Solving Constrained Stochastic Shortest Path Problems with Scalarisation

Johannes Schmalz[a,b,*] **and Felipe Trevizan**[b]

[a]Saarland University, Germany
[b]Australian National University, Australia
ORCID (Johannes Schmalz): https://orcid.org/0000-0003-3389-2490, ORCID (Felipe Trevizan): https://orcid.org/0000-0001-5095-7132

Abstract.

Constrained Stochastic Shortest Path Problems (CSSPs) model problems with probabilistic effects, where a primary cost is minimised subject to constraints over secondary costs, e.g., minimise time subject to monetary budget. Current heuristic search algorithms for CSSPs solve a sequence of increasingly larger CSSPs as linear programs until an optimal solution for the original CSSP is found. In this paper, we introduce a novel algorithm CARL, which solves a series of unconstrained Stochastic Shortest Path Problems (SSPs) with efficient heuristic search algorithms. These SSP subproblems are constructed with scalarisations that project the CSSP's vector of primary and secondary costs onto a scalar cost. CARL finds a maximising scalarisation using an optimisation algorithm similar to the subgradient method which, together with the solution to its associated SSP, yields a set of policies that are combined into an optimal policy for the CSSP. Our experiments show that CARL solves 50% more problems than the state-of-the-art on existing benchmarks.

1 Introduction

Stochastic Shortest Path Problems (SSPs) model problems where the effects of actions can be probabilistic and Constrained SSPs (CSSPs) extend the model by allowing constraints over secondary costs, enforcing that the incurred secondary costs do not exceed a specified threshold over expectation. This can represent interesting real-world problems, e.g., planning an aeroplane's route that minimises fuel usage while avoiding bad weather and satisfying certain timing requirements [14]. SSPs can be solved with the primal formulation, where we optimise over variables that represent the agent's cost-to-go (a.k.a. value functions); or with the dual formulation over occupation measures, which represent the expected number of times an action is applied in each state. Algorithms over the primal representation tend to be faster and indeed, the state-of-the-art algorithms for solving SSPs optimally use the primal representation, e.g., iLAO* [18], LRTDP [8], CG-iLAO* [26]. In contrast, the only heuristic search algorithms for CSSPs use the dual representation: i-dual [31] and i^2-dual [32]. In this paper, we present the first heuristic search algorithm that solves CSSPs optimally using the primal space, and show that it outperforms the existing algorithms.

Our algorithm CARL builds on existing work for finding deterministic policies for CSSPs (which are generally suboptimal) [22], with the key difference that CARL finds optimal (potentially stochastic) policies. CARL works by running heuristic search in the primal space using scalarisation, where a scalarisation $\boldsymbol{\lambda}$ projects the CSSP's cost vector onto a scalar cost function, thereby inducing an unconstrained SSP. CARL searches for an optimal scalarisation $\boldsymbol{\lambda}^*$ with optimsation techniques similar to the subgradient method [30] and solves SSPs induced by the encountered $\boldsymbol{\lambda}$ scalarisations as subproblems. Once CARL has found $\boldsymbol{\lambda}^*$, it computes the optimal costs-to-go V^* associated with that scalarisation, and builds an optimal policy for the CSSP by combining all of V^*'s greedy policies. CARL's subproblems are solved efficiently with heuristic search algorithms for SSPs and its outer problem over scalarisations is blind; this is reversed in the dual methods (i-dual and i^2-dual) where the construction of subproblems is guided with a heuristic, but the subproblems themselves are solved blindly by the LP solver. In our experiments, this tradeoff pays off and CARL can solve 1264 out of 1290 of our benchmark problems, whereas the state-of-the-art dual methods only solve 808. For some problems CARL is able to solve all instances while the state-of-the-art is not able to solve any, and over the problems where all algorithms solve all instances CARL offers an average speedup of $10\times$ w.r.t. the strongest baseline.

2 Background

Stochastic Shortest Path Problems (SSPs) [4] are defined by the tuple $\langle \mathsf{S}, s_I, \mathsf{G}, \mathsf{A}, P, C\rangle$ where S is a finite set of states; $s_I \in \mathsf{S}$ is the initial state; $\mathsf{G} \subset \mathsf{S}$ is a set of goal states that must be reached; A is a finite set of applicable actions, and we write $\mathsf{A}(s)$ to denote the applicable actions in state s; P is the probability transition function where $P(s'|s,a)$ is the probability of reaching s' after applying a to s; and $C : \mathsf{A} \to \mathbb{R}_{>0}$ is a cost function where $C(a)$ gives the cost of applying the action a.

Policies map states to actions and describe solutions to SSPs. Policies come in two flavours: **deterministic policies** $\pi : \mathsf{S} \to \mathsf{A}$ and **stochastic policies** $\pi : \mathsf{S} \to \text{distr}(\mathsf{A})$, which respectively map each state onto a single action, or onto a probability distribution over actions from which an action should be selected randomly; $\pi(s,a)$ is the probability that π applies a in s, and we may write $\pi(s) = a$ if $\pi(s,a) = 1$. The envelope of policy π from s, written $\mathsf{S}^{\pi,s}$, denotes the set of states reachable by following π from s. The policy π is closed w.r.t. s if π is defined for all $s' \in \mathsf{S}^{\pi,s}$, and π is proper w.r.t. s if following π from s guarantees that G is reached with proba-

* Corresponding Author. Email: schmalz@cs.uni-saarland.de.

bility 1. If π is not closed or not proper it is called open or improper, respectively. If s is omitted in these expressions, then we assume $s = s_I$. We extend the notion of an envelope, and write $\text{supp}(\pi)$ to denote the state-action pairs that might be encountered by π, i.e., $\text{supp}(\pi) = \{(s,a) : s \in \mathsf{S}^\pi, \pi(s,a) > 0\}$. We overload the C symbol and write $C(\pi)$ to denote the expected cost incurred by following the proper policy π from s_I. We make two standard assumptions for SSPs: a proper policy exists from each state (reachability), and any improper policy incurs infinite cost. Then, a policy π is optimal if it is closed and minimises $C(\pi)$. Under these assumptions, SSPs always have an optimal deterministic policy [4].

Primal algorithms for SSPs work over value functions $V : \mathsf{S} \to \mathbb{R}_{\geq 0}$, which represent the cost-to-go for each state, i.e., $V(s)$ indicates the expected cost that the agent must incur to reach the goal from s. For each V, its Q-values are $Q(s,a) = C(a) + \sum_{s' \in \mathsf{S}} P(s'|s,a)V(s')$. V^* is the optimal value function and denotes the cheapest possible cost-to-go for each state. V^* is the unique solution of the Bellman equations:

$$V(s) = \min_{a \in \mathsf{A}(s)} Q(s,a) \; \forall s \in \mathsf{S} \setminus \mathsf{G} \text{ and } V(s) = 0 \; \forall s \in \mathsf{G}.$$

For V, its greedy policy π_V is defined by $\pi_V(s) = \text{argmin}_{a \in \mathsf{A}(s)} Q(s,a)$, where the argmin operator breaks ties arbitrarily to yield a single policy. V^*'s greedy policy gives an optimal deterministic policy for the SSP. To compute V^*, we can start with some V and iteratively apply Bellman backups $V(s) \leftarrow \min_{a \in \mathsf{A}(s)} Q(s,a)$ over states s. Value Iteration (VI) [3] applies Bellman backups over the whole state space in each step, and converges to V^* in the limit as the number of steps increases.

Heuristic search algorithms use heuristic functions to focus on a promising subset of states and actions, and only apply backups there. For deterministic problems the canonical heuristic-search algorithm is A* [20], and the state-of-the-art algorithms iLAO* [18], LRTDP [8], and CG-iLAO* [26] generalise A* to the SSP setting. A heuristic function is a value function $H : \mathsf{S} \to \mathbb{R}_{\geq 0}$ that estimates V^*, so that the algorithm can avoid states with large $H(s)$ and prefer states with low $H(s)$. We require heuristics to be admissible, i.e., to be lower bounds on the optimal value function $H(s) \leq V^*(s) \; \forall s \in \mathsf{S}$. With admissible heuristics the heuristic-search algorithms we consider can guarantee optimality. For VI, we measure the change in successive value functions with Bellman residual $\text{RES}(s) = |V(s) - \min_{a \in \mathsf{A}(s)} Q(s,a)|$, and stop search once the changes in V are sufficiently small. In heuristic search, we stop when V is ϵ-consistent [7], i.e., when for some greedy policy π_V we have $\text{RES}(s) \leq \epsilon \; \forall s \in \mathsf{S}^{\pi_V}$. This condition guarantees that $V = V^*$ as $\epsilon \to 0$, and in practice, for sufficiently small ϵ, ϵ-consistent value functions induce an optimal policy.

CG-iLAO* [26] is a heuristic search algorithm over the primal space, which we explain because it provides crucial features to solve our subproblems efficiently. CG-iLAO* is presented in alg. 1. It works over value function V and a partial SSP $\widehat{\mathbb{S}}$, which contains a subset of the original SSP's states and actions. Non-goal states in $\widehat{\mathbb{S}}$ are called "expanded" if they have at least one applicable action in $\widehat{\mathbb{S}}$, and are called "fringes" if they have none. In each step, CG-iLAO* applies Bellman backups over the greedy policy $\widehat{\pi}_V$'s envelope $\mathcal{E}$, and expands $\mathcal{E}$'s fringes. Thus, CG-iLAO* simultaneously works towards making $\widehat{\pi}_V$ closed by expanding fringes, and towards making V ϵ-consistent with the Bellman backups. Since $\widehat{\mathbb{S}}$ need not contain all actions, CG-iLAO* looks for expanded states s with actions a such that $Q(s,a) < V(s)$, because this suggests a can improve $V(s)$ and should be added if it is missing. Γ is a set of all state-action pairs that potentially satisfy $Q(s,a) < V(s)$, i.e., it is a superset of improving actions. Γ is efficiently maintained by tracking changes to V and line 9 fixes any issues by setting $V(s) \leftarrow \min\{V(s), Q(s,a)\}$ for all $(s,a) \in \Gamma$ and adding a as required. Note that this may introduce more instances of $V(s) > Q(s,a)$, which are themselves recorded in Γ to be checked later.

Algorithm 1: CG-iLAO*

1 **Function** *CG-iLAO* (SSP $\mathbb{S}$, heuristic H, $\epsilon \in \mathbb{R}_{>0}$)*
2 $\quad \widehat{\mathbb{S}} \leftarrow$ partial SSP containing only s_I
3 $\quad V \leftarrow$ value function initialised by H
4 $\quad$ **repeat**
5 $\quad\quad \widehat{\pi}_V, \widehat{\pi}_{\text{old}} \leftarrow$ greedy policy for V, restricted to $\widehat{\mathbb{S}}$
6 $\quad\quad \mathcal{E} \leftarrow$ post-order DFS traversal of $\widehat{\pi}_V$ from s_I
7 $\quad\quad \widehat{\mathbb{S}}, \mathcal{E}, \widehat{\pi}_V \leftarrow$ expand fringes of $\mathcal{E}$
8 $\quad\quad V, \text{RES}, \widehat{\pi}_V, \Gamma \leftarrow$ apply Bellman backups over $\mathcal{E}$, recording increases and decreases to V in Γ
9 $\quad\quad V, \text{RES}, \Gamma, \widehat{\mathbb{S}} \leftarrow$ fix cases of $V(s) > Q(s,a)$ in Γ
10 $\quad$ **until** *fringes*$(\mathcal{E}) = \emptyset$ *and* $\widehat{\pi}_{old} = \widehat{\pi}_V$ *and* $\text{RES} \leq \epsilon$
11 $\quad$ **return** V

Constrained SSPs (CSSPs) [1, 31] are extensions to SSPs defined by $\mathbb{C} = \langle \mathsf{S}, s_I, \mathsf{G}, \mathsf{A}, P, \boldsymbol{C}, \boldsymbol{u} \rangle$ with two changes from SSPs: (1) the vector cost function $\boldsymbol{C} : \mathsf{A} \to \mathbb{R}_{>0} \times \mathbb{R}^n_{\geq 0}$, where $C_0 : \mathsf{A} \to \mathbb{R}_{>0}$ is the primary cost, and $C_i : \mathsf{A} \to \mathbb{R}_{\geq 0}$ for $i \in \{1, \ldots, n\}$ are the secondary costs; (2) we have a vector of upper bounds $\boldsymbol{u} \in \mathbb{R}^n_{\geq 0}$. Note that we use boldface to denote vectors. $C_i(\pi)$ denotes the expected cost of π on the i^{th} cost; we assume reachability and that improper policies incur infinite primary cost. A policy π is feasible if it satisfies all its secondary-cost constraints, i.e., $C_i(\pi) \leq u_i \; \forall i \in \{1, \ldots, n\}$, and π is optimal if it is closed, feasible, and minimises the primary cost $C_0(\pi)$ w.r.t. the other feasible policies. In contrast to SSPs, CSSPs may have no optimal deterministic policies and only strictly stochastic ones [1, 31]. This difference occurs because there may be deterministic policies that are infeasible on their own, but can be mixed to satisfy the secondary-cost constraints over expectation.

There are no primal method for optimally solving CSSPs and all optimal CSSP algorithms rely on the dual formulation. The dual formulation optimises over the space of occupation measures $\boldsymbol{x}$ where $x_{s,a}$ represents the expected number of times that s is reached and then a applied. LP 1 is the occupation measure LP for CSSPs [31], where $[\![\cdot]\!]$ is the Iverson bracket and we use the macros $out(s) = \sum_{a \in \mathsf{A}(s)} x_{s,a}$ and $in(s) = \sum_{s' \in \mathsf{S}, a' \in \mathsf{A}(s')} x_{s',a'} P(s|s',a')$ for each $s \in \mathsf{S}$. LP 1 can be interpreted as a network flow, where a unit of flow is injected into s_I, and must be routed through the actions so that all of it reaches the goals. An optimal solution $\boldsymbol{x}$ for LP 1 induces the optimal policy $\pi_{\boldsymbol{x}}$ with $\pi_{\boldsymbol{x}}(s,a) = x_{s,a}/out(s) \; \forall s \in \mathsf{S}, a \in \mathsf{A}(s)$.

$$\min_{\boldsymbol{x}} \sum_{s \in \mathsf{S}, a \in \mathsf{A}(s)} x_{s,a} C_0(a) \text{ s.t. C1–C4} \qquad \text{(LP 1)}$$

$$out(s) - in(s) = [\![s = s_I]\!] \qquad \forall s \in \mathsf{S} \setminus \mathsf{G} \quad \text{(C1)}$$

$$\sum_{g \in \mathsf{G}} in(g) = 1 \qquad \text{(C2)}$$

$$x_{s,a} \geq 0 \qquad \forall s \in \mathsf{S}, a \in \mathsf{A}(s) \quad \text{(C3)}$$

$$\sum_{s \in \mathsf{S}, a \in \mathsf{A}(s)} x_{s,a} C_i(a) \leq u_i \qquad \forall i \in \{1, \ldots, n\} \quad \text{(C4)}$$

There is an important connection between deterministic and stochastic policies: any stochastic policy can be represented as a convex combination (a.k.a. mixture) of deterministic ones [14]. We write this decomposition of the stochastic policy π as $\pi = \mu_0 \pi_0 + \cdots +$

$\mu_k \pi_k$ where $\mu_0, \ldots, \mu_k \in \mathbb{R}_{>0}$, $\mu_0 + \cdots + \mu_k = 1$, and $\pi_0, \ldots, \pi_k$ are π's constituent deterministic policies. We give an example of a CSSP and its policies in appendix A.

3 Solving CSSPs with Scalarisation

In this section we explain how to solve CSSPs with scalarisation, and implement this framework in our novel algorithm CARL. First, we give a high-level overview of the algorithm, and then give details for its three main steps.

Underlying the scalarisation approach is LP 2, the primal LP for LP 1. Intuitively, the V_s variables represent the cost-to-go $V(s)$, so we will use $V(s)$ and V_s interchangeably. In fact, if we construct LP 2 for an unconstrained SSP, then the λ_i and u_i terms disappear and the LP becomes the standard primal LP for solving SSPs which encodes the Bellman equations for V. If we fix $\boldsymbol{\lambda}$, then LP 2 again encodes the Bellman equations for an SSP, but with the modified cost function $C_{\boldsymbol{\lambda}}(a) = [1\ \boldsymbol{\lambda}] \cdot \boldsymbol{C}(a)$ (from C5) and the constant one-time terminal cost $-\sum_{i=1}^{n} \lambda_i u_i$ (from the objective). $\mathbb{S}(\boldsymbol{\lambda})$ denotes such an SSP parameterised by $\boldsymbol{\lambda}$.

Definition 1 (Scalarised SSP)**.** *Given $\mathbb{C}$ and a scalarisation $\boldsymbol{\lambda} \in \mathbb{R}^n_{\geq 0}$, $\mathbb{S}(\boldsymbol{\lambda})$ is an SSP relaxation of $\mathbb{C}$ with the cost function $C_{\boldsymbol{\lambda}} : \mathsf{A} \to \mathbb{R}_{\geq 0}$ s.t. $C_{\boldsymbol{\lambda}}(a) = [1\ \boldsymbol{\lambda}] \cdot \boldsymbol{C}(a)$ and the terminal cost $-\sum_{i=1}^{n} \lambda_i u_i$, which is incurred once when a goal is encountered, and is constant for $\boldsymbol{\lambda}$. We write a proper policy's cost as $C_{\boldsymbol{\lambda}}(\pi) = [1\ \boldsymbol{\lambda}] \cdot \boldsymbol{C}(\pi) - \sum_{i=1}^{n} \lambda_i u_i$.*

We can interpret each $\boldsymbol{\lambda}$ as a scalarisation that projects the CSSP's vector cost function onto a scalar cost function. An optimal solution $V^*, \boldsymbol{\lambda}^*$ for LP 2 gives the solution to the scalarised SSP's Bellman equations that are maximal over all scalarisations. Thanks to the strong duality of LPs, the solution $V^*, \boldsymbol{\lambda}^*$ can be transformed into an optimal solution for LP 1 using complementary slackness, yielding an optimal stochastic policy for the CSSP. This transformation can be interpreted as extracting all the greedy deterministic policies for V^*, and combining them into an optimal policy.

$$\max_{V, \boldsymbol{\lambda}} V_{s_I} - \sum_{i=1}^{n} \lambda_i u_i \text{ s.t. C5–C6} \qquad \text{(LP 2)}$$

$$V_s \leq C_0(a) + \sum_{i=1}^{n} \lambda_i C_i(a) + \sum_{s' \in \mathsf{S}} P(s'|s,a) V_{s'} \quad \forall s \in \mathsf{S} \setminus \mathsf{G}, a \in \mathsf{A}(s) \quad \text{(C5)}$$

$$V_g = 0 \quad \forall g \in \mathsf{G} \quad \text{(C6)}$$

$$\lambda_i \geq 0 \quad \forall i \in \{1, \ldots, n\} \quad \text{(C7)}$$

To avoid solving LP 2 directly, we separate the optimisation over V and $\boldsymbol{\lambda}$ into $\max_{\boldsymbol{\lambda}} L(\boldsymbol{\lambda})$, where $L(\boldsymbol{\lambda}) = \max_V V_{s_I} - \sum_{i=1}^{n} \lambda_i u_i$, i.e., $L(\boldsymbol{\lambda})$ is the optimal policy cost for $\mathbb{S}(\boldsymbol{\lambda})$. If we draw $L(\boldsymbol{\lambda})$ for all values of $\boldsymbol{\lambda}$, we get a surface that is piecewise linear concave [22]; we give some examples of this in appendix B. Thus, with an oracle that gives $L(\boldsymbol{\lambda})$ and a subgradient for each $\boldsymbol{\lambda}$, we can use any subgradient method to find $\boldsymbol{\lambda}^* = \max_{\boldsymbol{\lambda}} L(\boldsymbol{\lambda})$. Finally, we must find V^* that describes all optimal deterministic policies for $\mathbb{S}(\boldsymbol{\lambda}^*)$, from which we extract an optimal stochastic policy for the CSSP.

Alg. 2 presents the high-level pseudocode of this scalarisation algorithm, named CARL. In the remainder of this section, we explain CARL's three steps: (1) finding $\boldsymbol{\lambda}^*$ (line 3), (2) finding V^* given $\boldsymbol{\lambda}^*$ (line 4), and (3) extracting the optimal stochastic policy (line 5). To conclude, we prove CARL's correctness, and give more detail about CARL's error terms.

Algorithm 2: Solve CSSP with Scalarisation

1 **Function** *solve(CSSP $\mathbb{C}$, heuristic $\boldsymbol{H} : \mathsf{S} \to \mathbb{R}^{n+1}_{\geq 0}$)*
2 // $\boldsymbol{\lambda}^* \in \mathbb{R}^n_{\geq 0}, \boldsymbol{V} : \mathsf{S} \to \mathbb{R}^{n+1}_{\geq 0}, V^*_\forall : \mathsf{S} \to \mathbb{R}_{\geq 0}$
3 $\boldsymbol{\lambda}^*, \boldsymbol{V} \leftarrow$ *find-*$\boldsymbol{\lambda}^*(\mathbb{C}, \boldsymbol{H})$
4 $V^*_\forall \leftarrow$ *find-all-opts-for-*$\mathbb{S}(\boldsymbol{\lambda}^*, \boldsymbol{V}, \mathbb{C}, \boldsymbol{H})$
5 $\pi^* \leftarrow$ *extract-opt-policy*$(\boldsymbol{\lambda}^*, V^*_\forall, \mathbb{C})$
6 **return** π^*
7 **Function** *find-*$\boldsymbol{\lambda}^*$*(CSSP $\mathbb{C}$, $\boldsymbol{H} : \mathsf{S} \to \mathbb{R}^{n+1}_{\geq 0}$)*
8 // $\boldsymbol{\lambda} \in \mathbb{R}^n_{\geq 0}, \boldsymbol{V} : \mathsf{S} \to \mathbb{R}^{n+1}_{\geq 0}, \boldsymbol{g} \in \mathbb{R}^n$
9 $\boldsymbol{\lambda} \leftarrow 0, \boldsymbol{V} \leftarrow \boldsymbol{H}$
10 **repeat**
11 $\boldsymbol{V} \leftarrow$ *solve-*$\mathbb{S}(\boldsymbol{\lambda}, \boldsymbol{V}, \mathbb{C})$
12 subgradient $\boldsymbol{g} \leftarrow [V_1(s_I) - u_1\ \cdots\ V_n(s_I) - u_n]$
13 $\boldsymbol{\lambda} \leftarrow$ move from $\boldsymbol{\lambda}$ with $\boldsymbol{g}$ (see section 3.1)
14 **until** $\boldsymbol{\lambda}$ *has converged* (see section 3.5)
15 **return** $\boldsymbol{\lambda}, \boldsymbol{V}$
16 **Function** *solve-*$\mathbb{S}(\boldsymbol{\lambda}, \boldsymbol{V}, \mathbb{C})$
17 $\boldsymbol{V} \leftarrow$ solve $\mathbb{S}(\boldsymbol{\lambda})$ (see defn. 1) with SSP algorithm modified to work on vector $\boldsymbol{V}$ (see section 3.1)
18 **return** $\boldsymbol{V}$
19 **Function** *find-all-opts-for-*$\mathbb{S}(\boldsymbol{\lambda}, \mathbb{C}, \boldsymbol{H})$
20 $\boldsymbol{V}^*_\forall \leftarrow$ *solve-*$\mathbb{S}(\boldsymbol{\lambda}, \boldsymbol{V}, \mathbb{C})$ with modification that satisfies strong ϵ-consistency (see eq. (1) in section 3.2)
21 **return** $V^*_\forall$ *s.t.* $V^*_\forall(s) = [1\ \boldsymbol{\lambda}^*] \cdot \boldsymbol{V}^*_\forall(s)\ \forall s \in \mathsf{S}$
22 **Function** *extract-opt-policy(*$\boldsymbol{\lambda}$*,* $V^*_\forall$*, $\mathbb{C}$)*
23 $\boldsymbol{x}^* \leftarrow$ solution to SOL 1 with $\boldsymbol{\lambda}, V^*_\forall, \mathbb{C}$
24 **return** $\pi_{\boldsymbol{x}^*}$

We point out that CARL follows the concepts of Lagrangian decomposition and uses the same framework for finding $\boldsymbol{\lambda}^*$ as the algorithm by Hong and Williams [22]. The novelty of CARL is that it produces optimal (potentially stochastic) policies and improves on previous techniques, which we discuss further in section 4.

3.1 Finding a Maximal Scalarisation

In this section, we describe *find-*$\boldsymbol{\lambda}^*$ (line 7), which searches over $\boldsymbol{\lambda}$s with a subgradient method (outer problem). This technique must compute subgradients, which we do by solving $\mathbb{S}(\boldsymbol{\lambda})$ as subproblems

Outer optimisation over $\boldsymbol{\lambda}$. Suppose that we have an oracle that returns $L(\boldsymbol{\lambda})$ and a subgradient $\boldsymbol{g}$ for each $\boldsymbol{\lambda}$. Recall that our optimisation space, i.e., the surface obtained by plotting $L(\boldsymbol{\lambda})$ for all $\boldsymbol{\lambda}$s, is piecewise linear concave. This function has "sharp kinks" and can not be differentiated there, which is why we need subgradients [30].[1] To find $\max_{\boldsymbol{\lambda}} L(\boldsymbol{\lambda})$, we can use any optimisation method that accommodates concave and non-smooth search spaces. We follow Hong and Williams [22] and use an exact line search within coordinate search, which has been shown to be efficient for piecewise linear concave models with few constraints.

Coordinate search maximises $L(\boldsymbol{\lambda})$ by focusing on one coordinate at a time. That is, it sequentially solves for each $i \in \{1, \ldots, n\}$ the subproblem $\max_{\lambda_i} L(\boldsymbol{\lambda})$, where all λ_j for $j \neq i$ are fixed from the previous step. Thus, for each coordinate i it maximises $L([\lambda_1\ \ldots\ \lambda_n])$ where λ_i is the only variable and all other terms are fixed. This subproblem is solved with an exact line search which exploits that the subproblem is a piecewise linear concave problem with a single variable. The approach is reminiscent of binary search:

[1] Concave functions technically have supergradients, but these are still called subgradients in the literature.

it starts with $l = 0$ and $u \in \mathbb{R}_{>0}$ which give lower and upper bounds on the optimal assignment to λ_i. Then, the subgradients are computed at $\boldsymbol{\lambda}$ with $\lambda_i = l$ and $\lambda_i = u$, the intersection of these subgradients is computed as m, and either l or u is updated to m, depending on m's subgradient. This process repeats until l and u converge or their subgradients have the same sign (see [22] for more details). We visualise how $\boldsymbol{\lambda}$ is updated with coordinate search in appendix B. Unfortunately, coordinate search is incomplete for non-smooth problems [29], as we exemplify in appendix C. Fortunately, these edge cases occur rarely, as we see in our experiments (section 5), and are detectable: if we extract a stochastic policy from $V^\dagger, \boldsymbol{\lambda}^\dagger$ whose primary cost is greater than $L(\boldsymbol{\lambda}^\dagger)$, then we know that $\boldsymbol{\lambda}^\dagger \neq \boldsymbol{\lambda}^*$, because $L(\boldsymbol{\lambda}^*) = C_0(\pi^*)$ where π^* is extracted from $V^*, \boldsymbol{\lambda}^*$. If we detect that coordinate search failed we can fall back on a complete method to find $\boldsymbol{\lambda}^*$; we use the subgradient method with projection to ensure $\boldsymbol{\lambda} \geq 0$ [30]. Line 13 represents a single update of $\boldsymbol{\lambda}$ using coordinate search or the subgradient method as a fall-back, as we have described.

Oracle via $\mathbb{S}(\boldsymbol{\lambda})$ Subproblem. Recall that $\mathbb{S}(\boldsymbol{\lambda})$ (defn. 1) is an SSP relaxation of the CSSP, and therefore solved optimally by deterministic policies. $\mathbb{S}(\boldsymbol{\lambda})$'s optimal deterministic policy $\pi^*_{\boldsymbol{\lambda}}$ gives $L(\boldsymbol{\lambda})$ and a subgradient $\boldsymbol{g} \in \mathbb{R}^n$ in the following way: $L(\boldsymbol{\lambda}) = C_{\boldsymbol{\lambda}}(\pi^*_{\boldsymbol{\lambda}})$ and $\boldsymbol{g} = [C_1(\pi^*_{\boldsymbol{\lambda}}) - u_1 \ \cdots \ C_n(\pi^*_{\boldsymbol{\lambda}}) - u_n]$ [22]. These two steps correspond to lines 11 and 12, respectively. Thus, we can implement an oracle for $\boldsymbol{\lambda}$ by finding $\pi^*_{\boldsymbol{\lambda}}$ and then evaluating it w.r.t. $C_{\boldsymbol{\lambda}}$ and each secondary-cost function. To find $\pi^*_{\boldsymbol{\lambda}}$, we can use any optimal SSP algorithm with any heuristic that is admissible for SSPs, e.g., h-max [6], LMcut [21], or ROC [32]. In the remainder of this section, we show how to implement *solve*-$\mathbb{S}$ (line 16) efficiently.

Efficient Heuristic Search over $\mathbb{S}(\boldsymbol{\lambda})$s. The $\mathbb{S}(\boldsymbol{\lambda})$ subproblems are unconstrained SSPs and can be solved efficiently with existing heuristic-search algorithms such as CG-iLAO* [26], but we can improve efficiency further by exploiting some additional properties. First, instead of using a scalar value function $V : \mathsf{S} \to \mathbb{R}_{\geq 0}$, we use a vector value function $\boldsymbol{V} : \mathsf{S} \to \mathbb{R}^{n+1}_{\geq 0}$ with entries for each cost function. Then, we replace the scalar Bellman backups with a vectorised variant, where a Q-value is defined as $\boldsymbol{Q}(s,a) = \boldsymbol{C}(a) + \sum_{s' \in \mathsf{S}} P(s'|s,a)\boldsymbol{V}(s')$ and the Bellman backup becomes $\boldsymbol{V}(s) \leftarrow \boldsymbol{Q}(s, a_{\min})$ for some $a_{\min}$ that minimises $[1 \ \boldsymbol{\lambda}] \cdot \boldsymbol{Q}(s,a)$. With some minor technicalities (see appendix D for more details), SSP algorithms can be modified to use this vector Bellman backup, and they still solve $\mathbb{S}(\boldsymbol{\lambda})$ as an unconstrained SSP because the vector value function is projected onto $V_{\boldsymbol{\lambda}}(s) = [1 \ \boldsymbol{\lambda}] \cdot \boldsymbol{V}(s)$. However, we now work on $\boldsymbol{V}$ directly with the advantage that we simultaneously find an optimal policy π^* for $\mathbb{S}(\boldsymbol{\lambda})$ and evaluate the different costs of π^*, giving us $C_i(\pi^*_{\boldsymbol{\lambda}})$ for each cost function i.

To ensure optimality, we require an admissible heuristic to solve $\mathbb{S}(\boldsymbol{\lambda})$, i.e., $H(s) \leq V^*_{\boldsymbol{\lambda}}(s) \ \forall s \in \mathsf{S}$; in addition, we also require that the heuristic is a vector in $\mathbb{R}^{n+1}_{\geq 0}$ so that it can be used with $\boldsymbol{V}$. To address this, we introduce λ heuristics. These compute some admissible scalar heuristic $H(s)$, then identify which actions are selected by $H(s)$ and sum their vector costs to obtain a heuristic cost vector $\boldsymbol{H}$ with $[1 \ \boldsymbol{\lambda}] \cdot \boldsymbol{H}(s) = H(s) \leq V^*_{\boldsymbol{\lambda}}(s)$. For example, if a scalar heuristic uses actions a_0 and a_1 to construct its estimate $H(s) = [1 \ \boldsymbol{\lambda}] \cdot \boldsymbol{C}(a_0) + [1 \ \boldsymbol{\lambda}] \cdot \boldsymbol{C}(a_1)$, then we assign $\boldsymbol{H}(s) \leftarrow \boldsymbol{C}(a_0) + \boldsymbol{C}(a_1)$ which indeed satisfies $[1 \ \boldsymbol{\lambda}] \cdot \boldsymbol{H}(s) = H(s)$. For most heuristics, we can use their internal data-structures to extract the selected actions, e.g., for LMcut [21] we consider the representative action from each cut and its weight, and for ROC [32] we consider actions' operator counts. Another way to obtain admissible vector heuristics is with ideal-point (IP) heuristics [15]. Such heuristics are $\boldsymbol{H} = \langle H_0, \ldots, H_n \rangle$ where H_i are all admissible w.r.t. their cost function i. The advantage of λ heuristics is that they are computed for the specific $\mathbb{S}(\boldsymbol{\lambda})$ and are therefore more informative, but with the trade-off that they have to be recomputed for each $\mathbb{S}(\boldsymbol{\lambda})$, whereas an IP heuristic is admissible for all $\mathbb{S}(\boldsymbol{\lambda})$s.

Warm starting $\mathbb{S}(\boldsymbol{\lambda})$s. Notice that all $\mathbb{S}(\boldsymbol{\lambda})$s are the same problem only differing in their cost functions. This makes $\boldsymbol{V}$ from a previous $\mathbb{S}(\boldsymbol{\lambda}_{\text{old}})$ a good candidate to start solving a new $\mathbb{S}(\boldsymbol{\lambda}_{\text{new}})$, a technique known as warm start in Operations Research. However, we have to be careful in reusing $\boldsymbol{V}$. Suppose $\boldsymbol{V}$ is a solution to $\mathbb{S}(\boldsymbol{\lambda}_{\text{old}})$ and we have $V_{\boldsymbol{\lambda}_{\text{old}}}$ with $V_{\boldsymbol{\lambda}_{\text{old}}}(s) = [1 \ \boldsymbol{\lambda}_{\text{old}}] \cdot \boldsymbol{V}(s) \ \forall s \in \mathsf{S}$ and $V_{\boldsymbol{\lambda}_{\text{new}}}$ with $V_{\boldsymbol{\lambda}_{\text{new}}}(s) = [1 \ \boldsymbol{\lambda}_{\text{new}}] \cdot \boldsymbol{V}(s) \ \forall s \in \mathsf{S}$. Even if $V_{\boldsymbol{\lambda}_{\text{old}}}$ is admissible for $\mathbb{S}(\boldsymbol{\lambda}_{\text{old}})$, there is no guarantee that $V_{\boldsymbol{\lambda}_{\text{new}}}$ is admissible for $\mathbb{S}(\boldsymbol{\lambda}_{\text{new}})$, which may break the optimality of heuristic search algorithms. Conveniently, CG-iLAO* has a built-in mechanism for efficiently handling potentially inadmissible states by tracking states whose values have increased or decreased, i.e., we mark the states where $[1 \ \boldsymbol{\lambda}_{\text{new}}] \cdot \boldsymbol{V}(s)$ decreases or increases w.r.t. $[1 \ \boldsymbol{\lambda}_{\text{old}}] \cdot \boldsymbol{V}(s)$ in Γ. As explained in section 2, this ensures that CG-iLAO* fixes any issues in the value function and finds the optimal policy for $\mathbb{S}(\boldsymbol{\lambda}_{\text{new}})$. Reusing $\boldsymbol{V}$ in this way speeds up CARL significantly because the optimal policy for one $\mathbb{S}(\boldsymbol{\lambda})$ is often good for the next one, and in the best case, if the change in $C_{\boldsymbol{\lambda}}$ does not affect the optimal policy, then $\boldsymbol{V}$ immediately solves the new $\mathbb{S}(\boldsymbol{\lambda}')$.

3.2 Finding an Optimal Value Function

Once we have $\boldsymbol{\lambda}^*$, we need to compute $V^* : \mathsf{S} \to \mathbb{R}_{\geq 0}$ that describes all optimal deterministic policies for $\mathbb{S}(\boldsymbol{\lambda}^*)$, which is done in the pseudocode by *find-all-opts-for*-$\mathbb{S}$ (line 19). Let $\Pi(V)$ denote the set of tied-greedy policies for V, that is, the greedy policies of V obtained by breaking ties in all possible ways. Then, if V^* is the unique solution to the Bellman equations, $\Pi(V^*)$ gives precisely all optimal deterministic policies for the SSP. For us, V^* is produced by an SSP algorithm that only guarantees ϵ-consistency [7]. This means V^* need only have a small error within one policy's envelope, and any other policies' envelopes can have arbitrarily larger value functions, i.e., V^* need only encode a single greedy policy. We give an example to illustrate this in appendix E. In order to obtain a value function that encodes all (approximately) optimal policies, without solving the Bellman equations exactly, we introduce strong ϵ-consistency. This condition requires a value function to be ϵ-consistent for all of its greedy policies. Formally, V is strongly ϵ-consistent when

$$\forall s \in \bigcup_{\pi \in \Pi(V)} \mathsf{S}^\pi \quad \text{RES}(s) \leq \epsilon. \tag{1}$$

When a value function is strongly ϵ-consistent, we write it as $V^*_\forall$. Note that this has not been defined for SSPs before, because there was no need for more than one policy.

Finding $V^*_\forall$. Existing SSP algorithms are designed to find ϵ-consistent solutions, so we must modify them to find strongly ϵ-consistent solutions. This is done by updating the termination condition, and ensuring that Bellman backups are applied for all greedy policies for V. We make this modification concrete for the version of CG-iLAO* from section 3.1, but it can be done for other algorithms too. Recall that this version of CG-iLAO* works over the vector value function $\boldsymbol{V} : \mathsf{S} \to \mathbb{R}^{n+1}$, so we are looking for $\boldsymbol{V}^*_\forall$ that induces $V^*_\forall$ with $V^*_\forall(s) = [1 \ \boldsymbol{\lambda}^*] \cdot \boldsymbol{V}^*_\forall(s)$. Instead of tracking the candidate policy $\widehat{\pi}_V$, we track the union of

all tied-greedy policies (up to ϵ) with $\texttt{tied}(s) = \{a \in \mathsf{A}(s) : Q_{\boldsymbol{\lambda}}(s,a) \leq \min_{a' \in \mathsf{A}(s)} Q_{\boldsymbol{\lambda}}(s,a') + \epsilon\}$. Then, CG-iLAO*'s policy envelope $\mathcal{E}$ over all tied-greedy policies is found by running the depth-first search from s_I over $\texttt{tied}$. Additionally, any actions with $Q_{\boldsymbol{\lambda}}(s,a) \leq V_{\boldsymbol{\lambda}}(s) + \epsilon$ must be added to $\widehat{\mathbb{S}}$ in line 9 so that new tied-greedy actions are caught. Also, we replace the termination condition $\widehat{\pi}_{\text{old}} = \widehat{\pi}_V$, now requiring that no new actions have been added to any $\texttt{tied}(s)$. We only care about new actions and do not track removals, because they only shrink the envelope and do not affect V. These changes ensure that CG-iLAO* outputs a strongly ϵ-consistent solution which captures all ϵ-consistent policies. To prove this, we use similar arguments to Schmalz and Trevizan [26] to obtain that $\forall s \in \mathcal{E}\ \text{RES}(s) \leq \epsilon$. But $\mathcal{E}$, obtained by DFS over the tied-greedy actions (up to ϵ), captures all tied-greedy policies on V by construction, so V must be strongly ϵ-consistent.

3.3 Extracting Stochastic Policies

Suppose we have $\boldsymbol{\lambda}^*$ and $V^*_\forall$ which encodes all optimal deterministic policies for $\mathbb{S}(\boldsymbol{\lambda}^*)$. The final step of CARL is to extract an optimal (potentially stochastic) policy, which is done by *extract-opt-policy* (line 22). To implement this, recall that $V^*_\forall$ and $\boldsymbol{\lambda}^*$ give the primary cost of all (potentially stochastic) optimal policies for the CSSP with $C_0(\pi^*) = V^*_\forall(s_I) - \sum_{i=1}^n \lambda^*_i u_i$. Furthermore, we can extract π^*'s envelope and support from the envelopes and supports of $V^*_\forall$'s greedy policies. We now show how to use these insights to extract π^*, and thereby solve the CSSP optimally.

Given a policy π, we can evaluate its primary cost $C_0(\pi)$ with LP 1 by restricting the LP's variables to the support of π, adding the constraints $x_{s,a}/out(s) = \pi(s,a)$ for all $s,a \in \text{supp}(\pi)$, and removing the objective function. The resulting System of Linear Equations and Inequalities (SOL) finds $C_0(\pi)$ as $\sum_{s \in \mathsf{S}, a \in \mathsf{A}} x_{s,a} C_0(a)$. Our case is the converse since we have π's cost $C_0(\pi)$ but do not have the distributions $\pi(s,\cdot)$. So, instead of adding the constraints $x_{s,a}/out(s) = \pi(s,a)$, we add the constraint $\sum_{s \in \mathsf{S}, a \in \mathsf{A}(s)} x_{s,a} C_0(a) = C_0(\pi^*) = V^*_\forall(s_I) - \sum_{i=1}^n \lambda^*_i u_i$; and we approximate $\text{supp}(\pi^*)$ with $\text{supp}(V^*_\forall)$, which is defined as $\bigcup_{\pi \in \Pi(V^*_\forall)} \text{supp}(\pi)$, noting that $\text{supp}(V^*_\forall) \supseteq \text{supp}(\pi^*)$ as we explain later. This yields SOL 1 without C9 and C10 which we also explain later. Thus, a solution $\boldsymbol{x}$ for SOL 1 (with or without C9 and C10) induces a feasible policy $\pi_{\boldsymbol{x}}$ with the same cost as an optimal policy by construction. SOL 1's only decision is how to distribute the actions in $\text{supp}(V^*_\forall)$ so that the resulting policy has the required costs; in other words, it computes how to combine $\mathbb{S}(\boldsymbol{\lambda})$'s deterministic policies into an optimal feasible policy for $\mathbb{C}$.

$$\text{find } \boldsymbol{x} \text{ s.t. C1–C3 over } \text{supp}(V^*_\forall) \text{ and C8–C10} \qquad \text{(SOL 1)}$$

$$\sum_{s,a \in \text{supp}(V^*_\forall)} x_{s,a} C_0(a) = V^*_\forall(s_I) - \sum_{i=1}^n \lambda^*_i u_i \qquad \text{(C8)}$$

$$\sum_{s,a \in \text{supp}(V^*_\forall)} x_{s,a} C_i(a) \leq u_i \qquad \forall i \in \{1,\dots,n\} \text{ s.t. } \lambda^*_i = 0 \quad \text{(C9)}$$

$$\sum_{s,a \in \text{supp}(V^*_\forall)} x_{s,a} C_i(a) = u_i \qquad \forall i \in \{1,\dots,n\} \text{ s.t. } \lambda^*_i > 0 \quad \text{(C10)}$$

Formally, SOL 1 comes from complementary slackness, a technique that lets us transform an optimal solution of an LP into an optimal solution for its dual [5]. Suppose $V^*_\forall, \boldsymbol{\lambda}^*$ is an optimal basic feasible solution for LP 2 that is non-degenerate; intuitively, a basic feasible solution corresponds to a deterministic policy (rather than a stochastic one) and non-degeneracy is a technical requirement that the solution does not have too many variables set to zero. With such $V^*_\forall, \boldsymbol{\lambda}^*$, we can recover a solution for LP 2's dual (LP 1) with complementary slackness by constructing a system of linear equations according to the following rules, and then solving it:

$$V^*_\forall(s) > 0 \implies \text{C1 for } s \text{ is tight, i.e., } out(s) - in(s) = [\![s = s_I]\!]$$

$$\lambda_i > 0 \implies \text{C4 for } i \text{ is tight, i.e., } \sum_{s \in \mathsf{S}, a \in \mathsf{A}(s)} x_{s,a} C_i(a) = u_i$$

$$\text{C5 for } s,a \text{ is loose, i.e., } V^*_\forall(s) < Q^*_\forall(s,a) \implies x_{s,a} = 0.$$

These rules describe the constraints of SOL 1. In particular, note that these rules tighten the secondary-cost constraints to an equality $\sum_{s \in \mathsf{S}, a \in \mathsf{A}(s)} x_{s,a} C_i(a) = u_i$ whenever $\lambda^*_i > 0$, yielding the constraints C9 and C10. Unfortunately, our $V^*_\forall, \boldsymbol{\lambda}^*$ is often degenerate, so this system of linear equations is underspecified, and we must include additional constraints from the original problem. In particular, we have to reintroduce the secondary cost constraints C4 where $\lambda_i = 0$, which is what we do in C9. We also include C8 to ensure the resulting policy is indeed optimal.

We have been claiming that $\text{supp}(\pi^*) \subseteq \text{supp}(V^*_\forall)$ and that SOL 1 is constructing π^* from the optimal deterministic policies of $\mathbb{S}(\boldsymbol{\lambda}^*)$. This follows from the last complementary slackness rule: occupation measures $x_{s,a}$ are only allowed to be non-zero if $V^*_\forall(s) = Q^*_\forall(s,a)$, i.e., if a is a tied-greedy action and thereby $\pi(s) = a$ for some $\pi \in \Pi(V^*_\forall)$.

3.4 Correctness of CARL

In this section we show that CARL always finds an optimal solution. First, *find-*$\boldsymbol{\lambda}^*(\mathbb{C}, \boldsymbol{H})$ (alg. 2) is guaranteed to find $\boldsymbol{\lambda}^*$. This is because $\max_{\boldsymbol{\lambda}} L(\boldsymbol{\lambda})$ is a piecewise linear concave problem, solving $\mathbb{S}(\boldsymbol{\lambda})$s yields subgradients [22], and we use a complete subgradient method that is guaranteed to converge to $\boldsymbol{\lambda}^*$ (if coordinate search fails, it falls back on the subgradient method, see section 3.1). It remains to show that CARL finds V^* for $\mathbb{S}(\boldsymbol{\lambda}^*)$ such that the solution encodes an optimal policy.

Lemma 1. *If π^* is an optimal policy for the CSSP, then its constituent deterministic policies must be optimal for $\mathbb{S}(\boldsymbol{\lambda}^*)$.*

Proof. We first show that π^* is optimal for $\mathbb{S}(\boldsymbol{\lambda}^*)$, and then conclude that its constituent deterministic policies must also be optimal for $\mathbb{S}(\boldsymbol{\lambda}^*)$. Policy π^*'s cost in $\mathbb{S}(\boldsymbol{\lambda}^*)$ can be rewritten as $C_{\boldsymbol{\lambda}^*}(\pi^*) = C_0(\pi^*) + \sum_{i=1}^n \lambda^*_i (C_i(\pi^*) - u_i)$. But π^* is feasible, so $C_i(\pi^*) - u_i \leq 0$ for all $i \in \{1,\dots,n\}$, and therefore $C_{\boldsymbol{\lambda}^*}(\pi^*) \leq C_0(\pi^*)$. On the other hand, we know that $C_{\boldsymbol{\lambda}^*}(\pi) \geq C_0(\pi^*)$ for any policy π, due to the semantic of $\boldsymbol{\lambda}^*$ within strong duality. Thus, $C_{\boldsymbol{\lambda}^*}(\pi^*) = C_0(\pi^*)$, which is the minimal cost for $\mathbb{S}(\boldsymbol{\lambda}^*)$, and π^* is optimal for $\mathbb{S}(\boldsymbol{\lambda}^*)$. We know that π^* can be decomposed into the convex combination of deterministic policies $\mu_0\pi_0 + \cdots + \mu_k\pi_k$, and $C_{\boldsymbol{\lambda}^*}(\pi_i) = C_{\boldsymbol{\lambda}^*}(\pi^*)$ for each $i \in \{0,\dots,k\}$, because otherwise π^* can not be optimal for $\mathbb{S}(\boldsymbol{\lambda}^*)$. □

Theorem 1. *CARL's find-all-opts-for-*$\mathbb{S}(\boldsymbol{\lambda}^*, \mathbb{C}, \boldsymbol{H})$ *finds $\boldsymbol{V}^*_\forall$ so that SOL 1 with $\boldsymbol{\lambda}^*$ and $V^*_\forall$ constructed by $V^*_\forall(s) = \boldsymbol{\lambda}^* \cdot \boldsymbol{V}^*_\forall(s)$ produces an optimal stochastic policy.*

Proof. Suppose $V^*_\forall$ encodes all the optimal deterministic policies for $\mathbb{S}(\boldsymbol{\lambda}^*)$. If an optimal policy π^* exists for the CSSP, then its constituent deterministic policies must be optimal for $\mathbb{S}(\boldsymbol{\lambda}^*)$ by lem. 1; therefore its constituent policies are a subset of $\Pi(V^*_\forall)$. It follows that SOL 1 contains the whole support of such policy π^*. Since $C_0(\pi^*) = \boldsymbol{V}^*_\forall(s_I) - \sum_{i=1}^n \lambda^*_i u_i$ due to strong duality, it must be the case that SOL 1 induces an optimal policy, thanks to its connection

to LP 1. We have argued that the version of CG-iLAO* presented in section 3.2 finds all policies for which V can be ϵ-consistent, which gives approximately optimal policies for small ϵ, and precisely the optimal policies as $\epsilon \to 0$. We give more detail about CARL's error terms in section 3.5. □

3.5 Error Terms

In this section, we describe the error terms that are present in CARL and explain how to bound their effects on CARL's policies. We have already discussed that CG-iLAO*, which solves CARL's $\mathbb{S}(\boldsymbol{\lambda})$ subproblems, uses strong ϵ-consistency with $\epsilon \in \mathbb{R}_{>0}$ as its stopping condition, extending regular ϵ-consistency for SSPs (see section 3.2). The error associated with ϵ-consistency is well-understood and accepted in the SSP literature: for a fixed ϵ it is possible to construct a pathological problem where an ϵ-consistent policy is arbitrarily worse than the optimal policy, but for any SSP, as $\epsilon \to 0$, ϵ-consistent policies become optimal [7, 25, 17]. We inherit the same properties in strong ϵ-consistency, so our version of CG-iLAO* produces $V'_\forall \approx V^*_\forall$ where the error disappears as $\epsilon \to 0$. Another error term appears in *find-*$\boldsymbol{\lambda}^*$, which we have omitted until now for simplicity. Recall that to find $\boldsymbol{\lambda}^*$ we use coordinate search, and if it fails we fall back on the subgradient method (see section 3.1). Both require a tolerance term $\eta \in \mathbb{R}_{>0}$. In practice, coordinate search stops when it iterates over all coordinates and no single coordinate was able to improve $L(\boldsymbol{\lambda})$ by more than η. The subgradient method stops when its step size (which is monotonically decreasing over the algorithm's iterations) is smaller than η. In both cases, this yields an approximate solution $\boldsymbol{\lambda}' \approx \boldsymbol{\lambda}^*$, where the error disappears as $\eta \to 0$.

Both ϵ and η can affect CARL's solution, and we explain how this can be mitigated. Assume for now that ϵ is sufficiently small. If CARL finds $\boldsymbol{\lambda}' \approx \boldsymbol{\lambda}^*$ and an associated policy π, we have

$$L(\boldsymbol{\lambda}') \leq C_0(\pi^*) \leq C_0(\pi)$$

where π^* is an optimal policy for the CSSP. This gives us the optimality gap $C_0(\pi) - L(\boldsymbol{\lambda}')$, which upper bounds how far π's primary cost deviates from π^*'s. If the gap is too large, we can reduce η. Now, we return our attention to ϵ. The value of ϵ affects the subgradient search oracle's value of $L(\boldsymbol{\lambda})$ and subgradient $\boldsymbol{g}$ at each $\boldsymbol{\lambda}$, which in turn can significantly affect the found policy's quality and breaks the optimality gap for η. Recall that this is standard for algorithms using ϵ-consistency, and they only guarantee an optimal policy as $\epsilon \to 0$. For additional guarantees, there are "outer loop" methods that run the algorithm and select ϵ automatically in order to give the stronger guarantee of ϵ-optimality [17, 19]. CARL is in a similar position, and guarantees an optimal policy as $\epsilon \to 0$, but requires an outer loop method to select ϵ automatically for stronger guarantees. The outer loop methods for ensuring ϵ-optimality apply to the strongly ϵ-consistent subproblems with minimal changes, which lets us recover a correct optimality gap. Thus, it is possible to modify CARL to produce solutions with stronger guarantees. In practice, values of $\epsilon = \eta = 0.0001$ are usually "good enough."

4 Related Work

In the Operations Research (OR) community, it is common to consider the Lagrangian dual of a constrained problem, and then use some version of the subgradient method to find a solution. This framework has been used to find plans for constrained deterministic shortest path problems [16], stochastic policies for Constrained Partially Observable Markov Decision Processes (POMDPs) [23], and deterministic policies for CSSPs [22]. Our presentation differs from theirs, but LP 2 is functionally identical to the Lagrangian dual of LP 1, and CARL fits within this OR framework. They have the same $\boldsymbol{\lambda}$ as our scalarisation term, but they call it a Lagrangian multiplier.

Hong and Williams [22]'s algorithm is the most similar to CARL. Both solve CSSPs and start by finding the CSSPs' $\boldsymbol{\lambda}^*$. The difference is that CARL looks for optimal, potentially stochastic, policies whereas they focus on deterministic ones. Our method for finding $\boldsymbol{\lambda}^*$ is based on theirs but we make substantial algorithmic improvements. They solve $\mathbb{S}(\boldsymbol{\lambda})$s with iLAO* and reuse $\boldsymbol{V}$ as a warm start (see section 3.1); however, they need to run VI on iLAO*'s expanded states to make $\boldsymbol{V}$ admissible for the next step. CARL uses CG-iLAO*, allowing it to focus only on the changes to $\boldsymbol{\lambda}$ as opposed to VI's blind search. As a result, CARL finds $\boldsymbol{\lambda}^*$ up to $4.7\times$ faster than their method with a mean speedup of $2\times$.

Once $\boldsymbol{\lambda}^*$ has been found, the similarities between our algorithms end. Strong duality does not hold for deterministic policies, thus $\boldsymbol{\lambda}^*$ does not directly produce an optimal deterministic policy for the original CSSP. To overcome this, they use an expensive second stage that enumerates deterministic policies until the optimal one is found. CARL does not require this stage because strong duality holds for stochastic policies. Instead, we must ensure that V^* encodes all optimal policies for $\mathbb{S}(\boldsymbol{\lambda}^*)$, which can be done cheaply with dynamic programming, and then we use SOL 1 to extract the optimal policy for the CSSP. Both these steps have no analogue in their algorithm. Thus, CARL is guaranteed to find an optimal policy, stochastic or not, for CSSPs with relatively little computational effort after finding $\boldsymbol{\lambda}^*$, whereas Hong and Williams [22]'s method can only find an optimal *deterministic* policy, which can cost more than the optimal stochastic policy, and requires an expensive second stage to do so.

The algorithm by Lee et al. [23] shares with CARL that they get deterministic policies from their subproblems, which must be combined into a stochastic one. However, their setting and approach is quite different: they use Monte-Carlo sampling to approximate their Q-values, and their policies map histories onto action distributions, obtained by solving an LP for each history. We compute V^* and its Q-values with a modified SSP search, and then solve SOL 1 once.

Multi-Objective (MO) planning is also relevant. There are various models of probabilistic MO problems, e.g., MOMDPs with queries from model checking [13] and MOSSPs from planning [11]. These models are similar to CSSPs in that they have multiple cost functions, but the difference is that MO problems are solved by finding all undominated trade-offs between cost functions, e.g., consider π_1, π_2, π_3 with $\boldsymbol{C}(\pi_1) = [1\ 3], \boldsymbol{C}(\pi_2) = [3\ 1], \boldsymbol{C}(\pi_3) = [3\ 2]$; π_1 and π_2 are undominated and give different trade-offs of C_0 and C_1, and π_3 is dominated by π_2. The set of undominated policies is called the Pareto front. In contrast, CSSPs are solved by a single policy that minimises C_0 while satisfying the secondary-cost constraints. Consequently, MOSSPs are more general than CSSPs, and their algorithms can be applied to solve CSSPs, e.g., [13, 11], but these are much less efficient because they need to track an entire set of policies in the Pareto front, rather than searching for a single policy.

Scalarisation is a standard technique in MO planning [12], and has been applied to MOMDPs with queries [13]. Their algorithm's similarity to CARL is that both project the cost vector onto scalars to induce single-objective SSPs (or MDPs) as their subproblems. However, the way that the scalarisations are explored and the purpose is completely different: they use different scalarisations to find different trade-offs, and CARL uses scalarisation in the context of Lagrangian optimisation where scalarisations $\boldsymbol{\lambda}$ are also called Lagrangian multipliers. Importantly, CARL's scalarisation is a single-

objective technique, and it progressively finds a better scalarisation vector, whereas Forejt et al. [13] accumulate a set of non-dominated scalarisation vectors. Another difference is that Forejt et al. [13] use Value Iteration to solve the subproblems, and we use heuristic search, which increases CARL's efficiency but presents technical challenges as addressed in section 3.1 and section 3.2. Heuristic search has been shown to dominate non-guided search approaches from model checking, e.g., [2]. We point out that our method for efficiently solving subproblems can be applied to the algorithm from Forejt et al. [13], establishing another contribution of this work.

5 Experiments

We compare our novel algorithm CARL with i-dual [31] and i^2-dual [32], the state-of-the-art algorithms for finding optimal stochastic policies for CSSPs. For CARL, we consider the heuristics LMcut [21] and ROC [32] as λ heuristics (section 3.1), called λ-LMcut and λ-ROC. In appendix F we also test LMcut and ROC as ideal-point (IP) heuristics (section 3.1). For i-dual, we use ideal-point LMcut (IP-LMcut) and C-ROC [32]. I^2-dual uses C-POM [32], which is built into the algorithm. In our test problems, we allow CSSPs to have dead ends, but transform them into CSSPs without dead ends with the finite-penalty transformation, which adds give-up actions to each state that lead to a goal with a large cost penalty. We require a user-specified dead-end penalty for each cost function [31]. For problems with dead ends before the finite-penalty transformation, we augment ROC and i^2-dual's built-in C-POM with h-max as a dead-end detector [32]. We run experiments over the following benchmark domains:

Search and Rescue (SAR) [31]. A drone must rescue one of multiple survivors on an $n \times n$ grid as quickly as possible ($n \in \mathbb{N}$). One survivor's position is known and has distance $d \in \mathbb{N}$ from the drone's initial position. Some other locations, selected according to a density $r \in [0, 1]$, have a predetermined probability of having another survivor. There is a single secondary-cost constraint over fuel usage.

Elevators (Elev) [31]. In a 10-floor building, $w \in \mathbb{N}$ people have called the elevator and are waiting, and $h \in \mathbb{N}$ hidden people have not pressed the button yet, and do so probabilistically in each timestep. The $e \in \mathbb{N}$ elevators must be routed to deliver all people to their destinations in the fewest timesteps. Each person has a constraint over the time spent waiting, and the time spent travelling in an elevator, resulting in $2(w + h)$ secondary-cost constraints.

Exploding Blocks World (ExBW) [31]. Based on Exploding Blocks World for SSPs [9], we must rearrange $N \in \mathbb{N}$ blocks into a given arrangement with the minimal number of moves. This is complicated by the blocks being rigged with an explosive that can explode, destroying the block or table immediately underneath it. In the CSSP variant, the table can be repaired with a large primary cost, but destroyed blocks can not be repaired. There is a single secondary-cost constraint, limiting the expected number of destroyed blocks to $c \in \mathbb{R}_{\geq 0}$. We consider specific starting and goal arrangements of blocks from the 2008 International Probabilistic Planning Competition [10], and identify specific problems by the parameter id.

PARC Printer (PARC) [32]. Based on the IPC domain, a modular printer must print four pages with various requirements. Three of the printer's components are unreliable, where a page can get jammed with probability 0.1, simultaneously ruining the page and rendering the component unusable. There are two secondary-cost constraints: $f \in [0, 1]$ constrains how many components are allowed to fail and $u \in \mathbb{N}$ limits how often a particular component is allowed to be used ($u = \infty$ deactivates this constraint).

Triangle Tireworld (CTW) [27]. In the Triangle Tireworld for SSPs we must drive across a triangular network of cities, where $n \in \mathbb{N}$ specifies the size of the network, and $d \in \mathbb{N}$ gives the starting distance from the goal [24, 26]. Between any two cities the car gets a flat tyre with probability 0.5; with a spare tyre the flat can be replaced and the journey can continue; if no spare is loaded, the car is stuck and no actions are available. The car only fits one spare tyre at a time, and spare tyres can only be acquired in select cities. In the CSSP variant, tyres can be purchased in these cities for one of $c \in \mathbb{N}$ currencies, and the problem has secondary-cost constraints over the c-many currencies.

We ran each planner and heuristic on 30 instances of each problem on a cluster with Intel Xeon 3.2 GHz CPUs limited to one CPU core with 30 mins and 4GB RAM. For SAR and Elev, a problem instance is a randomly generated problem for the specified parameters. For the other domains, the parameters specify a unique problem and an instance couples this problem with a random seed that is passed to the algorithm. We use CPLEX Version 22.1.1 to solve LPs and SOLs. The source code and benchmarks are available at [28].

The results of our experiments are presented in tab. 1. We give each algorithm's coverage (the number of instances solved within its time and memory limits), and over the instances where the algorithm converged we give its mean runtime (and 95% C.I.). We highlight the fastest algorithms, where an algorithm is considered the fastest for a particular problem if it has the maximum coverage and then its mean time plus 95% C.I. is less than the next best algorithm's mean time minus 95% C.I. We now identify the best performing algorithm per domain.

SAR, ExBW, and CTW. CARL dominates in these domains: either CARL(λ-ROC) or CARL(λ-LMcut) is the best performing planner and, with exception of ExBW (1,5,.1) and (3,6,.91), the second best performing planning is also CARL. Moreover, CARL can obtain substantially larger coverage with the extreme case of ExBW (6,8,.3) in which CARL using both heuristics obtained full coverage while all other planners had zero coverage. Comparing only entries with full coverage in these domains, CARL has $10\times$, $19\times$, and $42\times$ speedup on average w.r.t. i-dual (C-ROC), i-dual (IP-LMcut) and i^2-dual, respectively. Between CARL(λ-ROC) and CARL(λ-LMcut), there is no clear winner on SAR; CARL(λ-LMcut) dominates on ExBW; and CARL(λ-ROC) dominates on CTW.

Elev. With one elevator i-dual dominates, and with two elevators CARL dominates. With one elevator, CARL fails to solve some instances for one of two reasons: (1) CARL's coordinate search fails and falls back on the subgradient method, which is significantly slower than coordinate search; (2) $\mathbb{S}(\boldsymbol{\lambda})$s are too difficult to solve in some cases, which suggests the heuristic does not give adequate guidance. Regarding (1), we note that these Elev instances are the only cases in our benchmarks where coordinate search fails and CARL is forced to fall back on the subgradient method to ensure completeness (see section 3.1). Here, the subgradient method is significantly slower than coordinate search and failed to converge in time. However, we reiterate that falling back to the subgradient method ensures that CARL is complete, and given enough time and memory CARL would solve the instances. With two elevators, the problem structure changes so that coordinate search succeeds and (1) does not occur, and any instances where CARL fails to find a solution is because of (2), that is, the coordinate search's subproblems take too long to

	algorithm	CARL				i-dual				i2-dual		algorithm	CARL				i-dual				i2-dual	
	heuristic	λ-ROC		λ-LMcut		C-ROC		IP-LMcut				heuristic	λ-ROC		λ-LMcut		C-ROC		IP-LMcut			
SAR (n, d, r)	4, 3, .75	**30**	**5.6± 1.4**	**30**	**4.3± 1.0**	30	104.1± 51.0	30	45.8± 21.1	30	69.2± 56.4	0.0, 1	30	311.2±13.6	30	270.9± 8.3	30	126.2± 9.1	30	103.1± 2.4	**30**	**42.8± 3.1**
	4, 4, .75	**30**	**41.9± 5.3**	**30**	**34.1± 4.2**	5	1345.7±534.1	9	979.8±296.6	8	495.5±417.8	0.0, ∞	30	100.7± 0.8	30	141.3± 1.0	30	47.1± 2.5	30	42.1± 1.8	**30**	**29.5± 1.8**
	5, 3, .50	**30**	**4.9± 0.9**	**30**	**4.8± 1.1**	30	25.1± 8.5	30	13.1± 4.6	30	11.1± 4.0	0.2, 1	30	600.7± 3.6	30	863.4± 9.7	0		0		**30**	**65.1± 11.1**
	5, 3, .75	**30**	**12.3± 2.5**	**30**	**13.3± 2.9**	30	253.8±110.6	30	113.3± 46.0	30	168.0±105.9	0.2, ∞	30	121.7± 1.9	30	185.7± 1.2	30	1537.7±72.8	0		**30**	**41.7± 4.9**
	5, 4, .50	**30**	**30.8± 6.8**	**30**	**36.2± 7.9**	17	695.0±268.5	24	550.6±166.1	27	637.8±213.1	PARC (f,u) 0.4, 1	30	882.8±31.7	**30**	**564.1±12.8**	0		0		3	801.5±536.2
	5, 4, .75	**30**	**126.6± 45.2**	**30**	**142.5± 49.3**	6	930.9±369.3	6	499.2±226.4	8	349.1±288.1	0.4, ∞	**30**	**130.9± 2.5**	30	188.1± 1.3	0		0		3	665.9±308.3
Elev (e,w,h)	1, 1, 1	30	1.6± 0.6	30	1.2± 0.3	30	1.5± 0.1	**30**	**0.8± 0.1**	30	6.2± 0.6	0.6, 1	30	318.4±10.3	**30**	**255.4± 6.9**	0		0		0	
	1, 1, 2	29	50.9± 26.8	29	35.0± 12.7	30	25.3± 1.0	**30**	**21.1± 2.6**	30	280.0± 46.5	0.6, ∞	**30**	**25.5± 0.5**	30	39.6± 0.3	26	1508.9±68.9	0		0	
	1, 2, 1	25	14.2± 5.6	25	13.1± 4.2	30	13.9± 0.9	**30**	**9.8± 1.4**	30	174.2± 38.4	0.8, 1	30	345.5± 7.4	**30**	**267.1± 8.0**	0		0		0	
	1, 2, 2	22	313.3±150.2	22	336.9±115.8	**30**	**340.1± 35.7**	**30**	**392.5± 72.1**	1	989.2	0.8, ∞	**30**	**25.7± 0.6**	30	39.8± 0.3	29	1375.1±89.9	0		0	
	2, 1, 1	**30**	**104.5± 23.2**	**30**	**121.8± 26.7**	23	670.6±173.7	26	576.1±181.4	1	1799.7	1.0, 1	**30**	**208.1± 6.8**	30	226.0± 5.4	0		0		0	
	2, 1, 2	**20**	**1321.5±161.1**	3	1344.2±341.5	1	1799.6	1	1799.1	0		1.0, ∞	**30**	**25.6± 0.6**	30	39.6± 0.3	30	1298.1±55.3	0		0	
	2, 2, 1	**28**	**569.2±131.7**	23	890.5±164.2	5	1608.9±229.1	5	1444.4±352.4	0		4, 4, 2	**30**	**1.6± 0.1**	**30**	**1.8± 0.1**	30	4.5± 0.1	30	24.2± 0.7	30	24.3± 0.8
ExBW (id,N,c)	1, 5, .1	30	8.6± 0.1	**30**	**1.6**	30	97.2± 1.0	30	7.9± 0.1	30	626.6± 18.8	4, 4, 4	**30**	**2.4**	30	3.2	30	8.2± 0.5	30	43.0± 1.0	30	36.1± 2.0
	2, 5, .07	30	40.4± 0.3	**30**	**18.7± 0.1**	0		30	732.9± 30.6	0		CTW (n,d,c) 4, 4, 6	**30**	**3.1**	30	4.6± 0.1	30	10.5± 0.4	30	56.0± 2.4	30	42.5± 1.5
	3, 6, .91	30	173.5± 5.1	**30**	**7.5± 0.1**	0		30	10.9± 0.2	0		4, 5, 2	**30**	**3.9± 0.2**	**30**	**4.2± 0.3**	30	18.7± 0.8	30	119.1± 5.5	30	115.3± 6.3
	4, 6, .16	30	161.2± 1.4	**30**	**63.7± 0.5**	0		27	1462.2± 79.6	0		4, 5, 4	**30**	**5.9± 0.1**	30	7.5± 0.1	30	32.4± 1.0	30	239.5± 9.0	30	157.0± 5.3
	5, 7, .01	30	30.7± 0.2	**30**	**25.8± 0.2**	30	82.7± 1.5	30	38.5± 0.6	30	451.0± 14.5	4, 5, 6	**30**	**7.3± 0.3**	30	10.3± 0.4	30	41.7± 1.0	30	310.6±14.7	30	194.5± 7.1
	6, 8, .3	30	300.4± 10.9	**30**	**91.6± 0.6**	0		0		0		4, 6, 2	**30**	**9.3± 0.5**	**30**	**10.1± 0.6**	30	90.9± 4.7	30	840.2±47.1	30	937.7± 45.7
	7, 8, .5	30	0.3	**30**	**0.2**	30	2.2± 0.1	30	0.9± 0.1	30	12.1± 0.4	4, 6, 4	**30**	**13.9± 0.3**	30	17.8± 0.3	30	187.0± 5.9	13	1656.6±76.8	30	1281.6± 64.0
	8, 8, .63	**30**	**35.1± 0.5**	**30**	**34.8± 0.4**	6	1799.2± 0.1	13	1691.9± 45.5	0		4, 6, 6	**30**	**18.2± 0.4**	30	24.9± 0.7	30	225.1±10.0	4	1424.3±88.9	30	1238.9± 81.6
	9, 8, .4	30	177.9± 1.4	**30**	**14.9± 0.1**	0		30	202.3± 7.5	0													

Table 1. For the benchmark problems, we show each planner and heuristic's coverage (out of 30) and over the converged runs the mean runtime (secs) with 95% C.I. For each problem, the fastest planner and heuristic pairs are in bold.

solve. This contrasts i-dual and i^2-dual, which are not affected by problem structure but scale worse with the larger search spaces as the second elevator is introduced.

PARC. i^2-dual dominates when the constraint over failure is tighter ($f \leq 0.2$) and CARL dominates when the constraint is looser ($f \geq 0.4$). CARL struggles with the tight constraints because its heuristic fails to take secondary-cost constraints into account. This is supported by the result that CARL becomes competitive with i^2-dual on the tightly constrained problems when used with C-ROC, a heuristic that takes secondary-cost constraints into account (see appendix F).[2] I^2-dual has the opposite weakness: its heuristic is very informative for highly constrained problems and it prunes a lot of the search space but, as the constraints become looser, its heuristic becomes less effective. On problems with the loosest constraints, namely $(0.6, 1)$, $(0.8, 1)$, and $(1.0, 1)$, CARL completely dominates by achieving full coverage while the baselines obtain zero coverage.

What makes CARL so effective on our benchmarks? The part of CARL that takes the most time is solving $\mathbb{S}(\boldsymbol{\lambda})$s and, over the solved instances, CARL only solves a mean and maximum of 6.7 and 30 $\mathbb{S}(\boldsymbol{\lambda})$s in its search. We identify two domain features that make coordinate search efficient and let CARL solve so few subproblems: single secondary-cost constraint and sparse $\boldsymbol{\lambda}$. In the former, coordinate search only has to solve a single coordinate and it is guaranteed to converge to $\boldsymbol{\lambda}^*$. SAR, ExBW, and PARC problems with $u = \infty$ have a single constraint. It is convenient if $\boldsymbol{\lambda}^* = 0$ because $\boldsymbol{\lambda} = 0$ is the initial guess and coordinate search then only has to verify that its candidate is optimal. CTW has $\boldsymbol{\lambda}^* = 0$ for almost all its problems. More generally, coordinate search tends to work well when $\boldsymbol{\lambda}^*$ is sparse because it is inexpensive to determine these, but this trend is not guaranteed and depends on the search space structure. Over the solved instances, $\boldsymbol{\lambda}^*$ is quite sparse having a mean and maximum of 0.62 and 2 non-zero entries. Note that if $C_i(\pi^*) < u_i$, i.e., a secondary-cost constraint is satisfied with some slack, then $\lambda_i^* = 0$ by complementary slackness, which suggests that many problems will have a sparse $\boldsymbol{\lambda}^*$. In contrast, we have seen that particular problems are structured such that coordinate search fails to find $\boldsymbol{\lambda}^*$, e.g., some Elev instances with a single elevator. Fortunately, these seem to appear infrequently. We emphasise that these benchmarks were taken from the existing literature on CSSPs, and were not hand-picked to accommodate CARL. This may suggest that problems are more often amenable to CARL than not.

[2] CARL (C-ROC) is not in the main results because C-ROC is inadmissible for $\mathbb{S}(\boldsymbol{\lambda})$s, so the algorithm has no optimality guarantee.

6 Conclusion and Future Work

We introduced CARL, a novel algorithm for finding optimal, potentially stochastic, policies for CSSPs. It works by solving a sequence of scalarised SSPs, called $\mathbb{S}(\boldsymbol{\lambda})$s, with efficient heuristic search algorithms for SSPs, and combining the optimal policies of the final $\mathbb{S}(\boldsymbol{\lambda})$ solved into an optimal policy for the original CSSP. CARL is the first heuristic search algorithm over the primal space for finding optimal CSSP policies and, although CARL follows the Lagrangian ascent framework from OR, this paper presents novel methods for solving $\mathbb{S}(\boldsymbol{\lambda})$s efficiently, adapting heuristics to $\mathbb{S}(\boldsymbol{\lambda})$s and extracting stochastic policies. CARL outperforms the state-of-the-art algorithms on nearly all our benchmark problems, solving 50% more instances and, for some problems, it solves all instances while the state-of-the-art solve none. Over the problems where all algorithms solved all problems, CARL offers an average $10\times$ speedup.

In terms of future work, we believe more investigation is required into optimisation procedures for finding $\boldsymbol{\lambda}^*$. CARL uses coordinate search, which works well and is significantly faster than the plain subgradient method, but it is unsatisfying that it can not guarantee an optimal solution and it may not scale well with many secondary-cost constraints. We also believe it is important to better understand the problems themselves and what features make them more or less amenable to CARL's search procedure.

7 Acknowledgements

This research project was undertaken with the assistance of resources and services from the National Computational Infrastructure (NCI), which is supported by the Australian Government. Johannes Schmalz received funding from DFG grant 389792660 as part of TRR 248 (see https://www.perspicuous-computing.science).

A CSSP Example

Here, we present an example of a CSSP.

Getting to Work. The agent needs to make its way from home to work. It can run, use a taxi, or walk to the train station where it can try to take the train. The train is cancelled with 50% probability. Each action has a cost vector $[t\ p\ e]$ in terms of time (t), price (p), and personal effort (e). The agent's task is to get to work in minimal time s.t. price ≤ 15 and effort ≤ 10 over expectation. The CSSP is shown in fig. 1. This problem has three proper deterministic policies:

- $\pi_{\text{run}} = \{s_I \mapsto \text{run}\}$,
- $\pi_{\text{taxi}} = \{s_I \mapsto \text{taxi}\}$,
- $\pi_{\text{train}} = \{s_I \mapsto \text{walk}, s_1 \mapsto \text{train}, s_2 \mapsto \text{train}\}$.

This problem has a unique optimal policy π^* which is stochastic with $\pi^*(s_I, \text{run}) = 0.5$ and $\pi^*(s_I, \text{taxi}) = 0.5$ where

- $C_0(\pi^*) = 1$ (time),
- $C_1(\pi^*) = 15 \leq 15$ (price),
- $C_2(\pi^*) = 10 \leq 10$ (effort).

Note that π^* can be expressed as $\pi^* = 0.5\pi_{\text{run}} + 0.5\pi_{\text{taxi}}$.

B Visualisation of $L(\boldsymbol{\lambda})$

In this section, we visualise the surface of $L(\boldsymbol{\lambda})$ for two different CSSPs.

Getting to Work. First, we consider the "Getting to Work" example from appendix A. Fig. 2 shows this problem's $L(\boldsymbol{\lambda})$ as $\boldsymbol{\lambda}$ varies. Each linear surface of the plot corresponds to one of the deterministic policies; we have labelled the linear surfaces associated with π_{run} and π_{train} with "run" and "train" respectively, and π_{taxi} is obscured. A linear surface indicates that the corresponding policy is optimal for the $\mathbb{S}(\boldsymbol{\lambda})$ at the relevant values of $\boldsymbol{\lambda}$; edges and vertices indicate that multiple deterministic policies are optimal at the $\mathbb{S}(\boldsymbol{\lambda})$. In particular, note that at $\boldsymbol{\lambda}^* = 0$ we have an edge between π_{run} and π_{taxi}, and π_{train} does not intersect there; this indicates that the optimal (in this case stochastic) policy consists of π_{run} and π_{taxi}, which we know to be true. This example is not interesting in terms of coordinate search, because $\boldsymbol{\lambda}^* = 0$ is found in the first step.

Interesting CSSP for Coordinate Search. Consider the CSSP in fig. 3, with $u_1 = u_2 = 15$. Its optimal policy π^* is stochastic with

- $\pi^*(s_I, a_2) = 1$
- $\pi^*(s_1, a_4) = \frac{1}{4} \quad \pi^*(s_1, a_5) = \frac{3}{4}$

and it has $C_0(\pi^*) = 4$ and $C_1(\pi^*) = C_2(\pi^*) = 15$. Fig. 4 shows this CSSP's $L(\boldsymbol{\lambda})$ as $\boldsymbol{\lambda}$ varies. As before, each linear surface corresponds to a deterministic policy that is optimal for the relevant $\mathbb{S}(\boldsymbol{\lambda})$. We visualise the values of $\boldsymbol{\lambda}$ considered by coordinate search with the red lines across the surface, i.e., the line starts at the initial value of $\boldsymbol{\lambda} = 0$, then moves to $\boldsymbol{\lambda} = [0.025\ 0]$, then $\boldsymbol{\lambda} = [0.025\ 0.025]$, and so on, eventually converging to $\boldsymbol{\lambda}^* = [0.2\ 0.2]$. Recall that coordinate search only changes a single coordinate per step, which gives the red lines' "stair-case" shape.

C Pathological Example for Coordinate Search

There are non-smooth problems where coordinate search fails. For example, consider the CSSP in fig. 5 with two secondary-cost constraints $u_1 = u_2 = 1$, i.e., a policy π is feasible if $C_1(\pi) \leq 1$ and $C_2(\pi) \leq 1$. The CSSP has three actions that lead deterministically from s_I to the goal with costs

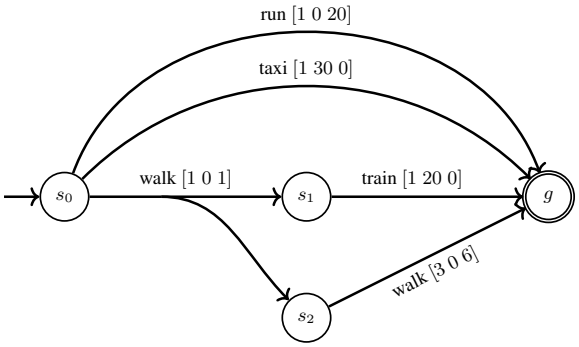

Figure 1. The CSSP associated with the "Getting to Work" example.

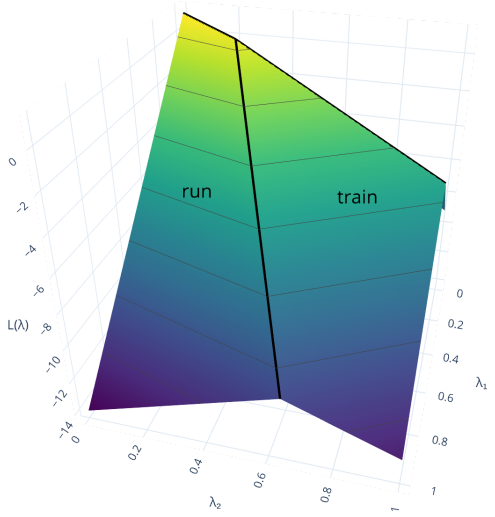

Figure 2. $L(\boldsymbol{\lambda})$ for the "Getting To Work" example.

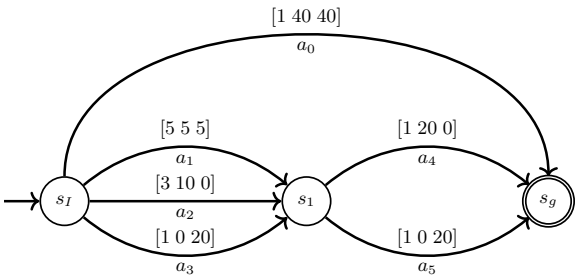

Figure 3. A CSSP that is interesting for coordinate search. It has $u_1 = u_2 = 15$.

- $\boldsymbol{C}(a_0) = [10\ 1\ 1]$,
- $\boldsymbol{C}(a_1) = [1\ 11\ 0]$, and
- $\boldsymbol{C}(a_2) = [1\ 0\ 11]$,

inducing the three deterministic policies π_0, π_1, π_2 with $\pi_i(s_I) = a_i$ for $i \in \{0, 1, 2\}$. Taking into account the terminal costs, the policy costs are

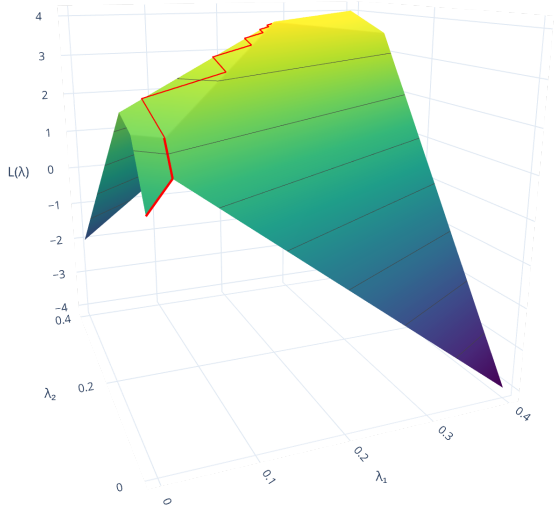

Figure 4. $L(\boldsymbol{\lambda})$ for the "Interesting CSSP for Coordinate Search" example.

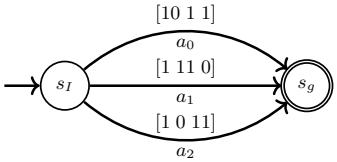

Figure 5. A pathological CSSP where coordinate search fails to find $\boldsymbol{\lambda}^*$. It has $u_1 = u_2 = 1$.

- $C_{\boldsymbol{\lambda}}(\pi_0) = 10 + \lambda_1(1 - u_1) + \lambda_2(1 - u_2) = 10$,
- $C_{\boldsymbol{\lambda}}(\pi_1) = 1 + \lambda_1(11 - u_1) + \lambda_2(0 - u_2) = 1 + 10\lambda_1 - \lambda_2$, and
- $C_{\boldsymbol{\lambda}}(\pi_2) = 1 + \lambda_1(0 - u_1) + \lambda_2(11 - u_2) = 1 - \lambda_1 + 10\lambda_2$.

$L(\boldsymbol{\lambda})$ selects the cheapest one of these, i.e., $L(\boldsymbol{\lambda}) = \min\{10, 1 + 10\lambda_1 - \lambda_2, 1 - \lambda_1 + 10\lambda_2\}$. This surface is shown in fig. 6; note there are three linear surfaces corresponding to the three deterministic policies (the flat one is π_0, and the sloped ones are π_1 and π_2). If we consider $\boldsymbol{\lambda} = 0$ as a starting point then $L([0\ 0]) = 1$ and we can see that there is no way to get a larger $L(\boldsymbol{\lambda})$ by moving in a single coordinate. Algebraically,

$$L([x\ 0]) = \min\{10, 1 + 10x, 1 - x\} = 1 - x < 1$$

and symmetrically

$$L([0\ x]) = \min\{10, 1 - x, 1 + 10x\} = 1 - x < 1$$

for all $x \in \mathbb{R}_{>0}$. Thus, coordinate search gets stuck at $\boldsymbol{\lambda} = [0\ 0]$ with $L(\boldsymbol{\lambda}) = 1$ and fails to progress to an optimal solution such as $\boldsymbol{\lambda}^* = [2\ 2]$ with $L(\boldsymbol{\lambda}^*) = 10$.

D Bellman Backups over Vector Value Function

Instead of using a scalar value function $V : \mathsf{S} \to \mathbb{R}_{\geq 0}$, we use a vector value function $\boldsymbol{V} : \mathsf{S} \to \mathbb{R}^{n+1}_{\geq 0}$, with entries for each cost function. This modification still solves the $\mathbb{S}(\boldsymbol{\lambda})$ SSP by projecting

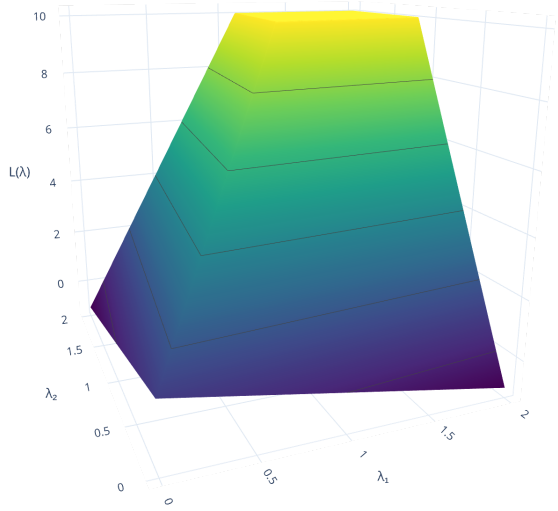

Figure 6. Surface of $L(\boldsymbol{\lambda})$ for the pathological CSSP where coordinate search fails to find $\boldsymbol{\lambda}^*$.

the vector value function onto $V_{\boldsymbol{\lambda}}(s) = [1\ \boldsymbol{\lambda}] \cdot \boldsymbol{V}(s)$, but we now apply Bellman backups on $\boldsymbol{V}$ directly. We write $\boldsymbol{Q}(s, a) = \boldsymbol{C}(a) + \sum_{s' \in \mathsf{S}} P(s'|s, a)\boldsymbol{V}(s')$, then the λ modified Bellman backups are given as follows:

Definition 2 (λ Bellman backup)**.** *Given the vector value function $\boldsymbol{V} : \mathsf{S} \to \mathbb{R}^{n+1}_{\geq 0}$ and a scalarisation $\boldsymbol{\lambda} \in \mathbb{R}^n_{\geq 0}$, a vector Bellman backup applied to state s sets*

$$\boldsymbol{V}(s) \leftarrow \boldsymbol{Q}(s, a_{\min}) \quad \textit{for} \quad a_{\min} \in \operatorname*{argmin}_{a \in \mathsf{A}(s)} \Big([1\ \boldsymbol{\lambda}] \cdot \boldsymbol{Q}(s, a)\Big).$$

By applying λ Bellman backups, we can solve $\mathbb{S}(\boldsymbol{\lambda})$ and evaluate its policy for each cost function $C_i(\pi^*_{\boldsymbol{\lambda}})$ at the same time. This idea was used by Hong and Williams [22].

The λ Bellman backups introduce two technical issues. First, to ensure that each $C_i(\pi^*_{\boldsymbol{\lambda}})$ is evaluated accurately, we must redefine the Bellman residual so that each value function converges, otherwise the backups may converge to an optimal vector value function for the $\mathbb{S}(\boldsymbol{\lambda})$ without the scalar value functions converging. We can do this by considering $\max_{i=0}^{n} |V_i(s) - Q(s, a_{\min})|$ for s's $a_{\min}$. The second technical issue follows from the redefined Bellman residual, because it is possible to get stuck in cycles where actions with similar scalarised Q-values change individual value functions significantly, so that the new residual never goes to 0. To address this, we need to use tie-breaking rules, e.g., if two actions have costs within ϵ of each other, iterate through the value function indices and pick the action that has a smaller Q-value first.

E Strong ϵ-consistency vs ϵ-consistency

For unconstrained SSPs, if a value function V is ϵ-consistent [7], then it is guaranteed to induce a deterministic optimal policy (as $\epsilon \to 0$); however, it may not induce all optimal deterministic policies. To produce all optimal deterministic policies, we require strong

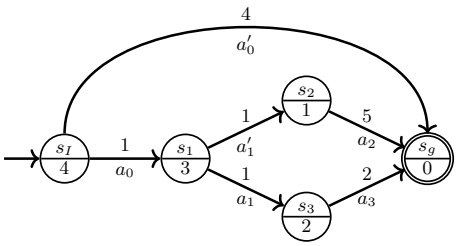

Figure 7. An SSP with a value function that is ϵ-consistent but not strongly ϵ-consistent, and fails to describe all optimal policies.

ϵ-consistency. We give an example where ϵ-consistency fails to capture all optimal deterministic policies: consider the SSP in fig. 7, with the value function V specified in the bottom of each node. The SSP has two optimal policies:

- $\pi^*_0 = \{s_I \mapsto a'_0\}$
- $\pi^*_1 = \{s_I \mapsto a_0, s_1 \mapsto a_1, s_3 \mapsto a_3\}$.

V has two greedy policies: the optimal π^*_0 and a suboptimal policy

- $\pi' = \{s_I \mapsto a_0, s_1 \mapsto a'_1, s_2 \mapsto a_2\}$.

V is ϵ-consistent w.r.t. the greedy policy π^*_0, and V is admissible; so, V could be produced by an optimal SSP algorithm such as CG-iLAO* [26] (which is what we use to solve our $\mathbb{S}(\boldsymbol{\lambda})$ SSPs). Importantly, V is not ϵ-consistent w.r.t. the other greedy policy π' because $\textsc{res}(s_2) = |1 - 5| > \epsilon$ (assume small ϵ such as 0.0001), and therefore V is not strongly ϵ-consistent. This has the effect that V fails to capture the second optimal policy π^*_1, i.e., π^*_1 is not a greedy policy w.r.t. V.

F More Results

We present an extended table of results in tab. 2. It has the same entries as tab. 1, and additionally includes CARL with ideal-point heuristics and C-ROC. Recall that C-ROC is not admissible for CARL's $\mathbb{S}(\boldsymbol{\lambda})$s, so it provides no guarantee of optimality nor convergence. Nevertheless, in our experiments, wherever CARL(C-ROC) has a coverage entry, it found the optimal solution.

References

[1] E. Altman. Constrained Markov Decision Processes. Technical report, INRIA, 1995. URL https://inria.hal.science/inria-00074109.

[2] P. Baumgartner, S. Thiébaux, and F. Trevizan. Heuristic Search Planning With Multi-Objective Probabilistic LTL Constraints. In *Proc. of Int. Conf. on Principles of Knowledge Representation and Reasoning (KR)*, pages 415–424, 2018. URL https://cdn.aaai.org/ocs/18039/18039-78663-1-PB.pdf.

[3] R. Bellman. *Dynamic programming*. Princeton University Press, 1957.

[4] D. Bertsekas and J. Tsitsiklis. An analysis of stochastic shortest path problems. *Mathematics of Operations Research*, pages 580–595, 1991. doi: 10.1287/moor.16.3.580.

[5] D. Bertsimas and J. Tsitsiklis. *Introduction to Linear Optimization*. Athena Scientific, 1997.

[6] B. Bonet and H. Geffner. Planning as heuristic search: New results. In *Recent Advances in AI Planning*, pages 360–372, 2000. doi: 10.1007/10720246_28.

[7] B. Bonet and H. Geffner. Faster heuristic search algorithms for planning with uncertainty and full feedback. In *Proc. of Int. Conf. on Automated Planning and Scheduling (ICAPS)*, pages 1233–1238, 2003. URL https://www.dtic.upf.edu/~hgeffner/html/reports/hdp.pdf.

[8] B. Bonet and H. Geffner. Labeled RTDP: Improving the Convergence of Real-Time Dynamic Programming. In *Proc. of Int. Conf. on Automated Planning and Scheduling (ICAPS)*, pages 12–21, 2003. URL https://dl.acm.org/doi/abs/10.5555/3036969.3036972.

[9] B. Bonet and R. Givan. 5th International Planning Competition: Non-deterministic track. call for participation, 2006. URL https://www.researchgate.net/publication/228724778_5th_International_Planning_Competition_Non-deterministic_track_call_for_participation.

[10] D. Bryce and O. Buffet. International Planning Competition Uncertainty Part: Benchmarks and Results, 2008. URL https://ipc08.icaps-conference.org/probabilistic/wiki/images/0/03/Results.pdf.

[11] D. Chen, F. Trevizan, and S. Thiébaux. Heuristic Search for Multi-Objective Probabilistic Planning. In *Proc. of AAAI Conf. on Artificial Intelligence*, pages 11945–11954, 2023. doi: 10.1609/aaai.v37i10.26409.

[12] M. Ehrgott. *Multicriteria Optimization*. Springer London, Limited, 2006. doi: 10.1007/3-540-27659-9.

[13] V. Forejt, M. Kwiatkowska, and D. Parker. Pareto Curves for Probabilistic Model Checking. In *Automated Technology for Verification and Analysis*, pages 317–332, 2012. doi: 10.1007/978-3-642-33386-6_25.

[14] F. Geißer, G. Povéda, F. Trevizan, M. Bondouy, F. Teichteil-Königsbuch, and S. Thiébaux. Optimal and heuristic approaches for constrained flight planning under weather uncertainty. *Proc. of Int. Conf. on Automated Planning and Scheduling (ICAPS)*, pages 384–393, 2020. doi: 10.1609/icaps.v30i1.6684.

[15] F. Geißer, P. Haslum, S. Thiébaux, and F. Trevizan. Admissible heuristics for multi-objective planning. *Proc. of Int. Conf. on Automated Planning and Scheduling (ICAPS)*, pages 100–109, 2022. doi: 10.1609/icaps.v32i1.19790.

[16] G. Y. Handler and I. Zang. A dual algorithm for the constrained shortest path problem. *Networks*, pages 293–309, 1980. doi: 10.1002/net.3230100403.

[17] E. Hansen and I. Abdoulahi. Efficient bounds in heuristic search algorithms for stochastic shortest path problems. *Proc. of AAAI Conf. on Artificial Intelligence*, pages 3283–3290, 2015. doi: 10.1609/aaai.v29i1.9643.

[18] E. Hansen and S. Zilberstein. LAO*: A heuristic search algorithm that finds solutions with loops. *Artificial Intelligence*, pages 35–62, 2001. doi: 10.1016/S0004-3702(01)00106-0.

[19] E. A. Hansen. Error bounds for stochastic shortest path problems. *Mathematical Methods of Operations Research*, pages 1–27, 2017. doi: 10.1007/s00186-017-0581-5.

[20] P. Hart, N. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, pages 100–107, 1968. doi: 10.1109/tssc.1968.300136.

[21] M. Helmert and C. Domshlak. Landmarks, Critical Paths and Abstractions: What's the Difference Anyway? *Proc. of Int. Conf. on Automated Planning and Scheduling (ICAPS)*, pages 162–169, 2009. doi: 10.1609/icaps.v19i1.13370.

[22] S. Hong and B. C. Williams. An anytime algorithm for constrained stochastic shortest path problems with deterministic policies. *Artificial Intelligence*, page 103846, 2023. doi: 10.1016/j.artint.2022.103846.

[23] J. Lee, G.-H. Kim, P. Poupart, and K.-E. Kim. Monte-Carlo Tree Search for Constrained POMDPs. In *Advances in Neural Information Processing Systems (NIPS)*, 2018. URL https://proceedings.neurips.cc/paper_files/paper/2018/file/54c3d58c5efcf59ddeb7486b7061ea5a-Paper.pdf.

[24] I. Little and S. Thiebaux. Probabilistic planning vs. replanning. In *ICAPS Workshop on IPC: Past, Present and Future*, 2007. URL http://users.cecs.anu.edu.au/~iain/icaps07.pdf.

[25] Mausam and A. Kolobov. *Planning with Markov Decision Processes: An AI Perspective*. Springer International Publishing, 2012. doi: 10.1007/978-3-031-01559-5.

[26] J. Schmalz and F. Trevizan. Efficient constraint generation for stochastic shortest path problems. *Proc. of AAAI Conf. on Artificial Intelligence*, pages 20247–20255, 2024. doi: 10.1609/aaai.v38i18.30005.

[27] J. Schmalz and F. Trevizan. Finding optimal deterministic policies for constrained stochastic shortest path problems. In *Proc. of European Conf. on Artificial Intelligence (ECAI)*, pages 4148–4156, 2024. doi: 10.3233/FAIA240986.

[28] J. Schmalz and F. Trevizan. Code and Benchmarks for ECAI 2025 paper "Solving Constrained Stochastic Shortest Path Problems with Scalarisation", 2025. doi: 10.5281/zenodo.16871056.

[29] H.-J. M. Shi, S. Tu, Y. Xu, and W. Yin. A primer on coordinate descent algorithms, 2017.

[30] N. Z. Shor. *Minimization Methods for Non-Differentiable Functions*. Springer Berlin Heidelberg, 1985. doi: 10.1007/978-3-642-82118-9.

[31] F. Trevizan, S. Thiébaux, P. Santana, and B. Williams. Heuristic Search in Dual Space for Constrained Stochastic Shortest Path Problems. In *Proc. of Int. Conf. on Automated Planning and Scheduling (ICAPS)*, pages 326–334, 2016. doi: 10.1609/icaps.v26i1.13768.

[32] F. Trevizan, S. Thiébaux, and P. Haslum. Occupation Measure Heuris-

		CARL														i-dual				i^2-dual		
		λ-ROC			IP-ROC			C-ROC			λ-LMcut			IP-LMcut			C-ROC			IP-LMcut			
SAR (n, d, r)	(4, 3, 0.75)	**30**	**5.6±**	**1.4**	30	11.5±	2.3	30	12.1±	2.4	**30**	**4.3±**	**1.0**	30	9.2±	1.8	30	104.1±	51.0	30	45.8±	21.1	30	69.2±	56.4
	(4, 4, 0.75)	**30**	**41.9±**	**5.3**	30	75.5±	7.6	30	79.5±	8.3	**30**	**34.1±**	**4.2**	30	65.3±	7.6	5	1345.7±	534.1	9	979.8±	296.6	8	495.5±	417.8
	(5, 3, 0.50)	**30**	**4.9±**	**0.9**	30	9.0±	1.6	30	9.3±	1.7	**30**	**4.8±**	**1.1**	30	10.0±	2.4	30	25.1±	8.5	30	13.1±	4.6	30	11.1±	4.0
	(5, 3, 0.75)	**30**	**12.3±**	**2.5**	30	26.2±	5.8	30	27.9±	6.0	**30**	**13.3±**	**2.9**	30	33.1±	9.0	30	253.8±	110.6	30	113.3±	46.0	30	168.0±	105.9
	(5, 4, 0.50)	**30**	**30.8±**	**6.8**	**30**	**52.3±**	**9.5**	**30**	**54.7±**	**9.8**	**30**	**36.2±**	**7.9**	**30**	**70.5±**	**12.9**	17	695.0±	268.5	24	550.6±	166.1	27	637.8±	213.1
	(5, 4, 0.75)	**30**	**126.6±**	**45.2**	**30**	**270.0±**	**80.7**	**30**	**282.9±**	**83.9**	**30**	**142.5±**	**49.3**	**30**	**353.9±**	**105.7**	6	930.9±	369.3	6	499.2±	226.4	8	349.1±	288.1
Elev (e,w,h)	(1, 1, 1)	30	1.6±	0.6	30	2.0±	0.4	30	2.1±	0.3	30	1.2±	0.3	30	1.4±	0.3	30	1.5±	0.1	**30**	**0.8±**	**0.1**	30	6.2±	0.6
	(1, 1, 2)	29	50.9±	26.8	29	42.7±	12.6	29	42.2±	11.9	29	35.0±	12.7	29	44.4±	14.1	30	25.3±	1.0	**30**	**21.1±**	**2.6**	30	280.0±	46.5
	(1, 2, 1)	25	14.2±	5.6	25	17.7±	3.7	25	17.7±	3.7	25	13.1±	4.2	25	15.8±	4.3	30	13.9±	0.9	**30**	**9.8±**	**1.4**	30	174.2±	38.4
	(1, 2, 2)	22	313.3±	150.2	22	399.8±	99.9	22	407.9±	102.0	22	336.9±	115.8	22	460.1±	118.5	**30**	**340.1±**	**35.7**	**30**	**392.5±**	**72.1**	1	989.2	
	(2, 1, 1)	**30**	**104.5±**	**23.2**	**30**	**131.7±**	**26.2**	**30**	**126.9±**	**25.1**	**30**	**121.8±**	**26.7**	**30**	**149.7±**	**31.5**	23	670.6±	173.7	26	576.1±	181.4	1	1799.7	
	(2, 1, 2)	**20**	**1321.5±**	**161.1**	4	1281.0±	114.2	5	1309.1±	230.8	3	1344.2±	341.5	0			1	1799.6		1	1799.1		0		
	(2, 2, 1)	**28**	**569.2±**	**131.7**	**28**	**830.8±**	**160.8**	**28**	**796.4±**	**158.7**	23	890.5±	164.2	18	1157.4±	180.4	5	1608.9±	229.1	5	1444.4±	352.4	0		
ExBW (id,N,c)	(01, 5, 0.1)	30	8.6±	0.1	30	8.5±	0.1	30	8.5±	0.1	30	1.6		**30**	**1.2**		30	97.2±	1.0	30	7.9±	0.1	30	626.6±	18.8
	(02, 5, 0.07)	30	40.4±	0.3	30	36.0±	0.2	30	35.3±	0.3	30	18.7±	0.1	**30**	**15.2±**	**0.1**	0			30	732.9±	30.6	0		
	(03, 6, 0.91)	30	173.5±	5.1	30	138.0±	1.4	30	135.7±	1.2	30	7.5±	0.1	**30**	**3.3**		0			30	10.9±	0.2	0		
	(04, 6, 0.16)	30	161.2±	1.4	30	142.9±	1.0	30	142.1±	1.7	30	63.7±	0.5	**30**	**45.6±**	**0.3**	0			27	1462.2±	79.6	0		
	(05, 7, 0.01)	30	30.7±	0.2	30	28.1±	0.2	30	27.2±	0.3	30	25.8±	0.2	**30**	**12.8±**	**0.1**	30	82.7±	1.5	30	38.5±	0.6	30	451.0±	14.5
	(06, 8, 0.3)	30	300.4±	10.9	30	276.0±	3.1	30	276.7±	4.1	30	91.6±	0.6	**30**	**48.2±**	**0.4**	0			0			0		
	(07, 8, 0.5)	30	0.3		30	0.5		30	0.5		**30**	**0.2**		**30**	**0.2**		30	2.2±	0.1	30	0.9±	0.1	30	12.1±	0.4
	(08, 8, 0.63)	**30**	**35.1±**	**0.5**	30	40.5±	0.8	30	41.1±	1.0	**30**	**34.8±**	**0.4**	**30**	**35.5±**	**0.7**	6	1799.2±	0.1	13	1691.9±	45.5	0		
	(09, 8, 0.4)	30	177.9±	1.4	30	374.6±	3.3	30	374.2±	5.2	**30**	**14.9±**	**0.1**	30	24.5±	0.1	0			30	202.3±	7.5	0		
PARC (f,u)	(0.0, 1)	30	311.2±	13.6	30	183.7±	6.9	**30**	**1.5±**	**1.4**	30	270.9±	8.3	30	252.5±	8.7	30	126.2±	9.1	30	103.1±	2.4	30	42.8±	3.1
	(0.0, ∞)	30	100.7±	0.8	30	52.1±	0.9	**30**	**0.1**		30	141.3±	1.0	30	64.7±	0.5	30	47.1±	2.5	30	42.1±	1.8	30	29.5±	1.8
	(0.2, 1)	30	600.7±	3.6	30	234.8±	3.1	13	184.2±	3.5	30	863.4±	9.7	30	337.1±	3.8	0			0			**30**	**65.1±**	**11.1**
	(0.2, ∞)	30	121.7±	1.9	30	91.0±	1.3	30	56.9±	0.6	30	185.7±	1.2	30	79.2±	0.6	30	1537.7±	72.8	0			**30**	**41.7±**	**4.9**
	(0.4, 1)	30	882.8±	31.7	**30**	**247.5±**	**3.8**	**30**	**241.8±**	**3.2**	30	564.1±	12.8	30	308.2±	4.0	0			0			3	801.5±	536.2
	(0.4, ∞)	30	130.9±	2.5	30	98.0±	1.8	30	94.0±	1.7	30	188.1±	1.3	**30**	**82.0±**	**1.2**	0			0			3	665.9±	308.3
	(0.6, 1)	30	318.4±	10.3	30	186.0±	3.6	30	148.3±	2.3	30	255.4±	6.9	**30**	**141.0±**	**2.1**	0			0			0		
	(0.6, ∞)	**30**	**25.5±**	**0.5**	30	40.0±	0.9	30	40.4±	0.9	30	39.6±	0.3	30	40.9±	0.3	26	1508.9±	68.9	0			0		
	(0.8, 1)	30	345.5±	7.4	30	185.3±	2.4	**30**	**148.2±**	**2.2**	30	267.1±	8.0	**30**	**148.1±**	**2.2**	0			0			0		
	(0.8, ∞)	**30**	**25.7±**	**0.6**	30	40.4±	1.1	30	40.8±	1.0	30	39.8±	0.3	30	41.3±	0.4	29	1375.1±	89.9	0			0		
	(1.0, 1)	30	208.1±	6.8	30	184.6±	2.2	**30**	**150.3±**	**3.1**	30	226.0±	5.4	**30**	**148.6±**	**1.6**	0			0			0		
	(1.0, ∞)	**30**	**25.6±**	**0.6**	30	40.5±	1.1	30	41.5±	1.1	30	39.6±	0.3	30	41.1±	0.4	30	1298.1±	55.3	0			0		
CTW (n,d,c)	(4, 4, 2)	**30**	**1.6±**	**0.1**	30	2.7±	0.1	30	3.2±	0.1	**30**	**1.8±**	**0.1**	**30**	**1.8±**	**0.1**	30	4.5±	0.1	30	24.2±	0.7	30	24.3±	0.8
	(4, 4, 4)	**30**	**2.4**		30	4.6±	0.1	30	5.4±	0.1	30	3.2		30	3.2		30	8.2±	0.5	30	43.0±	1.0	30	36.1±	2.0
	(4, 4, 6)	**30**	**3.1**		30	6.6±	0.1	30	7.4±	0.1	30	4.6±	0.1	30	4.7±	0.1	30	10.5±	0.4	30	56.0±	2.4	30	42.5±	1.5
	(4, 5, 2)	**30**	**3.9±**	**0.2**	30	6.3±	0.3	30	7.4±	0.3	**30**	**4.2±**	**0.3**	**30**	**4.2±**	**0.2**	30	18.7±	0.8	30	119.1±	5.5	30	115.3±	6.3
	(4, 5, 4)	**30**	**5.9±**	**0.1**	30	11.3±	0.3	30	13.4±	0.6	30	7.5±	0.1	30	7.6±	0.2	30	32.4±	1.0	30	239.5±	9.0	30	157.0±	5.3
	(4, 5, 6)	**30**	**7.3±**	**0.3**	30	15.9±	0.4	30	18.6±	0.5	30	10.3±	0.4	30	11.1±	0.5	30	41.7±	1.0	30	310.6±	14.7	30	194.5±	7.1
	(4, 6, 2)	**30**	**9.3±**	**0.5**	30	14.9±	0.6	30	17.8±	0.8	**30**	**10.1±**	**0.6**	**30**	**9.8±**	**0.6**	30	90.9±	4.7	30	840.2±	47.1	30	937.7±	45.7
	(4, 6, 4)	**30**	**13.9±**	**0.3**	30	27.1±	0.8	30	31.3±	0.9	30	17.8±	0.3	30	18.5±	0.3	30	187.0±	5.9	13	1656.6±	76.8	30	1281.6±	64.0
	(4, 6, 6)	**30**	**18.2±**	**0.4**	30	37.7±	0.9	30	44.7±	0.9	30	24.9±	0.7	30	26.5±	0.6	30	225.1±	10.0	4	1424.3±	88.9	30	1238.9±	81.6

Table 2. For the benchmark problems, we show each planner and heuristic's coverage (out of 30) and over the converged runs the mean runtime (secs) with 95% C.I. The fastest planner and heuristic per problem are in bold.

tics for Probabilistic Planning. In *Proc. of Int. Conf. on Automated Planning and Scheduling (ICAPS)*, pages 306–315, 2017. doi: 10.1609/icaps.v27i1.13840.