Efficient Constraint Generation for Stochastic Shortest Path Problems

Johannes Schmalz, Felipe Trevizan

School of Computing, Australian National University
johannes.schmalz@anu.edu.au, felipe.trevizan@anu.edu.au

Abstract

Current methods for solving Stochastic Shortest Path Problems (SSPs) find states' costs-to-go by applying Bellman backups, where state-of-the-art methods employ heuristics to select states to back up and prune. A fundamental limitation of these algorithms is their need to compute the cost-to-go for every applicable action during each state backup, leading to unnecessary computation for actions identified as sub-optimal. We present new connections between planning and operations research and, using this framework, we address this issue of unnecessary computation by introducing an efficient version of constraint generation for SSPs. This technique allows algorithms to ignore sub-optimal actions and avoid computing their costs-to-go. We also apply our novel technique to iLAO* resulting in a new algorithm, CG-iLAO*. Our experiments show that CG-iLAO* ignores up to 57% of iLAO*'s actions and it solves problems up to $8\times$ and $3\times$ faster than LRTDP and iLAO*.

1 Introduction

Planning is an important facet of AI that gives efficient algorithms for solving current real-world problems. Stochastic Shortest Path problems (SSPs) (Bertsekas and Tsitsiklis 1991) generalise classical (deterministic) planning by introducing actions with probabilistic effects, which lets us model problems where the actions are intrinsically probabilistic. Value Iteration (VI) (Bellman 1957) is a dynamic programming algorithm that forms the basis of optimal algorithms for solving SSPs. VI finds the cost-to-go for each state, which describes the solution of an SSP. A state s's cost-to-go is the minimum expected cost of reaching a goal from s, and similarly a action a's cost-to-go is the minimum after applying a. VI finds the optimal cost-to-go by iteratively applying *Bellman backups*, which update each state's cost-to-go with the minimal outgoing action's cost-to-go.

LRTDP (Bonet and Geffner 2003) and iLAO* (Hansen and Zilberstein 2001), the state-of-the-art algorithms for optimally solving SSPs, build on VI and offer significant speedup by using heuristics to apply Bellman backups only to promising states and pruning states that are deemed too expensive. A shortcoming of such algorithms is that each Bellman backup must consider all applicable actions. For instance, let s and a be a state and an applicable action, even if all successors of a will be pruned because they are too expensive, a Bellman backup on s still computes the Q-value of a, so these algorithms can prune unpromising states but not actions. This issue is compounded because algorithms for SSPs require arbitrarily many Bellman backups on each state s to find the optimal solution, thus wasting time on computing Q-values for such actions many times.

This issue of computing unnecessary Q-values for a state s is addressed by *action elimination* (Bertsekas 1995), which can be implemented in search algorithms to prune useless actions. Action elimination looks for pairs $(a, \widehat{a})$ of applicable actions in a state, such that a lower bound on $\widehat{a}$'s cost-to-go exceeds an upper bound on a's cost-to-go, in which case $\widehat{a}$ is proved to be a useless action and can be pruned. Although domain-independent lower bounds (heuristics) can be computed efficiently, finding an efficient, domain-independent upper bound remains an open question to the best of our knowledge. This gap has limited the use of action elimination in domain-independent planning. In the context of optimal heuristic planning for SSPs, the only algorithm we are aware of that utilises action elimination to prune actions is FTVI (Dai, Weld et al. 2009). Other algorithms, such as BRTDP (McMahan, Likhachev, and Gordon 2005), FRTDP (Smith and Simmons 2006) and VPI-RTDP (Sanner et al. 2009), use upper bounds in conjunction with lower bounds to guide their search. However, unlike FTVI, they do not perform action elimination.

We present a general technique for ignoring actions that does not rely on upper bounds. In contrast to action elimination that incrementally prunes useless actions, our approach initially treats all actions as inactive, i.e., not contributing to the solution. It then iteratively adds actions to the search that are guaranteed to improve the current solution. To develop our approach, we strengthen the connections between planning and operations research by relating heuristic search to *variable* and *constraint generation*. Similar to heuristic search, variable and constraint generation enable the solving of large Linear Programs (LPs) by considering only a subset of variables and constraints. We show that algorithms such as iLAO* implicitly perform constraint generation, albeit in a trivial manner, to the LP encoding of VI. Building on this, we introduce an efficient algorithm for constraint generation for SSPs that leads to inactive actions being ignored. We ap-

ply our approach to iLAO* to get the novel algorithm: CG-iLAO*.

In our experiments, CG-iLAO* solves problems up to $8\times$ and $3\times$ faster than LRTDP and iLAO*, respectively. CG-iLAO* is faster than the others over various problem difficulties: its improvement is apparent from problems that require only 4 minutes to solve, and the improvement gap increases as problems take longer to solve. To explain this, we quantify that CG-iLAO* only considers 43–65% of iLAO*'s total actions thus fewer actions' costs-to-go are computed. Investigating further, we empirically show that CG-iLAO* combines iLAO*'s efficient use of backups and LRTDP's strong pruning capability, thereby displaying the best characteristics of both in a single algorithm.

The structure of this paper is as follows: we first introduce the background for SSPs and existing methods for solving SSPs. Second, we give a brief background to linear programming and connect linear programming techniques to heuristic search. Then we explain and motivate our novel algorithm CG-iLAO* and prove its correctness. Finally, we empirically evaluate the performance of CG-iLAO*.

2 Background

A Stochastic Shortest Path problem (SSP) (Bertsekas and Tsitsiklis 1991) is a tuple $\langle \mathsf{S}, s_0, \mathsf{G}, \mathsf{A}, P, C\rangle$ where: S is a finite set of states; $s_0 \in \mathsf{S}$ is the initial state; $\mathsf{G} \subset \mathsf{S}$ are the goal states with $\mathsf{G} \neq \emptyset$; A is a finite set of actions and $\mathsf{A}(s) \subseteq \mathsf{A}$ denotes the actions applicable in state s; $P(s'|s,a)$ gives the probability that applying action a in state s results in state s'; $C(s,a) \in \mathbb{R}_{>0}$ is the cost of applying a in s.

The states immediately reachable by applying a to s are called successors and are given by $\text{succ}(s,a) \coloneqq \{s' \in \mathsf{S} : P(s'|s,a) > 0\}$. A solution to an SSP is given by a map $\pi : \mathsf{S} \to \mathsf{A}$, called a policy. A policy π is *closed* w.r.t. s_0 if each state s that can be reached by following π from s_0 is either a goal or π is defined for s. A policy is *proper* w.r.t. s_0 if it reaches the goal with probability 1 from s_0 and it is *improper* otherwise. An optimal policy π^* is any proper policy that minimises the expected cost to reach the goal from the initial state s_0.

For simplicity, we make two standard assumptions: (i) there exists at least one proper policy w.r.t. s_0, this is called the reachability assumption; and (ii) all improper policies have infinite expected cost. A consequence of assumption (ii) is that $\mathsf{A}(s) \neq \emptyset$ for all $\mathsf{S} \setminus \mathsf{G}$. Note that we define C to be strictly positive in order to avoid zero-cost cycles that would violate assumption (ii). In our experiments, we relax the reachability assumption by applying the fixed-penalty transformation of SSPs (Trevizan, Teichteil-Königsbuch, and Thiébaux 2017) resulting in a new SSP without dead ends. Other approaches for handling SSPs such as S3P (Teichteil-Königsbuch 2012) are also compatible with our approach.

An SSP's optimal solution is uniquely represented by the optimal value function V^*, which is the unique fixed-point solution to the Bellman equations:

$$V(s) = \begin{cases} 0 & \text{if } s \in \mathsf{G} \\ \min_{a \in \mathsf{A}(s)} Q(s,a) & \text{otherwise} \end{cases} \quad \forall s \in \mathsf{S} \qquad (1)$$

where $Q(s,a) \coloneqq C(s,a) + \sum_{s' \in \mathsf{A}(a)} P(s'|s,a)V(s)$ is known as the Q-value of s and a. A (optimal) value function $V(s)$ and the associated Q-value $Q(s,a)$ represent the (minimum) expected cost to reach the goal from state s and after executing action a on state s, respectively. Given a value function V, the policy associated with it is defined as $\pi_V(s) \coloneqq \operatorname{argmin}_{a \in \mathsf{A}(s)} Q(s,a)$ and is known as the greedy policy for V. Ties can be broken arbitrarily thanks to assumption (ii), and for simplicity we assume some tie-breaking rules that ensure the greedy policy is unique.

The Bellman equations (1) can be iteratively solved by Value Iteration (VI) (Bellman 1957): VI starts with an arbitrary value function V^0 and computes $V^{t+1}(s) \coloneqq \min_{a \in \mathsf{A}(s)} Q(s,a)$ over all states, where $Q(s,a)$ uses the previous value function V^t. This process of computing $V^{t+1}(s)$ using V^t is called a Bellman backup. VI is guaranteed to asymptotically converge to V^* regardless of V^0. For practical reasons, VI is terminated at iteration t when the *Bellman residual* $\text{RES}(s) \coloneqq |V^t(s) - \min_{a \in \mathsf{A}(s)} Q(s,a)|$ is less than or equal to $\epsilon \in \mathbb{R}_{>0}$ for all $s \in \mathsf{S}$.

Given a policy π, its policy envelope $\mathsf{S}^\pi \subseteq \mathsf{S}$ is the set of all reachable states when following π from s_0. Note that VI explores the complete state space S regardless of the optimal policy envelope S^{π^*}'s size. To address this shortcoming, heuristic search algorithms such as iLAO* (Hansen and Zilberstein 2001) and LRTDP (Bonet and Geffner 2003) use a heuristic function H to initialise V^0, which guides the exploration of the state space S in a way that expands as few states as possible. To find V^*, heuristic search algorithms require the heuristic to be *admissible*, that is, $H(s) \leq V^*(s)\ \forall s \in \mathsf{S}$. Often heuristics are also *monotonic*, which immediately implies admissibility. A value function V is monotonic if $V(s) \leq \min_{a \in \mathsf{A}(s)} Q(s,a)\ \forall s \in \mathsf{S}$, and the definition is analogous for H. Similar to VI, heuristic search algorithms converge to V^* asymptotically and require a practical stop criterion. This stop criterion is known as ϵ-consistency (Bonet and Geffner 2003) and is defined as:

Definition 1 (ϵ-consistency)**.** *A value function V is ϵ-consistent if* $\text{RES}(s) \leq \epsilon\ \forall s \in \mathsf{S}^{\pi_V}$.

Notice that ϵ-consistency checks the residual only on the policy envelope of a greedy policy and states outside the envelope are permitted to have a residual larger than ϵ.

We close this section by reviewing iLAO* (Hansen and Zilberstein 2001). iLAO* (alg. 1) is an iterative algorithm which works by incrementally growing a *partial SSP*:[1]

Definition 2 (Partial SSP)**.** *Given an SSP $\langle \mathsf{S}, s_0, \mathsf{G}, \mathsf{A}, P, C\rangle$ and a heuristic H, a partial SSP $\widehat{\mathbb{S}}$ is an SSP with* terminal costs *defined by $\widehat{\mathbb{S}} = \langle \widehat{\mathsf{S}}, s_0, \widehat{\mathsf{G}}, \widehat{\mathsf{A}}, P, C, H\rangle$ with $\widehat{\mathsf{S}} \subseteq \mathsf{S}, \widehat{\mathsf{G}} \subset \widehat{\mathsf{S}}, \mathsf{G} \cap \widehat{\mathsf{S}} \subseteq \widehat{\mathsf{G}}, \widehat{\mathsf{A}}(s) \subseteq \mathsf{A}(s)\ \forall s \in \widehat{\mathsf{S}}$ and terminal cost H.*

SSPs with terminal costs have a one-time terminal cost of $H(\widehat{g})$ that is incurred when $\widehat{g} \in \widehat{\mathsf{G}}$ is reached, so their Bellman equations are (1) with the goal case replaced by $V(\widehat{g}) = H(\widehat{g})$ for all $\widehat{g} \in \widehat{\mathsf{G}}$. It is trivial to see that SSPs with terminal costs are equivalent to SSPs, and we use them to simplify our presentation. In a partial SSP $\widehat{\mathbb{S}}$, we refer to

[1] Called the explicit graph in the original paper.

Algorithm 1: iLAO*

1 **Function** *iLAO** $(\mathbb{S}, H, \epsilon)$
2 $\quad \widehat{\mathbb{S}} \leftarrow$ partial SSP $\langle \{s_0\}, s_0, \{s_0\}, \emptyset, P, C, H \rangle$
3 $\quad V \leftarrow$ Value Function initialised by H
4 $\quad$ **repeat**
5 $\quad\quad \mathcal{E} \leftarrow$ post-order DFS traversal of $\widehat{\pi}_V$ from s_0
6 $\quad\quad \widehat{\mathbb{S}} \leftarrow$ EXPAND-FRINGES$(\mathbb{S}, \widehat{\mathbb{S}}, \mathcal{E})$
7 $\quad\quad \mathsf{F} \leftarrow \mathsf{S}^{\widehat{\pi}_V} \cap (\widehat{\mathsf{G}} \setminus \mathsf{G})$
8 $\quad\quad V, \text{RES}, \widehat{\pi}_{\text{old}} \leftarrow$ BACKUPS$(\widehat{\mathbb{S}}, \mathcal{E}, V, \mathsf{F})$
9 $\quad$ **until** $\mathsf{F} = \emptyset$ *and* $\widehat{\pi}_{old} = \widehat{\pi}_V$ *and* RES $< \epsilon$
10 $\quad$ **return** V
11 **Function** EXPAND-FRINGES$(\mathbb{S}, \widehat{\mathbb{S}}, \mathcal{E})$
12 $\quad$ **for** $s_f \in \mathcal{E} \cap (\widehat{\mathsf{G}} \setminus \mathsf{G})$ **do**
13 $\quad\quad \widehat{\mathsf{G}} \leftarrow \widehat{\mathsf{G}} \setminus \{s_f\}$
14 $\quad\quad$ ADD-ACTIONS$(\mathbb{S}, \widehat{\mathbb{S}}, s_f, \mathsf{A}(s_f))$
15 $\quad$ **return** $\widehat{\mathbb{S}}$
16 **Function** ADD-ACTIONS$(\mathbb{S}, \widehat{\mathbb{S}}, s, \mathsf{A}')$
17 $\quad \widehat{\mathsf{A}}(s) \leftarrow \widehat{\mathsf{A}}(s) \cup \mathsf{A}'$
18 $\quad$ **for** $a \in \mathsf{A}'$ **do**
19 $\quad\quad \widehat{\mathsf{G}} \leftarrow \widehat{\mathsf{G}} \cup (\text{succ}(s,a) \setminus \widehat{\mathsf{S}})$
20 $\quad\quad \widehat{\mathsf{S}} \leftarrow \widehat{\mathsf{S}} \cup \text{succ}(s,a)$
21 **Function** BACKUPS$(\widehat{\mathbb{S}}, \mathcal{E}, V, \mathsf{F})$
22 $\quad \widehat{\pi}_{\text{old}} \leftarrow \widehat{\pi}_V$
23 $\quad$ **repeat**
24 $\quad\quad \text{RES} \leftarrow 0$
25 $\quad\quad$ **for** $s \in \mathcal{E} \setminus \widehat{\mathsf{G}}$ **do**
26 $\quad\quad\quad Q_{\min} \leftarrow \min_{a \in \widehat{\mathsf{A}}(s)} Q(s,a)$
27 $\quad\quad\quad \text{RES} \leftarrow \max(|V(s) - Q_{\min}|, \text{RES})$
28 $\quad\quad\quad V(s) \leftarrow Q_{\min}$
29 $\quad$ **until** RES $< \epsilon$ *or* $\widehat{\pi}_V \neq \widehat{\pi}_{old}$ *or* $\mathsf{F} \neq \emptyset$
30 $\quad$ **return** $V, \text{RES}, \widehat{\pi}_{\text{old}}$

states in $\widehat{\mathsf{G}} \setminus \mathsf{G}$ as artificial goals, and we define $\widehat{\pi}_V$ to be the greedy policy over V restricted to $\widehat{\mathsf{S}} \setminus \widehat{\mathsf{G}}$.

At each iteration, iLAO* expands its partial SSP $\widehat{\mathbb{S}}$ by expanding the artificial goals reachable by $\widehat{\pi}_V$ into regular states. To expand $\widehat{g} \in \widehat{\mathsf{G}} \setminus \mathsf{G}$, iLAO* adds $\widehat{g}$'s applicable actions to $\widehat{\mathbb{S}}$ and adds new reachable states as artificial goals (alg. 1 line 6). This artificial goal expansion is done to make $\widehat{\pi}_V$ eventually closed w.r.t. s_0 for the original SSP $\mathbb{S}$. Simultaneously, iLAO* also works towards making V ϵ-consistent by applying a Bellman backup to all the states reachable by $\widehat{\pi}_V$ (alg. 1 line 8). These Bellman backups are ordered by a post-order DFS traversal of $\widehat{\pi}_V$, so states that occur closer to artificial goals are updated first, and s_0 is updated last. Note that, when a state is expanded, it may have a successor s already within the partial SSP. If this happens, the DFS in alg. 1 line 5 must keep traversing $\widehat{\pi}_V$ from s to ensure the policy envelope $\mathcal{E}$ is accurate.

In the next section, we show how to interpret iLAO* through the lens of Operations Research by relating it to techniques used for handling large linear programs.

3 iLAO* as Linear Program

SSPs can be solved by the Linear Program (LP) presented in LP 1. This LP is known as the primal LP or VI LP since it directly encodes the Bellman equations.

$$\max_{\boldsymbol{\mathcal{V}}} \quad V_{s_0} \qquad \text{(LP 1)}$$

$$\text{s.t.} \quad V_s \leq C(s,a) + \sum_{s' \in \mathsf{S}} P(s'|s,a) V_{s'} \qquad \forall s \in \mathsf{S} \setminus \mathsf{G}, a \in \mathsf{A}(s) \quad \text{(C1)}$$

$$V_g \leq 0 \qquad \forall g \in \mathsf{G} \quad \text{(C2)}$$

Each variable $V_s \in \boldsymbol{\mathcal{V}}$ represents $V(s)$, and for each state $s \in \mathsf{S} \setminus \mathsf{G}$ the relevant constraints C1 encode $V(s) \leq \min_{a \in \mathsf{A}(s)} Q(s,a)$. When clear from context, we use $V(s)$ to represent V_s, and $Q(s,a)$ for the right-hand side of s and a's constraint C1. Together with the objective that maximises $V(s_0)$, we obtain the Bellman equations (1) for the states in the optimal policy envelope $\mathsf{S}^{\pi *}$, i.e., the constraints are active (tight) for the pairs $(s, \pi^*(s))$ for all $s \in \mathsf{S}^{\pi *}$ and inactive (slack) everywhere else. We reframe iLAO*'s incremental growing of its partial SSP as solving LP 1 with *variable and constraint generation* (Bertsimas and Tsitsiklis 1997).[2]

Variable generation is a technique from Operations Research that enables us to solve LPs with a large number of variables by considering only a subset of variables. Given an LP with missing variables called the Reduced Master Problem (RMP), variable generation finds a set of variables outside the RMP whose addition lets the RMP's solution quality improve. Variable generation provides a sound and complete method to select such variables and a stop criterion that ensures the optimal solution for the RMP is also optimal for the original LP. Heuristic search algorithms, such as iLAO*, can be seen as a variable generation algorithm over LP 1, where each of its partial SSPs represents an RMP with the subset of variables $\{V_s : s \in \widehat{\mathsf{S}}\}$. For iLAO*, the variable selection mechanism is inherited from A* and is represented by the expansion of the artificial goals reachable by $\widehat{\pi}_V$ (alg. 1 lines 12 to 15): for all $s_f \in \mathsf{S}^{\widehat{\pi}_V} \cap (\widehat{\mathsf{G}} \setminus \mathsf{G})$ and $a \in \mathsf{A}(s_f)$, we add the variables V_s such that $s \in \text{succ}(s_f, a)$ and $s \notin \widehat{\mathsf{S}}$.

Constraint generation is a similar technique which enables the solving of LPs with a large (potentially infinite) number of constraints. The key idea is that the optimal solution of an LP with many constraints only makes a small number of constraints active, thus only a subset of constraints is needed to characterise this optimal solution. In constraint generation, the intermediate LPs are known as *relaxed LPs* since they relax the original LP by removing one or more constraints. Given a relaxed LP, constraint generation finds one or more constraints in the original LP that are violated by the optimal solution of the relaxed LP. By iteratively adding these violated constraints and re-optimising the new relaxed LP, a sequence of relaxed LPs with an increasing number of constraints is generated. When no violations are found, the optimal solution of the relaxed LP is an optimal solution for the original LP. The algorithm used to check constraint violations is called a separation oracle and the effectiveness of

[2] Variable generation is also known as column generation and constraint generation is also known as the cutting plane method.

constraint generation relies on the availability of an efficient separation oracle for the original LP.

In the case of iLAO*, constraint generation adds all the actions of the expanded artificial goal (alg. 1 line 14) where each action $a \in \mathsf{A}(s_f)$ added to the partial SSP implicitly represents the constraint $V(s_f) \leq Q(s_f, a)$. This separation oracle is computationally cheap since no checks are performed to detect if this new constraint is needed or not, at the cost that inactive constraints are unnecessarily added to the partial SSP. In the next section, we present our new algorithm that uses an efficient separation oracle that only adds violated constraints.

4 CG-iLAO*

In iLAO*, as most search algorithms, each state is either unexpanded or fully expanded, i.e., either none or all of its applicable actions are considered and it is not possible to ignore just a subset of the applicable actions. We start by defining which actions can be safely ignored in a partial SSP.

Definition 3 (Inactive Action)**.** *Consider an SSP $\mathbb{S}$, its partial SSP $\widehat{\mathbb{S}}$, a value function V, and a state $s \in \widehat{\mathsf{S}}$. An action $a \in \mathsf{A}(s) \setminus \widehat{\mathsf{A}}(s)$ is inactive in state s if $V(s) < Q(s, a)$.*

An inactive action $a \in \mathsf{A}(s) \setminus \widehat{\mathsf{A}}(s)$ for $s \in \widehat{\mathsf{S}}$ represents the inactive constraint C1 for s and a in LP 1. Since inactive actions are not in the partial SSP and their associated constraints are inactive, adding them to the partial SSP does not change the solution and only adds overhead in the form of Q-value computation for sub-optimal actions.

We generalise iLAO* by allowing states to be partially expanded, so these states only have a subset of actions available in the partial SSP. Under the lens of linear programming, we use constraint generation to identify and add actions that may be needed to encode the optimal solution and to ignore inactive actions. We call this algorithm Constraint-Generation iLAO* (CG-iLAO*).

CG-iLAO* is presented in alg. 2. One of the defining changes from iLAO* is in CG-iLAO*'s expanding phase (alg. 2 line 6) where PARTLY-EXPAND-FRINGES only expands a state with the greedy actions on V, rather than all the applicable actions. This introduces two challenges: (i) actions that were not added by the partial expansion may need to be added later when V is more accurate; and (ii) when we add such actions, V must be updated to reflect a's availability. If (ii) is not addressed, the reduction to V offered by a is not propagated, potentially leading to a suboptimal solution since V would overestimate V^*. The key insight of our algorithm is that both of these challenges are instances of constraint violation. Thus, we can solve both issues by finding which constraints are violated with a separation oracle and enforcing them in the style of constraint generation.

The trivial separation oracle checks all constraints (s, a) for $s \in \widehat{\mathsf{S}}$ and $a \in \mathsf{A}(s)$ for violations; this is needlessly expensive since some non-violated constraints remain non-violated from one iteration to the next. Our separation oracle exploits this persistence between iterations by tracking changes in V to compute a subset of constraints which could potentially be violated by the following rules (alg. 2 lines 25 to 28 and line 42): suppose $V(s)$ is assigned $\min_{a \in \widehat{\mathsf{A}}(s)} Q(s, a)$, then there are three cases:

1. $\boldsymbol{V(s)}$ **stays the same.** No new constraint violations.
2. $\boldsymbol{V(s)}$ **increases.** The constraints $V(s) \leq Q(s, a')$ may be violated for $(s, a') \in \text{EXT-SUCCS}(s, \mathbb{S}, \widehat{\mathbb{S}}) := \{(s, a) : a \in \mathsf{A}(s) \setminus \widehat{\mathsf{A}}(s)\}$. Note that if (s, a') is already

Algorithm 2: CG-iLAO*

```
1  Function CG-iLAO* (𝕊, H, ε)
2      Ŝ ← partial SSP ⟨{s_0}, s_0, {s_0}, ∅, P, C, H⟩
3      V ← value function initialised by H
4      repeat
5          E ← post-order DFS traversal of π̂_V from s_0
6          Ŝ ← PARTLY-EXPAND-FRINGES(𝕊, Ŝ, E, V)
7          F ← S^{π̂_V} ∩ (Ĝ \ G)
8          V, RES, π̂_old, Γ ← CG-BACKUPS(𝕊, Ŝ, E, V, F, Γ)
9          V, RES, Γ, Ŝ ← FIX-CONSTRS(𝕊, Ŝ, V, Γ, RES)
10     until F = ∅ and π̂_old = π̂_V and RES ≤ ε
11     return V
12 Function PARTLY-EXPAND-FRINGES(𝕊, Ŝ, E, V)
13     for s_f ∈ E ∩ (Ĝ \ G) do
14         Ĝ ← Ĝ \ {s_f}
15         Q_min ← min_{a∈A(s_f)} Q(s_f, a)
16         A' ← {a ∈ A(s_f) | Q(s_f, a) = Q_min}
17         ADD-ACTIONS(𝕊, Ŝ, s_f, A')
18     return Ŝ
19 Function CG-BACKUPS(𝕊, Ŝ, E, V, F, Γ)
20     π̂_old ← π̂_V
21     repeat
22         RES ← 0
23         for s ∈ E \ Ĝ do
24             Q_min ← min_{a∈Â(s)} Q(s, a)
25             if Q_min − V(s) > ε then
26                 Γ ← Γ ∪ EXT-SUCCS(s, 𝕊, Ŝ)
27             else if V(s) − Q_min > ε then
28                 Γ ← Γ ∪ PREDS(s, 𝕊)
29             RES ← max(|V(s) − Q_min|, RES)
30             V(s) ← Q_min
31     until RES ≤ ε or π̂_V ≠ π̂_old or F ≠ ∅
32     return V, RES, π̂_old, Γ
33 Function FIX-CONSTRS(𝕊, Ŝ, V, Γ, RES)
34     Γ' ← ∅
35     for (s, a) ∈ Γ s.t. V(s) > Q(s, a) + ε do
36         if a ∉ Â(s) then
37             Â(s) ← Â(s) ∪ {a}
38             Ĝ ← Ĝ ∪ (succ(s, a) \ Ŝ)
39             Ŝ ← Ŝ ∪ succ(s, a)
40         RES ← max(V(s) − Q(s, a), RES)
41         V(s) ← Q(s, a)
42         Γ' ← Γ' ∪ PREDS(s, 𝕊)
43     return V, RES, Γ', Ŝ
```

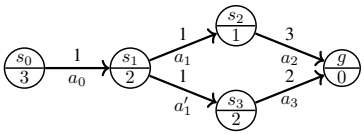

Figure 1: An SSP where CG-iLAO*'s value function is not monotonically non-decreasing.

inside $\widehat{\mathbb{S}}$, i.e., $a' \in \widehat{\mathsf{A}}(s)$, its constraint can not be violated since $V(s) \leftarrow \min_{a \in \widehat{\mathsf{A}}(s)} Q(s,a) \leq Q(s,a')$.

3. ***V*(*s*) decreases.** The only constraints that may be violated are $V(s') \leq Q(s',a')$ for $(s',a') \in \textsc{Preds}(s, \mathbb{S}) \coloneqq \{(s',a') : s \in \text{succ}(s',a'), a' \in \mathsf{A}(s')\}$.

We store potential violations in Γ, i.e., if $V(s)$ increases, the elements of $\textsc{Ext-Succs}(s, \mathbb{S}, \widehat{\mathbb{S}})$ are added to Γ (alg. 2 line 26); and if $V(s)$ decreases, the elements of $\textsc{Preds}(s, \mathbb{S})$ are added to Γ (alg. 2 line 28). Elements are removed from Γ after they are checked. As we prove later in this section, checking constraints in Γ is sufficient to find any constraint violations.

CG-iLAO* fixes a violated constraint (s,a) by setting $V(s) \leftarrow Q(s,a)$ (alg. 2 line 41). This change in V may create a new violation in another state, so we must track such potential violations in the same way as before. This ensures that all constraint violations are tracked and fixed eventually before termination.

Note that $V(s)$ may decrease after an update (case 3 of the constraint violations) in CG-iLAO* even if the heuristic used is monotonic. This is a departure from all other algorithms based on Bellman backups where V is guaranteed to be monotonically non-decreasing during their executions when initialised with a monotonic heuristic. To illustrate a scenario where V decreases in CG-iLAO*, consider the SSP in fig. 1 where H is monotonic and represented inside nodes. The first iterations of CG-iLAO* applied to this SSP are:

Iter. 1 expands s_0.

Iter. 2 partly expands s_1 with a_1.

Iter. 3 expands s_2 with a_2 and, after CG-Backups, we have $V(s_2) = 3$, $V(s_1) = 4$, $V(s_0) = 5$ and $\Gamma = \{(s_1, a'_1)\}$. Since $\Gamma \neq \emptyset$, Fix-Constrs verifies that (s_1, a'_1) is currently better than the existing action a_1 for s_1, so a'_1 is added to $\widehat{\mathsf{A}}(s_1)$ and $V(s_1)$ is changed from 4 to 3. Recall that $V(s_0) = 5$, so when $V(s_1)$ is updated to 3, $\{s_0, a_0\}$ is inserted into Γ, and no further changes are made in this iteration.

Iter. 4 expands s_3 and CG-Backups reduces $V(s_0)$ from 5 to 4, so V has been decreased by CG-Backups.

CG-iLAO* generalises iLAO* by using a more precise separation oracle that only adds violated constraints, which translates to CG-iLAO* ignoring inactive actions. For a state s, any action that has been ignored and left out of the partial SSP $\widehat{\mathbb{S}}$ is not considered by a Bellman backup of s in $\widehat{\mathbb{S}}$, so such actions' Q-values are not computed. However, CG-iLAO* needs to compute additional Q-values in its separation oracle to check violations. As our experiments in section 5 show, the Q-values saved by ignoring inactive actions outweigh the additional Q-values in the separation oracle, which lets CG-iLAO* outperform iLAO*.

To close the section, we prove that CG-iLAO*: (i) terminates (thm. 1); (ii) tracks all constraint violations (lem. 1); and (iii) returns an ϵ-consistent value function (thm. 2). Thus, CG-iLAO* is optimal for SSPs.

Theorem 1. *CG-iLAO* terminates.*

Proof. For contradiction, suppose CG-iLAO* does not terminate. $\mathbb{S}$ is finite and we do not add duplicate states nor constraints, so eventually $\widehat{\mathbb{S}}$ is fixed and $\mathsf{F} = \emptyset$. Then there must be a finite set of states $X \subseteq \widehat{\mathsf{S}}$ that are updated with Bellman backups infinitely often by CG-Backups and/or Fix-Constrs. But X induces a new partial SSP, and applying Bellman backups infinitely often to all X solves this new partial SSP with VI, so V must converge to a fixed point and the residual will be less than ϵ in finite time. Thus, V will not be updated, and all remaining termination conditions will be satisfied, giving us the desired contradiction. □

Lemma 1. *If there is $s \in \widehat{\mathsf{S}} \setminus \widehat{\mathsf{G}}$ and $a \in \mathsf{A}(s)$ such that $V(s) > Q(s,a) + \epsilon$, then $(s,a) \in \Gamma$.*

Proof. We prove by induction over n, the number of updates to V. In the base case, $n = 0$, $\widehat{\mathbb{S}}$ is the initial partial SSP with $\widehat{\mathsf{S}} \setminus \widehat{\mathsf{G}} = \emptyset$, so the claim is vacuously true. Now, we show the claim holds after $n+1$ updates to V, assuming that the claim holds for n updates. Any violations must have been introduced in the latest update to V by CG-Backups or Fix-Constrs, but we add any potential violations to Γ in alg. 2 lines 25 to 28 and line 42 respectively. □

Theorem 2. *CG-iLAO* outputs an ϵ-consistent V.*

Proof. For contradiction, suppose CG-iLAO* has terminated and outputs V with $s \in \mathsf{S}^{\widehat{\pi}_V}$ such that $\textsc{res}(s) > \epsilon$. By CG-iLAO*'s termination condition (alg. 2 line 10) we know that $\widehat{\pi}_V = \widehat{\pi}_{\text{old}}$ and $\mathsf{F} = \emptyset$, so CG-Backups applies Bellman backups to all states in the envelope until $\textsc{res} \leq \epsilon$ (alg. 2 line 31). Therefore, the inconsistency of s must be introduced by Fix-Constrs, either directly by updating V, or indirectly by forcing a policy change. But the residual is tracked (alg. 2 line 40) and policy changes are flagged when $\widehat{\pi}_V \neq \widehat{\pi}_{\text{old}}$, which are both checked in the termination condition. So, Fix-Constrs can not introduce any inconsistency either. But these two methods are the only ones affecting V, which yields the desired contradiction. This proves ϵ-consistency (defn. 1), but previous heuristic search methods rely on the invariant $V \leq V^*$ to safely prune states that can not be part of an optimal policy's envelope, which CG-iLAO* does not have. We must ensure that states s outside the policy envelope with $V(s) > V^*(s)$ can not lead to a cheaper policy if we apply more Bellman backups to them. Consider such s outside the greedy policy envelope with $V(s) > V^*(s)$, and for contradiction let $V(s) > Q(s,a) + \epsilon$ for some $a \in \mathsf{A}(s)$. Since states in $\widehat{\mathsf{G}}$ are initialised with an admissible H, we know that $s \in \widehat{\mathsf{S}} \setminus \widehat{\mathsf{G}}$, so $(s,a) \in \Gamma$ by lem. 1. Since Fix-Constrs overwrites Γ, (s,a) must have been added in the previous

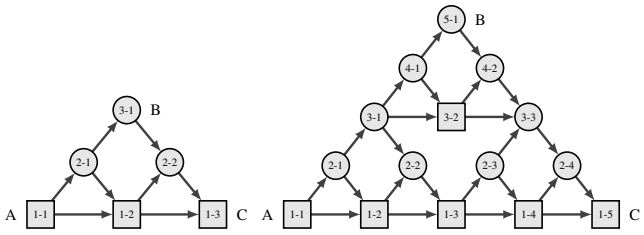

Figure 2: Triangle Tire World problems 1 (left) and 2 (right).

call, but then RES $\leftarrow \max(V(s) - Q(s,a), \text{RES}) > \epsilon$ (alg. 2 line 40), so the termination criteria (alg. 2 line 10) are not satisfied, giving us a contradiction. Therefore, all inadmissible states s satisfy $V(s) \leq Q(s,a) + \epsilon$. Thus, if CG-iLAO* terminates with RES $\leq \epsilon$, additional backups to states with $V(s) > V^*(s)$ would not change V, so we can conclude that CG-iLAO* outputs ϵ-consistent V. □

5 Experiments

In this section we empirically compare CG-iLAO* to two state-of-the-art optimal heuristic search planners: iLAO* (Hansen and Zilberstein 2001) and LRTDP (Bonet and Geffner 2003). We also compared CG-iLAO* against FTVI (Dai, Weld et al. 2009), the only algorithm we are aware of that uses action elimination to prune actions, but it is uncompetitive and FTVI's results are reported in the technical report (Schmalz and Trevizan 2024). We consider the following admissible heuristics: h-max ($h^{\max}$); lm-cut (h^{lmc}) (Helmert and Domshlak 2009); and h-roc (h^{roc}) (Trevizan, Thiébaux, and Haslum 2017). As in (Trevizan, Thiébaux, and Haslum 2017), we use h^{roc} with $h^{\max}$ as a dead-end detection mechanism for problems with dead ends. We use $\epsilon = 0.0001$ and convert SSPs into dead-end free SSPs (Trevizan, Teichteil-Königsbuch, and Thiébaux 2017) with a penalty of $D = 500$ for all domains except *Parc Printer variants* where $D = 10^7$ due to the large cost of single actions.

On all problems, we collected 50 runs with different random seeds for each combination of planner and heuristic. We refer to a problem paired with a fixed seed as an instance. All runs have a cutoff of 30 minutes of CPU time and 8GB of memory. The experiments were conducted in a cluster of Intel Xeon 3.2 GHz CPUs and each run used a single CPU core. The LP solver used for computing h^{roc} was CPLEX version 20.1. We consider the following domains:

Triangle Tire World with Head-start (TW) In the original Triangle Tire World domain (Little, Thiebaux et al. 2007; Buffet 2008), the agent is provided a map of locations in a triangular layout with corners A, B, C, as in fig. 2. The agent's task is to travel from corner A to C, but it gets a flat tyre with probability 0.5 every time it moves. Once the car gets a flat tyre, it must change the tyre if a spare is available, otherwise no action is available and the goal is no longer reachable. The agent can only store one spare at a time, and it can only obtain spare tyres in select locations (circles in fig. 2). A shortcoming of this domain is that its difficulty scales exponentially, so it may be easy to solve problem n and impractical for $n + 1$. For this reason we extend the domain by allowing the agent a head-start, that is, its starting location may be anywhere along the edge AB, for instance, on problem 2 these are locations 1-1, 2-1, …, 5-1 (fig. 2 (right)). Let TW(n, d) denote an instance of Triangle Tire World with Head-start where n is the problem size and d denotes the distance between the agent's starting location and corner B. When $d = 2n$, we obtain the original problem of size n and reducing d makes the problem easier until we reach $d = 1$, the easiest variant. Experiments using LRTDP and h^{roc} suggest the following relation in terms of CPU time: TW$(n+1, 2n-3) \leq$ TW$(n, 2n) \leq$ TW$(n+1, 2n-2)$. Therefore, for each size n, we consider 5 problems: TW$(n, 2n-4), \ldots,$ TW$(n, 2n)$.

Probabilistic Blocks World (BW) (Buffet 2008) As in the deterministic Blocks World from IPC, the agent is tasked with arranging blocks on a table into a particular configuration with actions to pick up blocks, put them down, or stack them. The probabilistic version adds a 0.25 probability to each action that the handled block falls onto the table. Furthermore, actions are added that allow the agent to pick up, put down, and stack a tower of three blocks; these have 0.9 probability of the whole tower falling onto the table.

Exploding Blocks World (ExBW) (Buffet 2008) Another variation for the deterministic Blocks World but this time each block is rigged with an explosive that can detonate once and destroy the table or block immediately underneath it. When a block is placed on the table or another block, it detonates with probability 0.4 and 0.1 respectively. Once the table or a block has been destroyed, the agent can not interact with them anymore; therefore, if they are not in their goal position, the goal will be unreachable.

Probabilistic PARC Printer (PARC) (Trevizan, Thiébaux, and Haslum 2017) This domain is a probabilistic extension of the PARC Printer domain from IPC. It models a modular printer consisting of various components and each page scheduled for printing needs to pass through multiple components in a particular order. The goal is to optimise how each page is directed through the different components to satisfy the printing requirements. With probability 0.1, a component jams ruining the relevant page and forcing it to be reprinted. The domain comes in two flavours: with repair (PARC-R), where jammed components can be repaired and then used again; and without repair (PARC-N), where jammed components remain unusable.

Our code and benchmarks are available at Schmalz and Trevizan (2023). We now present a summary of our findings.

What is the best planner and heuristic combination? A common metric to evaluate planners is *coverage*, i.e., the total number of instances solved in a given amount of time, thus larger coverage is better. Fig. 3 (left) shows the coverage of each combination of planner and heuristic as a function of time. The top three combinations and their total coverages are CG-iLAO$^*_{\text{roc}}$ (1300), LRTDP$_{\text{roc}}$ (1202), and iLAO$^*_{\text{roc}}$ (1200). Note that CG-iLAO$^*_{\text{roc}}$ and LRTDP$_{\text{roc}}$ alternate in the top spot up to 220 seconds and CG-iLAO$^*_{\text{roc}}$ has the best coverage after that until the experiment cutoff.

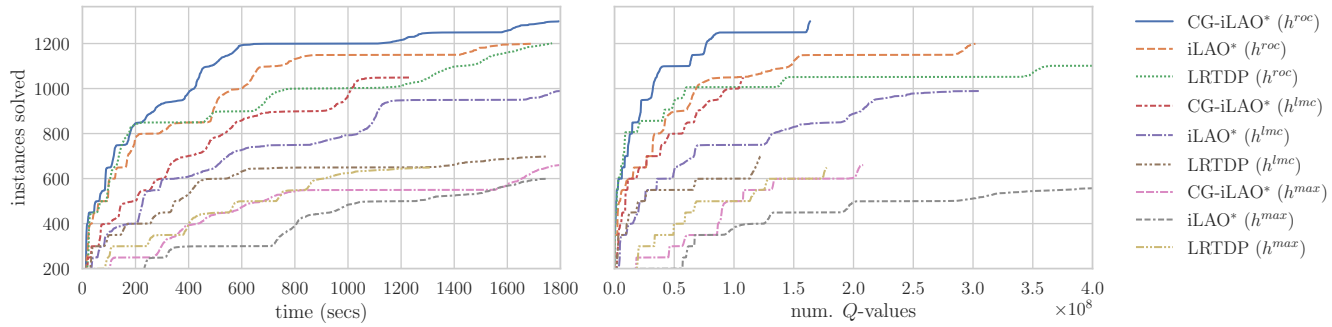

Figure 3: For each algorithm and heuristic, the cumulative plot of how many instances were solved w.r.t. time in seconds (left) and number of Q-values (right). Both plots start at 200 solved instances and (right) is cut off at 4×10^8 Q-values.

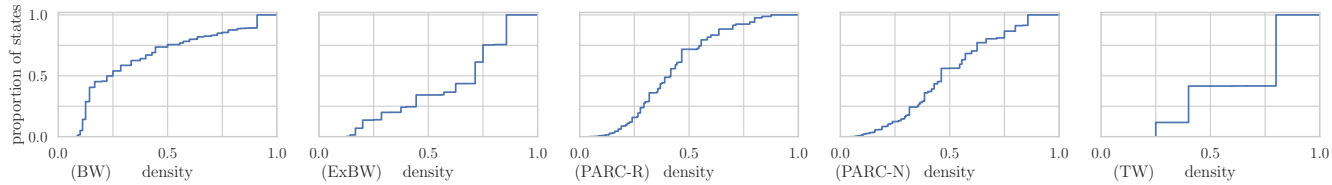

Figure 4: Cumulative plot of state density, i.e., $|\widehat{\mathsf{A}}(s)|/|\mathsf{A}(s)|$ over 50 instances for the largest solved problem per domain.

Moreover, from 300 seconds onwards, CG-iLAO$^*_{\text{roc}}$'s lead varies from 50 to 243 instances. We present a breakdown of coverage per domain in tab. 1. For each domain considered, CG-iLAO$^*_{\text{roc}}$ reaches the highest coverage over other planners and heuristics. For all three heuristics considered, CG-iLAO* also obtains the highest coverage in all domains against the other planners for the same heuristic. In tab. 2, we present the minimum and maximum speedup per domain of CG-iLAO* over the other planners for h^{roc}. The speedups w.r.t. iLAO$^*_{\text{roc}}$ vary from $0.9\times$ (i.e., 11% slower) in the largest PARC-R problem and $3\times$ in problem #9 of ExBW. Against LRTDP$_{\text{roc}}$, the speedups vary from $0.9\times$ in problem #8 of ExBW to $8.4\times$ in PARC-N problem s4-c3. For the performance of each planner and heuristic per problem, see the technical report in Schmalz and Trevizan (2024).

How many actions can CG-iLAO* ignore? To answer this question, we look at the density of states s, defined as $|\widehat{\mathsf{A}}(s)|/|\mathsf{A}(s)|$, in the final partial SSP of CG-iLAO$^*_{\text{roc}}$. Fig. 4 shows, for each domain, the cumulative plot of density, i.e., how many states contain up to and including a given proportion of their applicable actions. In all instances, at least one third of the states contains at most 50% of the applicable actions. The density is high for TW because many states only have one applicable action. The density is also high for ExBW because heuristics are comparatively weak for this domain. For the other domains the results are much stronger: between 56% and 75% of states contain at most 50% of the actions. Overall, CG-iLAO$^*_{\text{roc}}$ added between 38% and 66% of all possible actions in its own partial SSP, and added between 43% and 65% of iLAO*'s actions.

		BW	ExBW	PARC-N	PARC-R	TW	**Total**
Num. of instances		300	250	300	250	200	1300
h^{roc}	CG-iLAO*	**300**	**250**	**300**	**250**	**200**	**1300**
	iLAO*	**300**	200	**300**	**250**	150	1200
	LRTDP	257	**250**	**300**	200	195	1202
h^{lmc}	CG-iLAO*	150	**250**	**300**	200	150	1030
	iLAO*	150	200	**300**	200	140	990
	LRTDP	0	200	**300**	50	149	699
h^{max}	CG-iLAO*	150	200	150	0	161	661
	iLAO*	150	150	150	0	150	600
	LRTDP	150	200	150	0	150	650

Table 1: Coverage per domain. Best coverage for each domain (column) in bold.

	BW	ExBW	PARC-R	PARC-N	TW
iLAO$^*_{\text{roc}}$	1.1–1.4	1.0–3.0	0.9–1.3	1.3–2.2	1.4–1.6
LRTDP$_{\text{roc}}$	1.3–2.0	0.9–2.9	2.0–8.4	0.7–0.9	1.2–1.6

Table 2: CG-iLAO$^*_{\text{roc}}$'s speed-up over iLAO$^*_{\text{roc}}$ and LRTDP$_{\text{roc}}$. The speed-ups only consider problems solved by both algorithms.

Are Q-values being saved? CG-iLAO* can save Q-value computations by ignoring inactive actions, but at the cost of computing additional Q-values in its separation oracle. The cumulative plot over Q-values in fig. 3 (right) shows that the savings in Q-values outweigh the overhead, i.e., given a budget in Q-values computations, CG-iLAO* is ca-

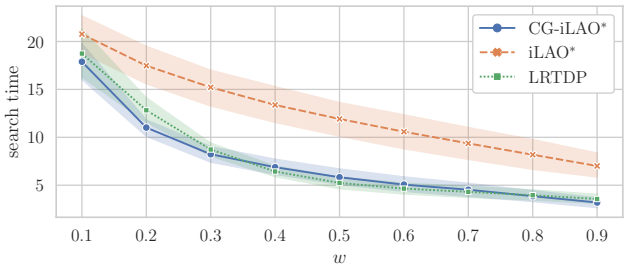

Figure 5: Search time (excludes compute time for the heuristic) of algorithm with h^{pert}_w as w varies. We show mean and 95% C.I. of all considered problems over 50 instances.

pable of solving more instances than the other planners for the same heuristic. At their maximum coverage, iLAO$^*_{\text{roc}}$ and LRTDP$_{\text{roc}}$ use $4\times$ and $10\times$ more Q-values than CG-iLAO$^*_{\text{roc}}$, respectively. Moreover, CG-iLAO$^*_{\text{roc}}$ reaches its maximum coverage of 1300 using 1.64×10^8 Q-values while iLAO$^*_{\text{roc}}$ and LRTDP$_{\text{roc}}$ only solve 1149 and 1052 instances, respectively, for the same number of Q-values. A similar trend is observed when using h^{lmc}; however, when using $h^{\max}$, the least informative heuristic considered, LRTDP$_{\max}$ is slightly more Q-value efficient than CG-iLAO$^*_{\max}$.

What is the impact of the heuristic on CG-iLAO*? Note that, in fig. 3, as the heuristic becomes more informative, the performance gains of CG-iLAO* over iLAO* and LRTDP increases. To explore this trend, we use the heuristic $h^{\text{pert}}_w(s)$ defined for $w \in [0, 1]$ as $V^*(s) \cdot r$ where r is a uniform randomly selected number from $(w, 1]$, which lets us quantify how informative a heuristic is (on average) with w. The randomness in the weight w ensures that the ordering of states induced by h^{pert}_w is different from the one induced by V^*. Due to the high cost of computing V^* we only consider the smallest problems of BW, ExBW, PARC-N, and TW. Over these problems and 50 instances each, fig. 5 shows the mean search time and 95% C.I. as w varies over $0.1, 0.2, \ldots, 0.9$. Search time excludes time spent computing the heuristic. The ratio between CG-iLAO*'s and iLAO*'s runtime supports that CG-iLAO* scales better with better heuristics: the ratio starts at 86% and decreases to 49% and 46% for $w = 0.5$ and $w = 0.9$ respectively. The reason for this behaviour is that good heuristics (i.e., tighter lower bounds) prevent CG-iLAO* from adding inactive actions to its partial SSP, resulting in more savings in Q-value computation. Over all values of w, there is no statistically significant difference between CG-iLAO* and LRTDP, but both offer substantial improvement over iLAO*. This suggests LRTDP's sampling approach can more efficiently leverage the information provided the heuristics than iLAO* and CG-iLAO* bridges the gap between them, allowing a non-sampling-based planner to use the heuristics as effectively as LRTDP.

6 Conclusion, Related and Future Work

Building on existing connections between operations research and planning, we presented a new interpretation of heuristic search on SSPs as solving LPs using variable and constraint generation. We exploit this equivalence to introduce a new and efficient separation oracle for SSPs, which enables a search algorithm to selectively add actions when they are deemed necessary to find the optimal solution. This addresses the shortcoming of state-of-the-art algorithms that add all applicable actions during state expansion, with no mechanism for ignoring actions that will not contribute to the solution. Using this principle, we generalised iLAO* into a new optimal heuristic search algorithm CG-iLAO*. Empirical evaluation showed that CG-iLAO*'s ability to consider a subset of actions results in significant savings in the number of Q-values computed, which in turn reduces the runtime of the algorithm compared to the state-of-the-art.

Regarding related work, LP 1 has been approximated for factored MDPs to get a more compact LP, called the approximate LP (ALP). Schuurmans and Patrascu (2001) apply constraint generation to the ALP; their separation oracle has a similar condition for adding constraints as CG-iLAO*; however, it checks for the condition naively, which is only practical on the compact ALP offered by factored MDPs, and is infeasible for SSPs. Constraint generation lends itself well to complex planning problems where a relaxation can be efficiently solved and constraint violations by the relaxed solution can be efficiently detected, e.g., in multiagent planning (Calliess and Roberts 2021) and metric hybrid factored planning in nonlinear domains (Say and Sanner 2019). For POMDPs, an LP with constraint generation can be used to prune unneeded vectors from the set of vectors used to represent the value function (Walraven and Spaan 2017). In all these works, the separation oracle either naively checks all possible constraints or relies on sampling to find violations.

As future work, we aim to expand the application of CG-iLAO* to more complex models that can benefit from our iterative method of generating applicable actions. Models with imprecise parameters, such as MDPIPs and MDPSTs (White III and Eldeib 1994; Trevizan, Cozman, and Barros 2007), are suitable candidates for our approach since they have a *minimax* semantics for the Bellman equations. In this minimax semantics, the value function minimises the expected cost-to-go assuming that an adversary aims to maximise the cost-to-go by selecting the values of the imprecise parameters. As a result, computing $Q(s, a)$ in these models requires solving a maximisation problem; therefore, ignoring inactive actions could lead to significant improvements in performance.

Other suitable models include SSPs with *PLTL constraints* (Baumgartner, Thiébaux, and Trevizan 2018; Mallet, Thiébaux, and Trevizan 2021) in which both the state space and action space are augmented to keep track of constraint violations. In these models, the concept of inactive actions can be extended to also prevent adding actions that lead to constraint violations to their partial problems. The methods presented in this paper may also be applicable to model checking more broadly. In particular, there has been work investigating how to use heuristics to guide the search for probabilistic reachability (Brázdil et al. 2014), in which action elimination is applicable.

Acknowledgements

We thank the anonymous reviewers for their feedback. 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.

References

Baumgartner, P.; Thiébaux, S.; and Trevizan, F. 2018. Heuristic Search Planning With Multi-Objective Probabilistic LTL Constraints. In *Proc. of 16th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR)*.

Bellman, R. 1957. *Dynamic programming*. Princeton University Press.

Bertsekas, D. 1995. *Dynamic Programming and Optimal Control*, volume 2. Athena Scientific.

Bertsekas, D.; and Tsitsiklis, J. 1991. An Analysis of Stochastic Shortest Path Problems. *Mathematics of Operations Research*.

Bertsimas, D.; and Tsitsiklis, J. 1997. *Introduction to Linear Optimization*. Athena Scientific.

Bonet, B.; and Geffner, H. 2003. Labeled RTDP: Improving the Convergence of Real-Time Dynamic Programming. In *Proc. of 13th Int. Conf. on Automated Planning and Scheduling (ICAPS)*.

Brázdil, T.; Chatterjee, K.; Chmelík, M.; Forejt, V.; Křetínský, J.; Kwiatkowska, M.; Parker, D.; and Ujma, M. 2014. Verification of Markov Decision Processes Using Learning Algorithms. In *Automated Technology for Verification and Analysis*.

Buffet, D. 2008. International Planning Competition Uncertainty Part: Benchmarks and Results.

Calliess, J.-P.; and Roberts, S. 2021. Multi-Agent Planning with Mixed-Integer Programming and Adaptive Interaction Constraint Generation (Extended Abstract). *Proc. of 14th Symposium on Combinatorial Search (SoCS)*.

Dai, P.; Weld, D.; et al. 2009. Focused topological value iteration. In *Proc. of 19th Int. Conf. on Automated Planning and Scheduling (ICAPS)*, volume 19, 82–89.

Hansen, E.; and Zilberstein, S. 2001. LAO*: A heuristic search algorithm that finds solutions with loops. *Artificial Intelligence*.

Helmert, M.; and Domshlak, C. 2009. Landmarks, Critical Paths and Abstractions: What's the Difference Anyway? *Proc. of 19th Int. Conf. on Automated Planning and Scheduling (ICAPS)*.

Little, I.; Thiebaux, S.; et al. 2007. Probabilistic planning vs. replanning. In *ICAPS Workshop on IPC: Past, Present and Future*.

Mallet, I.; Thiébaux, S.; and Trevizan, F. 2021. Progression Heuristics for Planning with Probabilistic LTL Constraints. In *Proc. of 35th AAAI Conference on Artificial Intelligence*.

McMahan, B.; Likhachev, M.; and Gordon, G. 2005. Bounded Real-Time Dynamic Programming: RTDP with Monotone Upper Bounds and Performance Guarantees. In *Proc. of 22nd Int. Conf. on Machine Learning*.

Sanner, S.; Goetschalckx, R.; Driessens, K.; and Shani, G. 2009. Bayesian real-time dynamic programming. In *Proc. of 21st Int. Joint Conf. on AI (IJCAI)*.

Say, B.; and Sanner, S. 2019. *Metric Hybrid Factored Planning in Nonlinear Domains with Constraint Generation*, 502–518. Springer International Publishing.

Schmalz, J.; and Trevizan, F. 2023. Code, benchmarks, and technical report for AAAI 2024 paper "Efficient Constraint Generation for Stochastic Shortest Path Problems". https://doi.org/10.5281/zenodo.10344842.

Schmalz, J.; and Trevizan, F. 2024. Efficient Constraint Generation for Stochastic Shortest Path Problems. arXiv:2401.14636.

Schuurmans, D.; and Patrascu, R. 2001. Direct value-approximation for factored MDPs. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Smith, T.; and Simmons, R. 2006. Focused real-time dynamic programming for MDPs: Squeezing more out of a heuristic. In *Proc. of 20th AAAI Conf. on Artificial Intelligence*.

Teichteil-Königsbuch, F. 2012. Stochastic safest and shortest path problems. In *Proc. of 26th AAAI Conf. on AI*.

Trevizan, F.; Cozman, F. G.; and Barros, L. N. 2007. Planning under Risk and Knightian Uncertainty. In *Proc. of 20th Int. Joint Conf. on AI (IJCAI)*.

Trevizan, F.; Teichteil-Königsbuch, F.; and Thiébaux, S. 2017. Efficient Solutions for Stochastic Shortest Path Problems with Dead Ends. In *Proc. of 33rd Int. Conf. on Uncertainty in Artificial Intelligence (UAI)*.

Trevizan, F.; Thiébaux, S.; and Haslum, P. 2017. Occupation Measure Heuristics for Probabilistic Planning. In *Proc. of 27th Int. Conf. on Automated Planning and Scheduling (ICAPS)*.

Walraven, E.; and Spaan, M. 2017. Accelerated Vector Pruning for Optimal POMDP Solvers. *Proc. of 31st AAAI Conf. on AI*.

White III, C. C.; and Eldeib, H. K. 1994. Markov decision processes with imprecise transition probabilities. *Operations Research*, 42(4): 739–749.