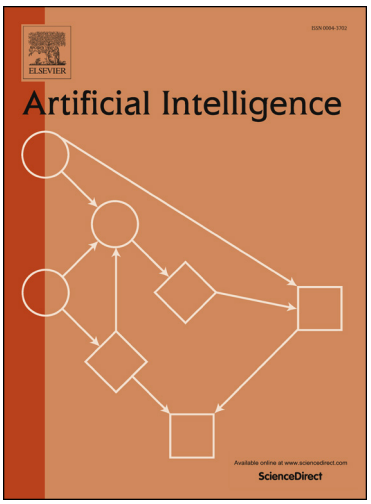
Journal Pre-proof



An Anytime Algorithm For Constrained Stochastic Shortest Path Problems With Deterministic Policies

Sungkweon Hong and Brian C. Williams

PII: S0004-3702(22)00186-2
DOI: https://doi.org/10.1016/j.artint.2022.103846
Reference: ARTINT 103846

To appear in: *Artificial Intelligence*

Received date: 14 October 2021
Revised date: 9 December 2022
Accepted date: 23 December 2022

Please cite this article as: S. Hong and B.C. Williams, An Anytime Algorithm For Constrained Stochastic Shortest Path Problems With Deterministic Policies, *Artificial Intelligence*, 103846, doi: https://doi.org/10.1016/j.artint.2022.103846.

An Anytime Algorithm For Constrained Stochastic Shortest Path Problems With Deterministic Policies

Sungkweon Hong[a,*], Brian C. Williams[a]

[a] *Massachusetts Institute of Technology, Cambridge, MA, USA*

Abstract

Sequential decision-making problems arise in every arena of daily life and pose unique challenges for research in decision-theoretic planning. Although there has been a wide variety of research in this field, most of the studies have largely focused on single objective problem without constraints. In many real-world applications, however, it is often desirable to bound certain costs or resources under some predefined level. Constrained stochastic shortest path problem (C-SSP), one of the most well-known mathematical frameworks for stochastic decision-making problems with constraints, can formally model such problems, by incorporating constraints in the model formulation. However, it remains an open challenge to produce a deterministic optimal policy with desirable computation time due to its intrinsic complexity.

In this paper, we propose a method that produces an optimal and deterministic policy for a C-SSP based on the Lagrangian duality theory and the heuristic forward search method. To address the intrinsic complexity of C-SSP, the proposed method is designed to have an anytime property. In other words, the proposed algorithm tries to find a feasible but decent solution quickly, then improves the solution incrementally until it converges to a true optimal solution. An extensive experimental evaluation on three problem domains shows that the proposed method outperforms the state-of-the-art methods in terms of

*Corresponding author

Email addresses: sk5050@mit.edu (Sungkweon Hong), williams@mit.edu (Brian C. Williams)

the near-optimal solution with an optimality gap of less than 0.1%.

Keywords: Constrained stochastic shortest path problem, risk-bounded planning, heuristic search

1. Introduction

Stochastic shortest path problem (SSP) is one of the most popular mathematical modeling frameworks for sequential decision-making problems under uncertain environments [1]. The solution of an SSP problem is a policy that minimizes the expected cost. In many of the real-world applications, however, naively minimizing the expected cost might not be desirable. For example, when we want an unmanned aerial vehicle (UAV) to navigate to the goal as fast as possible, it is also important to ensure the vehicle can complete the mission safely by staying outside of the unsafe regions. Moreover, many planning problems have not single but multiple constraints. In the previous UAV example, avoiding unsafe region is not enough and the UAV also should be able to complete the mission with the available battery for safety.

Constrained SSP (C-SSP), an extension of SSP, models such problems by having constraints on the secondary cost functions. Due to its generality, C-SSP has been widely studied and used in many practical applications, including telecommunication networks and resource management [2]. However, one of the major differences between SSP and C-SSP is that every optimal policy for C-SSP might be stochastic [2]. Although stochastic policies could be mathematically optimal, they are often undesirable in many real-world applications due to lack of reliability [3, 4, 5]. Executing a stochastic policy requires the system to decide an action based on a dice roll. This can raise questions about not only accountability but also explainability, especially in the safety-critical applications such as autonomous driving. In addition, Paruchuri et al. [6] described the difficulty of using stochastic policies in multi-agent settings without additional communications. Commercial aircraft operation is a representative example where multiple aircraft are operated by different stakeholders, such

as flight crews, airline ground center and air traffic control agents. Since a predictable and harmonized plan is crucial in this coordinated setting, it is impractical to use stochastic policies for aircraft routing [4, 5]. Therefore, it is appealing to produce deterministic policies; however, this significantly increases the complexity of the problem. Feinberg showed that finding an optimal deterministic policy for a C-SSP is NP-complete [7].

In this paper, our main contribution is an anytime algorithm that guarantees to produce a deterministic optimal policy for a C-SSP. Given the fact that solving the problem exactly is highly difficult, we focus on developing an algorithm that produces suboptimal but feasible solutions quickly, and is able to obtain an optimal solution eventually given enough time. Such anytime property is extremely important in the time-critical applications where constraints are safety-related. In those applications, it is highly desirable to have a feasible but decent solution quickly and we can improve the solution if time allows.

To accomplish such property, we propose a two-stage approach. In the first stage, we solve the Lagrangian dual of the original C-SSP. Then, if there is a duality gap, we obtain an optimal solution by closing the gap in the second stage. The crucial idea that makes our algorithm efficient is, throughout the process, using a heuristic forward search to avoid exploring the entire search space. In addition, we provide a provable approximation scheme that can further improve the performance of the algorithm at the cost of small efficiency loss.

Finally, we demonstrate our proposed algorithm in three different domains including widely used benchmark problems and a risk-bounded aircraft routing problem under uncertain weather conditions. For the evaluation, we compare our proposed anytime algorithm with MILP-based algorithm [3] and a variant of i-dual [8, 9].

1.1. Related work

The most widely used approach to a C-SSP is reducing it as a linear programming problem based on occupation measure [2, 10], similar to the LP-based approach for unconstrained SSP, which can be solved in polynomial time. Re-

cently, Trevizan et al. [8, 9] proposed a more efficient algorithm for C-SSP called i-dual which exploits a heuristic search. Instead of encoding and solving the entire search space of a C-SSP, as in [2] and [10], i-dual incrementally enlarges search spaces based on heuristics and solves sequence of LPs on those restricted search spaces, until the solution is found. They proved that the algorithm is optimal if the heuristics used in the algorithm are admissible. They also showed that i-dual dominates the LP-based method in most of their test cases, and the scalability could be even more improved with non-admissible heuristics at the cost of completeness and optimality.

Since the performance of i-dual is highly dependent on the quality of heuristics, Trevizan et al. [11] and Baumgartner et al. [12] introduced occupation measure heuristics for probabilistic planning problem when there is knowledge of its factored representation based on probabilistic SAS$^+$ [13]. In addition, they introduced i^2-dual, that couples i-dual and occupation measure heuristics, and showed it performed better than i-dual with well-known domain-independent heuristics such as determinization-based heuristics.

As mentioned before, however, every optimal policy for C-SSP might be stochastic [2], which is not desirable in many real-world applications due to practical reasons such as lack of reliability and difficulty in coordination [3, 4, 5, 6]. As an attempt to produce an optimal deterministic policy for a C-SSP, Dolgov and Durfee [3] proposed a mixed integer linear programming (MILP) based approach. Although theoretical complexity remains the same as proved in [7], the MILP-based approach can be solved by highly optimized commercial solvers and showed reasonable performance in their empirical results. However, it still suffers from high computational complexity, making it difficult to use in online planning, as we show in our experiments.

Taking another approach, there have been attempts to solve stochastic decision making problems with constraints using heuristic forward search methods. Pedro et al. [14] introduced an algorithm called RAO* to solve finite-horizon risk-bounded partially observable Markov decision process (POMDP) with deterministic policies. RAO* is basically based on AO* [15], but it keeps evalu-

ating a remaining risk budget that has not been used so far and an admissible risk that appears in the future so that it can prune overly risky parts of the tree. Although the pruning enables RAO* to find a risk-bounded policy fast, it is also possible to overly prune the search tree which results in suboptimality. In addition, the algorithm cannot deal with multiple constraints or a loop in the search graph which makes it limited to be applied to general C-SSP.

Motivated by RAO*, another heuristic forward search-based algorithm has been developed for finite-horizon risk-bounded SSP with deterministic policies [5]. Their work is a predecessor of this work in the sense that both algorithms share the same structure: the first stage solves the Lagrangian dual and the second stage closes the duality gap, if it exists, where heuristic forward search is used for both stages. However, the algorithm proposed in [5] is limited to finite-horizon without a loop in the search graph and also to a single constraint case, whereas this work extends it to general C-SSP.

To summarize, our approach differs from the LP-based methods, including i-dual and i^2-dual, in that it guarantees to produce a deterministic optimal policy for a C-SSP where the solutions produced by LP-based methods might be stochastic. Our approach also differs from MILP-based method in several different points. First, instead of solving C-SSP directly, the proposed method firstly solves an approximated problem of C-SSP based on Lagrangian dual and incrementally closes the gap, if it exists, which naturally gives an anytime property. In addition, the proposed approach uses a heuristic forward search to avoid exploring the entire search space. The latter point relates this work to recent works on risk-bounded probabilistic planning algorithms. However, this work generalizes the previous works by allowing the method to consider multiple constraints and loops in the solution graph. Finally, we provide an approximation scheme to scale the algorithm at the cost of optimality.

1.2. Organization

The remainder of this paper is organized as follows. Section 2 summarizes the definitions and formalism. Section 3 elaborates our proposed algorithm for

overview and explanation of two stages of the algorithm, respectively. Section 4 shows an approximation scheme that can be applied to our algorithm. Evaluation results are reported and analyzed in Section 5. Finally, we conclude and present future works in Section 6.

2. Background

2.1. Stochastic shortest path problem

A *stochastic shortest path problem* (SSP) is a tuple $\mathcal{P} = \langle \mathcal{S}, \bar{s}, \mathcal{G}, \mathcal{A}, T, C \rangle$ in which $\mathcal{S}$ is a set of finite states; $\bar{s} \in \mathcal{S}$ is the initial state; $\mathcal{G} \subset \mathcal{S}$ is a set of goal states; $\mathcal{A}$ is a set of finite actions; $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is the state transition function, where $T(s, a, s') = Pr(s'|s, a)$ is the probability of being in state s' after executing action a in the state s; $C : \mathcal{S} \times \mathcal{A} \to \mathbb{R}_+$ is the cost function, where $C(s, a)$ is the cost of executing action a in state s [1]. Note that each goal state is absorbing and cost-free, i.e., $T(s, a, s) = 1$ and $C(s, a) = 0$ for every $s \in \mathcal{G}$ and $a \in \mathcal{A}(s)$.

A solution to an SSP is a policy $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$ which maps states to a probability distribution over actions. If a policy maps every state to a probability distribution with a single outcome, the policy is called *deterministic*, and is called *stochastic* otherwise. In the former case, we denote $\pi(s)$ as the action selected by the policy π for the state s. In addition, a policy is called *proper* if it reaches a goal state with probability 1, and is called *improper* otherwise.

For a policy π, let

$$V^{\pi}(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} C(s_t, a_t) \middle| s_0 = s, \pi\right] \tag{1}$$

denote the value function of π that provides the total expected cost from s until one of the goals $\mathcal{G}$ is reached. Note that we assume that there is at least one proper policy. In addition, we assume that every improper policy yields infinite cost, i.e., $V^{\pi}(\bar{s}) = \infty$ for any improper policy π. Then, an optimal solution to an SSP is a policy π^* which minimizes the value function evaluated at the initial state $\bar{s}$, i.e.,

$$\pi^* = \underset{\pi \in \Pi}{\operatorname{argmin}}\, V^{\pi}(\bar{s})$$

where Π is a set of policies. Note that we denote V^* as an optimal value function

V^{π^*} which is given by the well-known *Bellman equation*:

$$V^*(s) = \begin{cases} 0 & \text{if } s \in \mathcal{G}, \\ \min_{a \in \mathcal{A}(s)} \left[C(s,a) + \sum_{s' \in \mathcal{S}} T(s,a,s') \cdot V^*(s') \right] & \text{otherwise.} \end{cases}$$

2.2. Constrained stochastic shortest path problem

A *constrained stochastic shortest path problem* (C-SSP) is a tuple $\mathcal{H} = \langle \mathcal{S}, \bar{s}, \mathcal{G}, \mathcal{A}, T, \vec{C}, \vec{\Delta} \rangle$ in which $\mathcal{S}$ is a set of finite states; $\bar{s} \in \mathcal{S}$ is the initial state; $\mathcal{G} \subset \mathcal{S}$ is a set of goal states; $\mathcal{A}$ is a set of finite actions; $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is the state transition function, where $T(s, a, s') = Pr(s'|s, a)$ is the probability of being in state s' after executing action a in state s; $\vec{C}$ is the indexed set of cost functions $\{C_0, ..., C_N\}$, where each $C_i : \mathcal{S} \times \mathcal{A} \to \mathbb{R}_+$ is the cost function and $C_i(s, a)$ is the i-th cost of executing action a in the state s; and $\vec{\Delta}$ is the indexed set of bounds $\{\Delta_1, \ldots, \Delta_N\}$, where Δ_i is the bound for i-th cost function for $i = 1, \ldots, N$ [2, 8]. Note that each goal state is absorbing and cost-free, i.e., $T(s, a, s) = 1$ and $C_i(s, a) = 0$ for every $s \in \mathcal{G}$, $a \in \mathcal{A}(s)$ and $i = 0, \ldots, N$. Also note that we refer to C_0 as the primary cost function and the rest as the secondary cost functions.

The UAV operation we introduced earlier is one of the examples of C-SSP, where we want the UAV to navigate to the goal as fast as possible while maintaining safety by avoiding unsafe regions. Suppose the set of states are joint positions of UAV and unsafe regions. Then each action takes the UAV from one waypoint to one of its adjacent waypoints while each unsafe region, such as a convective weather cell, stochastically moves based on the transition function. Then, since we want to minimize the flight time, the primary cost function can be defined as the flight time associated with each navigation action. Finally, the secondary cost function maps the state and action to the impact of encountering an unsafe region, which is upper bounded by the user-defined safety threshold Δ. Note that it is usually not trivial to define the secondary cost function that measures the impact of a safety violation, which will be discussed in Section 5.2.3 more in detail.

Similar to SSP, we assume that there exists a proper policy and every improper policy yields infinite primary cost. Then, an optimal solution to a C-SSP is a policy π^* which minimizes the total expected primary cost, but in addition, the expected cost of the secondary cost function C_i should be upper bounded by Δ_i for $i = 1, \ldots, N$. In other words, an optimal policy π^* for a C-SSP is defined as follows:

$$\pi^* = \underset{\pi \in \Pi}{\operatorname{argmin}} \mathbb{E}\left[\sum_{t=0}^{\infty} C_0(s_t, a_t) \middle| s_0 = \bar{s}, \pi\right]$$

subject to

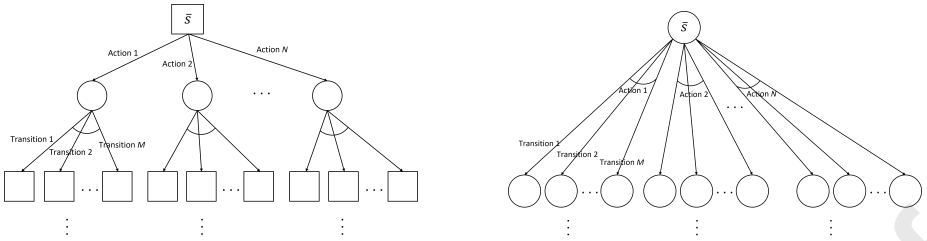
$$\mathbb{E}\left[\sum_{t=0}^{\infty} C_i(s_t, a_t) \middle| s_0 = \bar{s}, \pi\right] \leq \Delta_i,$$

for every $i = 1, \ldots, N$, where Π is the set of all policies. Note that every optimal policy for a C-SSP might be stochastic [2]. In this paper, however, we limit the policy space to deterministic ones as discussed in Section 1.

*2.3. (L)AO**

An SSP can be represented as an AND/OR graph as illustrated in Fig. 1a [15, 16] where the root node is the initial state $\bar{s}$. Each state, represented as a square in Fig. 1a, is an OR-node that can choose from different actions. On the other hand, each AND-node, represented as a circle in Fig. 1a, is an intermediate node that splits to different states reachable based on transition function. Note that each edge from an AND-node is a weighted edge where the weight is a probability associated with the transition. A policy of an SSP is a sub-graph with $\bar{s}$ as the root node, in which each OR-node activates one of its children and each activated AND-node activates all of its children, recursively, until all tip nodes are goal states. An equivalent representation using hyperedges is illustrated in Fig. 1b.

AO* is a heuristic search algorithm that finds an optimal solution to a problem that can be represented as an AND/OR graph [15]. AO* is one of the dynamic programming algorithms and it estimates value function V^* by incrementally expanding the graph guided by heuristics. AO* starts at the root node



(a) AND/OR graph representation.

(b) Hyperedges representation.

Figure 1: Graph representations of an SSP.

and alternates between two main steps: expansion and backup. In the expansion step, it expands one of the tip nodes in the current best solution graph. In the backup step, it updates values for the expanded node and its ancestors, by evaluating different actions, marking the best actions for the OR-nodes (or hyperedges) with the minimal expected cost, and updating the cost values based on the marked actions. The current best solution graph is then updated by following the marked actions from the root node. Finally, the algorithm terminates when all the tip nodes in the current best solution graph are terminal nodes. Similar to A* algorithm, AO* estimates the values of non-terminal tip states based on heuristic function h. Then, AO* is optimal, given that the heuristic function is admissible, i.e.,

$$h(s) \leq V^*(s)$$

for every $s \in \mathcal{S}$, similar to the A* algorithm [16].

However, AO* can only be applied to an acyclic AND/OR graph, which is a very strong assumption on the SSP. LAO*, a variant of AO*, is a heuristic algorithm that generalizes AO* to cyclic AND/OR graphs and has been widely used to solve SSPs [16]. To deal with a cycle, LAO* combines AO* with dynamic programming algorithms, such as the value iteration (VI) or policy iteration. In other words, LAO* expands nodes in the same manner as AO*, but instead of update values and best solution graph by backward induction, it performs a dynamic programming algorithm such as the VI on the set of states that might

Algorithm 1: LAO*

Input: SSP instance $\mathcal{P}$, VI error tolerance η

initialize AND/OR graph G with initial state $\bar{s}$
repeat
  perform postorder traversal for G and for each visited state s,
  if *s has not been expanded* **then**
    expand s and for each new state s' added to G,
    if $s' \in \mathcal{G}$ **then**
      $V(s') = 0$
    else
      $V(s') = h(s)$
  else
    $a^* \leftarrow \text{argmin}_{a \in \mathcal{A}(s)} \left[C(s,a) + \sum_{s'} T(s,a,s') \cdot V(s') \right]$
    mark a^* as the best action of s
    $V(s) \leftarrow C(s,a^*) + \sum_{s'} T(s,a^*,s') \cdot V(s')$
until *LAO*-Convergence-Test($\mathcal{P}, G, \eta$)*

be affected by the recently expanded node.

In fact, LAO* is a class of algorithms, and has several different variations. One of the most efficient and widely used versions of the LAO* is the one introduced in its original paper [16]. This version of LAO* reduces the number of dynamic programming algorithm runs by combining LAO* and depth-first search. Throughout this paper, LAO* refers to this version of the algorithm together with the VI as a dynamic programming algorithm, which is shown in Algorithm 1 and 2. Please refer to [16] for more details and grounded examples of LAO*.

2.4. Lagrangian Dual

In this section, we briefly review Lagrangian duality theory that is used for our proposed method. Although Lagrangian dual is general, we particularly

Algorithm 2: LAO*-Convergence-Test

Input: SSP instance $\mathcal{P}$, AND/OR graph G, VI error tolerance η

repeat
  if *best solution graph of* G *has unexpanded tip state* **then**
    return false
  $E \leftarrow 0$
  for s *in best solution graph* **do**
    $v \leftarrow V(s)$
    $a^* \leftarrow \text{argmin}_{a \in \mathcal{A}(s)} \left[C(s,a) + \sum_{s'} T(s,a,s') \cdot V(s') \right]$
    mark a^* as the best action of s
    $V(s) \leftarrow C(s,a^*) + \sum_{s'} T(s,a^*,s') \cdot V(s')$
    $E \leftarrow \max(E, |v - V(s)|)$
  update best solution graph of G
until $E < \eta$
return true

focus on Lagrangian dual for an optimization problem with a finite domain which is defined as follows:

$$\begin{aligned} \text{(P)} \quad & \min_{x \in \mathcal{X}} \quad f(x) \\ & \text{s.t.} \quad g_i(x) \leq 0 \quad \text{for } i = 1, \ldots, m, \end{aligned}$$

where $\mathcal{X}$ is a finite set, $f : \mathcal{X} \to \mathbb{R}$, $g_i : \mathcal{X} \to \mathbb{R}$ for $i = 1, \ldots, m$.

First, we define the Lagrangian function as follows.

Definition 2.1. (Lagrangian function) Consider the optimization problem (P). For $\lambda \in \mathbb{R}^m_+$ and $x \in \mathcal{X}$, the *Lagrangian function* is defined as follows:

$$L(\lambda, x) = f(x) + \lambda^\top \cdot g(x),$$

where $g = [g_1, \ldots, g_m]^\top$ and $\lambda = [\lambda_1, \ldots, \lambda_m]^\top \in \mathbb{R}^m_+$, and λ is said to be the *Lagrangian multiplier*.

Note that a Lagrangian function for any $x \in \mathcal{X}$ is linear in Lagrangian multiplier. Next, we define a relaxed problem of (P) with given Lagrangian multiplier as follows.

Definition 2.2. (Lagrangian relaxation) Consider an optimization problem (P). For some $\lambda \in \mathbb{R}_+^m$, consider another optimization problem that is defined as follows:

$$L(\lambda) = \min_{x \in \mathcal{X}} L(\lambda, x)$$

where $L(\lambda, x)$ is the Lagrangian function. Then, $L(\lambda)$ is said to be the Lagrangian relaxation of (P) with Lagrangian multiplier λ.

The Lagrangian relaxation can be thought of as a penalty version of the original problem where the second term of the Lagrangian function acts as a penalty for constraint violation with non-negative Lagrangian multiplier. In addition, $L(\lambda)$ is the pointwise minimum of finite linear functions, hence it is a piecewise linear concave function [17], which is shown in the following proposition.

Proposition 2.1. *Consider an optimization problem (P) and its Lagrangian functions $L(\lambda, x)$. Suppose $L(\lambda) = \min_{x \in \mathcal{X}} L(\lambda, x)$. Then $L(\lambda)$ is a piecewise linear concave function of λ.*

Proof. The concavity of $L(\lambda)$ can be shown as follows with any $\alpha \in [0, 1]$ and $\lambda_1, \lambda_2 \in \mathbb{R}_+^m$:

$$\begin{aligned} L(\alpha\lambda_1 + (1-\alpha)\lambda_2) &= L(\alpha\lambda_1 + (1-\alpha)\lambda_2, x) \quad \text{for some } x \in \mathcal{X} \\ &= \alpha L(\lambda_1, x) + (1-\alpha) L(\lambda_2, x) \\ &\geq \alpha \min_{x' \in \mathcal{X}} L(\lambda_1, x') + (1-\alpha) \min_{x' \in \mathcal{X}} L(\lambda_2, x') \\ &= \alpha L(\lambda_1) + (1-\alpha) L(\lambda_2). \end{aligned}$$

To show that $L(\lambda)$ is piecewise linear, we need to show that there is a finite family $\mathcal{Q}$ of closed domains such that $\mathbb{R}_+^m = \bigcup \mathcal{Q}$ and $L(\lambda)$ is linear on every domain in $\mathcal{Q}$ [18]. Let Q_x for $x \in \mathcal{X}$ be a subset of $\mathbb{R}_+^m$ such that $L(\lambda) = L(\lambda, x)$, i.e., $Q_x = \{\lambda \in \mathbb{R}_+^m \mid L(\lambda) = L(\lambda, x)\}$. Let $\mathcal{Q} = \{Q_x\}_{x \in \mathcal{X}}$. Then $\mathcal{Q}$ is finite

and $\mathbb{R}^m_+ = \bigcup \mathcal{Q}$, where the former is due to the finiteness of $\mathcal{X}$ and the latter is due to the definition of $L(\lambda)$. Now, we show that each $Q_x \in \mathcal{Q}$ is a convex region by contradiction. For this, let $q_1, q_2 \in Q_x$ and suppose there exists $q_3 = \alpha q_1 + (1-\alpha)q_2$ for some $\alpha \in [0,1]$ such that $q_3 \notin Q_x$ and $q_3 \in Q_y$ for $Q_y \in \mathcal{Q}$ that is different from Q_x. Then,

$$\begin{aligned}
L(q_3, y) &= \min_{x' \in \mathcal{X}} L(q_3, x') \\
&< L(q_3, x) \\
&= L(\alpha q_1 + (1-\alpha)q_2, x) \\
&= \alpha L(q_1, x) + (1-\alpha) L(q_2, x) \\
&= \alpha \min_{x' \in \mathcal{X}} L(q_1, x') + (1-\alpha) \min_{x' \in \mathcal{X}} L(q_2, x') \\
&\leq \alpha L(q_1, y) + (1-\alpha) L(q_2, y) \\
&= L(q_3, y),
\end{aligned}$$

which is a contradiction.

Finally, a boundary point p of a set Q_x either has 0 on some coordinate or at the intersection of $L(\lambda, x)$ and $L(\lambda, y)$ for some $y \in \mathcal{X}$. First, note that $L(\lambda, x)$ is well-defined for λ with 0 component. In addition, if p is at the intersection of $L(\lambda, x)$ and $L(\lambda, y)$ then $p \in Q_x$ and $p \in Q_y$ since $L(\lambda)$ is continuous. Therefore, Q_x is a closed convex region which completes the proof.

□

Finally, the Lagrangian dual is defined as follows.

Definition 2.3. (Lagrangian dual) Consider an optimization problem (P). Then

$$L^* = L(\lambda^*) = \max_{\lambda \in \mathbb{R}^m_+} L(\lambda) \tag{2}$$

is said to be the Lagrangian dual of (P).

Note that, due to Proposition 2.1, the Lagrangian dual problem is a piecewise linear concave optimization, which also can be interpreted as finding the tightest

possible lower bound on the optimal cost f^*. In addition, the Lagrangian dual has the following property:

$$L^* \leq f^*, \tag{3}$$

which is known as *weak duality* [17]. Note that it is called *strong duality* if the equality holds, which does not generally hold for discrete optimization such as (P).

Fig. 2 illustrates Lagrangian dual for an example of (P) with single constraint ($m = 1$), where x and $y-$axes represent Lagrangian multiplier and Lagrangian function value, respectively. In addition, each line shows Lagrangian function for $x \in \mathcal{X}$, where $y-$intercept and gradient are $f(x)$ and $g(x)$, respectively, so that green solid and red dotted lines are Lagrangian functions with feasible and infeasible solutions, respectively. Finally, bold lines show $L(\lambda)$, which is piecewise linear concave, with dual optimum L^* and associated Lagrangian multiplier λ^*.

By noting that any feasible solution has negative gradient, the weak duality can be visually proved. Given dual optimum L^*, none of the feasible solutions can have $f(x)$ ($y-$intercept) lower than L^* without intercepting piecewise linear concave function, in which case the assumption of dual optimum L^* is violated. The example also shows a counter-example of strong duality since $L^* < f^*$. In addition, the feasible solution associated with the dual optimum (denoted as x' in the figure) has larger cost than the optimal solution x^*.

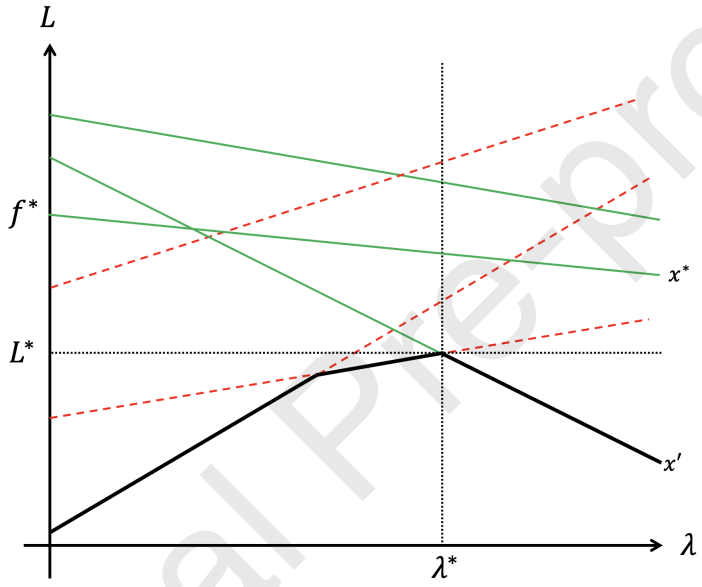


Figure 2: (λ, L) space representing solutions for an optimization problem (P) with a single constraint. Each line represents Lagrangian function of a solution x where green solid lines and red dotted lines show feasible and infeasible solutions, respectively. In addition, $y-$intercept and gradient of a line are $f(x)$ and $g(x)$, respectively, and $L(\lambda)$ is shown as bold lines.

3. Anytime Algorithm for C-SSP

In this section, we introduce an anytime algorithm that finds an optimal deterministic policy for a C-SSP. The algorithm leverages two main theories, which are Lagrangian duality theory and heuristic forward search. In the proposed method, we assume that admissible heuristics for costs are available. Note that the method is still valid even though there is no available heuristics because the "zero heuristic" can always be an admissible heuristic.

The proposed algorithm has two stages. In the first stage, we Lagrangianize the constraints on the secondary costs of a C-SSP, then obtain an optimal solution to the Lagrangian dual. Then in the second stage, we reduce a duality gap from the dual optimum, if the gap exists, by incrementally enumerating k-best policies in terms of a Lagrangian function, until it converges to a primal optimal solution. We begin this section with an overview of the proposed algorithm, then present technical details of the algorithm in the later sections more in detail.

3.1. Overview of the algorithm

A C-SSP is an optimization problem that can be formulated as follows:

$$\begin{aligned} \min_{\pi \in \Pi} \quad & f(\pi) \\ \text{s.t.} \quad & g_i(\pi) \leq 0 \quad \text{for } i = 1, \ldots, N, \end{aligned}$$

where Π is the set of all deterministic policies and

$$\begin{aligned} f(\pi) &= \mathbb{E}\left[\sum_{t=0}^{\infty} C_0(s_t, a_t) \middle| s_0 = \bar{s}, \pi \right] \\ g_i(\pi) &= \mathbb{E}\left[\sum_{t=0}^{\infty} C_i(s_t, a_t) \middle| s_0 = \bar{s}, \pi \right] - \Delta_i. \end{aligned}$$

Now, we can define the Lagrangian function as follows:

$$L(\lambda, \pi) = f(\pi) + \lambda^\top \cdot g(\pi), \tag{4}$$

where $\lambda = [\lambda_1, \ldots, \lambda_N]^\top \in \mathbb{R}_+^N$ and $g = [g_1, \ldots, g_N]^\top$.

In addition, we can define the Lagrangian relaxation as follows:

$$L(\lambda) = \min_{\pi \in \Pi} L(\lambda, \pi). \tag{5}$$

Note that the number of policies ($|\Pi|$) is finite since there are finite number of states and actions in a C-SSP. Therefore, the optimization problem for C-SSP has the form of (P) introduced in Section 2.4, and $L(\lambda)$ is piecewise linear concave (Proposition 2.1).

Finally, the best lower bound, or dual optimum, for f^* can be found by solving the Lagrangian dual problem, defined as follows:

$$L^* = L(\lambda^*) = \max_{\lambda \in \mathbb{R}^N_+} L(\lambda). \tag{6}$$

Algorithm 3 summarizes our dual-based anytime algorithm. In line 1, we first solve the Lagrangian dual problem in Eq. (6) and compute a dual optimal policy π^d. According to the weak duality, the Lagrangian function value $L(\lambda^*, \pi^d)$ is a lower bound (LB) of the primal optimal cost (f^*). Also, the cost of the incumbent policy π^{inc} is an upper bound (UB) of the primal optimal cost. Lines 2–4 correspond to the case when the dual optimal solution coincides with the primal optimal solution (i.e., the strong duality holds) in which case we found an optimal solution to C-SSP. Lines 5–11 correspond to the case when there might be a duality gap (i.e., only the weak duality holds), in which case we close the gap until the primal optimal solution is obtained (lines 9–11) or we prove the infeasibility (lines 7–8).

In the rest of this section, we begin by showing an algorithm to solve a Lagrangian relaxation of a C-SSP with fixed λ (Eq. (5)), which is used as a subroutine throughout the proposed method. Then we continue to introduce two main pieces of Algorithm 3 which are 1) how to solve the Lagrangian dual problem in line 1 (*Lagrangian-Dual*) and 2) how to close the gap in line 6 (*Closing-Gap*), in the following subsections.

3.2. Solving Lagrangian relaxation of C-SSP with fixed λ

One of the best advantages of the Lagrangian relaxation of the C-SSP in Eq. (5) is that, given a fixed λ, it is now simply an unconstrained SSP with a

Algorithm 3: Anytime Algorithm for C-SSP

Input: C-SSP instance $\mathcal{H}$, upper bound of Lagrangian multiplier λ^{UB}, VI error tolerance η, approximation parameters M and ρ

$\pi^d, \pi^{\mathrm{inc}}, \mathrm{LB}, \mathrm{UB}, \lambda^*, G \leftarrow$ Lagrangian-Dual$(\mathcal{H}, \lambda^{\mathrm{UB}}, \eta, M, \rho)$
if $LB = UB$ **then**
  $\pi^* \leftarrow \pi^d$
  return π^*
else
  $\pi^{\mathrm{inc}}, \mathrm{UB} \leftarrow$ Closing-Gap$(\pi^d, \pi^{\mathrm{inc}}, \mathrm{LB}, \mathrm{UB}, \lambda^*, \mathcal{H}, G, \eta)$
  if $UB = \infty$ **then**
    return infeasible
  else
    $\pi^* \leftarrow \pi^{\mathrm{inc}}$
    return π^*

cost function $f(\pi) + \lambda^\top \cdot g(\pi)$ instead of $f(\pi)$. Accordingly, the value function of a policy π for the Lagrangian relaxation is defined as follows:

$$V^\pi(s) = \bar{\lambda}^\top \cdot \vec{V}^\pi(s) \tag{7}$$

where $\bar{\lambda} = [1, \lambda_1, \ldots, \lambda_N]^\top$, $\vec{V}^\pi(s) = [V_0^\pi(s), \ldots, V_N^\pi(s)]^\top$,

$$V_0^\pi(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} C_0(s_t, a_t) \middle| s_0 = s, \pi\right]$$

and

$$V_i^\pi(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} C_i(s_t, a_t) \middle| s_0 = s, \pi\right] - \Delta_i$$

for $i = 1, \ldots, N$. Note that the proper policy assumptions that we made for SSP in Section 2.1 hold for the Lagrangian relaxation since a proper policy for a C-SSP is simply still proper and any improper policy yields infinite cost given $\lambda \in \mathbb{R}_+^N$.

Algorithm 4: Lagrangian-Relaxation

Input: C-SSP instance $\mathcal{H}$, AND/OR graph G, Lagrangian multiplier λ, VI error tolerance η

1 **if** G *is not empty* **then**
2 perform VI on G
3 update best solution graph of G
4 **else**
5 initialize graph G with the initial state $\bar{s}$
6 $f, g, \pi \leftarrow \text{WLAO}^*(\mathcal{H}, G, \lambda, \eta)$
7 $L(\lambda) \leftarrow f + \lambda^\top \cdot g$
8 **return** $L(\lambda), f, g, \pi$

Then, an optimal solution to the Lagrangian relaxation with fixed λ is a policy π^* which minimizes the modified value function in Eq. (7) evaluated at the initial state $\bar{s}$, i.e.,

$$\pi^* = \underset{\pi \in \Pi}{\operatorname{argmin}}\, V^\pi(\bar{s})$$

where Π is a set of policies. Therefore, the Lagrangian relaxation of the C-SSP can be solved by a modified LAO* which estimates the value function V^* based on Eq. (7) instead of Eq. (1), with a set of admissible heuristic functions h_i for every $i = 0, \ldots, N$. Note that defining individual value functions is not necessary for solving Lagrangian relaxation of a C-SSP, and we could use original LAO* by using weighted cost function $C' = C_0 + \sum_{i=1}^{N} \lambda^\top \cdot C_i$. However, individual value functions for each secondary cost functions are needed to check the feasibility of a policy in terms of the original C-SSP.

Algorithm 4 shows the Lagrangian relaxation algorithm of a C-SSP with fixed λ, and Algorithm 5 and 6 show modified LAO* (WLAO*). The only notable difference between WLAO* and LAO* is that WLAO* chooses a best action based on the weighted cost with λ and updates values for each cost separately (lines 10–13 of Algorithm 5). This difference is also applied to the

Algorithm 5: WLAO*

Input: C-SSP instance $\mathcal{H}$, AND/OR graph G, Lagrangian multiplier λ, VI error tolerance η

1 **repeat**
2 perform postorder traversal for G and for each visited state s,
3 **if** *s has not been expanded* **then**
4 expand s and for each new state s' added to G,
5 **if** $s' \in \mathcal{G}$ **then**
6 $V_0(s') = 0$, $V_i(s') = -\Delta_i$ for every $i = 1, \ldots, N$
7 **else**
8 $V_0(s') = h_0(s)$, $V_i(s') = h_i(s)$ for every $i = 1, \ldots, N$
9 **else**
10 $a^* \leftarrow \text{argmin}_{a \in \mathcal{A}(s)} \bar{\lambda}^\top \cdot \left[\vec{C}(s,a) + \sum_{s'} T(s,a,s') \cdot \vec{V}(s')\right]$
11 mark a^* as the best action of s
12 **for** $i = 0, \ldots, N$ **do**
13 $V_i(s) \leftarrow C_i(s,a^*) + \sum_{s'} T(s,a^*,s') \cdot V_i(s')$
14 **until** *WLAO*-Convergence-Test($\mathcal{H}, G, \lambda, \eta$)*
15 $f \leftarrow V_0(\bar{s})$, $g \leftarrow [V_1(\bar{s}), \ldots, V_N(\bar{s})]$, $\pi \leftarrow$ best solution graph of G
16 **return** f, g, π

convergence test (Algorithm 6) as well. In addition, as will be clear later, we need to solve Lagrangian relaxation several times with different λ's in the proposed algorithm. However, it is usually better to use already generated, and partially expanded graph, instead of starting with new graph from scratch for solving a Lagrangian relaxation with changed λ value. Therefore, if an existing AND/OR graph G is provided, Algorithm 4 initially performs value iteration with changed λ to ensure all the states' values are computed correctly (lines 1–3 of Algorithm 4), and then proceed to run WLAO* to solve the Lagrangian relaxation.

Algorithm 6: WLAO*-Convergence-Test

Input: C-SSP instance $\mathcal{H}$, AND/OR graph G, Lagrangian multiplier λ, VI error tolerance η

repeat
  if *best solution graph of G has unexpanded tip state* **then**
    return false
  $E \leftarrow 0$
  for *s in best solution graph* **do**
    $v \leftarrow \bar{\lambda}^\top \cdot \vec{V}(s)$
    $a^* \leftarrow \text{argmin}_{a \in \mathcal{A}(s)} \bar{\lambda}^\top \cdot \left[\vec{C}(s,a) + \sum_{s'} T(s,a,s') \cdot \vec{V}(s')\right]$
    mark a^* as the best action of s
    for $i = 0, \ldots, N$ **do**
      $V_i(s) \leftarrow C_i(s,a^*) + \sum_{s'} T(s,a^*,s') \cdot V_i(s')$
    $E \leftarrow \max(E, |v - \bar{\lambda}^\top \cdot \vec{V}(s)|)$
  update best solution graph
until $E < \eta$
return true

3.3. Stage 1: Solving Lagrangian dual of C-SSP

We now introduce an algorithm for solving the Lagrangian dual of a C-SSP shown in Eq. (6). Equipped with *Lagrangian-Relaxation* algorithm (Algorithm 4) that is able to compute a Lagrangian function value $L(\lambda)$ and gradients g at λ, we can use general optimization methods such as the subgradient method [19]. However, it has been shown that a piecewise linear concave optimization can be efficiently solved with an exact line search, especially when there are only a few side constraints [20, 21, 22, 23, 24]. We follow one of the simple and popular choices with the coordinate search, where the exact line search is performed for each coordinate direction sequentially [22, 23, 24]. In the following, we first illustrate the algorithm based on an example with two secondary costs

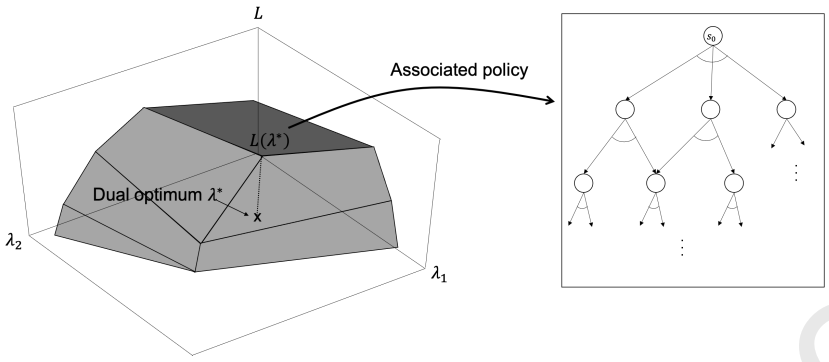
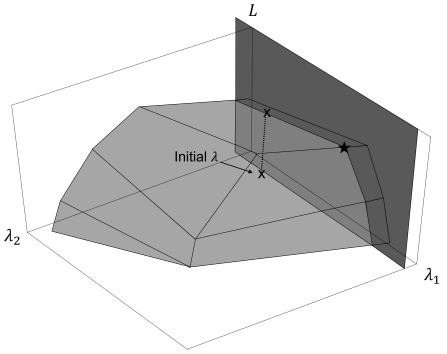


Figure 3: Graphical representation of a Lagrangian function $L(\lambda)$ of an example C-SSP with two secondary cost functions. The graph shows (λ, L) space on the left where Lagrangian functions of each policy are represented as planes, and one of the associated policies that is represented with the dark plane is shown on the right.

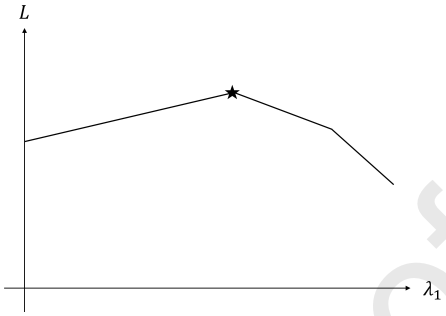
without details. Then we proceed to show detailed algorithm with Algorithm 7 in the later of this section.

For the illustration, consider an example Lagrangian function $L(\lambda)$ of a C-SSP with two secondary cost functions shown in Fig. 3. The $x, y-$axes represent each coordinate of Lagrangian multiplier λ, and the $z-$axis represents Lagrangian function value $L(\lambda, \pi)$. Note that Lagrangian functions for each deterministic policy π can be represented as planes in the graph. Also note that $z-$intercept is $f(\pi)$ and the gradient of a plane in each λ_i direction is $g_i(\pi)$, which implies that a plane with non-positive gradients for every λ_i direction is a feasible policy. Finally, Fig. 3 shows a lower envelope of Lagrangian functions for all the deterministic policies (i.e., $L(\lambda)$), which can be theoretically found by solving Lagrangian relaxations of C-SSP (Eq. (5)) for every λ.

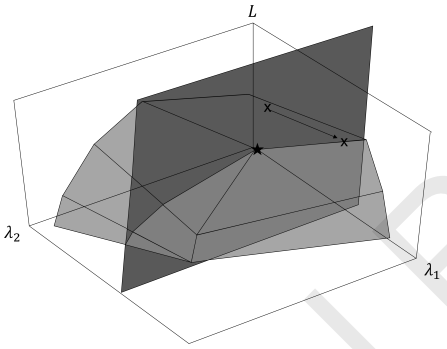
The goal of the first stage is finding the dual optimum λ^* from some initial λ. In the initial iteration, we perform an exact line search on the direction of the first coordinate, i.e., λ_1 (Fig. 4a). The exact line search is an one-dimensional piecewise linear concave optimization (Fig. 4b), which will be explained later. Then, given an updated λ, we perform an exact line search on the direction of the second coordinate (Fig. 4c and Fig. 4d). After performing exact line



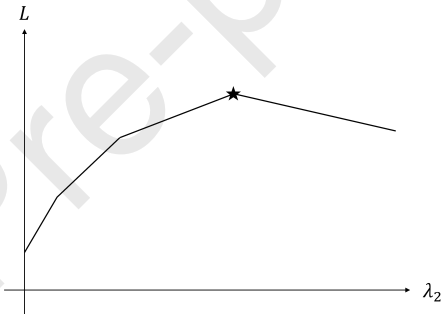
(a) In the first iteration of the algorithm, we optimize the first coordinate of λ given the initial λ.



(b) Optimizing a single coordinate of λ in (a) results in a piecewise linear concave optimization.



(c) Given updated λ from the first iteration, we optimize the second coordinate of λ.



(d) Optimizing a single coordinate of λ in (c) results in a piecewise linear concave optimization.

Figure 4: Illustration of the Lagrangian-Dual algorithm (Algorithm 7).

search(es) for every coordinate(s), if any of the line searches for a coordinate has improved the Lagrangian function value, then we keep iterating the process, and terminate otherwise.

Although a coordinate search works well in many cases, it has some pathological cases where the search gets stuck at the edge of the hyperplanes which might be a non-optimal point [25]. When those pathological cases happen, Algorithm 7 may not be able to find a dual optimal solution for the C-SSP.

However, this does not affect the optimality of the proposed algorithm, and a true optimal solution can still be obtained in the second stage of the algorithm, which will be shown in the subsequent section. In fact, the proposed algorithm remains optimal if a solution returned by Algorithm 7 is based on a Lagrangian relaxation with any λ. Therefore, we can optionally terminate Algorithm 7 before it converges to the dual optimum, when the successive Lagrangian function value change is small enough, or when the number of iterations is large enough, compared to the given approximation parameters.

Algorithm 7 shows the Lagrangian dual algorithm illustrated above. The inputs of the algorithm are a C-SSP problem instance $\mathcal{H}$, upper bound of Lagrangian multiplier λ^{UB}, a VI error tolerance η for WLAO*, and approximation parameters M and ρ. The algorithm starts by initializing an AND/OR graph and iteration count k as 1. Since none of the solutions has been found at this point, UB is set as infinity and incumbent solution π^{inc} is *None*. In addition, initially λ is set as zero vector. Note that any initial λ can be used, but we use zero vector as initial λ since we can detect the trivially feasible problem and return the solution without further computation by solving the relaxed C-SSP with zero Lagrangian multiplier in the beginning of the algorithm. For this, after initialization, the algorithm solves a Lagrangian relaxation (Eq. (4)) with initial λ (line 2). If the policy found with zero Lagrangian multiplier is feasible, then it implies that the problem is trivially feasible, hence this policy is optimal to the C-SSP problem (lines 3–5). Otherwise, we proceed to solve for the dual optimum (lines 6–13).

If the problem is not trivially feasible, then we iteratively perform exact line searches as illustrated in Fig. 4b and Fig. 4d for each coordinate (lines 8 and 9). Finally, there are two termination conditions. First of all, if the number of side constraints is 1 ($N = 1$), then the coordinate search with exact line search converges with a single iteration, hence we return. Second, if the difference between subsequent Lagrangian function values is below the approximation parameter ρ, then it implies that the search has almost converged and we terminate. Finally, if the number of iterations has reached the maximum iteration parameter M,

Algorithm 7: Lagrangian-Dual

Input: C-SSP instance $\mathcal{H}$, upper bound of Lagrangian multiplier λ^{UB}, VI error tolerance η, maximum iteration number M, optimality tolerance ρ

1 initialize AND/OR graph G with initial state $\bar{s}$, iteration count $k \leftarrow 1$, upper bound UB $\leftarrow \infty$, incumbent policy $\pi^{\mathrm{inc}} \leftarrow$ none, Lagrangian multiplier $\lambda \leftarrow \mathbf{0}$

2 $L, f, g, \pi \leftarrow$ Lagrangian-Relaxation$(\mathcal{H}, G, \lambda, \eta)$

3 **if** $g_i \leq 0$ *for every* $i = 1, \ldots, N$ **then**

4 LB $\leftarrow f$, UB $\leftarrow f$, $\pi^d \leftarrow \pi$, $\pi^{\mathrm{inc}} \leftarrow \pi$, $\lambda^* \leftarrow \lambda$

5 **return** π^d, π^{inc}, LB, UB, λ^*, G

6 **while** *true* **do**

7 $L_{prev} \leftarrow L$

8 **for** $i = 1, ..., N$ **do**

9 $L, \pi, \pi^{\mathrm{inc}}, \mathrm{LB}, \mathrm{UB}, \lambda, G \leftarrow$ 1d-Lagrangian-Dual$(\mathcal{H}, G, \lambda, \lambda^{\mathrm{UB}}, \mathrm{UB}, \pi^{\mathrm{inc}}, i, \eta)$

10 $k \leftarrow k + 1$

11 **if** $N = 1$ *or* $|L - L_{prev}| \leq \rho$ *or* $k > M$ **then**

12 $\pi^d \leftarrow \pi$, $\lambda^* \leftarrow \lambda$

13 **return** $\pi^d, \pi^{\mathrm{inc}}, \mathrm{LB}, \mathrm{UB}, \lambda^*, G$

then we terminate the first stage as well. Note that ρ and M can be set as zero and ∞, respectively, if no approximation is used in the first stage.

Now, we turn our attention to the exact line search that has been used as a subroutine in line 9 of the Algorithm 7. In fact, the exact line search is an one-dimensional Lagrangian dual, which is also an one-dimensional piecewise linear concave optimization. We apply a bisection-like method shown in Algorithm 8, which is proved to be effective for one-dimensional piecewise linear concave optimizations [26, 27, 28]. Similar to Algorithm 7, Algorithm 8 starts by solving

Algorithm 8: 1d-Lagrangian-Dual

Input: C-SSP instance $\mathcal{H}$, AND/OR graph G, Lagrangian multiplier λ, upper bound of Lagrangian multiplier λ^{UB}, upper bound UB, incumbent policy π^{inc}, coordinate i, VI error tolerance η

$\lambda^0 \leftarrow \lambda$, $\lambda_i^0 \leftarrow 0$, $\lambda^\infty \leftarrow \lambda$, $\lambda_i^\infty \leftarrow \lambda_i^{\text{UB}}$

$L(\lambda^0), f^0, g^0, \pi^0 \leftarrow$ Lagrangian-Relaxation$(\mathcal{H}, G, \lambda^0, \eta)$

if $g_j^0 \leq 0$ *for every* $j = 1, \ldots, N$ *and* $f^0 < UB$ **then**

  UB $\leftarrow f^0$, $\pi^{\text{inc}} \leftarrow \pi^0$

$L(\lambda^\infty), f^\infty, g^\infty, \pi^\infty \leftarrow$ Lagrangian-Relaxation$(\mathcal{H}, G, \lambda^\infty, \eta)$

if $g_j^\infty \leq 0$ *for every* $j = 1, \ldots, N$ *and* $f^\infty < UB$ **then**

  UB $\leftarrow f^\infty$, $\pi^{\text{inc}} \leftarrow \pi^\infty$

if $g_i^0 \cdot g_i^\infty \geq 0$ **then**

  $z \leftarrow \text{argmax}_{j \in \{0, \infty\}} L(\lambda^j)$

  LB $\leftarrow L(\lambda^z)$

  return $L(\lambda^z), \pi^z$, π^{inc}, LB, UB, λ, G

repeat

  update λ_i using Eq. (8)

  $L \leftarrow f^0 + \lambda g^0$

  $L(\lambda), f^{\text{new}}, g^{\text{new}}, \pi^{\text{new}} \leftarrow$ Lagrangian-Relaxation$(\mathcal{H}, G, \lambda, \eta)$

  if $L(\lambda) < L$ *and* $g_i^{new} > 0$ **then**

    $\pi^0 \leftarrow \pi^{\text{new}}, f^0 \leftarrow f^{\text{new}}, g^0 \leftarrow g^{\text{new}}$

  else if $L(\lambda) < L$ *and* $g_i^{new} \leq 0$ **then**

    $\pi^\infty \leftarrow \pi^{\text{new}}, f^\infty \leftarrow f^{\text{new}}, g^\infty \leftarrow g^{\text{new}}$

  if $g_j^{new} \leq 0$ *for every* $j = 1, \ldots, N$ *and* $f^{new} < UB$ **then**

    UB $\leftarrow f^{\text{new}}$, $\pi^{\text{inc}} \leftarrow \pi^{\text{new}}$

until $L = L(\lambda)$

LB $\leftarrow L(\lambda)$

return $L, \pi^\infty, \pi^{\text{inc}}, \text{LB}, \text{UB}, \lambda, G$

two Lagrangian relaxations with lower and upper bounds λ^0 and λ^∞ (lines 2 and 5), where λ^0 and λ^∞ are computed in line 1. For each case, we update UB and π^{inc} if a policy is feasible and better than the current incumbent policy (lines 3–4 and 6–7). Then, if it has been proved that the lower envelope is monotonic (line 8), we finish the search and return the better solution in terms of the Lagrangian function value. Otherwise, it iteratively updates λ and solves for Eq. (5) using Algorithm 4 to find the dual optimum (lines 12–22).

Figure 5 illustrates the iterative process in lines 12–22 with a simple example. Note that a graph in each sub-figure shows the i-th coordinate subsection of (λ, L) space where each policy is represented as a line. Also note that solid bold lines represent policies that have been found and gray lines represent policies that have not been found so far. Given π^0 and π^∞ and their corresponding function and constraint values f^0, g^0, f^∞ and g^∞ that have been found in lines 2–7, we update λ as the intersection of two solutions (Fig. 5a), which satisfy the following equality:

$$L(\lambda, \pi^0) = L(\lambda, \pi^\infty)$$

where

$$L(\lambda, \pi^0) = f(\pi^0) + \sum_{i=1}^{N} \lambda_i \cdot g_i(\pi^0)$$

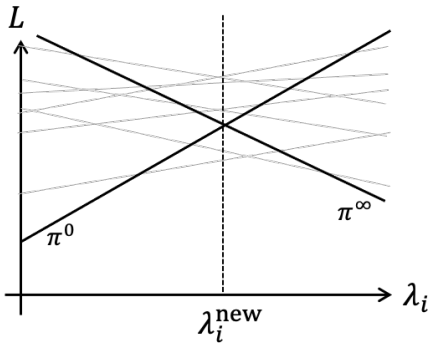
and

$$L(\lambda, \pi^\infty) = f(\pi^\infty) + \sum_{i=1}^{N} \lambda_i \cdot g_i(\pi^\infty).$$

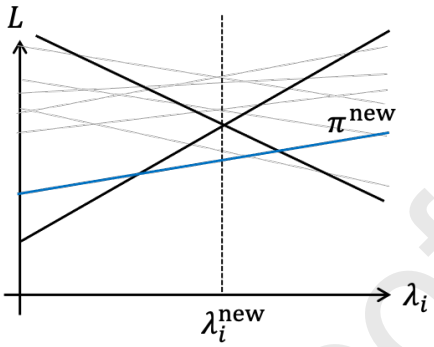
Therefore, new i-th coordinate of λ at the intersection of two policies can be computed as follows:

$$\lambda_i = \frac{f(\pi^0) - f(\pi^\infty) + \sum_{j \neq i} \lambda_j \Big(g_j(\pi^0) - g_j(\pi^\infty) \Big)}{g_i(\pi^\infty) - g_i(\pi^0)}. \tag{8}$$

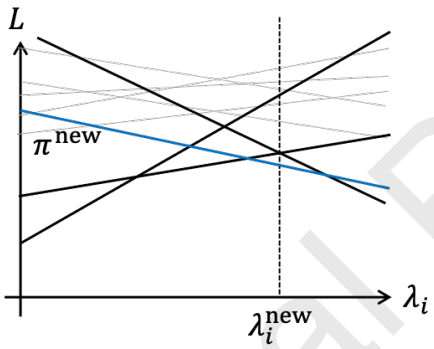
Then we obtain a new π by solving Eq. (5) with the updated λ (line 15 and Fig. 5b). Since new π has positive gradient in i-th coordinate, we update π^0 (lines 16 and 17). Then we repeat the process, which updates λ using Eq. (8)



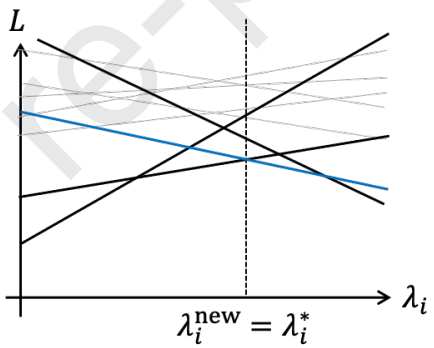
(a) The process starts with two initial solutions π^0 and π^∞ (lines 2–7 in Algorithm 8).



(b) Update π^0 since π^{new} has positive gradient in λ_i direction (lines 16 and 17 in Algorithm 8).



(c) Update π^∞ since π^{new} has negative gradient in λ_i direction (lines 18 and 19 in Algorithm 8).



(d) The process is terminated since λ^{new} has converged (line 22 in Algorithm 8).

Figure 5: Illustration of the process of solving one-dimensional Lagrangian dual problem with respect to i-th coordinate of λ. Each graph shows (λ_i, L) space where each policy is represented as a line.

and solve Eq. (5) with the updated λ, as shown in Fig. 5c. Note that we update π^∞ in this case (lines 18 and 19) since new π has negative gradient. In addition, if a new policy is feasible and better than the incumbent, then UB and π^{inc} are updated (lines 20 and 21).

If we repeat the process again (Fig. 5d), the Lagrangian function value remains the same if we solve Lagrangian relaxation with updated λ, in which case we have found the dual optimal solution λ_i^* for this one-dimensional case (line 22). Note that since we update UB throughout the process (lines 4, 7 and 21), it comes with the anytime property. In addition, the algorithm is complete and terminates with finite iterations. This is because, roughly speaking, the algorithm only evaluates $L(\lambda)$'s at the intersections between policies, which are finite due to a finite number of deterministic policies. For more details about the one-dimensional piecewise linear concave optimization, please refer to [27].

However, if the given λ_i^{UB} is not large enough, there might be a case that Algorithm 8 fails to optimize the 1-d Lagrangian dual for the $i-$th coordinate. Although this does not affect the optimality of the proposed algorithm, which will be discussed in the following section, obtaining Lagrangian dual can be beneficial to achieve a better solution during the first stage and also to have better convergence in the second stage. One of the possible ideas that overcomes this issue without prior knowledge on λ_i^{UB} is repeatedly solving Lagrangian relaxation in line 5 of Algorithm 8 by adaptively increasing λ_i^{UB} until g_i^{∞} is negative or λ_i^{UB} reaches some relatively large number.

3.4. Stage 2: Closing the duality gap

Since finding an optimal deterministic policy for C-SSP can be viewed as a discrete optimization problem, the strong duality does not generally hold for our problem. Even though the dual optimal policy found in the first stage can serve as an approximated solution, our goal is finding a primal optimal solution eventually. Therefore, the objective of the second stage of the proposed method is closing the duality gap if there is any. In addition, we would like to emphasize that the algorithm is anytime, which incrementally updates the incumbent policy until we find a true optimal policy.

Conceptually, the way the algorithm obtains a primal optimal policy is quite simple. From the dual optimal policy, we can find a primal optimal solution by iteratively finding the next best solutions, with respect to the Lagrangian func-

tion value $L(\lambda^*, \pi)$. This procedure is illustrated in Fig. 6 with an intuitive example which only includes a single secondary cost. The figure shows all the deterministic policies in (λ, L) space, including feasible ones ($g_1(\pi) \leq 0$) and infeasible ones ($g_1(\pi) > 0$), in green solid lines and red dotted lines, respectively. Note that the true primal optimal policy π^* is indicated with a bold line. Fig. 6 also shows the Lagrangian function value $L(\lambda^*, \pi)$ for each policy as an intersection of the policy with the dotted vertical line at λ^*. Similarly, a y-intercept of a policy π is the primal cost $f(\pi)$. In addition, $\pi^\infty = \pi^1$ is the solution associated with λ^* which can be obtained by Algorithm 7, where π^k denotes k-th best policy with respect to the Lagrangian function value $L(\lambda^*, \pi)$. Starting from π^1, we find the next best policies in terms of $L(\lambda^*, \pi)$, in non-decreasing manner, until we obtain a primal optimal policy π^*, and update the incumbent policy whenever we obtain a better feasible policy. In the example shown in Fig. 6, the primal optimal policy can be found as the fifth best policy, and the incumbent solution can be updated once we find the fourth and fifth best policies.

To prevent us from searching exhaustively, we need to have a termination condition that indicates that the current incumbent policy is optimal. For this purpose, we need to keep track of LB and UB of the primal cost, $f(\pi)$. Let $L^l(\lambda^*)$ be a Lagrangian function value of the l-th best policy. Then $L^l(\lambda^*)$ is a lower bound of the primal cost for all the k-th best policies for $k > l$, since $g_i^k(\pi) \leq 0$ for every $i = 1, \ldots, N$ for any feasible solution, hence none of the k-th best solution for $k > l$ can intersect the y-axis with a lower value than $L^l(\lambda^*)$. On the other hand, the primal cost of the current incumbent solution is obviously an upper bound of the primal optimal cost. Therefore, whenever the condition LB $\geq$ UB is satisfied, we can assure that the current incumbent solution is a primal optimal policy. In the example shown in Fig. 6, the condition is satisfied when the sixth best policy is found, and the algorithm terminates with the optimal policy $\pi^5 = \pi^*$.

Algorithm 9 shows the Closing-Gap algorithm illustrated above. The algorithm starts with initialization, where the best policy order k is set as 1, the

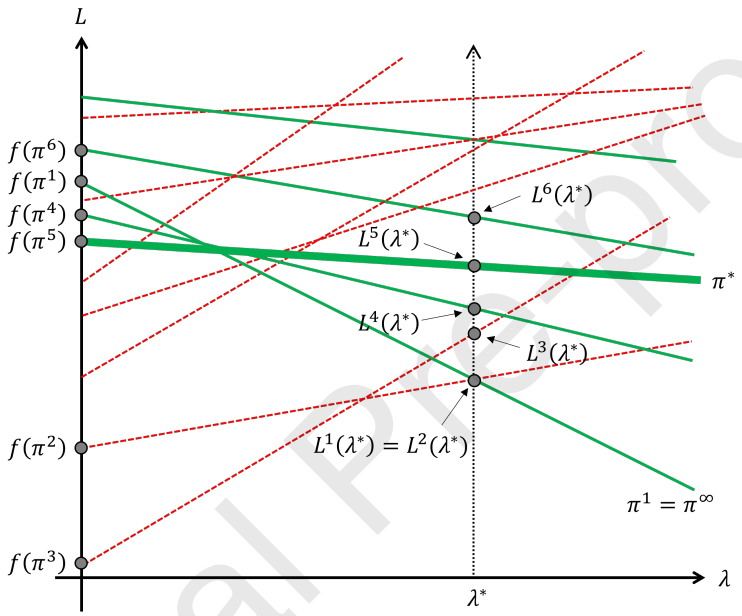


Figure 6: (λ, L) space representing policies for an example C-SSP with a single constraint. Green solid lines and red dotted lines show feasible and infeasible policies, respectively, where bold green line shows the optimal policy. Each policy intersects y-axis and vertical line at λ^* (shown as dotted line) with the primal cost f and the Lagrangian function value L, respectively.

Algorithm 9: Closing-Gap

Input: Policy π^d, incumbent policy π^{inc}, lower bound LB, upper bound UB, Lagrangian multiplier λ^*, C-SSP instance $\mathcal{H}$, AND/OR graph G, VI error tolerance η

1 initialize best policy order $k \leftarrow 1$, current best policy $\pi^1 \leftarrow \pi^d$, set of best policies $\Pi \leftarrow \{\pi^1\}$, candidates set $\Gamma \leftarrow \{\}$, set of fixed state-action pairs $\pi_F^1 \leftarrow \{\}$, set of blocked state-action pairs $\pi_B^1 \leftarrow \{\}$

2 **repeat**

3 $\quad k \leftarrow k + 1$

4 $\quad$ set $\mathcal{S}_{\pi^{k-1}}$ as a set of non-terminal reachable states from $\bar{s}$ by π^{k-1}.

5 $\quad (L^k, \pi^k, f^k, g^k, \pi_F^k, \pi_B^k), \Pi, \Gamma, G \leftarrow$ Next-Best-Policy(k, π^{k-1}, $\mathcal{S}_{\pi^{k-1}}, \pi_F^{k-1}, \pi_B^{k-1}, \Pi, \Gamma, \lambda^*, \mathcal{H}, G, \eta$)

6 $\quad$ LB $\leftarrow L^k$.

7 $\quad$ **if** $g_i^k \leq 0$ *for every* $i = 1, \ldots, N$ *and* $f^k < UB$ **then**

8 $\quad\quad$ UB $\leftarrow f^k$ and $\pi^{\text{inc}} \leftarrow \pi^k$

9 **until** $LB \geq UB$

10 **return** π^{inc}, UB

first best policy is initialized as the policy provided by Lagrangian-dual (Algorithm 7), and the k-best policy set Π and candidate set Γ are initialized as $\{\pi^1\}$ and the empty set, respectively. We also initialize two sets, fixed state-action pair set π_F^1 and blocked state-action pair set π_B^1, as empty sets. Then the algorithm iteratively finds the next best policy (line 5) until the termination condition is satisfied (line 9). While finding the next best policies, if a newly found policy is feasible and better than the current incumbent policy, then we update the incumbent policy and the upper bound (lines 7 and 8).

Note that, although we assumed that the second stage starts with the dual optimum λ^* and an associated policy π^d in the above illustration, the optimality of the algorithm remains the same even with any Lagrangian multiplier λ and an associated policy. In other words, if π' and λ' are given to Algorithm 9

instead of π^d and λ^*, where π' is an optimal policy to the Lagrangian relaxation with λ', then π^{inc} returned by Algorithm 9 is guaranteed to be optimal. The optimality of the algorithm is formally described in the following theorem.

Theorem 3.1. *Suppose we are given a C-SSP that has at least one feasible policy. Also, suppose π^d and λ^* are given to Algorithm 9 where π^d is an optimal policy to the Lagrangian relaxation with some Lagrangian multiplier λ^*. Then, once Algorithm 9 is terminated, the incumbent policy π^{inc} is a primal optimal policy. Moreover, the algorithm is complete.*

Proof. The optimality can be proved by contradiction. Let f^* be the primary cost of the incumbent policy π^{inc}. Suppose $k = J$ is the index when Algorithm 9 is terminated. Let Π be a set of every policy and suppose there exists $\pi \in \Pi \backslash \{\pi^k | k = 1, ..., J\}$, such that $g_i(\pi) \leq 0$ for every i and $f(\pi) < f^*$. Since π is not included in the first J best solutions,

$$L(\pi, \lambda^*) = f(\pi) + \lambda^* g(\pi) \geq L^J(\lambda^*).$$

Also, at the termination, the following inequality holds:

$$L^J(\lambda^*) = \text{LB} \geq \text{UB} = f^*,$$

which result in

$$f(\pi) + \lambda^* g(\pi) \geq f^*.$$

However, this is contradiction, since $\lambda^* \cdot g(\pi)$ is non-positive, which shows that $f(\pi) \geq f^*$.

To show the algorithm is complete, suppose the Lagrangian function value of an optimal policy evaluated at λ^* is L'. Now, recall that the number of policies ($|\Pi|$) is finite since there are finite numbers of states and actions in C-SSP. Therefore, there are finite, say K, number of policies that have Lagrangian function values lower than or equal to L' at λ^*. Therefore the algorithm can obtain the optimal solution with at most K iterations in the Algorithm 9. □

The crucial part of the second stage given in Algorithm 9 is finding the next best policy given the first k-best policies (line 5), which is shown in Algorithm 10. The algorithm can be viewed as an AND/OR graph version of Yen's algorithm [29], or a special case of Lawler's k-best solution method [30]. The intuition behind Algorithm 10 is the following. Suppose π^k is the current best policy. Then, for each non-terminal state reachable from the initial state $\bar{s}$ by following π^k, we generate a best deviated policy, and keep them in a candidate set. Note that the candidate set is cumulative during the entire run of Algorithm 9 and 10. Then a best policy among the candidates becomes the next best policy, π^{k+1}.

Each candidate generation can be thought of generating and solving a new SSP instance by adding more constraints to an SSP instance from which the current best policy was generated, where the additional constraints are represented as fixed and blocked state-action pair sets π_F and π_B. In other words, if $(s, a) \in \pi_F$ for some state s and action a, then we fix the action selection from s as a. Similarly, if $(s', a') \in \pi_B$ for some state s' and action a', then we remove action a' from available actions of s'. Suppose a set of non-terminal states reachable from the initial state $\bar{s}$ by following π^k is ordered with a breadth-first search (BFS) order, where $s_0 = \bar{s}$. Then a new SSP instance for a candidate generation with s_i from the current best policy π^k is formed with the following three steps:

1. copy π_F and π_B from an SSP instance that generated the current best policy π^k,
2. add $(s_i, \pi^k(s_i))$ to π_B,
3. for every $j < i$, add $(s_j, \pi^k(s_j))$ to π_F.

Roughly speaking, the second step enables us to find a deviated policy that is worse than the current best policy by blocking the action selected by the current best policy. In addition, the third step enables us to find deviated policies for each state reachable from the current best policy given that the earlier states do not deviate from the current best policy. Note that a rigorous justification

and the correctness of the algorithm is given in Appendix A.

Fig. 7 illustrates an example of the first run of the Next-Best-Policy algorithm (Algorithm 10). As shown in the left of the Fig. 7, the algorithm is fed the first best policy π^1 with the fixed and blocked state-action pair sets π_F and π_B. The policy is represented as an AND/OR graph, where each node is a state, which are numbered with BFS order, and each hyperedge is an action, which is labeled with the action name. In addition, the sets π_F and π_B are empty, as initialized in Algorithm 9. Then from the current best policy, which is π^1 in this case, Algorithm 10 generates candidates with each non-terminal reachable state from s_0 with BFS order. On the right of Fig. 7, each box represents a modified SSP for each candidate generation, where the red thick circle is a deviating state and shaded circles are fixed states. For example, for the candidate generation with s_6, states s_0 to s_5 are fixed and action a_2 is blocked from s_6. Then a candidate can be generated by solving an SSP with those new constraints given by π_F and π_B.

Similarly, Fig. 8 shows an example of the k-th run of Algorithm 10. Since π_F and π_B are copied and augmented when we generate a candidate, they are cumulative and might not be empty for the n-th run for $n > 1$, which is also the case for the example in Fig. 8. One notable difference from the first run of the algorithm is that candidates are not generated from the states which are included in π_F. For example, in Fig. 8, the first candidate generation was with s_2 instead of s_0 since s_0 and s_1 are in x_F. This is because any candidate generation for a state in x_F results in an infeasible SSP, since a candidate generation will add a state-action pair in x_F to x_B as well, which cannot be satisfied at the same time.

Algorithm 10 shows the illustrated Next-Best-Policy algorithm. The algorithm begins by initializing fixed and blocked state-action pair sets π_F^{new} and π_B^{new} as the sets that were computed when we generated π^{k-1}, i.e., π_F^{k-1} and π_B^{k-1}. Then the algorithm iteratively generates candidates for each non-terminal state that are reachable from the initial state by π^{k-1} with BFS order (lines 2–14). As explained previously with Fig. 8, we skip a candidate generation if

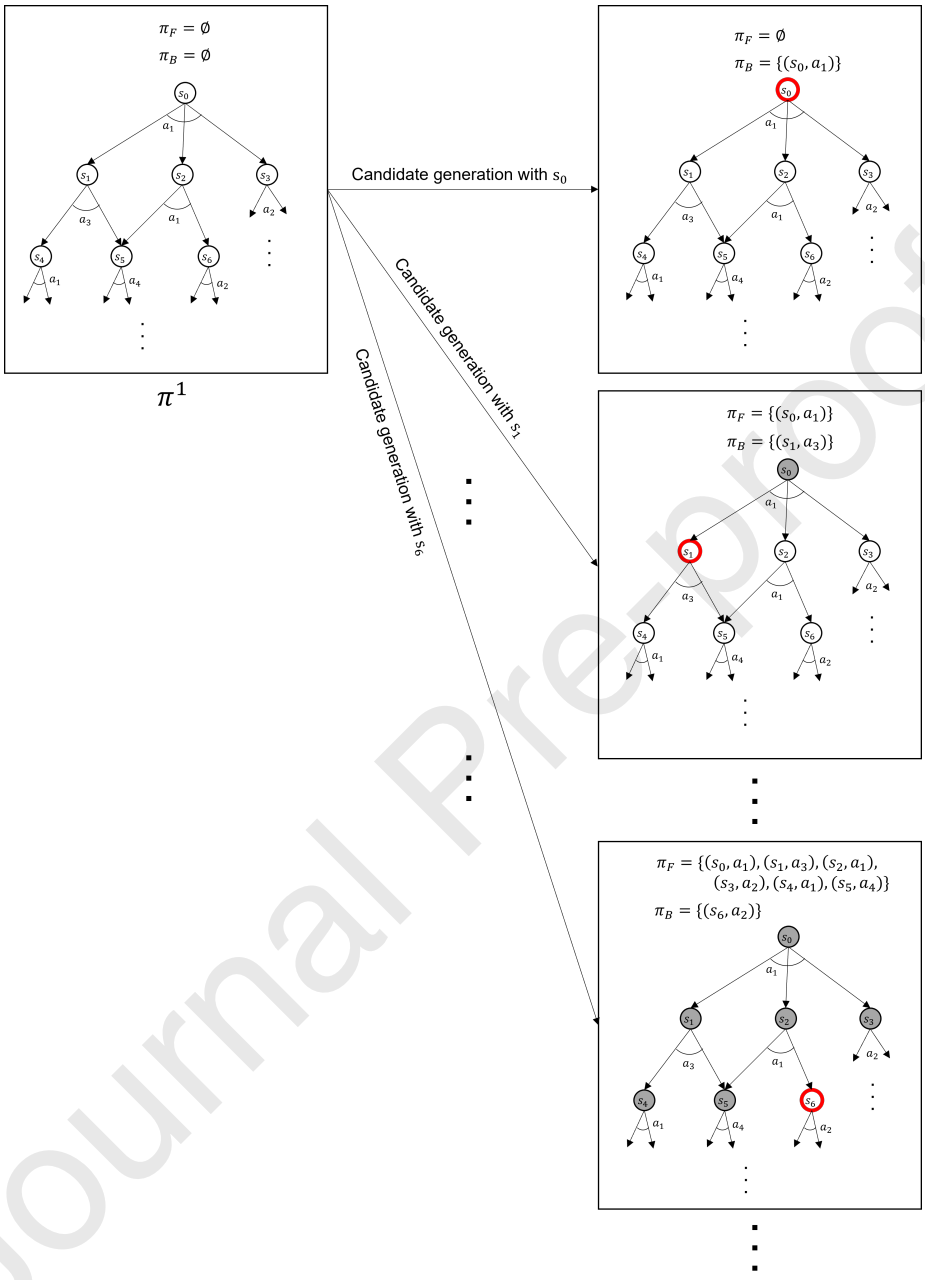


Figure 7: Illustration of the first iteration of the Next-Best-Policy (Algorithm 10) for an example C-SSP. The graph on the left is the policy from which we want to generate candidates, and the graphs on the right are candidate generating problems with added constraints. For each candidate generating problem, a red circle represents deviating state from which we block the previously selected action, and shaded circles represent fixed states from which we fix the previously selected actions.

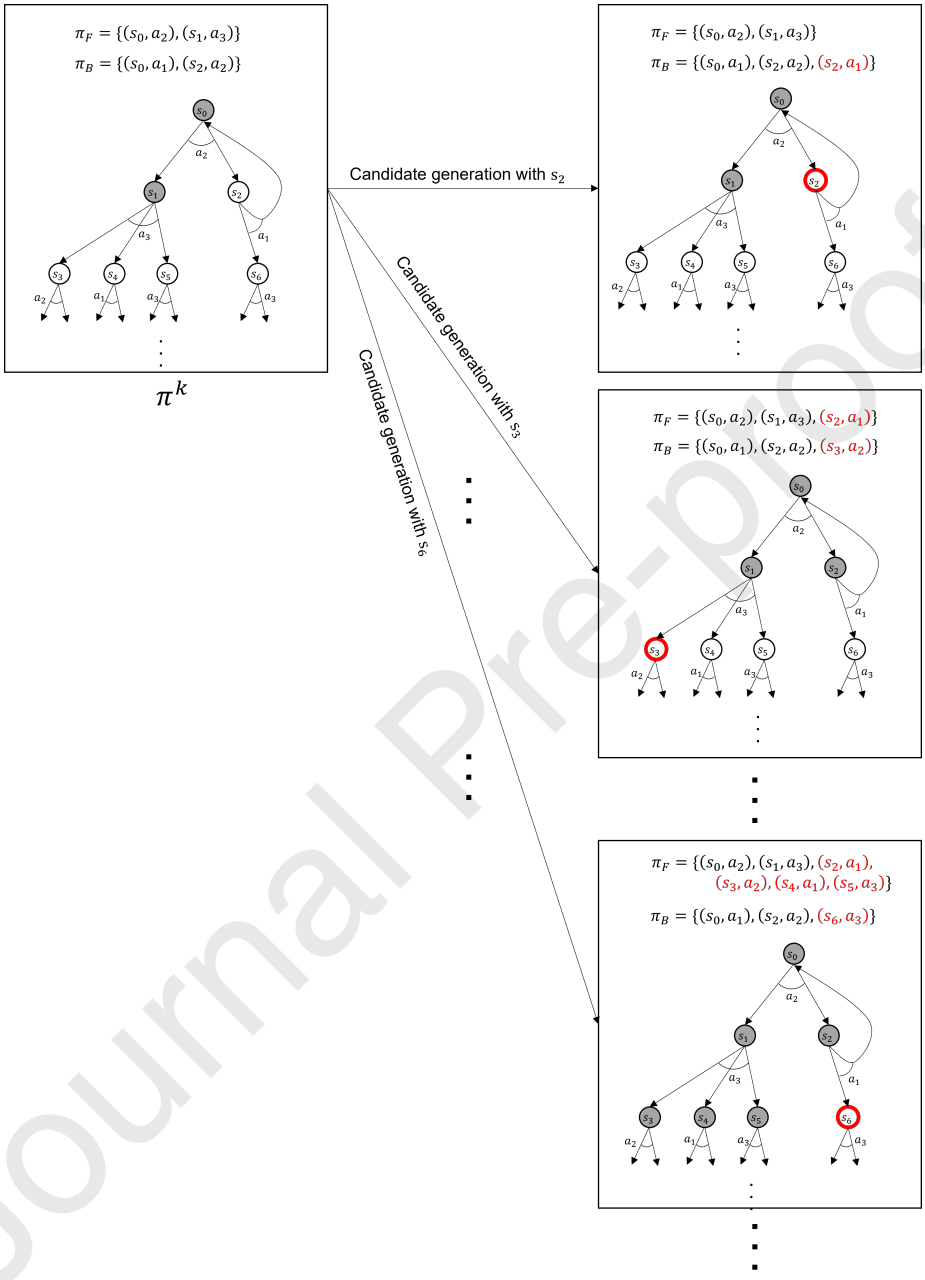


Figure 8: Illustration of the k-th iteration of the Next-Best-Policy (Algorithm 10) for an example C-SSP. The figure illustrates the process in the same way as in Fig. 7.

a deviating state is in π_F^{new} (line 3). Then in lines 4–8, we add new blocked state-action pair(s) and modify the C-SSP instance based on constraints given by π_F^{new} and π_B^{new}. Then we solve the modified C-SSP using the WLAO* (line 9) and add the newly computed candidate into the candidates set (lines 10 and 11). Finally, we add the current deviating state with its action selected by π^{k-1} into π_F^{new} (line 12), remove currently blocked state-action pair from π_B^{new} (line 13) and reset the action model of the C-SSP instance (line 14) for the next iteration. After generating all the candidates, the next k-th best policy can be obtained by selecting the best policy from the candidate set with respect to the Lagrangian function value (lines 15 and 16). Note that Γ is a priority queue with the Lagrangian function value $L(\lambda^*, \pi)$.

Algorithm 10: Next-Best-Policy

Input: best policy order k, current best policy π^{k-1}, set of reachable states $\mathcal{S}_{\pi^{k-1}}$, set of fixed state-action pairs π_F^{k-1}, set of blocked state-action pairs π_B^{k-1}, set of best policies Π, candidates set Γ, Lagrangian multiplier λ, C-SSP instance $\mathcal{H}$, AND/OR graph G, VI error tolerance η

$\pi_F^{\text{new}} \leftarrow \pi_F^{k-1}$, $\pi_B^{\text{new}} \leftarrow \pi_B^{k-1}$
for $s \in \mathcal{S}_{\pi^{k-1}}$ *in BFS order* **do**
  if $\nexists (s', a) \in \pi_F^{new}$ *s.t.* $s = s'$ **then**
    $\pi_B^{\text{new}} \leftarrow \pi_B^{\text{new}} \cup \{(s, \pi^{k-1}(s))\}$
    for $(s', a) \in \pi_F^{new}$ **do**
      $\mathcal{A}(s') \leftarrow \{a\}$
    for $(s', a) \in \pi_B^{new}$ **do**
      $\mathcal{A}(s') \leftarrow \mathcal{A}(s') \backslash \{a\}$
    $f^{new}, g^{new}, \pi^{new} \leftarrow \text{WLAO}^*(\mathcal{H}, G, \lambda, \eta)$
    $L^{\text{new}} \leftarrow f^{\text{new}} + \lambda^\top \cdot g^{\text{new}}$
    $\Gamma \leftarrow \Gamma \cup \{(L^{\text{new}}, \pi^{\text{new}}, f^{\text{new}}, g^{\text{new}}, \pi_F^{\text{new}}, \pi_B^{\text{new}})\}$
    $\pi_F^{\text{new}} \leftarrow \pi_F^{\text{new}} \cup \{(s, \pi^{k-1}(s))\}$
    $\pi_B^{\text{new}} \leftarrow \pi_B^{\text{new}} \backslash \{(s, \pi^{k-1}(s))\}$
    reset action model $\mathcal{A}$ of $\mathcal{H}$
$(L^k, \pi^k, f^k, g^k, \pi_F^k, \pi_B^k) \leftarrow \text{pop}(\Gamma)$
$\Pi \leftarrow \Pi \cup \{\pi^k\}$
return $(L^k, \pi^k, f^k, g^k, \pi_F^k, \pi_B^k), \Pi, \Gamma, G$

4. Approximation: Candidate Pruning

Although the proposed algorithm can produce an optimal policy eventually, in practical applications, especially in safety-critical and time-limited applications, we are most interested in having a near-optimal solution in a given amount of the time. In those applications, it is often preferred to have a better convergence rate at the cost of optimality. To accomplish this desired property, we propose an approximation scheme that is guaranteed to produce an ϵ-optimal policy given $\epsilon > 0$ with better convergence rate.

The key idea of the approximation scheme is skipping generating a candidate in the second stage of the proposed algorithm if we can prove the values of the candidate and its further generation candidates are not much better than the value of the current best solution. We begin by introducing a notion of potential, which is an estimation of value reduction that could be obtained by generating candidates from a subset of states. Then we present the candidate pruning algorithm based on the potential. Finally we provide speed-up techniques that can reduce the computational overhead of the approximation scheme.

4.1. Estimating the value reduction by candidates

We propose a simple estimation for the value reduction by a set of candidates based on artificial goals. To elaborate, we briefly review the policy evaluation in a matrix form as follows. Given a policy π, let $\mathcal{S}_\pi$ be a set of states that are reachable from the initial state $\bar{s}$ by π. For simplicity, let the states $\mathcal{S}_\pi$ be ordered from 0 to $|\mathcal{S}_\pi| - 1$ so that the non-terminal states come first and s_0 be the initial state $\bar{s}$. In addition, let M be the number of terminal states where $M < |\mathcal{S}_\pi| - 1$. Then the j-th cost vector of the given policy π is defined as follows:

$$\mathcal{C}_j^\pi = \left[\begin{array}{c} \bar{\mathcal{C}}_j^\pi \\ \hline 0 \end{array}\right],$$

where $\bar{\mathcal{C}}_j^\pi$ is the length $|\mathcal{S}_\pi| - M - 1$ column vector, $\left(\bar{\mathcal{C}}_j^\pi\right)_i = C_j(s_i, \pi(s_i))$, and 0 is the length M zero vector. Similarly, the transition matrix of π is defined as

follows:

$$P^{\pi} = \left[\begin{array}{c|c} Q^{\pi} & R^{\pi} \\ \hline 0 & I \end{array}\right],$$

where $\left(P^{\pi}\right)_{i,k} = Pr(s_k|s_i, \pi(s_i))$. Note that Q^{π} is the $(|\mathcal{S}_{\pi}| - M - 1)$-by-$(|\mathcal{S}_{\pi}| - M - 1)$ matrix, R^{π} is the $(|\mathcal{S}_{\pi}| - M - 1)$-by-$M$ matrix, 0 is the M-by-$(|\mathcal{S}_{\pi}| - M - 1)$ zero matrix and I is the M-by-M identity matrix.

Then, if we define the j-th value function of the policy π as

$$V_j^{\pi} = \left[\begin{array}{c} \bar{V}_j^{\pi} \\ \hline \bar{\bar{V}}_j^{\pi} \end{array}\right],$$

where $\bar{V}_j^{\pi}$ and $\bar{\bar{V}}_j^{\pi}$ are value functions for non-terminal and terminal states, respectively, the value function can be computed as follows:

$$\left[\begin{array}{c} \bar{V}_j^{\pi} \\ \hline \bar{\bar{V}}_j^{\pi} \end{array}\right] = \left[\begin{array}{c} \bar{\mathcal{C}}_j^{\pi} \\ \hline 0 \end{array}\right] + \left[\begin{array}{c|c} Q^{\pi} & R^{\pi} \\ \hline 0 & I \end{array}\right] \cdot \left[\begin{array}{c} \bar{V}_j^{\pi} \\ \hline \bar{\bar{V}}_j^{\pi} \end{array}\right].$$

Therefore, $\bar{\bar{V}}_j^{\pi}$ is the zero vector, and $\bar{V}_j^{\pi}$ can be computed as follows:

$$\bar{V}_j^{\pi} = (I - Q^{\pi})^{-1} \cdot \bar{\mathcal{C}}_j^{\pi}.$$

Now, we want to compute an upper bound of the potential value reduction by a set of candidates generated from states in $\hat{\mathcal{S}}_{\pi} \subset \mathcal{S}_{\pi}$. One of the simple choices that we use in this paper is temporarily changing each state in $\hat{\mathcal{S}}_{\pi}$ into a goal state by modifying the cost vector and transition matrix accordingly. Let $\hat{\mathcal{C}}_j^{\pi}$ and $\hat{Q}^{\pi}$ be modified j-th cost vector and transition matrix for non-terminal states, which are defined as follows:

$$\left(\hat{\mathcal{C}}_j^{\pi}\right)_i = \begin{cases} \left(\bar{\mathcal{C}}_j^{\pi}\right)_i & \text{if } s_i \notin \hat{\mathcal{S}}_{\pi} \\ 0 & \text{if } s_i \in \hat{\mathcal{S}}_{\pi} \end{cases}$$

for every $j = 0, \ldots, N$, and

$$\left(\hat{Q}^{\pi}\right)_{i,k} = \begin{cases} \left(Q^{\pi}\right)_{i,k} & \text{if } s_i \notin \hat{\mathcal{S}}_{\pi} \\ 0 & \text{if } s_i \in \hat{\mathcal{S}}_{\pi} \end{cases}$$

for every $k = 0, \ldots, |\mathcal{S}_\pi| - M - 1$. Then the modified value function, which we denoted as $\hat{V}_j^\pi$, also can be computed as follows:

$$\hat{V}_j^\pi = (I - \hat{Q}^\pi)^{-1} \cdot \hat{C}_j^\pi. \tag{9}$$

Finally, we define a potential of j-th value, which is an upper bound of the j-th value reduction by deviating from a state in $\hat{\mathcal{S}}_\pi$ as follows:

Definition 4.1. (Potential of j-th value)

$$\phi_j^\pi(\hat{\mathcal{S}}_\pi) = \bar{V}_j^\pi(\bar{s}) - \hat{V}_j^\pi(\bar{s}).$$

4.2. Candidate pruning algorithm

Now we present the candidate pruning algorithm shown in Algorithm 11 that is based on the potential introduced in the previous section. The general idea of the Algorithm 11 is the following. Given an approximation bound ϵ and the current best policy π^k with $\mathcal{S}_{\pi^k}$, we collect as many states from $\mathcal{S}_{\pi^k}$ as possible into $\hat{\mathcal{S}}_{\pi^k}$ with the reverse order of BFS, until no more states can be included without the potential of $\hat{\mathcal{S}}_{\pi^k}$ exceeding ϵ (lines 8–12). However, the algorithm works a little differently based on whether π^k is feasible or not. If π^k is feasible, the algorithm tries to prune candidates that cannot be better than π^k by ϵ in terms of the primary cost. On the other hand, if π^k is infeasible (i.e., there is an index i such that $g_i > 0$), then the algorithm tries to prune candidates that cannot become feasible. For this purpose, we use an index set $\mathcal{I}$ in the algorithm to indicate which potential(s) should be computed for the pruning (lines 2–7).

One caveat of the candidate pruning is that the order of the policies computed during the second stage of the algorithm is no longer guaranteed to be monotonic increasing in terms of $L(\lambda^*)$ due to pruning. Therefore, we cannot use the termination condition in Algorithm 9. Although the termination condition is no longer available, the candidate pruning can be preferred when we have a certain computation time budget and finding a true optimal solution is not the primary interest.

Algorithm 12 shows a modified Closing-Gap algorithm which uses the candidate pruning as an approximation scheme to improve the convergence rate

Algorithm 11: Candidate-Pruning

Input: current best policy π^{k-1}, set of reachable states $\mathcal{S}_{\pi^{k-1}}$, approximation parameter ϵ

1 initialize index set $\mathcal{I} \leftarrow \{\}$ and pruned states set $\hat{\mathcal{S}} \leftarrow \{\}$
2 **if** π^{k-1} *is feasible* **then**
3 $\mathcal{I} \leftarrow \{0\}$, $\epsilon_0 \leftarrow \epsilon$
4 **else**
5 **for** $i = 1, ..., n$ **do**
6 **if** $g_i > 0$ **then**
7 $\mathcal{I} \leftarrow \mathcal{I} \cup \{i\}$, $\epsilon_i \leftarrow g_i$
8 **for** $s \in \mathcal{S}_{\pi^{k-1}}$ *in reverse order of BFS* **do**
9 $\hat{\mathcal{S}} \leftarrow \hat{\mathcal{S}} \cup \{s\}$
10 compute $\phi_i(\hat{\mathcal{S}})$ for every $i \in \mathcal{I}$.
11 **if** $\phi_i(\hat{\mathcal{S}}) \geq \epsilon_i$ *for every* $i \in \mathcal{I}$ **then**
12 **return** $\hat{\mathcal{S}} \backslash \{s\}$

at the cost of the termination condition. The major difference from the original Closing-Gap algorithm (Algorithm 9) is the process of removing a subset of states from the set of reachable states from which we generate candidates (lines 5 and 6). In addition, the termination condition has been replaced as the computation time budget (line 2).

Now, we show that our proposed algorithm with the approximation scheme in Algorithms 11 and 12 will produce an ϵ-optimal solution.

Theorem 4.1. *The proposed algorithm with candidate pruning (Algorithms 11 and 12) will produce an ϵ-optimal solution given the approximation parameter ϵ.*

Proof. To show that our approximation scheme guarantees ϵ-optimality with given $\epsilon > 0$, it suffices to show that every policy pruned by the approximation scheme for a policy π is either i) not better than π by ϵ in terms of the primary

Algorithm 12: Closing-Gap-With-Pruning

Input: Lagrangian dual optimal policy π, incumbent policy π^{inc}, lower bound LB, upper bound UB, Lagrangian multiplier λ, AND/OR graph G, VI error tolerance η, approximation parameter ϵ

initialize best policy order $k \leftarrow 1$, current best policy $\pi^1 \leftarrow \pi$, set of fixed state-action pairs $\pi_F^1 \leftarrow \{\}$, set of blocked state-action pairs $\pi_B^1 \leftarrow \{\}$, set of best policies $\Pi \leftarrow \{\pi^1\}$, candidates set $\Gamma \leftarrow \{\}$

while *time left* **do**

  $k \leftarrow k + 1$

  set $\mathcal{S}_{\pi^{k-1}}$ as a set of non-terminal reachable states from $\bar{s}$ by π^{k-1}.

  $\hat{\mathcal{S}} \leftarrow$ Candidate-Pruning$(\pi^{k-1}, \mathcal{S}_{\pi^{k-1}}, \epsilon)$

  $\mathcal{S}_{\pi^{k-1}} \leftarrow \mathcal{S}_{\pi^{k-1}} \backslash \hat{\mathcal{S}}$

  $(L^k, \pi^k, f^k, g^k, \pi_F^k, \pi_B^k), \Pi, \Gamma, G \leftarrow$ Next-Best-Policy$(k,\ \pi^{k-1},\ \mathcal{S}_{\pi^{k-1}}, \pi_F^{k-1}, \pi_B^{k-1}, \Pi, \Gamma, \lambda, G, \eta)$

  if $g_i^k \leq 0$ *for every* $i = 1, \ldots, N$ *and* $f^k < UB$ **then**

    UB $\leftarrow f^k$ and $\pi^{\text{inc}} \leftarrow \pi^k$

return π^{inc}

cost, or ii) infeasible. Throughout the proof, let π be the current best policy, $\mathcal{S}_\pi$ be a set of reachable states from $\bar{s}$ by π, $\hat{\mathcal{S}}_\pi$ be a set of pruned states based on Algorithm 11, and $\mathcal{S}_\pi^- = \mathcal{S}_\pi \backslash \hat{\mathcal{S}}_\pi$. Also, let g_i be the i-th value of π.

Suppose π is feasible. Then it implies that

$$\phi_0^\pi(\hat{\mathcal{S}}_\pi) = \bar{V}_0^\pi(\bar{s}) - \hat{V}_0^\pi(\bar{s}) < \epsilon.$$

Since we generate candidates with BFS order in Algorithm 10 and collect pruned candidates with the reverse order of BFS in Algorithm 11, the part of the policy for $\mathcal{S}_\pi^-$ is fixed whenever we generate a candidate from a state in $\hat{\mathcal{S}}_\pi$. In addition, since the fixed state-action pair set π_F is copied and augmented whenever we generate a candidate, $\mathcal{S}_\pi^-$ is fixed whenever we further generate a candidate from

a generated policy from π by deviating from a state in $\hat{\mathcal{S}}_\pi$. Since $\hat{V}_0^\pi(\bar{s})$ is a lower bound of having fixed policy for $\mathcal{S}_\pi^-$, any candidate that is pruned by the approximation scheme cannot be better than π by ϵ in terms of the primary cost.

Suppose π is infeasible. Then it implies that there is some i such that

$$\phi_i^\pi(\hat{\mathcal{S}}_\pi) = \bar{V}_i^\pi(\bar{s}) - \hat{V}_i^\pi(\bar{s}) < g_i,$$

where $g_i > 0$. Then with the same reasoning given in the feasible case, any candidate that is pruned by the approximation scheme cannot be better than π by g_i in terms of the i-th cost, and is infeasible. □

4.3. Speed-up techniques in candidate pruning

Although candidate pruning can increase the convergence rate of the proposed algorithm, the overhead of computing potential in line 10 of Algorithm 11 can be prohibitively high. In this section, we present two approaches that can reduce the computational overhead.

Batch update. The computational overhead in the candidate pruning mainly comes from taking the inverse in Eq. (9) to compute a potential. Given the fact that the computational cost of computing this inverse does not depend on the number of modified states, we can facilitate the candidate pruning process if we can prune a bunch of states at once instead of over multiple iterations. Therefore, we can balance the overhead and efficiency of the candidate pruning by adding more than 1 state at a time to the pruned set in Algorithm 11.

Rank-1 update. As pointed out previously, the computational overhead in the candidate pruning mainly comes from taking the inverse of the modified transition matrix in Eq. (9). However, for each iteration in Algorithm 11, only one row of the transition matrix is modified as the zero vector. This is actually rank-1 update of the matrix, and it is known that the Sherman-Morrison formula enables us to find the inverse of rank-1 updated matrix more efficiently

than taking the inverse from scratch given that the inverse of the original matrix was already computed [31, 32].

To compute the potential, we need to compute $(I-\hat{Q}^{\pi}_{\mathcal{S}'})^{-1}$ in Eq. (9), where we used subscription $\mathcal{S}'$ to emphasize that $\mathcal{S}'$ is a set of pruned states. However, if we already computed $(I-\hat{Q}^{\pi}_{\mathcal{S}'\setminus\{s_i\}})^{-1}$ for some $s_i \in \mathcal{S}'$, then we can apply the Sherman-Morrison formula as follows to compute $(I-\hat{Q}^{\pi}_{\mathcal{S}'})^{-1}$:

$$
\begin{aligned}
(I-\hat{Q}^{\pi}_{\mathcal{S}'})^{-1} &= (I-\hat{Q}^{\pi}_{\mathcal{S}'\setminus\{s_i\}}+uv^T)^{-1} \\
&= (I-\hat{Q}^{\pi}_{\mathcal{S}'\setminus\{s_i\}})^{-1} - \frac{(I-\hat{Q}^{\pi}_{\mathcal{S}'\setminus\{s_i\}})^{-1}uv^T(I-\hat{Q}^{\pi}_{\mathcal{S}'\setminus\{s_i\}})^{-1}}{1+v^T(I-\hat{Q}^{\pi}_{\mathcal{S}'\setminus\{s_i\}})^{-1}u}
\end{aligned}
$$

where u and v are column vectors defined as follows:

$$
u_j = \begin{cases} 0 & \text{if} \quad j \neq i, \\ 1 & \text{if} \quad j = i, \end{cases}
$$

and

$$
v_j = \big(\hat{Q}^{\pi}_{\mathcal{S}'\setminus\{s_i\}}\big)_{i,j}.
$$

5. Experimental Results

In this section, we evaluate the proposed method with three different state-of-the-art methods. The first is the MILP-based method which reduces a C-SSP to a MILP formulation and uses a commercial solver to find an optimal deterministic policy [3]. The second and the third algorithms are i-dual-LP and its variant, i-dual-MILP [8, 9]. Note that in this section, i-dual-LP refers to i-dual to make a clear distinction from i-dual-MILP, and the term i-dual is used to refer to either i-dual-LP or i-dual-MILP. I-dual is similar to the MILP-based method in the sense that it encodes a C-SSP to a mathematical programming formulation and uses a commercial solver to find a solution. However, instead of encoding and solving the problem at once, the i-dual incrementally expands the search space and solves a sequence of (MI)LPs until it finds an optimal solution. Note that the policy found by i-dual-LP might be stochastic but we compare the proposed method against i-dual-LP as well for reference.

We evaluate the proposed method in comparison with the benchmark methods on three domains to show various aspects of the proposed method in addition to its strength and limitations. In the racetrack domain, we evaluate the vanilla version of the algorithm for C-SSPs with a single constraint. Then in the elevators domain, we investigate the effect of candidate pruning for C-SSPs with multiple constraints under various parameter settings. Similarly, we study the effect of candidate pruning in the aircraft routing domain, but we also study how the algorithm scales with varying sizes of problems. In addition, in both elevators and aircraft routing domains, we demonstrate the advantage of using speed-up techniques developed in Section 4.3. Throughout the experiments, we set error tolerances η and ρ as 0.1^{10}, and all values of λ^{UB} are set as 0.1. In addition, as mentioned at the end of Section 3.3, λ_i^{∞} was adaptively increased by a factor of 10 until g_i^{∞} is negative or λ_i^{UB} reaches 10^{10} in line 5 of Algorithm 8.

The proposed algorithm was implemented in Python 3, and LPs and MILPs in the benchmark methods were solved using Gurobi 9. All of the algorithms were executed on an Intel core i5-7200U with 8GB of RAM. In addition, the

time-out was set to 1800 seconds. In the rest of this section, we provide the details of the comparison methods and problem domains. Then we provide the evaluation results to show the performance of the proposed method. Note that the source code of the proposed algorithm with all the test scenarios used in the experimental results can be found at `https://github.com/sk5050/Anytime-CSSP`.

5.1. Benchmark methods

5.1.1. MILP-based method [3]

If we allow to have a stochastic policy as our solution, then a C-SSP can be solved over the dual space with the following LP [10, 2, 8]:

$$\text{(LP 1)} \quad \min_{x} \quad \sum_{s\in\mathcal{S}}\sum_{a\in\mathcal{A}(s)} x_{s,a}\cdot C_0(s,a) \tag{10}$$

$$\text{s.t.} \quad \sum_{s\in\mathcal{S}}\sum_{a\in\mathcal{A}(s)} x_{s,a}\cdot C_i(s,a) \le \Delta_i \quad \forall i\in\{1,...,N\} \tag{11}$$

$$\sum_{a\in\mathcal{A}(s)} x_{s,a} - \sum_{s'\in\mathcal{S}}\sum_{a\in\mathcal{A}(s')} x_{s',a}\cdot Pr(s|s',a) = 0 \quad \forall s\in\mathcal{S}\backslash(\mathcal{G}\cup\{\bar{s}\}) \tag{12}$$

$$\sum_{a\in\mathcal{A}(\bar{s})} x_{\bar{s},a} - \sum_{s'\in\mathcal{S}}\sum_{a\in\mathcal{A}(s')} x_{s',a}\cdot Pr(\bar{s}|s',a) = 1 \tag{13}$$

$$\sum_{s\in\mathcal{G}}\sum_{s'\in\mathcal{S}}\sum_{a\in\mathcal{A}(s')} x_{s',a}\cdot Pr(s|s',a) = 1 \tag{14}$$

$$x_{s,a} \ge 0 \quad \forall s\in\mathcal{S}, a\in\mathcal{A}(s) \tag{15}$$

where $x_{s,a}$ is called occupation measure that represents the expected number of times action $a\in\mathcal{A}(s)$ is executed in s and $\bar{s}$ is the initial state. Once we obtain an optimal solution for occupation measure x^*, then an optimal policy can be obtained by normalizing x^* as follows:

$$\pi^*(s,a) = x^*_{s,a} / \sum_{a\in\mathcal{A}(s)} x^*_{s,a}.$$

Dolgov and Durfee [3] proposed a MILP-based solution method that finds an optimal deterministic policy for a C-SSP. The MILP formulation is basically based on LP 1 but with additional constraints as follows:

$$
\begin{aligned}
\text{(MILP 1)} \quad \min_{x} \quad & \sum_{s\in\mathcal{S}} \sum_{a\in\mathcal{A}(s)} x_{s,a} \cdot C_0(s,a) \\
\text{s.t.} \quad & (11)\text{–}(15) \\
& \sum_{a\in\mathcal{A}(s)} y_{s,a} \leq 1 \quad \forall s \in \mathcal{S} \\
& x_{s,a}/X \leq y_{s,a} \quad \forall s \in \mathcal{S}, a \in \mathcal{A}(s) \\
& y_{s,a} \in \{0,1\} \quad \forall s \in \mathcal{S}, a \in \mathcal{A}(s)
\end{aligned}
$$

where X is some constant that can be computed by solving LP 1 with the objective function replaced by $\max_{s,a} x_{s,a}$. Finally, an optimal deterministic policy can be obtained by

$$\pi^*(s) = a \Leftrightarrow y^*_{s,a} = 1.$$

5.1.2. i-dual-LP [8, 9]

Instead of solving LP 1 directly, starting with the initial state $\bar{s}$, i-dual-LP incrementally enlarges the search space guided by heuristics, and solves a sequence of LP 1 instances on the restricted search spaces. The procedure iteratively enlarges the search space until all of the tip states are goal states, by which time the solution is a closed policy. Moreover, if we use admissible heuristics, the solution is guaranteed to be an optimal policy. Similar to LP 1, however, the policy found by i-dual-LP may be stochastic.

5.1.3. i-dual-MILP [8]

As mentioned by the authors in [8], the i-dual-LP can be modified to find an optimal deterministic policy for a C-SSP by replacing LP with MILP formulation in its iterative process. In other words, for each iteration, the i-dual-MILP solves MILP 1 instead of LP 1 for a restricted search space. Although this guarantees finding an optimal deterministic policy, the computational overhead for

solving multiple MILPs might be too high. Therefore, we use a modified version of i-dual-MILP in this paper. In the modified version, to reduce the computational cost for enlarging part of the i-dual algorithm, we first run i-dual-LP until we find an optimal stochastic policy. Then we switch from i-dual-LP to i-dual-MILP to find an optimal deterministic policy.

5.2. Problem domains

5.2.1. Racetrack

The racetrack domain was first introduced in [33] and is one of the most widely used domains in SSP studies. Fig. 9 shows an example of the racetrack problem, where the goal of the problem is passing the finish line from an initial state located at the starting line as fast as possible, by accelerating or decelerating properly. The set of states consists of integer vectors $(x, y, \dot{x}, \dot{y})$, where the first two indicate the position in the grid world, and the latter two indicate speed in the x and y directions. For each action, the car can decide whether to accelerate or decelerate by 1 or stay in the current speed for each direction, all of which results in 9 different available actions. However, the controller is not perfect, so with a probability p, an action has no effect, and the speed remains the same as before. In addition, if the car hits any boundary of the racetrack except the finish line, the car is randomly located at one of the grids in the starting line with zero velocity. Finally, the goal is reached if the car passes the finish line.

In addition to the original problem, we added one more aspect to make the problem as a C-SSP. As marked as the red shaded area in the examples in Fig. 10, there are bumpy grids that cause tire wear. Therefore, in the constrained racetrack problem, there is a secondary cost that damages the tire only from bumpy grids and the maximum damage should be bounded.

In the evaluation, we used four different racetrack configurations as shown in Fig. 10, where green grids show initial locations. Note that the layouts of the racetrack are from [33] and [34], and we made two bumpy grids modifications for each layout. In addition, we used $p = 0.1$ for the "slippery" scenario for all the

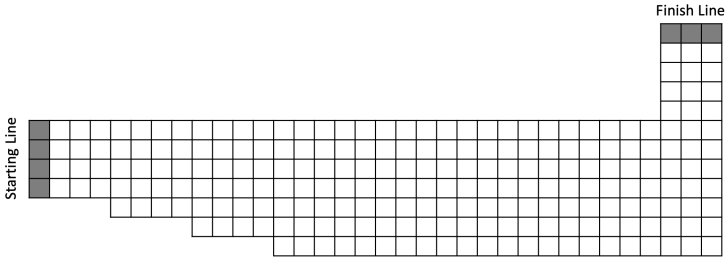


Figure 9: An example problem instance of the racetrack domain.

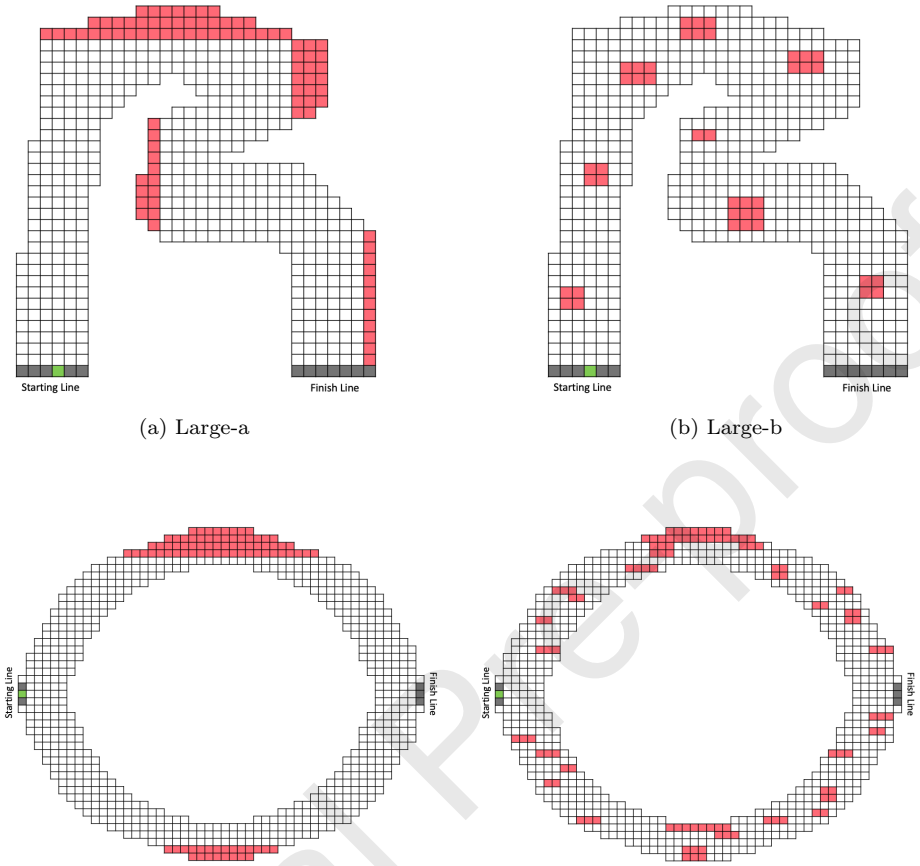
evaluations. For heuristics, we used "shortest-path heuristic" for the primary cost [16] and "zero heuristic" for the secondary cost. Finally, the secondary cost induced by a bumpy grid is 10 and the secondary cost is bounded by 1 for all the problem instances.

5.2.2. Elevators

The elevators domain was introduced in [8, 9] for the evaluation of i-dual method. There are five parameters that define a problem: the number of floors in the building n, the number of elevators e, the number of persons that are known to be waiting for an elevator w, the number of hidden persons that are expected to arrive and request an elevator h, and the probability p of arrival of any hidden person for each time step.

The objective of the problem is minimizing the total number of elevators' movements. In addition, there are two constraints for each person, where one upper bounds the waiting time and the other upper bounds the travel time in an elevator. For each person, the former is bounded by $0.75n$ and the latter is bounded by $n + 1$.

We use the simple admissible heuristic functions defined as follows. An obvious optimistic waiting time for a person is the distance from her current position to the closest elevator if she is currently waiting, and 0 otherwise. Similarly, an optimistic heuristic for travel time is the difference between the current position and the destination if she has not arrived at the destination, and 0 otherwise. Then, for each person, we can compute the total admissible heuristic time to



(a) Large-a (b) Large-b

(c) Ring-a (d) Ring-b

Figure 10: Racetrack configurations used in the evaluation.

the destination by summing optimistic heuristic waiting and travel times. Finally, an admissible heuristic for the primary cost can be obtained by taking the maximum of every person's total admissible heuristic time.

In our experiments, we set $n = 20$, $e = 2$, $h = 1$, $w = 2$ and $p = 0.75$, which result in 6 constraints in each problem. In addition, to study how the performance of the algorithm is changed based on the number of constraints, we tested on problems with 4 constraints where constraints on non-hidden passengers' travel time are relaxed.

5.2.3. Risk-bounded aircraft routing problem

Chance-constrained SSP and its reduction to C-SSP. The chance-constrained approach has been widely studied in various domains including operations research, motion planning, control, to name a few [35, 36, 37]. One of the recent applications of the chance-constrained approach is risk-bounded planning under uncertain environments [14, 38]. In their applications, the user can define the level of safety guarantees and systematically balance the efficiency and the risk.

Among different risk-bounded planning approaches, we focus on the risk-bounded conditional planning which is formulated as a chance-constrained SSP (CC-SSP) [14]. CC-SSP is defined as an SSP with two added elements $\overline{\mathcal{S}}$ and Δ. The objective of a CC-SSP is finding an optimal policy that maintains the risk of violating a constraint in $\overline{\mathcal{S}}$ below the given threshold Δ. The latter constraint can be formulated mathematically with the notion of execution risk, which is defined as follows:

$$er(\bar{s}|\pi) = 1 - Pr\left(\bigwedge_{i=0}^{\infty} Sa_i = 1 \middle| s_0 = \bar{s}, \pi\right), \tag{16}$$

where Sa_i is a Bernoulli random variable with value 1, when an agent has not violated constraints at time i. Then we can enforce the risk-bound with the following constraint:

$$er(\bar{s}|\pi) \leq \Delta. \tag{17}$$

To apply our proposed method, we reduce a CC-SSP $\mathcal{Q} = \langle \mathcal{S}, \bar{s}, \mathcal{G} \subset \mathcal{S}, \overline{\mathcal{S}}, \mathcal{A}, T, C, \Delta \rangle$ problem to C-SSP $\mathcal{H} = \langle \mathcal{S}', \bar{s}', \mathcal{G}' \subset \mathcal{S}', \mathcal{A}', T', \vec{C}', \vec{\Delta}' \rangle$ problem with the following mapping:

- $\mathcal{S}' = \mathcal{S} \cup \{g_0\}$, where g_0 is an artificial goal.
- $\bar{s} = \bar{s}'$
- $\mathcal{G}' = \mathcal{G} \cup \{g_0\}$

- A set of actions $\mathcal{A}'$ is defined as follows:

$$\mathcal{A}'(s) = \begin{cases} a_0 \quad \text{for} \quad s \in \overline{\mathcal{S}} \\ \mathcal{A}(s) \quad \text{otherwise.} \end{cases}$$

  where a_0 is an artificial action.

- Transition function T' is defined as follows:

$$T'(s,a,s') = \begin{cases} 1 \quad \text{for} \quad s \in \overline{\mathcal{S}}, s' = g_0, \\ 0 \quad \text{for} \quad s \in \overline{\mathcal{S}}, s' \neq g_0, \\ T(s,a,s') \quad \text{otherwise.} \end{cases}$$

- $\vec{C}' = [C_0, C_1]$, where

$$C_0(s,a) = \begin{cases} 0 \quad \text{for} \quad s \in \overline{\mathcal{S}} \\ C(s,a) \quad \text{otherwise,} \end{cases}$$

  and

$$C_1(s,a) = \begin{cases} 1 \quad \text{for} \quad s \in \overline{\mathcal{S}} \\ 0 \quad \text{otherwise.} \end{cases}$$

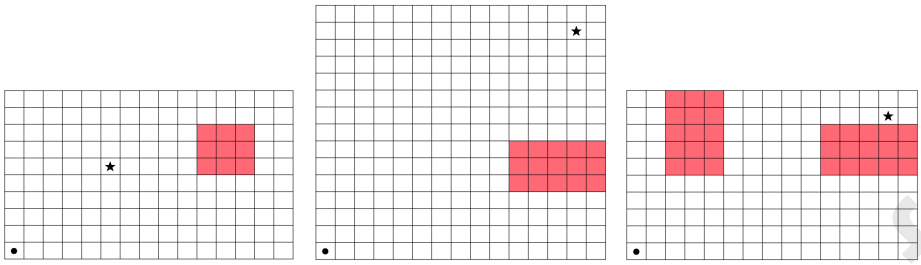
- $\vec{\Delta}' = [\Delta_1]$ where $\Delta_1 = \Delta$.

Then,

$$\mathbb{E}\left[\sum_{k=0}^{\infty} C_1(s_k, a_k) \middle| s_0 = \bar{s}, \pi\right] \leq \Delta_1$$

is equivalent to Eq. (17) for a policy π, and an optimal solution to $\mathcal{Q}$ can be obtained by solving $\mathcal{H}$.

Note that a chance-constrained SSP enables us to handle dead-ends in a probabilistic way. In other words, instead of using the common way of handling dead-ends with finite-penalty [39], we can define dead-ends as constrained states and bound the probability of reaching a dead-end by some given allowable risk Δ.



(a) Configuration 1. (b) Configuration 2. (c) Configuration 3.

Figure 11: Airspace configuration examples used in the evaluation. Aircraft initial and goal positions are shown with circles and stars, respectively, and convective weather cells are shown with the red shaded regions. A random problem instance is generated by randomly selecting initial location(s) and moving direction(s) of weather cell(s).

Risk-bounded Aircraft Routing Problem in the Presence of Adverse Weather Conditions. Now we present the risk-bounded aircraft routing problem in the presence of adverse weather conditions as an application of CC-SSP. The problem is inspired by [40]. In their work, an aircraft routing problem was formulated as a stochastic shortest path problem where aircraft movement is deterministic and a convective weather cell transitions stochastically. To avoid encountering a convective weather cell, they introduced a secondary cost when an aircraft encounters a weather cell, and penalize the secondary cost with some fixed weight on top of the primary cost.

Different from the originally introduced penalty version of the problem, we formulate the problem as a CC-SSP, which explicitly bounds the probability of encountering a convective weather cell. Fig. 11 shows examples of the problem, where a circle and the red shaded region describe an aircraft position and a weather cell, respectively. Similar to [40], for each time step, an aircraft can deterministically move to an adjacent cell and a weather cell moves stochastically based on a given probability distribution. Then the objective is to get to the destination waypoint (marked as a star) as fast as possible while maintaining the probability of encountering a weather cell below the given threshold Δ. In our evaluations, we assumed that a weather cell moves towards a given

direction for a given probability p, and it moves $+90$ and -90 directions with $(1-p)/2$, respectively. In addition, we used Euclidean distance for a heuristic of the primary cost function, and zero heuristic for the risk.

We tested on 3 different configurations with varying sizes and the number of weather cells, where the first two configurations have a single weather cell and the third configuration has two weather cells. Note that we generated an instance by randomly selecting the initial position(s) and moving direction(s) for weather cell(s), where Fig. 11 shows example instances for each configuration. Also note that p and Δ were set as 0.8 and 0.01, respectively, for every instance.

5.3. Results

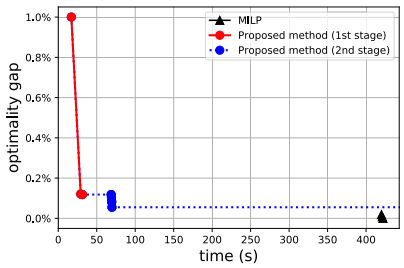
5.3.1. Racetrack

Table 1 shows the evaluation results of three different methods for the racetrack problems in Fig. 10, where the first column shows the configuration name. Columns 2–5 show the results of the MILP-based method, where the first two columns show the time and optimality gap of the initial feasible solution and the latter two columns show the time and the total number of states expanded when a true optimal solution was found. Then the results of the i-dual-LP and i-dual-MILP are followed where time and the number of states expanded are recorded for each method. Finally, the remaining columns show the results of the proposed method. To show the anytime property of the proposed method, we report two solutions which obtained $< 1\%$ and $< 0.1\%$ optimality gaps, respectively, where the optimality gap, computation time, and the number of states expanded are shown for each solution.

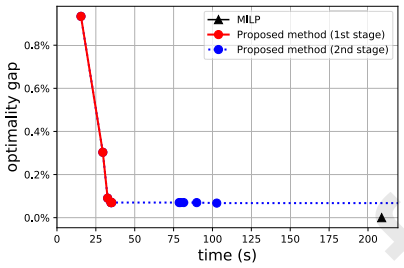
As shown in the table, the proposed method could obtain solutions that have only $< 1\%$ optimality gap with less than 10% of computation time of the MILP-based method for the first three racetrack configurations (Large-a, Large-b and Ring-a). Although the speed-up of the proposed method for the last configuration was smaller than for the other configurations, the proposed method still could obtain the solution with $< 1\%$ optimality gap with less than half of the computation time used by the MILP-based method. In addition,

Config.	MILP				i-dual				Proposed method							
	Initial		Opt		LP		MILP		$< 1\%$ optimality gap				$< 0.1\%$ optimality gap			
	time (s)	gap	time (s)	states	time (s)	states	time (s)	states	gap	time (s)	states		gap	time (s)	states	
Large-a	264.2	1.42%	350.6	21620	655.6	12906	-	-	0.12%	29.2	14083		0.09%	69.2	14083	
Large-b	291.5	0.00%	291.5	21620	527.3	13060	-	-	0.93%	15.4	14715		0.09%	32.6	14726	
Ring-a	392.1	0.00%	392.1	30446	466.6	12954	708.9	12992	0.65%	15.0	16797		0.04%	21.1	16797	
Ring-b	151.8	0.00%	151.8	30446	1082.0	16091	1494.5	16205	0.06%	34.0	22396		0.06%	34.0	22396	

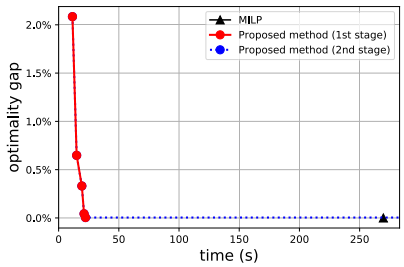
Table 1: Evaluation results of the racetrack problem instances.



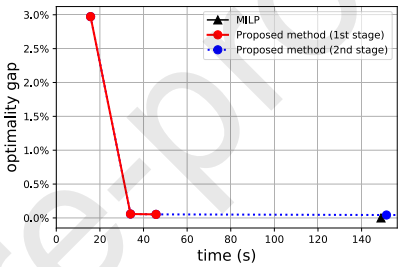
(a) Results of the large-a configuration.



(b) Results of the large-b configuration.



(c) Results of the ring-a configuration.



(d) Results of the ring-b configuration.

Figure 12: Solution histories of the MILP-based and the proposed method on racetrack problem instances.

the proposed method could reduce the gap further down below 0.1% with only marginal time except for the first configuration. Again, the computation time of the proposed method for the first configuration for $< 0.1\%$ solution was smaller than the computation time used by the MILP-based method for the initial feasible solution.

Fig. 12 shows the anytime solution histories for both MILP-based and the proposed methods, where the black solid line with triangle markers shows the solutions of the MILP-based method, the red solid line and blue dotted line with circle markers show the solutions of the first stage and the second stage of the proposed method, respectively. The results in the figures show that the first stage of the method could improve the solution quality very quickly and the second stage could further reduce the gap with the remaining time. However,

the results show that the proposed method struggles to close the gap after it obtains a certain optimality gap ($< 0.1\%$ in all cases), although the gap was small.

Finally, it is interesting to notice that both i-dual-LP and i-dual-MILP performed poorly even compared with the MILP-based method. Although it is true that the i-dual expands only part of the search space, there is computational overhead by solving a sequence of (MI)LPs whenever it expands the states. Therefore, the overhead can dominate the advantage when the difference between the number of states expanded by the (MI)LP-based method and i-dual is small. This can be also observed in experimental results of [8, 9], where the i-dual dominates the LP-based method when the number of states expanded for i-dual was only $1 - 2\%$ of the number of states expanded by the LP-based method.

5.3.2. Elevators

In this experiment, we investigate the effects of approximations that are used in Algorithms 7 and 12 as well as the advantage of using speed-up techniques introduced in Section 4.3. For M, which controls the number of iterations for solving Lagrangian dual (Algorithm 7), we test two different values, where the one is the number of constraints and the other is ∞. In other words, with the former M value, we basically optimize each coordinate of λ once, and with the latter M value, we run Algorithm 7 until it converges. On the other hand, we also can approximate the second stage of the proposed method by pruning candidates with parameter ϵ at the cost of optimality, as introduced in Section 4. For ϵ, we test four different values, 0, 0.1, 0.2 and 0.3, where $\epsilon = 0$ implies that we don't use candidate pruning and larger ϵ prunes candidates more aggressively. Finally, for each ϵ value except 0, we evaluate the algorithm with or without speed-up techniques to demonstrate the advantage of using speed-up techniques. Note that we use the rank-1 update for the speed-up technique throughout the experiment for simplicity.

For the comparison methods, we report the number of problems solved within

the time limit (1800 seconds), the average and standard deviation of the computation times for solved problems, and the average number of states for solved problems. For the proposed method, we summarize three anytime solutions that obtained $< 10\%$, $< 1\%$ and $< 0.1\%$ optimality gaps, respectively. In addition to the metrics that are reported for the comparison methods, we also report the average speed-up from either MILP or i-dual-MILP that performed better along with minimum and maximum speed-up. Note that the speed-up is calculated as $\frac{\min\{t_{\text{MILP}}, t_{\text{i-dual-MILP}}, 1800\}}{t_{\text{proposed}}}$, where t_{MILP}, $t_{\text{i-dual-MILP}}$ and t_{proposed} are computation times for MILP, i-dual-MILP and the proposed method, respectively. Also note that the speed-up is 1 if every method could not find a solution within the time limit and 0 if at least one of the comparison methods could find a solution while the proposed method could not find a solution.

Table 2 shows the results for the elevators problems with 6 constraints, where we tested on 30 randomly generated problem instances. Note that the first two columns of the table for the proposed method show approximation parameters M and ϵ, respectively, and the star marks ($\star$) next to ϵ values indicate that the speed-up technique was used. For the case without any approximation ($M = \infty$ and $\epsilon = 0$), the proposed method could not solve 8–9 out of 30 problem instances for the optimality gap less than 10%. With candidate pruning ($\epsilon = 0.1, 0.2, 0.3$), however, the proposed algorithm performed better than the case without approximation and found near-optimal solutions with $< 0.1\%$ optimality gap for 26–27 out of 30 problem instances.

This tendency is further emphasized with the approximation in Lagrangian dual with $M = 6$. In other words, although the proposed algorithm found $< 0.1\%$ optimality gap solutions for 22 out of 30 cases without candidate pruning ($\epsilon = 0$), the number of problems solved increases as it prunes candidates more aggressively, and found $< 0.1\%$ optimality gap solutions for 28–29 out of 30 instances with candidate pruning ($\epsilon = 0.1, 0.2, 0.3$). The best results overall without speed-up technique were obtained with $M = 6$ and $\epsilon = 0.3$, where the proposed method achieved 22x of average speed-up against the comparison methods for $< 1\%$ optimality gap, and achieved 21.7x of average speed-up ex-

cept one instance for $< 0.1\%$ optimality gap. The results shown in Table 2 also compare the algorithm performance with or without using speed-up techniques. Although the number of solved problems remains the same, the average computation times were consistently reduced for almost every case. This shows that the speed-up technique indeed reduced the computational overhead induced by candidate pruning and resulted in better efficiency in terms of the overall computation time. In addition, it is worth noting that the average number of states expanded by the proposed algorithm is only about 20% of the entire state space that has been expanded by the MILP-based method.

During the experiment, the proposed method with limited M performed better because there is not much benefit of optimizing Lagrangian dual exactly and it is rather more efficient to proceed to the second stage after it finds an approximated solution to the Lagrangian dual. This is well shown in Fig. 13, where the solid lines with circle markers show the solution histories with $M = 6$, and the dotted lines with square markers show the solution histories with $M = \infty$. Note that the speed-up technique was used for every case in the figure. The cases with both $M = 6$ and $M = \infty$ have similar histories, but the histories with $M = \infty$ are shifted to the right over time. This is resulted from the fact that, with $M = \infty$, the algorithm spent more time in the first stage to optimize the Lagrangian dual, which didn't provide any benefit in this problem instance.

In addition, the proposed method generally converged to near-optimal solutions faster as it pruned candidates more aggressively. The candidate pruning, however, has a trade-off between efficiency and the optimality, as discussed in Section 4. This trade-off can be observed in one of the test instances that is shown in Fig 14. In this instance, the algorithm achieved $< 0.1\%$ optimality gap successfully with $\epsilon = 0.1$ and 0.2. With $\epsilon = 0.3$, however, although the convergence rate up to $< 1\%$ optimality gap is the highest, it could not obtain a solution with $< 0.1\%$ optimality gap, which might be due to aggressive candidate pruning. Note that this is an expected behavior, and in fact what we desire with approximation: better convergence at the cost of optimality. However, the experimental results show that the candidate pruning increases convergence rate

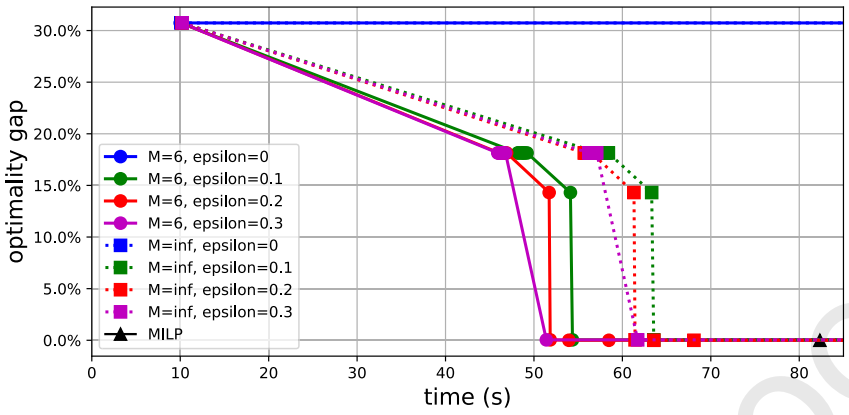


Figure 13: Solution histories of the MILP-based method and the proposed method with different parameters for an instance of elevators problem.

without losing near-optimality in most of the cases.

Similar to Table 2, Table 3 shows the results for the elevators problems with 4 constraints, where we tested on 30 randomly generated problem instances. A similar comparison can be made between the proposed method and the comparison methods with the case with 6 constraints. Without using the speed-up technique, the proposed algorithm performed the best with $M = 4$ and $\epsilon = 0.3$ where it achieved more than 30x average speed-up against the comparison methods for $< 1\%$ optimality gap, and found $< 0.1\%$ optimality gap solutions for 28 out of 30 instances. Moreover, the speed-up technique further reduced the average computation time for every case.

However, a major difference between Table 2 and Table 3 is that different M values have a bigger effect in the experiments with 6 constraints than 4 constraints. As mentioned earlier, the smaller M can be advantageous if optimizing Lagrangian dual requires too much effort, as shown in Fig. 14. Therefore, the approximation in Lagrangian dual with M can affect the algorithm more significantly as the Lagrangian dual has a larger dimension with more constraints.

Finally, throughout the experiment, MILP-based and the proposed methods outperformed i-dual-LP and i-dual-MILP. Possible reasons that i-dual particularly showed poor performance on the elevators domain are the following. First,

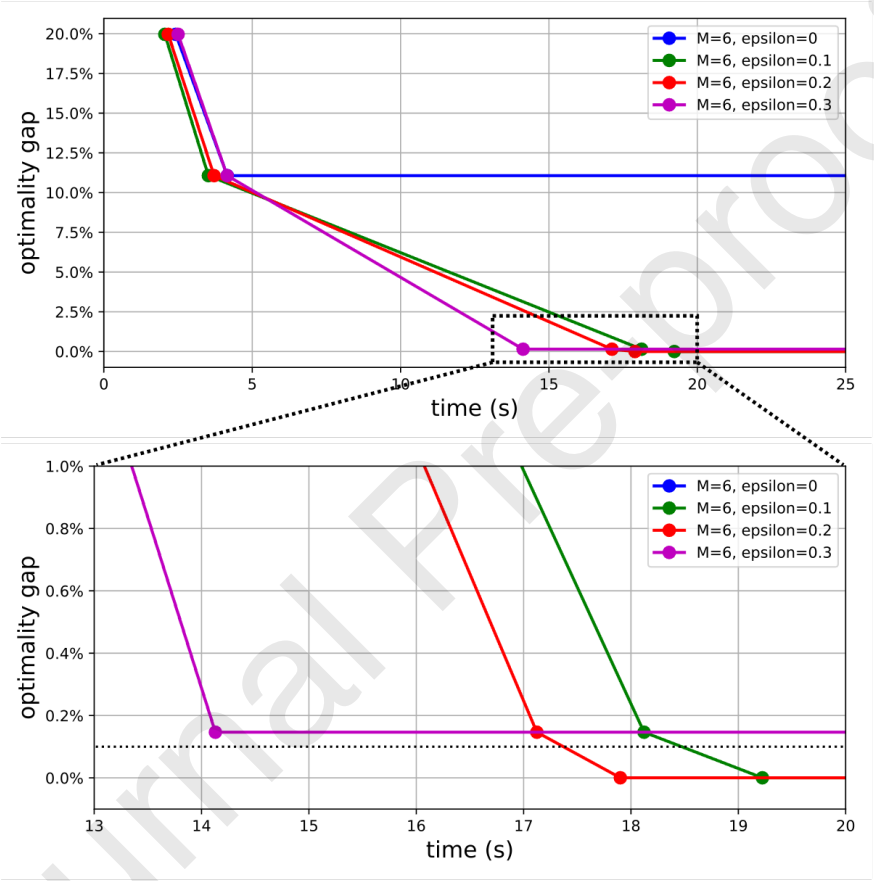


Figure 14: Solution histories of the proposed method for an instance of elevators problem with different parameters (top) and zoom in view of near-optimal solutions (bottom).

Baseline	# solved / time (s)	states
MILP	30 (140.4 ± 81.5)	104495.3
i-dual-LP	14 (486.8 ± 527.2)	4400.6
i-dual-MILP	13 (467.3 ± 559.1)	3955.9

		Proposed method							
		< 10% optimality gap			< 1% optimality gap			< 0.1% optimality gap		
M	ϵ	# solved / time (s)	states	speed-up	# solved / time (s)	states	speed-up	# solved / time (s)	states	speed-up
∞	0	22 (33.8 ± 50.2)	21859.5	23.3 [0, 286.2]	21 (43.1 ± 60.1)	22743.9	20.5 [0, 286.2]	21 (43.4 ± 60.5)	22759.5	20.5 [0, 286.2]
	0.1	28 (38.9 ± 59.1)	19820.8	24.5 [0, 281.2]	27 (42.9 ± 60.0)	20594.4	21.4 [0, 281.2]	26 (44.7 ± 61.0)	21241.8	20.8 [0, 281.2]
	0.1 (*)	28 (36.3 ± 51.5)	19820.8	24.8 [0, 287.5]	27 (40.0 ± 52.3)	20594.4	21.8 [0, 287.5]	26 (41.7 ± 53.1)	21241.8	21.2 [0, 287.5]
	0.2	28 (38.7 ± 59.1)	19831.6	24.6 [0, 284.6]	27 (42.6 ± 59.8)	20607.2	21.6 [0, 284.6]	27 (43.0 ± 60.0)	20619.7	21.3 [0, 284.6]
	0.2 (*)	28 (35.4 ± 50.1)	19831.6	24.9 [0, 283.9]	27 (39.2 ± 50.9)	20607.2	21.8 [0, 283.9]	27 (39.5 ± 51.2)	20619.7	21.6 [0, 283.9]
	0.3	29 (35.6 ± 53.2)	20075.8	25.0 [0, 286.2]	27 (40.7 ± 53.4)	19869.7	21.1 [0, 286.2]	26 (41.8 ± 54.5)	20482.3	20.8 [0, 286.2]
	0.3 (*)	29 (38.5 ± 54.0)	20075.8	25.3 [0, 283.8]	27 (35.8 ± 45.7)	19869.7	21.5 [0, 283.8]	26 (36.8 ± 46.8)	20482.3	21.1 [0, 283.8]
6	0	22 (25.2 ± 27.9)	21595.2	23.7 [0, 285.5]	22 (27.3 ± 28.2)	21784.0	21.3 [0, 285.5]	22 (27.5 ± 29.1)	21803.6	21.2 [0, 285.5]
	0.1	30 (26.7 ± 27.1)	19383.7	25.7 [1.4, 283.5]	29 (29.5 ± 27.8)	20082.2	22.5 [0, 283.5]	28 (30.7 ± 28.7)	20669.6	21.8 [0, 283.5]
	0.1 (*)	30 (25.9 ± 26.5)	19383.7	26.1 [1.5, 282.6]	29 (28.2 ± 26.8)	20082.2	22.7 [0, 282.6]	28 (29.4 ± 27.6)	20669.6	21.9 [0, 282.6]
	0.2	30 (26.9 ± 27.1)	19392.9	25.6 [1.4, 283.9]	29 (29.7 ± 27.9)	20093.5	22.4 [0, 283.9]	29 (30.0 ± 28.5)	20109.1	22.3 [0, 283.9]
	0.2 (*)	30 (25.6 ± 26.3)	19392.9	26.4 [1.6, 285.0]	29 (27.9 ± 26.6)	20093.5	22.9 [0, 285.0]	29 (28.2 ± 27.3)	20109.1	22.6 [0, 285.0]
	0.3	30 (26.3 ± 26.8)	19376.8	26.1 [1.4, 285.9]	30 (31.5 ± 27.9)	19574.8	22.0 [1.2, 285.9]	29 (32.2 ± 28.8)	20117.8	21.7 [0, 285.9]
	0.3 (*)	30 (25.3 ± 26.3)	19376.8	26.4 [1.6, 285.5]	30 (29.4 ± 27.0)	19574.8	22.5 [1.3, 285.5]	29 (30.1 ± 28.0)	20117.8	22.1 [0, 285.5]

Table 2: Evaluation results of the elevators problems with 6 constraints.

as mentioned in the racetrack results, the overhead of solving a sequence of (MI)LPs in i-dual cannot be compensated if the difference between the number of states expanded by i-dual and the size of the state space is small. Secondly, which is also related to the previous point, the quality of the heuristic functions is important to reduce the search space, but our naive optimistic heuristics are weak in the sense that they do not guide the search well. Finally, the ratio of the branching factor to the size of the state space does matter for i-dual. For example, if the branching factor is big compared to the size of the state space, the number of (MI)LP instances that i-dual solves will be small, and vice versa. Since the branching factor of the elevator is 1 or 2 for each action (either a hidden person arrives or not) and the size of the state space is comparatively big, i-dual solves too many intermediate (MI)LPs until reaching the required search space.

5.3.3. Risk-bounded aircraft routing problem

In this experiment, we demonstrate the proposed method on the risk-bounded aircraft routing problem. Similar to the elevators problem, we investigate the effects of the candidate pruning and speed-up techniques, where we test four different ϵ values, 0, 0.1, 0.2 and 0.3, for the candidate pruning. Note that the approximation for the Lagrangian dual with the parameter M is not applicable for this problem since there is only a single constraint. In addition, to study how the algorithm scales with the size of the problems, we test on three different configurations shown in Fig. 11, ordered in the size of the state space. For each configuration, we tested on 30 problem instances which were generated by randomly selecting initial position(s) of weather cell(s) and moving direction(s).

Similar to the previous section, we report the number of problems solved within the time limit (1800 seconds), the average and standard deviation of the computation times for solved problems, and the average number of states for solved problems for comparison methods. Again, we report three anytime solutions that achieve $< 10\%$, $< 1\%$ and $< 0.1\%$ optimality gaps, respectively, with the same metrics used for comparison methods and speed-up against the

Baseline	# solved / time (s)	states
MILP	30 (138.9 ± 74.1)	109112.9
i-dual-LP	15 (856.1 ± 496.8)	6784.5
i-dual-MILP	12 (738.1 ± 407.1)	6347.0

		Proposed method							
		< 10% optimality gap			< 1% optimality gap		< 0.1% optimality gap		
M	ϵ	# solved / time (s)	states	speed-up	# solved / time (s)	states	speed-up	# solved / time (s)	states	speed-up
8	0	27 (23.9 ± 42.7)	18956.3	36.3 [0, 205.9]	26 (29.0 ± 47.8)	19978.2	31.0 [0, 205.9]	26 (29.0 ± 47.8)	20035.0	25.6 [0, 188.9]
	0.1	30 (33.5 ± 57.6)	18137.5	35.5 [0.5, 189.2]	27 (40.2 ± 63.1)	19547.1	28.6 [0, 189.2]	26 (35.3 ± 58.6)	19953.3	24.8 [0, 189.2]
	0.1 (*)	30 (30.5 ± 50.3)	18137.5	36.4 [0.6, 186.8]	27 (36.9 ± 55.9)	19547.1	29.4 [0, 186.8]	26 (32.6 ± 52.1)	19953.3	25.0 [0, 186.8]
	0.2	30 (34.1 ± 59.1)	18125.1	37.3 [0.5, 212.2]	28 (39.9 ± 64.3)	19106.0	32.3 [0, 212.2]	27 (34.8 ± 59.4)	19482.2	26.4 [0, 189.2]
	0.2 (*)	30 (28.7 ± 46.9)	18125.1	35.6 [0.7, 188.9]	28 (33.8 ± 52.2)	19106.0	31.3 [0, 188.9]	27 (30.3 ± 49.5)	19482.2	26.6 [0, 188.9]
	0.3	30 (32.6 ± 54.9)	18141.2	36.3 [0.5, 189.2]	29 (38.0 ± 58.4)	18681.9	32.1 [0, 189.2]	27 (34.0 ± 56.5)	19434.3	26.4 [0, 188.4]
	0.3 (*)	30 (23.7 ± 41.0)	18141.2	36.2 [1.7, 189.4]	29 (31.0 ± 40.1)	18681.9	32.3 [0, 189.4]	27 (30.7 ± 50.8)	19434.3	27.1 [0, 189.4]
4	0	27 (24.5 ± 44.2)	18979.4	35.3 [0, 189.6]	26 (28.1 ± 49.4)	19967.4	29.8 [0, 189.6]	26 (28.2 ± 46.4)	20030.6	24.8 [0, 189.6]
	0.1	30 (32.2 ± 53.6)	18169.4	35.3 [0.5, 188.7]	28 (37.0 ± 56.4)	19493.6	29.5 [0, 188.7]	27 (33.7 ± 53.5)	19888.9	24.4 [0, 188.7]
	0.1 (*)	30 (31.1 ± 51.4)	18169.4	36.5 [0.6, 190.2]	28 (35.6 ± 53.8)	19493.6	30.1 [0, 190.2]	27 (31.4 ± 49.5)	19888.9	24.7 [0, 190.2]
	0.2	30 (35.1 ± 60.7)	18149.2	36.0 [0.5, 188.8]	29 (38.9 ± 62.5)	19071.3	31.0 [0, 188.8]	28 (34.1 ± 57.6)	19438.9	25.6 [0, 188.8]
	0.2 (*)	30 (28.8 ± 47.1)	18149.2	36.1 [0.7, 195.4]	29 (32.2 ± 49.0)	19071.3	31.4 [0, 195.4]	28 (29.0 ± 46.4)	19438.9	25.7 [0, 188.8]
	0.3	30 (34.4 ± 59.1)	18145.3	36.4 [0.5, 189.1]	30 (38.3 ± 59.8)	18641.5	31.4 [0.5, 189.1]	28 (34.2 ± 57.7)	19379.1	25.7 [0, 189.1]
	0.3 (*)	30 (23.9 ± 41.7)	18145.3	36.5 [1.7, 189.9]	30 (29.6 ± 45.9)	18641.5	31.6 [0.9, 189.9]	28 (29.2 ± 47.4)	19379.1	26.4 [0, 189.9]

Table 3: Evaluation results of the elevators problems with 4 constraints.

baselines. Note that when the optimal cost is unknown (due to failure from both MILP and i-dual), the optimality gap is estimated as the best lower bound on the optimal cost computed by the proposed method.

Table 4 shows the results for the first configuration (Fig. 11a), where the MILP-based method performed the best among baselines by solving every problem with average of 12.7 seconds. First of all, the proposed method without candidate pruning solved every instance for $< 1\%$ optimality gap with average of 126.6x speed-up compared to the comparison methods. For $< 0.1\%$ optimality gap, however, it failed to find solutions for 2 out of 30 cases. Given that the size of the state space for the first configuration is small (6305.2 number of states in average), these results show how the proposed algorithm possibly struggles to converge to (near-)optimal solution when there are too many policies with almost the same primary cost.

Roughly speaking, the purpose of the candidate pruning is to skip such policies by examining their potential in cost improvement, which in fact is demonstrated in Table 4. By using the candidate pruning with $\epsilon = 0.1$ and without the speed-up technique, the algorithm found $< 0.1\%$ optimality gap solutions for every case. However, the computation time also increased significantly. Although the average computation time for $< 0.1\%$ gap solutions are still lower than with the MILP-based method, the standard deviation is very high. In addition, as the minimum speed-up value shows, there is a case that found $< 0.1\%$ gap solution five times slower than a comparison method. In fact, this is not surprising, since candidate pruning has computational overhead that comes from performing matrix inversions.

The speed-up techniques have been introduced to reduce such computational overhead, which is well demonstrated in the case with $\epsilon = 0.1$ and the speed-up technique. With the speed-up technique, the average computation time has reduced from 8.9 to 1.8 seconds with a much smaller standard deviation. In addition, the minimum speed-up value shows that the algorithm achieved $< 0.1\%$ optimality gap faster than comparison methods for every instance.

However, as discussed in the previous section with Fig. 14, there is a trade-off

in the candidate pruning: candidate pruning improves convergence rate at the cost of optimality. This is shown in Table 4 for $\epsilon = 0.2$ and 0.3 cases. Although the proposed algorithm could still find more $< 0.1\%$ optimality gap solutions than the case without candidate pruning, the number of problems solved was decreased compared to the case with $\epsilon = 0.1$ due to aggressive pruning.

Table 5 shows the results for the second configuration (Fig. 11b). The proposed method found $< 1\%$ gap solutions for every case with or without candidate pruning with more than 300x of average speed-up against the comparison methods, whereas the MILP-based method, which performed the best among baselines, only could solve 18 out of 30 test instances.

Similar to the results for the first configuration, the algorithm failed to achieve $< 0.1\%$ optimality gap for 5 out of 30 instances without the candidate pruning. It is interesting to note, however, the algorithm performed even worse with the candidate pruning with $\epsilon = 0.1$ and without the speed-up technique both in terms of the number of problems solved and computation time. Again, it is not surprising because the computational overhead from the candidate pruning is based on the matrix inversion, which becomes more significant as the size of a matrix (which depends on the size of a policy) increases. Therefore, in this case, the computational overhead is so large to prevent the algorithm from taking advantage of the candidate pruning. However, again, the algorithm could take advantage of the candidate pruning with the speed-up technique by reducing the computational overhead, which results in solving 27 out of 30 instances for $< 0.1\%$ gap with average of 276.4x speed-up against the comparison methods.

Fig. 15 shows the solution histories for one of the test instances of the second configuration and well demonstrates the overall tendency described above. Without pruning (blue line), the algorithm achieved the near-optimal solution slower than the case with both candidate pruning and speed-up technique with $\epsilon = 0.1$ (red line). Also, the case with the candidate pruning without speed-up technique poorly performed due to computational overhead and could not even obtain near-optimal solution in this case (green line).

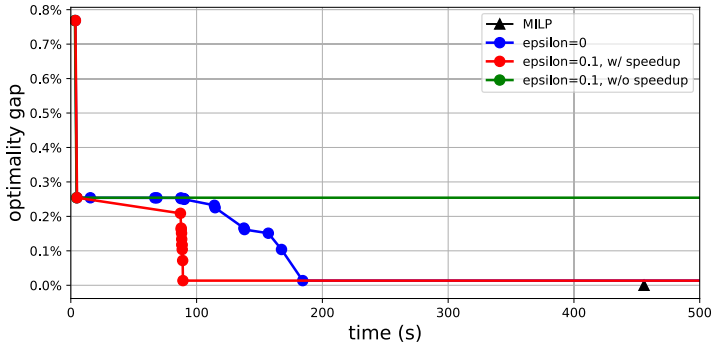


Figure 15: Solution histories of the MILP-based method and the proposed method with different settings for an instance of aircraft routing problem with the second configuration.

The advantage of using the speed-up technique also can be found for $\epsilon = 0.2$ and 0.3 cases as well in Table 5. More specifically, the number of problems solved was improved from 22 to 25 for $\epsilon = 0.2$ case, and from 22 to 24 for $\epsilon = 0.3$ case. However, the overall numbers of problems solved, with or without the speed-up technique, were smaller than the $\epsilon = 0.1$ case, which again demonstrates the trade-off in the candidate pruning.

Finally, Table 6 shows the results for the third configuration (Fig. 11c), in which case none of the baseline methods solved any problem instance given 1800 seconds time limit. It is worth noting that the results for the proposed method have no statistically meaningful difference with or without pruning or speed-up technique. In fact, the state space and the size of policy for the third configuration are too huge to make improvement in the second stage for the proposed method. Therefore, all the results are based on the first stage of the algorithm and have no difference between different cases. However, it is interesting to note that the proposed algorithm still achieved $< 1\%$ and $< 0.1\%$ optimality gap for 27 and 13 out of 30 problem instances, respectively, solely based on the first stage of the algorithm with much less time than the time limit.

5.4. Summary

In this section, we evaluated the performance of the proposed method on

Baseline	# solved / time (s)	states
MILP	30 (12.7 ± 9.9)	6305.2
i-dual-LP	30 (34.3 ± 133.4)	1666.1
i-dual-MILP	30 (82.8 ± 241.7)	1728.8

	Proposed method									
	< 10% optimality gap			< 1% optimality gap			< 0.1% optimality gap			
ϵ	# solved / time (s)	states	speed-up	# solved / time (s)	states	speed-up	# solved / time (s)	states	speed-up
0	30 (0.2 ± 0.4)	1578.3	127.3 [15.1, 441.2]	30 (0.3 ± 0.7)	1578.6	126.6 [9.7, 441.2]	28 (0.4 ± 0.6)	1443.9	101.9 [0, 441.2]
0.1	30 (0.3 ± 0.6)	1578.3	118.5 [11.7, 441.2]	30 (0.4 ± 1.0)	1578.6	117.9 [7.1, 441.2]	30 (8.9 ± 41.5)	1652.2	101.1 [0.2, 441.2]
0.1 (⋆)	30 (0.2 ± 0.4)	1578.3	129.7 [17.0, 441.2]	30 (0.3 ± 0.7)	1578.6	129.0 [10.6, 441.2]	30 (1.8 ± 5.9)	1652.2	102.1 [1.1, 441.2]
0.2	30 (0.3 ± 0.5)	1578.3	116.4 [8.1, 441.2]	30 (0.4 ± 0.8)	1578.6	115.7 [8.1, 441.2]	29 (11.5 ± 46.7)	1604.9	100.8 [0, 441.2]
0.2 (⋆)	30 (0.3 ± 0.6)	1578.3	116.1 [8.2, 441.2]	30 (0.4 ± 0.9)	1578.6	115.5 [8.2, 441.2]	29 (2.2 ± 7.8)	1604.9	101.5 [0, 441.2]
0.3	30 (0.3 ± 0.6)	1578.3	115.5 [9.2, 441.2]	30 (0.4 ± 0.9)	1578.6	114.9 [8.2, 441.2]	29 (10.8 ± 44.0)	1604.9	100.9 [0, 441.2]
0.3 (⋆)	30 (0.3 ± 0.6)	1578.3	115.8 [8.6, 441.2]	30 (0.4 ± 0.9)	1578.6	115.1 [8.1, 441.2]	29 (2.3 ± 8.1)	1604.9	101.3 [0, 441.2]

Table 4: Evaluation results of the aircraft routing problems with the first configuration.

Baseline	# solved / time (s)	states
MILP	18 (256.9 ± 274.9)	10565.3
i-dual-LP	12 (451.6 ± 479.1)	8071.3
i-dual-MILP	3 (577.4 ± 464.1)	4974.0

	Proposed method									
	< 10% optimality gap			< 1% optimality gap			< 0.1% optimality gap			
	# solved / time (s)	states	speed-up	# solved / time (s)	states	speed-up	# solved / time (s)	states	speed-up
0	30 (5.1 ± 7.0)	11140.3	319.5 [1.7, 1538.3]	30 (5.1 ± 7.0)	11140.3	319.5 [1.7, 1538.3]	25 (26.8 ± 44.7)	11607.2	271.6 [0, 1538.3]
0.1	30 (4.5 ± 7.0)	11140.3	323.7 [1.7, 1538.2]	30 (4.5 ± 7.0)	11140.3	323.7 [1.7, 1538.2]	24 (58.6 ± 112.2)	11682.8	274.0 [0, 1538.2]
0.1 (⋆)	30 (4.5 ± 6.9)	11140.3	322.8 [1.7, 1538.8]	30 (4.5 ± 6.9)	11140.3	322.8 [1.7, 1538.8]	27 (21.4 ± 27.9)	11504.6	276.4 [0, 1538.8]
0.2	30 (4.6 ± 7.0)	11140.3	316.1 [1.7, 1538.6]	30 (4.6 ± 7.0)	11140.3	316.1 [1.7, 1538.6]	22 (35.2 ± 74.4)	11421.6	271.6 [0, 1538.6]
0.2 (⋆)	30 (4.6 ± 7.0)	11140.3	318.3 [1.7, 1538.5]	30 (4.6 ± 7.0)	11140.3	318.3 [1.7, 1538.5]	25 (20.5 ± 30.5)	11508.8	273.5 [0, 1538.5]
0.3	30 (4.9 ± 7.1)	11140.3	318.0 [1.7, 1538.3]	30 (4.9 ± 7.1)	11140.3	318.0 [1.7, 1538.3]	22 (36.3 ± 77.4)	11421.6	273.2 [0, 1538.3]
0.3 (⋆)	30 (4.6 ± 7.0)	11140.3	318.2 [1.7, 1538.4]	30 (4.6 ± 7.0)	11140.3	318.2 [1.7, 1538.4]	24 (19.2 ± 31.8)	11400.3	272.3 [0, 1538.4]

Table 5: Evaluation results of the aircraft routing problems with the second configuration.

Baseline	# solved / time (s)	states
MILP	0 (-)	-
i-dual-LP	0 (-)	-
i-dual-MILP	0 (-)	-

Proposed method									
	$< 10\%$ optimality gap			$< 1\%$ optimality gap			$< 0.1\%$ optimality gap		
	# solved / time (s)	states	speed-up	# solved / time (s)	states	speed-up	# solved / time (s)	states	speed-up
0	30 (145.1 ± 138.3)	254934.2	69.3 [3.5, 526.7]	27 (131.6 ± 133.9)	232942.7	68.5 [1.0, 526.7]	13 (266.3 ± 343.3)	255011.2	51.5 [1.0, 526.7]
0.1	30 (145.0 ± 138.2)	254934.2	69.4 [3.5, 526.1]	27 (131.6 ± 133.9)	232942.7	68.5 [1.0, 526.1]	13 (266.5 ± 343.6)	255011.2	51.5 [1.0, 526.1]
0.1 (⋆)	30 (145.1 ± 138.1)	254934.2	69.4 [3.5, 525.9]	27 (131.5 ± 133.6)	232942.7	68.5 [1.0, 525.9]	13 (266.6 ± 343.8)	255011.2	51.5 [1.0, 525.9]
0.2	30 (145.5 ± 139.5)	254934.2	69.3 [3.4, 522.0]	27 (131.7 ± 134.9)	232942.7	68.5 [1.0, 522.0]	13 (266.7 ± 343.2)	255011.2	51.3 [1.0, 522.0]
0.2 (⋆)	30 (145.0 ± 138.2)	254934.2	69.1 [3.5, 525.7]	27 (131.8 ± 134.2)	232942.7	68.3 [1.0, 525.7]	13 (266.5 ± 343.5)	255011.2	51.3 [1.0, 525.7]
0.3	30 (144.9 ± 137.5)	254934.2	69.0 [3.5, 521.7]	27 (131.4 ± 133.2)	232942.7	68.2 [1.0, 521.7]	13 (266.6 ± 342.9)	255011.2	51.3 [1.0, 521.7]
0.3 (⋆)	30 (145.1 ± 138.5)	254934.2	69.8 [3.5, 530.6]	27 (131.3 ± 133.8)	232942.7	68.9 [1.0, 530.6]	13 (268.4 ± 347.3)	255011.2	51.9 [1.0, 530.6]

Table 6: Evaluation results of the aircraft routing problems with the third configuration.

three domains in comparison with three baseline methods. The evaluation results well demonstrated the anytime property of the proposed method. In other words, the proposed method could obtain solutions with $< 10\%$ and $< 1\%$ optimality gaps much faster than the baseline methods for most of the test instances. In addition, for the aircraft routing problem with the third configuration where none of the baseline methods could solve the problem, the proposed algorithm still could find solutions with optimality gap less than 10%, 1% and 0.1% for 30, 27 and 13 cases out of 30, respectively.

However, the results also showed that the method suffers from slow convergence rate and cannot achieve near-optimality with minimal ($< 0.1\%$) optimality gap in several cases given limited planning time. In addition to the vanilla version of the algorithm, we also proposed candidate pruning to facilitate the search at the cost of optimality in Section 4. Throughout the experiments, we could observe that candidate pruning generally improves the convergence rate of the proposed method and helps us to find near-optimal solutions with $< 0.1\%$ optimality gap faster in comparison with the cases without candidate pruning. However, we could also observe the trade-off between the convergence rate and optimality while using candidate pruning. For example, candidate pruning with $\epsilon = 0.3$ was outperformed by the case with $\epsilon = 0.1$ for the aircraft routing problems with the first and the second configurations due to aggressive pruning. In general, the effect of the parameter ϵ for candidate pruning depends on the domain and problem instance, and it is difficult to find a universally outperforming parameter. Although it is out of the scope of this paper, finding a systematic way to balance the trade-off between efficiency and optimality in candidate pruning with parameter ϵ should be a promising direction for future research.

Although candidate pruning potentially facilitates the search by skipping generating a candidate in the second stage of the proposed algorithm, the computational overhead might dominate its advantage and make the performance of the method even worse. To overcome such computational overhead, we introduced the speed-up techniques in Section 4.3, that reduce the computational

effort of matrix inversion which is the main source of computational overhead in candidate pruning. The advantage of using the speed-up techniques was well demonstrated in both elevators and aircraft routing domains. In the elevators domain, we could observe that the speed-up technique reduced the computation time consistently for almost every case compared to the cases without using speed-up technique. In the aircraft routing domain, the advantage was more dramatic and the speed-up technique even increased the number of problems solved with $< 0.1\%$ optimality gap compared to the cases without using the speed-up technique. Note that the difference in the magnitude of the advantage of using the speed-up techniques is mainly due to the difference in the sizes of policies between the domains, since the sizes of matrices that we need to take inverse during candidate pruning depend on the sizes of policies. Therefore, the experimental results suggest that the use of speed-up technique is more important in the domains where the sizes of policies are usually large.

Finally, we also studied the effect of approximation in Lagrangian dual with parameter M that controls the number of iterations for solving Lagrangian dual when there are multiple constraints. The experimental results for the elevators domain showed that, in general, solving Lagrangian dual only approximately with limited M performs better than solving it optimally. These results might come from the fact that the Lagrangian dual algorithm could iterate too much with marginal improvements, in which case the closing gap procedure can improve the solution quality faster especially with candidate pruning.

6. Conclusion

This work introduces an anytime algorithm that is able to find an optimal deterministic policy for constrained stochastic shortest path problems (C-SSPs). We argue that the anytime property is one of the most crucial requirements for a solution method for C-SSPs, especially with safety-critical applications. Our proposed method focuses on finding a feasible but decent solution quickly, then updating the solution as much as we can, until time is up. The algorithm is proved able to obtain a true optimal solution eventually, if we have enough planning time. We achieve this by dividing the solution method in two stages, where the first stage finds a Lagrangian dual optimal solution, and the second stage incrementally updates the solution to close a duality gap if it exists. Throughout the process, we leverage the heuristic forward search which improves efficiency in many ways. First, heuristics guide us to explore only the part of the search space where the admissibility of a heuristic guarantees optimality. Second, throughout the process of our proposed method, we solve a series of SSPs with only slight modifications. Heuristic forward search enables us to reuse previously generated AND/OR graphs and reduce redundant computations. In many time- and safety-critical real-world applications, it is often desirable to obtain a suboptimal solution with limited computation time. In those applications, increasing convergence rate of the solution quality is worth pursuing at the cost of optimality. Motivated by this, we propose an approximation scheme which prunes candidate generations in the second stage of the algorithm, which is guaranteed to obtain an ϵ-optimal solution with user specified approximation parameter ϵ. The evaluation results presented in this paper show that the proposed approach could obtain suboptimal solutions with $< 10\%$ and $< 1\%$ optimality gaps much faster than the baseline methods for most of the test problem instances. However, the results also show that the algorithm suffers from slow convergence rate and cannot achieve near-optimality with minimal ($< 0.1\%$) optimality gap in several cases especially with multiple constraints. In those cases, the approximation scheme and speed-up technique

could improve the convergence rate so that it can achieve near-optimality for some cases, while there are still some cases that could not achieve it. Another interesting observation from the experiments is that the proposed approach could produce solutions with at most 10% optimality gap for extremely large problems that could not be solved by baseline methods given limited planning time. The observations made from the experiments identify when the users take advantage of using the proposed method. As argued above, the proposed approach can be beneficial especially with safety-critical applications when a user needs to have a feasible policy in a limited planning time, as it can produce suboptimal solutions much faster than the baseline methods. However, it might be beneficial to use the MILP-based methods when optimality is the primary concern, since the proposed method has slow tail convergence rate. In addition, the proposed method can be thought of as an approximation method when the problem is too huge to be solved by the baseline methods. There are a number of avenues for future work. In the experimental results, the proposed approach showed various ranges of performance based on different parameter selections. This leads us to further study ways to systematically choose such parameters as discussed in Section 5.4. For example, one of the promising research directions is deciding approximation parameter ϵ in candidate pruning by considering the given planning time. More specifically, if the given planning time is enough to solve the problem without approximation, then there is no reason to use candidate pruning. On the other hand, if the given planning time is too limited compared to the size of the problem, we might want to prune candidates more aggressively with bigger ϵ. Second, the proposed approach is actually a class of algorithms where we can plug-in different algorithms for each step of the method. Therefore, another interesting future work is highly optimizing the algorithm by plugging in state-of-the-art methods for those steps. For example, we can substitute a coordinate search used in the first stage of the algorithm with more recent algorithms such as a quasi-Newton approach [41]. In addition, various approximation schemes are worth exploring in order to improve the convergence rate of the algorithm, which is particularly important with applications where

planning time is highly limited. Finally, although this paper proposes a general framework without consideration of choosing heuristics, the quality of heuristics affects the performance of the proposed method a lot. Therefore, it would be valuable to explore various domain-independent heuristics and integrate them as in [11, 12]. There are also interesting ongoing research works that generalize the problem model while the proposed method still can be beneficial to use. One of the ideas that we are currently pursuing is generalizing the proposed approach for hierarchical planning where the higher level of the planning problem is based on a model-based approach, such as a temporal plan network (TPN) [42], which has symbolic constraints and decisions. Another avenue that we are pursuing is including path constraints in stochastic sequential decision making, such as temporal logic constraints [43], to generalize the expressibility of the safety and goal achievement in the problem.

Declaration of competing interest

We wish to confirm that there are no known conflicts of interest associated with this publication and there has been no significant financial support for this work that could have influenced its outcome.

Acknowledgements

This work was supported by the Boeing Corporation under grant No. 6943358.

Appendix A. Correctness of the k-best stochastic shortest path algorithm

In this section, we prove that the k-best stochastic shortest path algorithm in Algorithm 9 and 10 is correct. For this, it suffices to show that our algorithm is a special case of Lawler's algorithm [30], which is a general k-best solution method for discrete optimization. First, we briefly review Lawler's algorithm. Given an optimization problem with a set of binary decision variables x_i for $i = 1, \ldots, n$, the Lawler's algorithm finds k-best solutions with the following procedure [30]:

0) (Initialization) Set the solution number k and candidate set Γ as 1 and $\emptyset$, respectively. Compute an optimal solution and put this solution in the candidate set Γ. Note that none of the variables has fixed value in this step.

1) (Compute k-th solution) Remove a solution with the lowest cost from Γ. Let denote this solution as $x^k = (x_1^k, x_2^k, \ldots, x_n^k)$ and indicate x^k as the k-th best solution.

2) (Test termination condition) If a termination condition is satisfied, then stop.

3) (Generate candidates) Suppose, without loss of generality, the values of $x_1, x_2, \ldots, x_s$ were fixed when we computed x^k. With those variables fixed as before, generate $n - s$ new problems by fixing the remaining variables as follows:

$$
\begin{aligned}
&(1) \quad x_{s+1} = 1 - x_{s+1}^k, \\
&(2) \quad x_{s+1} = x_{s+1}^k, x_{s+2} = 1 - x_{s+2}^k, \\
&(3) \quad x_{s+1} = x_{s+1}^k, x_{s+2} = x_{s+2}^k, x_{s+3} = 1 - x_{s+3}^k, \\
&\quad \vdots \\
&(n-s) \quad x_{s+1} = x_{s+1}^k, x_{s+2} = x_{s+2}^k, \ldots, x_{n-1} = x_{n-1}^k, x_n = 1 - x_n^k.
\end{aligned}
$$

Compute optimal solutions for each of newly generated $n - s$ problems.

Put each optimal solution with a record of the fixed variables in Γ. Increase k by 1. Continue to step 1).

Now, to show that our k-best stochastic shortest path algorithm for an SSP is a special case of Lawler's algorithm, let define a set of binary decision variables $x(s,a)$ for every $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$, where an action a is selected from a state s if and only if $x(s,a) = 1$. Let π^k be the k-th best policy, $\mathcal{S}_{\pi^k}$ be a set of states reachable from the initial state $\bar{s}$ by π^k, and $\pi^k(s)$ be the action selected by π^k from s. Also, let $\mathcal{A}(s) = \{a^1(s), a^2(s), \ldots, a^{|\mathcal{A}(s)|}(s)\}$.

Theorem Appendix A.1. *Suppose we apply Lawler's algorithm to find k-best stochastic shortest paths for an SSP. Suppose, without loss of generality, the states in $\mathcal{S}_{\pi^k}$ for the k-th best policy π^k are indexed with BFS order, i.e., $s_0, s_1, s_2, \ldots,$ with $s_0 = \bar{s}$, and the other states are indexed arbitrarily. Now, suppose for step 3) of k-th iteration of Lawler's algorithm, we create problems for $x(s_i, \pi^k(s_i))$'s first for $i = 0, \ldots, |\mathcal{S}_{\pi^k}|-1$ in ascending order, then create the rest of the problems in arbitrary order. Then, in step 3) of k-th iteration, the fixed variables can be partitioned into $x_F^k = \{x(s_i, \pi^k(s_i)) \mid i = 0, \ldots, |x_F^k|-1, |x_F^k| < |\mathcal{S}_{\pi^k}|\}$ and x_B^k, where the variables in the former set have value of 1 and the variables in the latter set have value of 0. Moreover, the set of created problems are in the following form:*

$$
\begin{aligned}
(1)\quad & x(s_{|x_F^k|}, \pi^k(s_{|x_F^k|})) = 0, \\
(2)\quad & x(s_{|x_F^k|}, \pi^k(s_{|x_F^k|})) = 1, x(s_{|x_F^k|+1}, \pi^k(s_{|x_F^k|+1})) = 0, \\
(3)\quad & x(s_{|x_F^k|}, \pi^k(s_{|x_F^k|})) = 1, x(s_{|x_F^k|+1}, \pi^k(s_{|x_F^k|+1})) = 1, \\
& x(s_{|x_F^k|+2}, \pi^k(s_{|x_F^k|+2})) = 0, \\
& \vdots \\
(|\mathcal{S}_{\pi^k}| - |x_F^k|)\quad & x(s_{|x_F^k|}, \pi^k(s_{|x_F^k|})) = 1, x(s_{|x_F^k|+1}, \pi^k(s_{|x_F^k|+1})) = 1, \ldots, \\
& x(s_{|\mathcal{S}_{\pi^k}|-2}, \pi^k(s_{|\mathcal{S}_{\pi^k}|-2})) = 1, x(s_{|\mathcal{S}_{\pi^k}|-1}, \pi^k(s_{|\mathcal{S}_{\pi^k}|-1})) = 0.
\end{aligned}
$$

Proof. The proof can be done by induction. Suppose π^1 has been computed by any stochastic shortest path algorithm.

(**Base case**) In the first iteration, it is obvious that $x_F^1 = x_B^1 = \emptyset$. Then we create problems as follows based on Lawler's algorithm with given order in the theorem:

$$
\begin{aligned}
&(1) \quad x(s_0, \pi^1(s_0)) = 0,\\
&(2) \quad x(s_0, \pi^1(s_0)) = 1, x(s_1, \pi^1(s_1)) = 0,\\
&(3) \quad x(s_0, \pi^1(s_0)) = 1, x(s_1, \pi^1(s_1)) = 1, x(s_2, \pi^1(s_2)) = 0,\\
&\quad \vdots\\
&(|\mathcal{S}_{\pi^1}|) \quad x(s_0, \pi^1(s_0)) = 1, x(s_1, \pi^1(s_1)) = 1, \ldots,\\
&\qquad\qquad x(s_{|\mathcal{S}_{\pi^1}|-2}, \pi^1(s_{|\mathcal{S}_{\pi^1}|-2})) = 1, x(s_{|\mathcal{S}_{\pi^1}|-1}, \pi^1(s_{|\mathcal{S}_{\pi^1}|-1})) = 0,\\
&(|\mathcal{S}_{\pi^1}|+1) \quad x(s_i, \pi^1(s_i)) = 1 \quad \forall i = 0, \ldots, |\mathcal{S}_{\pi^1}| - 1,\\
&\qquad\qquad x(s_{|\mathcal{S}_{\pi^1}|}, a^1(s_{|\mathcal{S}_{\pi^1}|})) = 0,\\
&(|\mathcal{S}_{\pi^1}|+2) \quad x(s_i, \pi^1(s_i)) = 1 \quad \forall i = 0, \ldots, |\mathcal{S}_{\pi^1}| - 1,\\
&\qquad\qquad x(s_{|\mathcal{S}_{\pi^1}|}, a^1(s_{|\mathcal{S}_{\pi^1}|})) = 1, x(s_{|\mathcal{S}_{\pi^1}|}, a^2(s_{|\mathcal{S}_{\pi^1}|})) = 0,\\
&\quad \vdots
\end{aligned}
$$

However, the problems (p) with $p > |\mathcal{S}_\pi^1|$ include constraints

$$x(s_i, \pi^1(s_i)) = 1 \quad \forall i = 0, \ldots, |\mathcal{S}_{\pi^1}| - 1,$$

which form the closed policy π^1. Therefore, any problem (p) with $p > |\mathcal{S}_\pi^1|$ is infeasible and can be ignored because it has at least one constraint that either fixes an action from a non-reachable state from π^1 or fixes an action from a reachable state from π^1 different that the already fixed action. This concludes that the created problems are of the form as claimed in the theorem.

Now, suppose π^2 is the second best policy, which was obtained by solving (j)-th problem. Then, in the second iteration, the fixed variables are

$$x(s_0, \pi^1(s_0)) = x(s_0, \pi^2(s_0)) = 1, \ldots, x(s_{j-2}, \pi^1(s_{j-2})) = x(s_{j-2}, \pi^2(s_{j-2})) = 1,$$

and

$$x(s_{j-1}, \pi^1(s_{j-1})) = 0,$$

where the former and the latter can be identified as x_F^2 and x_B^2, respectively. This proves the base case.

(**induction step**) Suppose the claim holds for the first k iterations. Let π^k be the k-th best policy, and the fixed variables are partitioned as $x_F^k = \left\{x(s_i, \pi^k(s_i)) \mid i = 0, \ldots, |x_F^k| - 1, |x_F^k| < |\mathcal{S}_{\pi^k}|\right\}$ and x_B^k, where the variables in the former set have value of 1 and the variables in the latter set have value of 0. Then we create problems as follows based on Lawler's algorithm with given order in the theorem:

$$\begin{aligned}
&(1) \quad && x(s_{|x_F^k|}, \pi^k(s_{|x_F^k|})) = 0, \\
&(2) \quad && x(s_{|x_F^k|}, \pi^k(s_{|x_F^k|})) = 1, x(s_{|x_F^k|+1}, \pi^k(s_{|x_F^k|+1})) = 0, \\
&(3) \quad && x(s_{|x_F^k|}, \pi^k(s_{|x_F^k|})) = 1, x(s_{|x_F^k|+1}, \pi^k(s_{|x_F^k|+1})) = 1, \\
& && x(s_{|x_F^k|+2}, \pi^k(s_{|x_F^k|+2})) = 0, \\
& \vdots && \\
&(|\mathcal{S}_{\pi^k}| - |x_F^k|) \quad && x(s_{|x_F^k|}, \pi^k(s_{|x_F^k|})) = 1, x(s_{|x_F^k|+1}, \pi^k(s_{|x_F^k|+1})) = 1, \ldots, \\
& && x(s_{|\mathcal{S}_{\pi^k}|-2}, \pi^k(s_{|\mathcal{S}_{\pi^k}|-2})) = 1, x(s_{|\mathcal{S}_{\pi^k}|-1}, \pi^k(s_{|\mathcal{S}_{\pi^k}|-1})) = 0, \\
&(|\mathcal{S}_{\pi^k}| - |x_F^k| + 1) \quad && x(s_i, \pi^k(s_i)) = 1 \quad \forall i = 0, \ldots, |\mathcal{S}_{\pi^k}| - 1, \\
& && x(s_{|\mathcal{S}_{\pi^k}|}, a^1(s_{|\mathcal{S}_{\pi^k}|})) = 0, \\
&(|\mathcal{S}_{\pi^k}| - |x_F^k| + 2) \quad && x(s_i, \pi^k(s_i)) = 1 \quad \forall i = 0, \ldots, |\mathcal{S}_{\pi^k}| - 1, \\
& && x(s_{|\mathcal{S}_{\pi^k}|}, a^1(s_{|\mathcal{S}_{\pi^k}|})) = 1, x(s_{|\mathcal{S}_{\pi^k}|}, a^2(s_{|\mathcal{S}_{\pi^k}|})) = 0, \\
& \vdots &&
\end{aligned}$$

where the problems (p) with $p > |\mathcal{S}_{\pi^k}| - |x_F^k|$ can be ignored based on the same reason as the base case. Therefore, the created problems are of the form as claimed in the theorem.

Finally, suppose, without loss of generality, we selected π^{k+1} as the $(k+1)$-th best policy from (j)-th problem created in the l-th iteration, where $l \leq k$, which has the following constraints:

$$x(s_{|x_F^l|}, \pi^l(s_{|x_F^l|})) = 1, \ldots, x(s_{j-2}, \pi^l(s_{j-2})) = 1, x(s_{j-1}, \pi^l(s_{j-1})) = 0.$$

Therefore, x_F^{k+1} and x_B^{k+1} becomes

$$\begin{aligned} x_F^{k+1} &= x_F^l \cup \{x(s_{|x_F^l|}, \pi^l(s_{|x_F^l|})), \ldots, x(s_{j-2}, \pi^l(s_{j-2}))\}, \\ &= \{x(s_0, \pi^{k+1}(s_0)), x(s_1, \pi^{k+1}(s_1)), \ldots, x(s_{j-2}, \pi^{k+1}(s_{j-2}))\}, \end{aligned}$$

and

$$x_B^{k+1} = x_B^l \cup \{x(s_{j-1}, \pi^l(s_{j-1}))\},$$

which proves the theorem. □

References

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

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

[3] D. Dolgov, E. Durfee, Stationary deterministic policies for constrained MDPs with multiple rewards, costs, and discount factors, in: International Joint Conference on Artificial Intelligence, Vol. 19, LAWRENCE ERLBAUM ASSOCIATES LTD, 2005, p. 1326.

[4] F. Geißer, G. Povéda, F. Trevizan, M. Bondouy, F. Teichteil-Königsbuch, S. Thiébaux, Optimal and heuristic approaches for constrained flight planning under weather uncertainty, in: Proceedings of the International Conference on Automated Planning and Scheduling, Vol. 30, 2020, pp. 384–393.

[5] S. Hong, S. U. Lee, X. Huang, M. Khonji, R. Alyassi, B. C. Williams, An anytime algorithm for chance constrained stochastic shortest path problems and its application to aircraft routing, in: 2021 IEEE International Conference on Robotics and Automation (ICRA), IEEE, 2021, pp. 475–481.

[6] P. Paruchuri, M. Tambe, F. Ordonez, S. Kraus, Towards a formalization of teamwork with resource constraints, in: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems-Volume 2, IEEE Computer Society, 2004, pp. 596–603.

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

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

[9] F. Trevizan, S. Thiébaux, P. Santana, B. Williams, I-dual: solving constrained SSPs via heuristic search in the dual space, in: Proceedings of the 26th International Joint Conference on Artificial Intelligence, AAAI Press, 2017, pp. 4954–4958.

[10] L. C. Kallenberg, Linear programming and finite Markovian control problems, Vol. 148 of Mathematical Centre Tracts, Mathematisch Centrum, 1983.

[11] F. Trevizan, S. Thiébaux, P. Haslum, Occupation measure heuristics for probabilistic planning, in: Twenty-Seventh International Conference on Automated Planning and Scheduling, 2017.

[12] P. Baumgartner, S. Thiébaux, F. Trevizan, Heuristic search planning with multi-objective probabilistic LTL constraints, in: Sixteenth International Conference on Principles of Knowledge Representation and Reasoning, 2018.

[13] C. Bäckström, Equivalence and tractability results for SAS+ planning., in: KR, 1992, pp. 126–137.

[14] P. Santana, S. Thiébaux, B. Williams, RAO*: an algorithm for chance constrained POMDPs, in: Proc. AAAI Conference on Artificial Intelligence, 2016.

[15] N. J. Nilsson, Principles of artificial intelligence, Morgan Kaufmann, 2014.

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

[17] S. Boyd, L. Vandenberghe, Convex optimization, Cambridge university press, 2004.

[18] S. Ovchinnikov, Max-min representation of piecewise linear functions, arXiv preprint math/0009026.

[19] S. Boyd, L. Xiao, A. Mutapcic, Subgradient methods, lecture notes of EE392o, Stanford University, Autumn Quarter 2004 (2003) 2004–2005.

[20] P. Neame, N. Boland, D. Ralph, An outer approximate subdifferential method for piecewise affine optimization, Mathematical programming 87 (1) (2000) 57–86.

[21] W. W. Hager, S. Park, The gradient projection method with exact line search, Journal of Global Optimization 30 (1) (2004) 103–118.

[22] W. M. Carlyle, J. O. Royset, R. Kevin Wood, Lagrangian relaxation and enumeration for solving constrained shortest-path problems, Networks: an international journal 52 (4) (2008) 256–270.

[23] J. O. Royset, W. M. Carlyle, R. K. Wood, Routing military aircraft with a constrained shortest-path algorithm, Military Operations Research (2009) 31–52.

[24] D. D. Dewolfe, J. G. Stevens, R. K. Wood, Setting military reenlistment bonuses, Naval Research Logistics (NRL) 40 (2) (1993) 143–160.

[25] H.-J. M. Shi, S. Tu, Y. Xu, W. Yin, A primer on coordinate descent algorithms, arXiv preprint arXiv:1610.00040.

[26] B. L. Fox, D. M. Landi, Searching for the multiplier in one-constraint optimization problems, Operations Research 18 (2) (1970) 253–262.

[27] C. Beltran, F.-J. Heredia, An effective line search for the subgradient method, Journal of optimization theory and applications 125 (1) (2005) 1–18.

[28] G. Y. Handler, I. Zang, A dual algorithm for the constrained shortest path problem, Networks 10 (4) (1980) 293–309.

[29] J. Y. Yen, Finding the k shortest loopless paths in a network, management Science 17 (11) (1971) 712–716.

[30] E. L. Lawler, A procedure for computing the k best solutions to discrete optimization problems and its application to the shortest path problem, Management science 18 (7) (1972) 401–405.

[31] J. Sherman, W. J. Morrison, Adjustment of an inverse matrix corresponding to a change in one element of a given matrix, The Annals of Mathematical Statistics 21 (1) (1950) 124–127.

[32] M. S. Bartlett, An inverse matrix adjustment arising in discriminant analysis, The Annals of Mathematical Statistics 22 (1) (1951) 107–111.

[33] A. G. Barto, S. J. Bradtke, S. P. Singh, Learning to act using real-time dynamic programming, Artificial intelligence 72 (1-2) (1995) 81–138.

[34] B. Bonet, H. Geffner, Labeled RTDP: Improving the convergence of real-time dynamic programming., in: ICAPS, Vol. 3, 2003, pp. 12–21.

[35] A. Charnes, W. W. Cooper, Chance-constrained programming, Management science 6 (1) (1959) 73–79.

[36] L. Blackmore, M. Ono, B. C. Williams, Chance-constrained optimal path planning with obstacles, IEEE Transactions on Robotics 27 (6) (2011) 1080–1094.

[37] S. U. Lee, R. Gonzalez, K. Iagnemma, Robust sampling-based motion planning for autonomous tracked vehicles in deformable high slip terrain, in: 2016 IEEE International Conference on Robotics and Automation (ICRA), IEEE, 2016, pp. 2569–2574.

[38] M. Ono, B. C. Williams, Iterative risk allocation: A new approach to robust model predictive control with a joint chance constraint, in: Decision and Control, 2008. CDC 2008. 47th IEEE Conference on, Citeseer, 2008, pp. 3427–3432.

[39] Mausam, A. Kolobov, Planning with Markov decision processes: An AI perspective, Synthesis Lectures on Artificial Intelligence and Machine Learning 6 (1) (2012) 1–210.

[40] E. Balaban, I. Roychoudhury, L. Spirkovska, S. Sankararaman, C. S. Kulkarni, T. Arnon, Dynamic routing of aircraft in the presence of adverse weather using a POMDP framework, in: 17th aiaa aviation technology, integration, and operations conference, 2017, p. 3429.

[41] J. Yu, S. Vishwanathan, S. Günter, N. N. Schraudolph, A quasi-newton approach to nonsmooth convex optimization problems in machine learning, The Journal of Machine Learning Research 11 (2010) 1145–1200.

[42] P. Kim, B. C. Williams, M. Abramson, Executing reactive, model-based programs through graph-based temporal planning, in: IJCAI, Citeseer, 2001, pp. 487–493.

[43] A. Pnueli, The temporal logic of programs, in: 18th Annual Symposium on Foundations of Computer Science (SFCS 1977), IEEE, 1977, pp. 46–57.

Declaration of interests

☒ The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

☐The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: