdot R_k$ for all $i \in \mathcal{N}$ and $j \in \mathcal{M}_i$
5 **for** $i = 1, ..., |\mathcal{N}|$ **do**
6 **for** $\rho \in \mathcal{R}_k = \left\{0, R_k, 2R_k, ..., \left\lfloor \frac{D + \max_{i,j} v_{i,j}}{R_k} \right\rfloor R_k \right\}$ **do**
7 $\texttt{TB}(i, \rho) \leftarrow \min_{j \in \mathcal{M}_i} \texttt{TB}(i-1, [\rho - \overline{v}_{i,j}]^+) + w_{i,j}$
8 $\texttt{Alc}(i, \rho) \leftarrow \arg\min_{j \in \mathcal{M}_i} \texttt{TB}(i-1, [\rho - \overline{v}_{i,j}]^+) + w_{i,j}$, where $[x]^+ = x$ if $x \geq 0$ and $[x]^+ = 0$ otherwise, for any $x \in \mathbb{R}$
9 **end for**
10 **end for**
11 Find minimum ρ' such that $\rho' \geq D$
12 **for** $i = |\mathcal{N}|, |\mathcal{N}| - 1, ..., 1$ **do**
13 $j = \texttt{Alc}(i, \rho')$; $\rho' \leftarrow \rho' - \overline{v}_{i,j}$; $\ell^i \leftarrow j$
14 **end for**
15 **return** $\boldsymbol{\ell}$

as

$$\mathcal{R}_k := \left\{0, 1R_k, ..., \left\lfloor \frac{D + \max_{i,j} v_{i,j}}{R_k} \right\rfloor R_k \right\}, \tag{15}$$

where R_k is a discretization factor defined below. Let $\boldsymbol{\ell}$ be an allocation returned by algorithm `Dyn-MinKS` and $\boldsymbol{\ell}^*$ be an optimal solution.

Lemma III.6. *Algorithm* `Dyn-MinKS` *obtains a solution $\boldsymbol{\ell} = (\ell^1, ..., \ell^{|\mathcal{N}|})$ that satisfies*

$$\sum_{i \in \mathcal{N}} w_{i,\ell^i} \leq \sum_{i \in \mathcal{N}} w_{i,\ell^{i*}}, \text{ and } \sum_{i \in \mathcal{N}} v_{i,\ell^i} \geq \sum_{i \in \mathcal{N}} v_{i,\ell^{i*}} - |\mathcal{N}| R_k,$$

where $\boldsymbol{\ell}^*$ is an optimal solution.

Proof: Define $\overline{v}_{i,\ell^i} := \lfloor v_{i,\ell^i} / R_k \rfloor R_k$ as in Line 4 of `Dyn-MinKS` (also define $\overline{v}_{i,\ell^{i*}} := \lfloor v_{i,\ell^{i*}} / R_k \rfloor R_k$). The algorithm maintains a table $\texttt{TB}(i, \rho)$ of minimum total item weights up to category i that satisfies a total value of at least ρ. Since the algorithm discretizes values (Line 4), and the largest element of $\mathcal{R}$ is an upper bound on $\sum_{i \in \mathcal{N}} \overline{v}_{i,\ell^{i*}}$ (by the definition of $\mathcal{R}$ in Eq. (15)), then any discretized optimal total values are considered in the table. Therefore in steps 11-13, the algorithm obtains a minimum $\rho \geq D$ that accrues the least total weight, hence $\sum_{i \in \mathcal{N}} w_{i,\ell^i} \leq \sum_{i \in \mathcal{N}} w_{i,\ell^{i*}}$. By the feasibility of optimal solutions $\sum_{i \in \mathcal{N}} \overline{v}_{i,\ell^{i*}} \geq D$, and since ρ is the least element that satisfies $\rho \geq D$, we have

$$\rho = \sum_{i \in \mathcal{N}} \overline{v}_{i,\ell^i} \geq \sum_{i \in \mathcal{N}} \overline{v}_{i,\ell^{i*}}. \tag{16}$$

Thus, by Eq. (16) and rounding values down (Line 4 of `Dyn-MinKS`), $\sum_{i \in \mathcal{N}} v_{i,\ell^i} \geq \sum_{i \in \mathcal{N}} \overline{v}_{i,\ell^i} \geq \sum_{i \in \mathcal{N}} \overline{v}_{i,\ell^{i*}} \geq \sum_{i \in \mathcal{N}} (v_{i,\ell^{i*}} - R_k) = \sum_{i \in \mathcal{N}} v_{i,\ell^{i*}} - |\mathcal{N}| R_k$, which completes the proof.

We define algorithm `dis-DynMDP` by replacing `Update` at Line 4 of `lim-DynMDP` by `KS-Update`, and using the following discretization factors,

$$L_k = \frac{\epsilon U_{\max}}{3(h-k)(\ln h + 1)} \quad \text{and} \quad R_k = \frac{L_k}{\gamma}. \tag{17}$$

Lemma III.7. *Let π be a policy obtained by* `dis-DynMDP` *and π^* be an optimal deterministic policy. The solution π satisfies $\overline{v}_\pi(s_k) \geq \overline{v}_{\pi^*}(s_k) - 2 \sum_{k'=k}^{h-1} \cdot L_{k'}$.*

Proof: We show (by induction) that for some $\ell_k \in \mathcal{L}_k$ and action a, we have $\ell_k = \overline{v}_\pi(s_k) \geq \overline{v}_{\pi^*}(s_k) - 2 \sum_{k'=k}^{h-1} \cdot L_{k'}$.

We proceed with the induction proof; for the base case, $k = h-1$, we have $\overline{v}_\pi(s_{h-1}) = \lfloor U(s_{h-1}, a)/L_{h-1} \rfloor L_{h-1}$. Clearly, there is an action that satisfies the claim. For the inductive step, suppose the claim holds at step k; we show that the claim also holds for step $k-1$.

Since the `dis-DynMDP` considers all possible values for ℓ_{k-1} at time $k-1$, there exists an ℓ_{k-1}, an action a, and a solution $\bar{\boldsymbol{\ell}}_k$ (Line 9 of `KS-Update`) such that,

$$\begin{aligned}\ell_{k-1} &= \Big\lfloor \tfrac{1}{L_{k-1}} \Big(\sum_{s_k^i \in N_a(s_{k-1})} T(s_{k-1}, a, s_k^i) \bar{\ell}_k^i \\ &\quad + U(s_{k-1}, a) \Big) \Big\rfloor L_{k-1} \\ &\geq \Big\lfloor \tfrac{1}{L_{k-1}} \Big(\sum_{s_k^i \in N_a(s_{k-1})} T(s_{k-1}, a, s_k^i) \bar{\ell}_k^{i*} - |N_a(s_{k-1})| R_k \\ &\quad + U(s_{k-1}, a) \Big) \Big\rfloor L_{k-1} \qquad (18)\end{aligned}$$

where Eq. (18) follows by Lemma III.6 (where $v_{i,j} := T(s_{k-1}, a, s_k^i) \cdot \overline{v}_\pi(s_k^i)$ and $w_{i,j} := T(s_{k-1}, a, s_k^i) \cdot \mathtt{DP}_{\mathrm{ER}}(s_k^i, \bar{\ell}_k^i)$ as per Line 6 of `KS-Update`). By the inductive assumption and Eq. (18),

$$\begin{aligned}\ell_{k-1} \geq \Big\lfloor \tfrac{1}{L_{k-1}} \Big(\sum_{s_k \in \mathcal{S}} T(s_{k-1}, a, s_k) \big(\overline{v}_{\pi^*}(s_k) - 2 \sum_{k'=k}^{h-1} L_{k'}\big) \\ - |\mathcal{N}_a(s_{k-1})| R_{k-1} + U(s_{k-1}, a) \Big) \Big\rfloor L_{k-1}\end{aligned}$$

Therefore,

$$\begin{aligned}\ell_{k-1} \geq \Big\lfloor \tfrac{1}{L_{k-1}} \Big(\sum_{s_k \in \mathcal{S}_k} T(s_{k-1}, a, s_k) \overline{v}_{\pi^*}(s_k) + U(s_{k-1}, a) \\ - 2 \sum_{k'=k}^{h-1} L_{k'} - \gamma R_{k-1} \Big) \Big\rfloor L_{k-1} \qquad (19)\end{aligned}$$

Thus, by the definition of R_k in Eq. (17), the r.h.s of Eq. (19) can be written as,

$$\begin{aligned}\ell_{k-1} \geq \Big\lfloor \tfrac{1}{L_{k-1}} \Big(\sum_{s_k \in \mathcal{S}_k} T(s_{k-1}, a, s_k) \overline{v}_{\pi^*}(s_k) + U(s_{k-1}, a) \\ - 2 \sum_{k'=k}^{h-1} L_{k'} - L_{k-1} \Big) \Big\rfloor L_{k-1} \qquad (20)\end{aligned}$$

$$\geq \overline{v}_{\pi^*}(s_k) - 2 \sum_{k'=k-1}^{h-1} L_{k'}, \qquad (21)$$

By the disjoint transition assumption (following the feasibility argument in the proof of Lemma III.2), policy π is feasible.

Corollary III.8. *Algorithm* `dis-DynMDP` *is an FPTAS for (C)C-MDP under disjoint transition assumption.*

Proof: First, observe that the algorithm maintains a dynamic programming table with minimum execution risk. Lines 2 of subroutine `Fetch-Policy` ensures that a feasible solution with such property is constructed. The algorithm runs in polynomial time as the sizes of $\mathcal{L}_k$ and $\mathcal{R}_k$ are polynomial. By Lemma III.3 and Lemma III.7, expanding for s_0, and by the definition of L_k and R_k, we obtain

$$\begin{aligned}\overline{v}_\pi(s_0) &\geq \overline{v}_{\pi^*}(s_0) - 2 \sum_{k=0}^{h-1} L_k \qquad (22) \\ &\geq v_{\pi^*}(s_0) - \sum_{k=0}^{h-1} L_k - 2 \sum_{k=0}^{h-1} L_k \qquad (23)\end{aligned}$$

Substituting L_k obtains,

$$\begin{aligned}&= v_{\pi^*}(s_0) - 3 \sum_{k=0}^{h-1} \frac{\epsilon U_{\max}}{3(h-k)(\ln h + 1)} \qquad (24) \\ &= v_{\pi^*}(s_0) - \frac{\epsilon U_{\max}}{\ln h + 1} \sum_{k=0}^{h-1} \frac{1}{(h-k)} \qquad (25)\end{aligned}$$

Using the upper bound on the harmonic series $\sum_{n=1}^{k} \frac{1}{n} \leq \ln k + 1$ obtains

$$\begin{aligned}\overline{v}_\pi(s_0) &\geq v_{\pi^*}(s_0) - \frac{\epsilon U_{\max}}{\ln h + 1} (\ln h + 1) \qquad (26) \\ &\geq (1-\epsilon) \cdot v_{\pi^*}(s_0), \qquad (27)\end{aligned}$$

Therefore, $v_\pi(s_0) \geq \overline{v}_\pi(s_0) \geq (1-\epsilon) v_{\pi^*}(s_0)$, which completes the proof.

C. FPTAS for (C)C-MDP under Local Transition

Algorithm 6: `DynMDP`$[M, \epsilon]$

1 $\mathtt{DP}_{\mathrm{ER}}(s_k, \ell_k) \leftarrow \infty$; $\mathtt{DP}_\pi(s_k, \ell_k) \leftarrow \varnothing$, $\mathtt{DP}_{\bar{\boldsymbol{\ell}}}(s_k, \ell_k) \leftarrow \mathbf{0}$, for all $k = 0, ..., h-1$; $s_k \in \mathcal{S}_k$; $\ell_k \in \mathcal{L}_k$
2 $\mathtt{DP}_{\mathrm{ER}}(s_h, 0) \leftarrow r(s_h)$
3 **for** $k = h-1, ..., 0$ **do**
4 $\mathcal{J}_k \leftarrow$ Partition states in $\mathcal{S}_k$ based on reachability given by Eqn. (28)
5 **for** $J_k \in \mathcal{J}_k$ **do**
6 **for** $\boldsymbol{\ell}_k = (\ell_k^c)_{s_k^c \in J_k} \in \mathcal{L}_k^{|J_k|}$ **do**
7 $\big(\mathtt{DP}_{\mathrm{ER}}(s_k^c, \ell_k^c), \mathtt{DP}_\pi(s_k^c, \ell_k^c), \mathtt{DP}_{\bar{\boldsymbol{\ell}}}(s_k^c, \ell_k^c)\big) \leftarrow$ `mKS-Update`$[J_k, \boldsymbol{\ell}_k, \{\mathtt{DP}_{\mathrm{ER}}(\cdot, \cdot)\}]$
8 **end for**
9 **end for**
10 **end for**
11 $\pi \leftarrow$ `Fetch-Policy`$[\mathtt{DP}]$
12 **return** π

To tackle (C)C-MDP with local transitions, we perform a tree decomposition: a transformation of the MDP graph into a tree where each node in the tree consists of a set of states. The tree nodes define a family of disjoint sets of states $\mathcal{J}_k \subseteq 2^{\mathcal{S}_k}$ at each time k (level in the MDP graph) as

$$\begin{aligned}\mathcal{J}_k := \{ J_k \subseteq \mathcal{S}_k \mid \textsc{Reach}(s_k) \cap \textsc{Reach}(s_k') \neq \varnothing, \\ \text{for any pair } s_k, s_k' \in J_k \}. \qquad (28)\end{aligned}$$

According to the local transition assumption, $|J_k|$ is at most a constant ψ, a property that is necessary to obtain a polynomial-time algorithm. Next, we show how to convert the allocation subproblem into a *multi-dimensional* version of McMinKS (denoted as MMcMinKS). MMcMinKS extends definition III.5, allowing the value of each item to be a $|J_k|$-dimensional vector. The goal is to compute the total minimum weight such that the total value for each dimension is at least $D^c \in \mathbb{R}_+$ for $c = \{1, ..., |J_k|\}$. More formally, the problem can be defined as an ILP by replacing Cons. (13) by $\sum_{i \in \mathcal{N}} \sum_{j \in \mathcal{M}_i} v_{i,j}^c x_{i,j} \geq D^c$, for $c = 1, ..., |J_k|$.

The key idea, presented in Alg. 6 (denoted by `DynMDP`), is to operate on clusters of states J_k instead of individual states as in `dis-DynMDP`. `mKS-Update` (Alg. 7) provides a reduction from the allocation problem into MMcMinKS. As we have a tree structure, the problem structure remains similar to `dis-DynMDP` except that here we require to solve an instance of multi-dimensional McMinKS. This can be done by a slight modification of `Dyn-MinKS`. The basic idea is to round off the set of possible values to obtain a range,

Algorithm 7: mKS-Update$\left[J_k, \boldsymbol{\ell}_k = (\ell_k^c)_{s_k^c \in J_k}, \{\text{DP}_{\text{ER}}(s, \ell)\}\right]$

$\text{ER}(s_k^c) \leftarrow \infty$ for $s_k^c \in J_k$
$\text{ACT} \leftarrow \varnothing$
$\text{ALLOC} \leftarrow \mathbf{0}$
for $\mathbf{a} = (a^c)_{s_k^c \in J_k} \in \mathcal{A}^{|J_k|}$ **do**
  // Find an allocation that achieves the minimum execution risk of states in J_k such that the total utility at each $s_k^c \in J_k$ is at least ℓ_k^c
  Let $\mathcal{N} := \bigcup_{s_k^c \in J_k} N_{a^c}(s_k^c)$
  Let $\mathcal{M}_i := \mathcal{L}_{k+1}$ for $i \in \mathcal{N}$
  Let $w_{i,j} := \sum_{s_k^c \in J_k} T(s_k^c, a^c, s_{k+1}^i) \cdot \text{DP}_{\text{ER}}(s_{k+1}^i, \overline{\ell}_{k+1}^i)$ for all $j = \overline{\ell}_{k+1}^i \in \mathcal{M}_i$ and i such that $s_{k+1}^i \in N_a(s_k^c)$
  for $s_k^c \in J_k$ **do**
    Let $v_{i,j}^c := T(s_k^c, a^c, s_{k+1}^i) \cdot \overline{\ell}_{k+1}^i$ for all $j = \overline{\ell}_{k+1}^i \in \mathcal{M}_i$ and i such that $s_{k+1}^i \in N_a(s_k^c)$
    Let $D^c := \ell_k^c - U(s_k^c, a^c)$
  end for
  $(\overline{\boldsymbol{\ell}}_{k+1}^i)_{i \in \mathcal{N}} \leftarrow$ Solve MMcMinKS$[\{(w_{i,j}, v_{i,j}^c)_{i \in \mathcal{N}, j \in \mathcal{M}_i}, D^c\}_{s_k^c \in J_k}]$
  $\text{ER}_{a^c}(s_k^c, \overline{\boldsymbol{\ell}}_{k+1}) := r(s_k^c) + (1 - r(s_k^c)) \sum_{s_{k+1}^i \in N_{a^c}(s_k^c)} T(s_k^c, a^c, s_{k+1}^i) \cdot \text{DP}_{\text{ER}}(s_{k+1}^i, \overline{\ell}_{k+1}^i)$ for $s_k^c \in J_k$
  if $\sum_{s_k^c \in J_k} \text{ER}_{a^c}(s_k^c, \overline{\boldsymbol{\ell}}_{k+1}) < \sum_{s_k^c \in J_k} \text{ER}(s_k^c)$ **then**
    for $s_k^c \in J_k$ **do**
      $\text{ER}(s_k^c) \leftarrow \text{ER}_{a^c}(s_k^c, \overline{\boldsymbol{\ell}}_{k+1})$
      $\text{ACT}^c \leftarrow a^c$
      $\text{ALLOC}^c \leftarrow (\overline{\boldsymbol{\ell}}_{k+1}^i)_{s_{k+1}^i \in N_{a^c}(s_k^c)}$
    end for
  end if
end for
return $(\text{ER}(s_k^c), \text{ACT}^c, \text{ALLOC}^c)_{s_k^c \in J_k}$

by which we can optimize over in polynomial time using dynamic programming. Thus, we create $|J_k|$ dimensional dynamic programming table $\text{TB}(j, \rho^1, ..., \rho^{|J_k|})$. Since $|J_k| \leq \psi$ is a constant, the size of the table is polynomial in the input size.

Theorem III.9. *Algorithm* DynMDP *is an FPTAS for (C)C-MDP under local transition assumption.*

A proof of the theorem can be obtained using that of Corollary III.8 with a slight modification of Lemma III.6 to account for a higher dimensional dynamic programming table.

IV. Conclusion

This work provides the first fully polynomial-time approximation scheme for a class of constrained MDP under local transition. Since the problem is NP-Hard, our algorithm is the best polynomial-time approximation algorithm attainable in theory. We believe our results provide fundamental insights into the problem and can lead to the future development of algorithms and faster heuristics for (C)C-MDP and constrained reinforcement learning.

References

[1] M. Ahmadi, U. Rosolia, M. D. Ingham, R. M. Murray, and A. D. Ames. Constrained risk-averse markov decision processes. In *The 35th AAAI Conference on Artificial Intelligence (AAAI-21)*, 2021.

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

[3] R. Alyassi and M. Khonji. Dual formulation for chance constrained stochastic shortest path with application to autonomous vehicle behavior planning. In *2021 60th IEEE Conference on Decision and Control (CDC)*, pages 4486–4492. IEEE, 2021.

[4] C. Bentz and P. L. Bodic. A note on" approximation schemes for a subclass of subset selection problems", and a faster fptas for the minimum knapsack problem. *arXiv preprint arXiv:1607.07950*, 2016.

[5] D. P. Bertsekas and J. N. Tsitsiklis. An analysis of stochastic shortest path problems. *Mathematics of Operations Research*, 16(3):580–595, 1991.

[6] B. Bonet and H. Geffner. Labeled rtdp: Improving the convergence of real-time dynamic programming. In *ICAPS*, volume 3, pages 12–21, 2003.

[7] C.-K. Chau, K. Elbassioni, and M. Khonji. Truthful mechanisms for combinatorial allocation of electric power in alternating current electric systems for smart grid. *ACM Transactions on Economics and Computation (TEAC)*, 5(1):1–29, 2016.

[8] F. De Nijs, E. Walraven, M. de Weerdt, and M. Spaan. Bounding the probability of resource constraint violations in multi-agent mdps. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31, 2017.

[9] F. de Nijs, E. Walraven, M. De Weerdt, and M. Spaan. Constrained multiagent markov decision processes: a taxonomy of problems and algorithms. *Journal of Artificial Intelligence Research*, 70:955–1001, 2021.

[10] F. d'Epenoux. A probabilistic production and inventory problem. *Management Science*, 10(1):98–108, 1963.

[11] D. Ding, K. Zhang, T. Basar, and M. Jovanovic. Natural policy gradient primal-dual method for constrained markov decision processes. *Advances in Neural Information Processing Systems*, 33:8378–8390, 2020.

[12] D. A. Dolgov and E. H. Durfee. Approximating optimal policies for agents with limited execution resources. In *IJCAI*, pages 1107–1112, 2003.

[13] E. A. Feinberg. Constrained discounted markov decision processes and hamiltonian cycles. *Mathematics of Operations Research*, 25(1):130–140, 2000.

[14] E. A. Feinberg and A. Shwartz. Constrained discounted dynamic programming. *Mathematics of Operations Research*, 21(4):922–945, 1996.

[15] V. Freire, K. V. Delgado, and W. A. S. Reis. An exact algorithm to make a trade-off between cost and probability in ssps. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 29, pages 146–154, 2019.

[16] E. A. Hansen and S. Zilberstein. Lao*: A heuristic search algorithm that finds solutions with loops. *Artificial Intelligence*, 129(1-2):35–62, 2001.

[17] S. Hong, S. U. Lee, X. Huang, M. Khonji, R. Alyassi, and B. Williams. An anytime algorithm for chance constrained stochastic shortest path problems and its application to aircraft routing. In *ICRA*. https://bit.ly/2Pb9LPr/, 2021.

[18] R. A. Howard. Dynamic programming and markov processes. 1960.

[19] X. Huang, S. Hong, A. Hofmann, and B. C. Williams. Online risk-bounded motion planning for autonomous vehicles in dynamic environments. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 29, pages 214–222, 2019.

[20] H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack Problems*. Springer, 2004.

[21] M. Khonji, A. Jasour, and B. Williams. Approximability of constant-horizon constrained pomdp. In *IJCAI*, pages 5583–5590, 2019.

[22] M. Ono, M. Pavone, Y. Kuwata, and J. Balaram. Chance-constrained dynamic programming with application to risk-aware robotic space exploration. *Autonomous Robots*, 39(4):555–571, 2015.

[23] K. Pruhs and G. J. Woeginger. Approximation schemes for a class of subset selection problems. *Theoretical Computer Science*, 382(2):151–156, 2007.

[24] P. Santana, S. Thiébaux, and B. Williams. Rao*: an algorithm for chance constrained pomdps. In *Proc. AAAI Conference on Artificial Intelligence*, 2016.

[25] F. Trevizan, S. Thiébaux, P. Santana, and B. Williams. Heuristic search in dual space for constrained stochastic shortest path problems. In *Twenty-Sixth International Conference on Automated Planning and Scheduling*, 2016.

[26] V. V. Vazirani. *Approximation algorithms*. Springer Science & Business Media, 2013.

[27] J. Wei, J. M. Snider, T. Gu, J. M. Dolan, and B. Litkouhi. A behavioral planning framework for autonomous driving. In *2014 IEEE Intelligent Vehicles Symposium Proceedings*, pages 458–464. IEEE, 2014.