Heuristic Search in Dual Space for Constrained Stochastic Shortest Path Problems

Felipe Trevizan and **Sylvie Thiébaux**
Machine Learning and Optimisation Groups, NICTA
Research School of Computer Science, ANU
first.last@anu.edu.au

Pedro Santana and **Brian Williams**
MERS Group
Computer Science and AI Laboratory, MIT
{psantana,williams}@mit.edu

Abstract

We consider the problem of generating optimal stochastic policies for Constrained Stochastic Shortest Path problems, which are a natural model for planning under uncertainty for resource-bounded agents with multiple competing objectives. While unconstrained SSPs enjoy a multitude of efficient heuristic search solution methods with the ability to focus on promising areas reachable from the initial state, the state of the art for constrained SSPs revolves around linear and dynamic programming algorithms which explore the entire state space. In this paper, we present i-dual, which, to the best of our knowledge, is the first heuristic search algorithm for constrained SSPs. To concisely represent constraints and efficiently decide their violation, i-dual operates in the space of dual variables describing the policy occupation measures. It does so while retaining the ability to use standard value function heuristics computed by well-known methods. Our experiments on a suite of PPDDL problems augmented with constraints show that these features enable i-dual to achieve up to two orders of magnitude improvement in run-time and memory over linear programming algorithms.

1 Introduction

Stochastic Shortest Paths problems (SSPs) are widely used models for planning under uncertainty. Given an initial state, a set of goal states, actions with probabilistic outcomes, and an action cost function, the objective is to find an action policy minimising the expected cost of reaching the goal from the initial state. A nice feature of SSPs is that optimal policies are deterministic mappings from states to actions. Such policies can be efficiently computed using heuristic search algorithms, e.g. LAO* (Hansen and Zilberstein 2001), (L)RTDP (Bonet and Geffner 2003), and many other variants. These algorithms are capable of focusing the search on promising regions reachable from the initial state. To achieve this, they use guidance in the form of an admissible heuristic function, which estimates the expected cost to goal from newly encountered states. This focused search sets these algorithms apart from linear programming formulations and dynamic programming algorithms, such as Value and Policy Iteration, which explore the entire state space.

In many application domains, SSPs have limited value, as multiple competing performance criteria need to be considered and are difficult to encapsulate in a single scalar cost function (Undurti and How 2010). For example, in a UAV search and rescue mission, as many targets as possible need to be found as fast as possible, whilst minimising battery usage and the probability of reaching dangerous areas. A natural approach to deal with such situations is to optimise a primary cost function, and constrain the others (Altman 1999, Feinberg and Shwarz 1995, Dolgov and Durfee 2005). This leads to *constrained SSPs* which augment SSPs with upper bounds on the expected value of the secondary costs. In the above example, we may minimise the number of targets missed, keeping the expected mission time, the expected battery usage and the probability of visiting dangerous areas within acceptable limits. As for unconstrained SSPs, the worst-case complexity of finding the optimal solution for constrained SSPs is polynomial; however, constrained SSPs are harder solve in practice because of the potential conflicts between minimising the primary cost and satisfying all the constraints. Moreover, there is no guarantee that the optimal policy for constrained SSPs is deterministic.

In this paper, we present the first algorithm for constrained SSPs that exploits the advantages of heuristic search. Our algorithm, i-dual, generates optimal stochastic policies. Similarly to *unconstrained* planning algorithms, such as LAO*, i-dual incrementally builds the search space reachable from the initial state (search envelope) by repeatedly expanding the fringe states reachable under the current best policy, and updating that policy using linear programming. The use of an LP naturally enables the formulation of the secondary cost constraints, ensures that the resulting policy satisfies them, and facilitates the generation of stochastic policies.

Importantly, in order to represent and update the search envelope, i-dual uses the dual LP formulation (D'Epenoux 1963). In this formulation, the variables being optimised are the policy *occupation measures* and they represent the expected number of times a given action is executed in a given state. This contrasts with the more common primal LP formulation where the variables being optimised represent the expected cost to reach the goal. The main advantage of the dual LP representation is that it allows solving constraint SSPs in polynomial time while the primal LP representation results in a non-convex problem; therefore, no

polynomial algorithm is known for constrained SSPs using the primal LP representation (Kallenberg 1983, Dolgov and Durfee 2005). A disadvantage of the dual space is that it is difficult to obtain lower bounds for occupation measures, thus early pruning of infeasible solutions based on them is also difficult. To address this issue, i-dual uses heuristics (lower bounds) for the value functions, which can be easily computed by domain-independent methods (Teichteil-Königsbuch, Vidal, and Infantes 2011, Mausam and Kolobov 2012). This makes i-dual the first heuristic search algorithm for constrained SSPs.

We have implemented a domain-independent planner based on i-dual, which solves constrained SSPs described in an extension of the Probabilistic Planning Domain Definition Language (PPDDL) (Younes et al. 2005). We use this planner to evaluate the performance of i-dual on a number of benchmark problems and compare it with the dual LP without heuristic search. We find that i-dual successfully leverages the heuristics over the primary and secondary cost functions to prune suboptimal and infeasible solutions, and is able to solve problems up to two orders of magnitude larger than the regular dual LP approach. Moreover, we show how non-admissible heuristics can be used by i-dual to trade-off optimality with scalability and quickly compute policies that still respect the constraints, an option not offered by the dual LP approach.

The paper is organised as follows: Section 2 presents the background on SSPs and constrained SSPs; i-dual, our novel algorithm, is introduced in Section 3; the empirical evaluation of i-dual is presented in Section 4; and we conclude with a discussion of related and future work in Section 5.

2 Background

2.1 Stochastic Shortest Path Problems

A Stochastic Shortest Path problem (SSP) (Bertsekas and Tsitsiklis 1991) is a tuple $\mathbb{S} = \langle \mathsf{S}, s_0, \mathsf{G}, \mathsf{A}, P, C \rangle$ in which: S is the finite set of states; $s_0 \in \mathsf{S}$ is the initial state; $\mathsf{G} \subseteq \mathsf{S}$ is the non-empty set of goal (or terminal) states; A is the finite set of actions; $P(s'|s,a)$ represents the probability that $s' \in \mathsf{S}$ is reached after applying action $a \in \mathsf{A}$ in state $s \in \mathsf{S}$; and $C(s,a) \in \mathbb{R}^*_+$ is the immediate cost of applying action a in state s. For simplicity, we assume $s_0 \notin \mathsf{G}$ and we represent by $\mathsf{A}(s)$ the actions applicable in state s.

A solution to an SSP is a policy π, i.e., a mapping from states to actions, and we denote by S^π the domain of π. A policy π is complete if $\mathsf{S}^\pi = \mathsf{S}$. If all states s' reachable from s_0 when following π are contained in S^π, then π is a closed policy (i.e., no replanning is needed when executing π), otherwise π is a partial policy. A policy can also be classified as deterministic or stochastic. A deterministic policy π maps each state $s \in \mathsf{S}^\pi$ to one action $a \in \mathsf{A}(s)$; alternatively, a stochastic policy π maps each state $s \in \mathsf{S}^\pi$ to a probability distribution over $\mathsf{A}(s)$ and we denote by $\pi(s,a)$ the probability of executing a in s.

The optimal solution for an SSP is a closed policy π^* that minimises the total expected cost of reaching the goal from s_0. For SSPs, π^* might not be unique and there always exists at least one optimal policy that is deterministic. The optimal solution of an SSP can be uniquely described by the optimal value function V^*, i.e., a function from S to $\mathbb{R}_+$ representing the minimum total expected cost to reach the goal from each state s (Bertsekas and Tsitsiklis 1996). The optimal value function V^* is defined by the set of fixed-point equations known as *Bellman equations*:

$$V^*(s) = \min_{a \in \mathsf{A}(s)} C(s,a) + \sum_{s' \in \mathsf{S}} P(s'|s,a) V^*(s')$$

for $s \notin \mathsf{G}$ and $V^*(s) = 0$ for $s \in \mathsf{G}$. Any optimal policy π^* can be extracted from V^* by replacing $\min$ by argmin in the definition of the Bellman equations. V^* can be computed iteratively (e.g., Value Iteration (Howard 1960)), using linear programming (Puterman 1994), and through heuristic search (e.g., LAO* (Hansen and Zilberstein 2001), (L)RTDP (Bonet and Geffner 2003)). We refer to these algorithms that explicitly compute V^* as primal-space algorithms. An admissible heuristic H for primal-space algorithms is any lower bound on V^*.

Alternatives to primal-space algorithms are policy space algorithms (e.g., Policy Iteration (Howard 1960)) and linear programming over the dual space (D'Epenoux 1963). The latter is presented in (LP 1) and the optimisation variables $x_{s,a}$, known as occupation measure, represent the expected number of times action $a \in \mathsf{A}(s)$ is executed in s.

$$\min_x \sum_{s \in \mathsf{S}, a \in \mathsf{A}(s)} x_{s,a} C(s,a) \quad \text{s.t.} \quad \text{(C1) – (C6)} \qquad \text{(LP 1)}$$

$$x_{s,a} \geq 0 \qquad \forall s \in \mathsf{S}, a \in \mathsf{A}(s) \quad \text{(C1)}$$

$$in(s) = \sum_{s' \in \mathsf{S}, a \in \mathsf{A}(s')} x_{s',a} P(s|s',a) \qquad \forall s \in \mathsf{S} \quad \text{(C2)}$$

$$out(s) - in(s) = 0 \qquad \forall s \in \mathsf{S} \setminus (\mathsf{G} \cup \{s_0\}) \quad \text{(C3)}$$

$$out(s_0) - in(s_0) = 1 \quad \text{(C4)}$$

$$out(s) = \sum_{a \in \mathsf{A}(s)} x_{s,a} \qquad \forall s \in \mathsf{S} \setminus \mathsf{G} \quad \text{(C5)}$$

$$\sum_{s_g \in \mathsf{G}} in(s_g) = 1 \quad \text{(C6)}$$

This dual formulation can be interpreted as a *flow problem*, where: (C2) and (C5) define expected flow entering and leaving the state s, respectively; (C3) is the flow conservation principle, i.e., all flows reaching s must leave s (for all states s that are neither the *source* nor a *sink*); and (C4) and (C6) define, respectively, the source (initial state s_0) and the sinks (goal states). The objective function of (LP 1) captures the minimisation of the total expected cost to reach the goal (sink) from the initial state (source). Once we have the solution x^* of (LP 1), the optimal policy is $\pi^*(s,a) = x_{s,a}/out(s)$ and is guaranteed to be deterministic, i.e., $x_{s,a} > 0$ for exactly one action $a \in \mathsf{A}(s)$ for all $s \in \mathsf{S}$ such that $out(s) > 0$.

In this work, we consider SSPs with dead ends, that is, we do not assume that, for all $s \in \mathsf{S}$, the probability for reaching the goal is 1. We use the fixed-cost approach for dead ends (Mausam and Kolobov 2012), i.e., if s is a dead end, then $V^*(s)$ is defined as d, where d is a large positive

penalty for not reaching the goal.[1] As a side effect, every state s such that $V^*(s) \geq d$ is treated as a dead end, even if its probability of reaching the goal is greater than 0.

LP 2 presents the dual-space linear program for SSPs with dead ends. For each state s, a new occupation measure x_s^D is added to represent the flow *escaping* from s if s is proved to be a dead end. x_s^D is necessary in order to enforce the flow conservation constraints since flows can only be absorbed by sinks. A new sink constraint (C9) is also required and it enforces that the flows not reaching the goal must be directed to the dead-end sink, i.e., the flow trapped in a dead-end state s is forced to leave s using x_s^D by (C3) and (C8). Since one unit of flow is injected into the system (C4) and all flows using x_s^D are directed to a sink, $\sum_{s\in\mathsf{S}} x_s^D$ is equivalent to the probability of reaching a dead end from the initial state. The objective function of LP 2 is obtained by adding the total expected penalty for reaching dead ends ($\sum_{s\in\mathsf{S}} x_s^D d$) to the objective function of LP 1.

$$\min_x \sum_{s\in\mathsf{S},a\in\mathsf{A}(s)} x_{s,a}C(s,a) + \sum_{s\in\mathsf{S}} x_s^D d$$
$$\text{s.t.} \quad \text{(C1) – (C4), (C7) – (C9)} \qquad \text{(LP 2)}$$

$$x_s^D \geq 0 \qquad \forall s \in \mathsf{S} \qquad \text{(C7)}$$

$$out(s) = \sum_{a\in\mathsf{A}(s)} x_{s,a} + x_s^D \qquad \forall s \in \mathsf{S}\setminus\mathsf{G} \qquad \text{(C8)}$$

$$\sum_{s_g\in\mathsf{G}} in(s_g) + \sum_{s\in\mathsf{S}} x_s^D = 1 \qquad \text{(C9)}$$

2.2 Constrained SSPs

A constrained SSP (C-SSP) is an SSP with $k+1$ cost functions in which one cost function is optimised while the remaining k costs are constrained by an upper bound. We refer to the former as *primary* cost and all other cost functions as *secondary* costs. Formally, a C-SSP $\mathbb{C}$ is the tuple $\langle \mathsf{S}, s_0, \mathsf{G}, \mathsf{A}, P, \vec{C}, \vec{\hat{c}}\rangle$ where $\mathsf{S}, s_0, \mathsf{G}, \mathsf{A}$, and P are defined as in the SSP case and $\vec{C} = [C_0, C_1, C_2, \ldots, C_k]$ is the vector of cost functions ($C_j \colon (\mathsf{S},\mathsf{A}) \to \mathbb{R}_+^*$ for all j) and $\vec{\hat{c}} = [\hat{c}_1, \ldots, \hat{c}_k]$ is the cost upper bound vector ($\hat{c}_j > 0$ for all j). Since each C_j can differ in scale (e.g., cost in dollars, energy in joules, time in minutes), the dead-end penalty is also represented by a vector $\vec{d}$ and d_j is the penalty incurred in C_j when a dead end is reached.

The constraints on costs C_j, for $j \in \{1,\ldots,k\}$, are on expectation, i.e., given a policy π, the *expected* value of C_j over all executions of π from s_0 to the goal must be less or equal to $\hat{c}_j$. Thus, for a particular execution of π, the observed cost C_j may be larger than $\hat{c}_j$. Even though an upper bound on the maximum incurred cost (as opposed to its expectation) might be desired, such bounds make most SSPs infeasible because they require that none of the possible executions of π violate the constraints. In the general case, this can only be guaranteed when the execution of π is loop-free because, if the execution π can revisit any state, then there is a non-zero probability of π being executed for at least n steps before reaching the goal for all $n \in \mathbb{N}$. Moreover, since the constraints are over the expected execution of π, they restrict the policy space instead of the state space. For instance, a given state could be reached by two actions that consume different amounts of resources and only one of them violate the constraints.

The optimal solution to a C-SSP is defined similarly to the optimal solution of an SSP where the primary cost function C_0 is being minimised and the expected value of the secondary cost functions C_j, for $j \in \{1,\ldots,k\}$, is upper bounded by $\hat{c}_j$. Thus, the dual LP for solving C-SSPs is the LP 2 subject to the additional constraint C10:

$$\sum_{s\in\mathsf{S},a\in\mathsf{A}(s)} x_{s,a}C_j(s,a) + \sum_{s\in\mathsf{S}} x_s^D d_j \leq \hat{c}_j \quad \forall j \in \{1,\ldots,k\} \qquad \text{(C10)}$$

In contrast to SSPs, there is no guarantee that the optimal policy for C-SSPs is deterministic (Altman 1999). Nevertheless, the complexity of finding the (potentially stochastic) optimal policy for C-SSPs is polynomial (Dolgov and Durfee 2005) as in the case of SSPs.

The occupation measure space is particularly interesting for C-SSPs because only one set of fixed-point equations needs to be solved to obtain x, regardless of the number of secondary cost functions. Given x, the total expected j-th cost of following x from s_0 to a goal can be easily computed by $\sum_{s,a} x_{s,a}C_j(s,a) + \sum_s x_s^D d_j$. In contrast, primal-space algorithms have to represent and compute the fixed-point solution for each value function V_j associated with C_j ($j \in \{0,\ldots,k\}$). Furthermore, the set of costs computed need to be those for a single policy, and imposing this constraint in the primal-space is non-trivial. Specifically, it requires an extra set of continuous variables to represent the current stochastic policy $\pi(s,a)$ and they multiply the right-hand side of the Bellman equation, resulting in the following bilinear constraints:

$$V_j(s) \leq \sum_{a\in\mathsf{A}(s)} \pi(s,a)\Big[C_j(s,a) + \sum_{s'\in\mathsf{S}} P(s'|s,a)V_j(s')\Big].$$

More importantly, these bilinear constraints are non-convex, replacing the LP with a non-convex problem, whose solution is intractable for all but the smallest problems.

Although the dual approach (LP 2) is tractable, a significant computational cost is that it requires all the reachable states from s_0 to be encoded, and thus explored, even if S^{π^*} is only a fraction of the reachable space. Furthermore, deriving a non-zero lower bound (admissible heuristic) for the occupation measure x is not trivial because $x_{s,a} > 0$ if and only if $a \in \pi^*(s)$ for at least one optimal policy π^*. In fact, the authors are not aware of any existing work that derives domain-independent lower bounds for $x_{s,a}$.

3 i-dual

In this section, we explain how to perform heuristic search in the occupation measure space using heuristics for the primal space. That is, instead of using a lower bound on occupation measures $x_{s,a}$, we use a heuristic vector

[1] I-dual can be easily adapted to work with other approaches for handling dead-ends; see Section 5.

over costs $\vec{H} = [H_0, \ldots, H_k]$, where $H_j \colon \mathsf{S} \to \mathbb{R}_+$ for all $i \in \{0, \ldots, k\}$ is a heuristic for the expected value of C_j. A heuristic H_j is admissible for C_j if it is a lower bound for the j-th total expected cost.

In contrast to a heuristic for $x_{s,a}$, H_j can be easily derived by relaxing the C-SSP into an unconstrained SSP as a range of heuristics exist for SSPs (Mausam and Kolobov 2012) which generalise classical planning heuristics (Teichteil-Königsbuch, Vidal, and Infantes 2011). Formally, given a C-SSP $\langle \mathsf{S}, s_0, \mathsf{G}, \mathsf{A}, P, \vec{C}, \vec{\hat{c}} \rangle$ and $j \in \{0, \ldots, k\}$, let $\mathbb{S}_j$ be the SSP $\langle \mathsf{S}, s_0, \mathsf{G}, \mathsf{A}, P, C_j \rangle$, then the optimal value function V^* for $\mathbb{S}_j$ is an admissible heuristic for C_j. Therefore, any admissible heuristic H for $\mathbb{S}_j$ is also admissible for C_j.

Our novel approach consists in defining incrementally larger *partial problems* starting from s_0, using artificial goals to represent non-explored areas of the occupation measure space. Artificial goals are prioritised using their heuristic values, incurred as a one-time *terminal cost* when the artificial goal state is first reached. Each partial problem is thus represented as a C-SSP with terminal costs[2] $\langle \hat{\mathsf{S}}, s_0, \hat{\mathsf{G}}, \hat{\mathsf{A}}, P, \vec{C}, \vec{\hat{c}}, \vec{H} \rangle$, where: $\langle \hat{\mathsf{S}}, s_0, \hat{\mathsf{G}}, \hat{\mathsf{A}}, P, \vec{C}, \vec{\hat{c}} \rangle$ is a regular C-SSP, $\hat{\mathsf{S}} \subseteq \mathsf{S}$, $\mathsf{G} \cap \hat{\mathsf{S}} \subseteq \hat{\mathsf{G}}$, $\hat{\mathsf{A}} \subseteq \mathsf{A}$, and $\vec{H}$ is the heuristic vector. The dual linear program for C-SSPs with such terminal costs is presented in LP 3.

$$\min_{x} \sum_{s \in \hat{\mathsf{S}}, a \in \hat{\mathsf{A}}(s)} x_{s,a} C_0(s,a) + \sum_{s \in \hat{\mathsf{S}}} x_s^D d_0 + \sum_{s_g \in \hat{\mathsf{G}}} in(s_g) H_0(s_g)$$

$$\text{s.t.} \quad \text{(C1) – (C4),(C7) – (C9), (C11)} \qquad \text{(LP 3)}$$

$$\sum_{s \in \hat{\mathsf{S}}, a \in \hat{\mathsf{A}}(s)} x_{s,a} C_j(s,a) + \sum_{s \in \hat{\mathsf{S}}} x_s^D d_j + \sum_{s_g \in \hat{\mathsf{G}}} in(s_g) H_k(s_g) \leq \hat{c}_k \quad \forall j \in \{1, \ldots, k\} \quad \text{(C11)}$$

Our new algorithm, i-dual, is depicted in Algorithm 1. The key elements that vary from one iteration of the algorithm to the next are: the set $\hat{\mathsf{S}}$ of states considered so far; the set $\mathsf{F} \subseteq \hat{\mathsf{S}}$ of fringe states (i.e., the unexpanded states in $\hat{\mathsf{S}}$); the fringe states $\mathsf{F}_R \subseteq \mathsf{F}$ reachable from s_0 by following the policy encoded by x; and the set $\hat{\mathsf{G}}$ of artificial goals for the current partial problem. At each iteration, i-dual computes the set N of all the new states reachable from F_R by applying one action (line 7), and updates the set of states, the fringe, and the artificial goals accordingly (lines 8-10). A new C-SSP with heuristic terminal costs is set up for this set of states and artificial goals, and an optimal policy for it is produced by solving LP 3. This process is repeated until the set F_R of reachable fringe states is empty (line 6). When this occurs, a closed policy has been found, that is, all of the flow injected into the system is absorbed only by goals and dead ends specified in the original problem. At this point, the search stops and this policy is returned (lines 14-15).

I-dual handles both avoidable and unavoidable dead ends and, since it always enforces the cost constraints (C11), the returned policy π (if any) is always feasible regardless of the heuristic vector used and its admissibility. Moreover, if all the heuristics in $\vec{H}$ are admissible, then π is optimal:

[2] A C-SSP with terminal cost can be trivially encoded as a regular C-SSP by adding extra actions. The use of terminal costs simplifies our notations.

1 I-DUAL(C-SSP $\langle \mathsf{S}, s_0, \mathsf{G}, \mathsf{A}, P, \vec{C}, \vec{\hat{c}} \rangle$, dead-end penalties $\vec{d}$, and vector of heuristics $\vec{H}$)
2 **begin**
3 $\quad \hat{\mathsf{S}} \leftarrow \{s_0\}$
4 $\quad \mathsf{F} \leftarrow \{s_0\}$
5 $\quad \mathsf{F}_R \leftarrow \{s_0\}$
6 $\quad$ **while** $\mathsf{F}_R \neq \emptyset$ **do**
7 $\quad\quad \mathsf{N} \leftarrow \{s' \notin \hat{\mathsf{S}} | \exists s \in \mathsf{F}_R, a \in \mathsf{A}(s), P(s'|s,a) > 0\}$
8 $\quad\quad \hat{\mathsf{S}} \leftarrow \hat{\mathsf{S}} \cup \mathsf{N}$
9 $\quad\quad \mathsf{F} \leftarrow (\mathsf{F} \setminus \mathsf{F}_R) \cup (\mathsf{N} \setminus \mathsf{G})$
10 $\quad\quad \hat{\mathsf{G}} \leftarrow \mathsf{F} \cup (\mathsf{G} \cap \hat{\mathsf{S}})$
11 $\quad\quad \hat{\mathsf{A}} \leftarrow \{a \mid \exists s \in \hat{\mathsf{S}} \setminus \mathsf{F}, a \in \mathsf{A}(s)\}$
12 $\quad\quad x \leftarrow$ SOLVE(LP 3 for $\langle \hat{\mathsf{S}}, s_0, \hat{\mathsf{G}}, \hat{\mathsf{A}}, P, \vec{C}, \vec{\hat{c}}, \vec{H} \rangle, \vec{d}$)
13 $\quad\quad \mathsf{F}_R \leftarrow \{s \in \mathsf{F} \mid in(s) > 0\}$
14 $\quad$ **for** (s,a) *s.t.* $x_{s,a} > 0$ **do**
15 $\quad\quad \pi(s,a) \leftarrow x_{s,a} / out(s)$
16 $\quad$ **return** π

Algorithm 1: i-dual algorithm for solving C-SSPs using incrementally larger linear programs over the occupation measure (dual) space and heuristic search over the value function (primal) space. The functions $in(s)$ and $out(s)$ are defined by (C2) and (C5), respectively.

Theorem 1. *Given a C-SSP $\mathbb{C} = \langle \mathsf{S}, s_0, \mathsf{G}, \mathsf{A}, P, \vec{C}, \vec{\hat{c}} \rangle$ and a vector of admissible heuristics $\vec{H}$, the policy π returned by i-dual is an optimal policy for $\mathbb{C}$.*

Proof Sketch. Let $\hat{\mathbb{C}} = \langle \hat{\mathsf{S}}, s_0, \hat{\mathsf{G}}, \hat{\mathsf{A}}, P, \vec{C}, \vec{\hat{c}}, \vec{H} \rangle$ be the last C-SSP with terminal costs solved by line 12 (Algorithm 1). By definition, π is an optimal policy for $\hat{\mathbb{C}}$. Moreover, since F_R is empty when i-dual terminates, then, by line 13, $\sum_{s \in \mathsf{F}} in(s) = 0$. Thus, by line 10, $\sum_{s \in \hat{\mathsf{G}}} in(s) = \sum_{s_g \in \mathsf{G}} in(s_g)$, i.e., all goal states reached by following π from s_0 in $\hat{\mathbb{C}}$ are goals of the original problem $\mathbb{C}$; therefore π is a closed policy for $\mathbb{C}$. Since the terminal costs of $\hat{\mathbb{C}}$ are a lower bound (admissible heuristic) for all expected costs C_j ($j \in \{0, \ldots, k\}$), then any other policy for $\mathbb{C}$ has expected cost greater or equal than π. □

Similarly to primal-space heuristic search algorithms for SSPs, such as LAO* and LRTDP, i-dual is complete but suboptimal when the heuristics $H_1, \ldots, H_k$ for the secondary costs are admissible but the heuristic H_0 for the primary cost is not. When any secondary heuristic is not admissible, i-dual is incomplete: it might not find a solution even if one exists because the non-admissible heuristic can incorrectly indicate that a feasible solution violates the constraints and i-dual would prune this solution from the search space.

I-dual can be viewed through two different prisms: as a heuristic search algorithm and as a delayed column and row generation algorithm. In the former, an Artificial Intelligence point of view, i-dual is analogous to A* where the cost

$g(n)$ to reach an artificial goal (fringe) n from s_0 is computed using LP 3. The LP handles the search in the cyclic space of occupation measures to find a solution that minimises the expected value of $g(n) + H_0(s)$. As in A*, the selected fringe states (F_R in Algorithm 1) are expanded and the search continues until a solution is found.

Alternatively, from an Operations Research point of view, i-dual is a column and row generation algorithm. That is, at each iteration, new variables $x_{s,a}$ are added (columns) as well as new flow conservation constraints (rows) for the newly expanded states, and their corresponding occupation measures. The heuristics $\vec{H}$ prioritise the candidate columns and the solution of LP 3 indicates what columns should be added, namely, $x_{s,a}$ for all $s \in \mathsf{F}_R$ and $a \in \mathsf{A}(s)$. Columns are added in batch because, if a is applied in s, then all states s' s.t. $P(s'|s,a) > 0$ need to be considered in order to obtain a closed policy.

The column and row generation point of view is not only theoretically relevant, it also increases the performance of i-dual. The overhead of building the LPs in line 12 (Algorithm 1) is minimised by keeping only one LP in memory and adding columns and rows to it. More importantly, this approach can take advantage of "warm starts", i.e., to solve the new LP by reusing the solution of the previous one.

4 Empirical Evaluation

In this section we empirically evaluate i-dual and the dual LP approach which directly solves LP 2 with the additional cost constraint C10. We use as metric the number of problems solved and the average time to solve them. Our hypothesis is that the i-dual approach, namely, incremental generation of the occupation measure space guided by value function heuristics, is able to scale up to larger problems than the dual LP approach.

For these experiments, we use Gurobi as linear solver and enforce a 30-minutes cpu-time and 4-Gb memory cutoff on all planners. Our implementation of i-dual follows the column and row generation approach described in Section 3 and we denote the different parametrisations of i-dual as "i-dual(X,Y)", where X is the heuristic used for the primary cost C_0 and Y is the heuristic used for each of the secondary costs C_j; therefore, the heuristic vector $\vec{H}$ used by i-dual(X,Y) is $[X, Y, Y, \ldots, Y]$. We consider the following domain-independent heuristics: always zero (h_0), maximal ($h_{\max}$), additive (h_{add}), and lm-cut (h_{lmc}) (Helmert and Domshlak 2009) – see (Teichteil-Königsbuch, Vidal, and Infantes 2011) for a description of domain-independent heuristics in the context of SSPs. The heuristics for C_j are obtained by applying them in the all-outcomes determinisation of the SSP (Jimenez, Coles, and Smith 2006) in which the cost function is C_j (Section 3).

4.1 Problems

The following three domains were considered in our empirical evaluation. For all problems in these domains, the dead-end penalty d_j equals 1000 for all cost functions C_j, $j \in \{0, \ldots, k\}$.

Search and Rescue. This is an $n \times n$ grid vehicle navigation problem where the goal is to find one survivor, board her on the vehicle and bring her to a safe location. The primary objective is to do this as fast as possible, and the constraint is to keep the expected fuel consumption under a certain threshold (the vehicle fuel autonomy). The time for moving from a location to an adjacent one depends on the load of the vehicle and is therefore higher if a survivor has boarded, and changes with the speed of the vehicle which can be slow, normal, or fast. The fuel consumption increases with load and speed, and some of the more demanding moves (e.g., moving fast with survivor boarded) can fail with a small probability. The location of one survivor at Hamming distance $d \in \{1, \ldots, 4\}$ is known a priori; whereas the presence or absence of a survivor at each other location can be known or unknown, and in the latter case, has an initial probability in the range low (5%), medium (10%) and high (20%). The presence or absence of a survivor is revealed by a coin flip when visiting the location, and remains known thereafter. We used random problem instances with a density $r \in \{25\%, 50\%, 75\%\}$ of locations with initially unknown survivor presence.

Elevators. This domain represents a n-storey building with e elevators in which w persons are known to be waiting for an elevator and h persons are expected to arrive and request an elevator. The arrival of each "hidden" person follows a geometric distribution with probability of success $p = 0.75$. The origin and destination of all persons are known a priori, and the maximum expected waiting time and travel time of each person are constrained; therefore, each instance of this domain has $2(w + h)$ cost constraints. The expected waiting time and travel time for each passenger is constrained to be less or equal to $1.5n$, except for a randomly chosen hidden person, which is considered a VIP passenger and her expected waiting time and travel time are upper bounded by $0.75n$ and $n + 1$, respectively. The primary cost function is the total number of elevators movements. For all the elevators problems, we fix the number of floors n to 20.

Exploding Blocks World. This domain is an adaptation of the exploding blocks world domain of the 2008 International Probabilistic Planning Competition (IPPC) (Bryce and Buffet 2008). The exploding blocks world is a probabilistic extension of the deterministic blocks world in which blocks can explode and destroy other blocks or the table. Once a block or the table is destroyed, nothing can be placed on them, and destroyed blocks cannot be moved. Therefore, the original IPPC domain has unavoidable dead-ends. We modified the exploding blocks world domain by adding a new action to fix a destroyed table, allowing blocks to be placed on it again, with cost 100; the other actions have cost 1.[3] Due do the *fix-table* action, dead-ends are avoidable in our extension. In the C-SSP version, the primary objective is to minimise the action costs and the constraint is to keep the expected number of exploded blocks under a given threshold.

[3] Additionally, we removed the flaw from the original domain that allowed a block to be placed on top of itself.

	r	planner	$d=1$	$d=2$	$d=3$	$d=4$
n = 4	**0.25**	dual-lp	30 (1.6 ± 0.0)	30 (2.3 ± 0.1)	30 (2.5 ± 0.2)	30 (2.6 ± 0.2)
		i-dual(h_{lmc},$h_{\max}$)	30 (0.0 ± 0.0)	30 (0.1 ± 0.0)	30 (0.7 ± 0.1)	30 (2.8 ± 0.5)
		i-dual(h_{add},h_{add})	30 (0.0 ± 0.0)	30 (0.1 ± 0.0)	30 (0.4 ± 0.1)	30 (2.1 ± 0.4)
	0.50	dual-lp	30 (598.4 ± 67.7)	30 (540.6 ± 66.3)	30 (546.5 ± 65.4)	30 (622.6 ± 87.8)
		i-dual(h_{lmc},$h_{\max}$)	30 (0.1 ± 0.0)	30 (0.2 ± 0.0)	30 (8.1 ± 3.6)	30 (220.2 ± 85.5)
		i-dual(h_{add},h_{add})	30 (0.0 ± 0.0)	30 (0.1 ± 0.0)	30 (4.4 ± 1.9)	30 (142.4 ± 51.2)
	0.75	dual-lp	0 (–)	0 (–)	0 (–)	0 (–)
		i-dual(h_{lmc},$h_{\max}$)	30 (0.1 ± 0.0)	30 (0.6 ± 0.1)	30 (126.9 ± 60.8)	7 (2291.6 ± 829.1)
		i-dual(h_{add},h_{add})	30 (0.0 ± 0.0)	30 (0.3 ± 0.0)	30 (78.1 ± 39.0)	7 (1217.7 ± 530.5)
n = 5	**0.25**	dual-lp	30 (131.0 ± 12.4)	30 (109.2 ± 10.1)	30 (121.2 ± 11.0)	30 (128.4 ± 9.2)
		i-dual(h_{lmc},$h_{\max}$)	30 (0.1 ± 0.0)	30 (0.3 ± 0.0)	30 (1.4 ± 0.4)	30 (30.6 ± 16.9)
		i-dual(h_{add},h_{add})	30 (0.1 ± 0.0)	30 (0.1 ± 0.0)	30 (0.6 ± 0.2)	30 (15.6 ± 7.5)
	0.50	dual-lp	0 (–)	0 (–)	0 (–)	0 (–)
		i-dual(h_{lmc},$h_{\max}$)	30 (0.1 ± 0.0)	30 (0.7 ± 0.1)	30 (29.9 ± 11.5)	16 (1370.0 ± 605.6)
		i-dual(h_{add},h_{add})	30 (0.1 ± 0.0)	30 (0.3 ± 0.1)	30 (13.5 ± 5.2)	25 (1360.7 ± 437.9)
	0.75	dual-lp	0 (–)	0 (–)	0 (–)	0 (–)
		i-dual(h_{lmc},$h_{\max}$)	30 (0.1 ± 0.0)	30 (1.0 ± 0.2)	30 (444.2 ± 225.0)	6 (2106.9 ± 1112.0)
		i-dual(h_{add},h_{add})	30 (0.1 ± 0.0)	30 (0.4 ± 0.1)	30 (200.8 ± 99.4)	7 (1517.3 ± 785.9)

Table 1: Results of the search and rescue experiment presented as coverage and average cpu-time (with 95% confidence interval) in seconds. The problems parameters are: grid size (n), density of locations with potential survivors (r), and distance from the robot to known survivor (d).

For each problem, this threshold is the smallest value with only two decimal places that yields a feasible problem. The problem instances are those of the IPPC augmented with the constraint, and the instances with more than 8 blocks have their blocks b9, b10, etc removed.

4.2 Results

The results for 30 randomly generated problems for different parametrisations of the search and rescue and elevators domain are shown in Tables 1 and 2, respectively. Table 3 presents the results for the augmented IPPC exploding blocks world instances over 30 runs with different random seeds. For each problem, the results are presented as "$X(Y \pm Z)$", where X is the number of problems solved by the planner (coverage) and $Y \pm Z$ is the average (and its 95% confidence interval) cpu-time in seconds over the X solved problems. If a problem of the search and rescue or elevators domain is proved to be infeasible (impossible to meet the constraints), it is replaced by a new random problem. Table 4 shows the average number of states explored and percentage of the total cpu-time spent encoding LP 3, computing the vector of heuristics $\vec{H}$, and solving LP 3 for different problems of each domain.

For the search and rescue problems (Table 1), the dual LP fails to scale up and is unable to solve even trivial problems (e.g., distance to known survivor $d = 1$) for higher density of locations with potential survivor presence r and larger grid size n. For trivial problems ($d \in \{1, 2\}$), i-dual finds the optimal solution in all its parametrisations in less than a second since S^{π^*} encompasses only a few hundred states. As shown in Table 4, i-dual is able to explore at least one order of magnitude less states than the dual LP, allowing it to scale up to larger problems. The results also show that the scalability of i-dual can be further improved by giving up optimality and using non-admissible heuristics (e.g., h_{add}), a trade-off not present in the dual LP approach. As expected, h_{add} also speeds up i-dual in part because h_{add} allows pruning infeasible solutions earlier; however, it might also prune feasible solutions due to its non-admissibility.

For the elevator problems (Table 2), the dual LP dominates all i-dual parametrisations when there is only one elevator ($e = 1$). This is expected since all the random problems with $e = 1$ have less than 40,000 reachable states; therefore, it is feasible to encode the complete dual LP. In the instances with two elevators, the dual LP fails to scale up to the larger problems and the configuration $e = 2, h = 2, w = 2$ (0.5 million reachable states on average) contains the largest problems in which dual LP obtained non-zero coverage. The admissible parametrisations of i-dual were also dominated by the dual LP when $e = 2$. The reason for this under performance of i-dual using admissible heuristics, as illustrated in Table 4, are: (i) the high computational cost of h_{lmc}, which consumes about 41% of the cpu-time for the problems in $e = 2, h = 1, w = 2$; and (ii) the weak guidance of $h_{\max}$ for the primary cost. For instance, in $e = 2, h = 1, w = 2$, $h_{\max}$ consumes only 2.3% of the cpu-time of i-dual($h_{\max}$,$h_{\max}$) but this parametrisation of i-dual reaches the 30 minutes cpu-time limit in 4 instances that i-dual with h_{lmc} successfully finds the optimal solution. In contrast, h_{add} was able to offer good guidance and allowed i-dual to solve problems with up to three elevators. For instance, the configuration $e = 2, h = 2, w = 4$ contains the largest instances in which we could explicitly compute the set of reachable states, and i-dual(h_{add},h_0) explored on average only 5000 states out of the average 0.8 million reachable states in those instances.

For the exploding blocks world (Table 3), the dual LP is unable to solve problems with more than 5 blocks while i-dual is able to solve problems with up to 8 blocks. Furthermore, i-dual is two orders of magnitude faster than the

		planner	$w = 1$	$w = 2$	$w = 3$	$w = 4$
e = 1	**h = 1**	dual-lp	30 (0.1 ± 0.0)	30 (0.2 ± 0.0)	30 (0.8 ± 0.1)	30 (3.5 ± 0.5)
		i-dual($h_{\max}$,$h_{\max}$)	30 (0.2 ± 0.0)	30 (1.1 ± 0.2)	30 (11.3 ± 3.2)	30 (186.7 ± 76.0)
		i-dual(h_{lmc},$h_{\max}$)	30 (0.9 ± 0.1)	30 (6.1 ± 0.9)	30 (87.1 ± 20.0)	17 (1119.6 ± 202.6)
		i-dual(h_{lmc},h_0)	30 (0.9 ± 0.2)	30 (6.6 ± 1.0)	30 (101.9 ± 23.7)	14 (1110.8 ± 207.5)
		i-dual(h_{add},h_0)	30 (0.1 ± 0.0)	30 (0.2 ± 0.1)	30 (3.3 ± 2.3)	30 (16.8 ± 4.1)
		i-dual(h_{add},h_{add})	30 (0.1 ± 0.0)	30 (0.3 ± 0.1)	30 (2.7 ± 0.9)	30 (19.5 ± 4.8)
	h = 2	dual-lp	30 (0.4 ± 0.0)	30 (1.7 ± 0.4)	30 (9.8 ± 3.2)	30 (137.9 ± 42.3)
		i-dual($h_{\max}$,$h_{\max}$)	30 (1.6 ± 0.3)	30 (11.2 ± 2.0)	30 (185.3 ± 67.6)	11 (736.2 ± 217.9)
		i-dual(h_{lmc},$h_{\max}$)	30 (9.3 ± 1.4)	30 (91.6 ± 14.6)	19 (1117.2 ± 218.1)	0 (–)
		i-dual(h_{lmc},h_0)	30 (10.0 ± 1.4)	30 (106.5 ± 17.4)	15 (1115.5 ± 214.2)	0 (–)
		i-dual(h_{add},h_0)	30 (0.4 ± 0.2)	30 (2.6 ± 1.2)	30 (57.2 ± 42.4)	30 (217.0 ± 89.2)
		i-dual(h_{add},h_{add})	30 (0.6 ± 0.2)	30 (3.0 ± 0.8)	30 (27.6 ± 8.0)	30 (254.8 ± 79.9)
e = 2	**h = 1**	dual-lp	30 (38.8 ± 7.3)	30 (375.0 ± 117.5)	9 (968.3 ± 356.4)	0 (–)
		i-dual($h_{\max}$,$h_{\max}$)	23 (607.1 ± 229.7)	9 (790.7 ± 355.4)	2 (525.8 ± 283.8)	0 (–)
		i-dual(h_{lmc},$h_{\max}$)	25 (511.7 ± 180.3)	13 (681.5 ± 246.8)	2 (918.7 ± 46.2)	0 (–)
		i-dual(h_{lmc},h_0)	26 (436.7 ± 159.7)	16 (753.7 ± 260.9)	2 (1166.0 ± 25.2)	0 (–)
		i-dual(h_{add},h_0)	30 (0.5 ± 0.3)	30 (18.7 ± 28.3)	30 (66.9 ± 73.5)	30 (369.3 ± 220.3)
		i-dual(h_{add},h_{add})	30 (0.5 ± 0.2)	30 (8.6 ± 8.9)	30 (35.8 ± 32.7)	30 (272.4 ± 110.5)
	h = 2	dual-lp	24 (770.1 ± 150.5)	2 (1727.2 ± 55.0)	0 (–)	0 (–)
		i-dual($h_{\max}$,$h_{\max}$)	3 (662.0 ± 121.0)	0 (–)	0 (–)	0 (–)
		i-dual(h_{lmc},$h_{\max}$)	8 (989.3 ± 329.5)	0 (–)	0 (–)	0 (–)
		i-dual(h_{lmc},h_0)	11 (1089.5 ± 294.1)	0 (–)	0 (–)	0 (–)
		i-dual(h_{add},h_0)	30 (3.5 ± 1.8)	30 (80.5 ± 51.0)	28 (342.3 ± 156.7)	9 (816.4 ± 335.3)
		i-dual(h_{add},h_{add})	30 (4.6 ± 2.7)	30 (68.3 ± 45.9)	28 (258.7 ± 144.9)	11 (960.3 ± 296.4)
e = 3	**h = 1**	dual-lp	0 (–)	0 (–)	0 (–)	0 (–)
		i-dual($h_{\max}$,$h_{\max}$)	0 (–)	0 (–)	0 (–)	0 (–)
		i-dual(h_{lmc},$h_{\max}$)	0 (–)	0 (–)	0 (–)	0 (–)
		i-dual(h_{lmc},h_0)	0 (–)	0 (–)	0 (–)	0 (–)
		i-dual(h_{add},h_0)	30 (36.2 ± 37.6)	25 (84.1 ± 68.2)	15 (89.9 ± 63.3)	9 (294.8 ± 303.3)
		i-dual(h_{add},h_{add})	30 (57.8 ± 101.5)	27 (129.9 ± 115.6)	17 (386.9 ± 276.3)	9 (289.2 ± 214.9)
	h = 2	dual-lp	0 (–)	0 (–)	0 (–)	0 (–)
		i-dual($h_{\max}$,$h_{\max}$)	0 (–)	0 (–)	0 (–)	0 (–)
		i-dual(h_{lmc},$h_{\max}$)	0 (–)	0 (–)	0 (–)	0 (–)
		i-dual(h_{lmc},h_0)	0 (–)	0 (–)	0 (–)	0 (–)
		i-dual(h_{add},h_0)	26 (128.0 ± 124.9)	14 (193.6 ± 162.4)	10 (268.8 ± 128.4)	2 (545.8 ± 160.3)
		i-dual(h_{add},h_{add})	29 (223.8 ± 171.5)	15 (442.2 ± 272.8)	8 (386.2 ± 315.0)	1 (1478.0 ± n/a)

Table 2: Results of the elevator experiment presented as coverage and average cpu-time (with 95% confidence interval) in seconds. The problems parameters are: number of elevators (e), number of calls to appear (h), and number of known calls (w).

problem	# of blocks	threshold	dual-lp	i-dual(h_{lmc},$h_{\max}$)	i-dual(h_{add},h_{add})
p01	5	0.10	30 (849.2 ± 108.47)	30 (9.7 ± 0.33)	30 (9.0 ± 0.29)
p02	5	0.07	14 (1603.1 ± 85.36)	30 (865.1 ± 38.86)	30 (665.3 ± 16.95)
p03	6	0.91	0 (–)	30 (14.4 ± 0.57)	30 (7.9 ± 0.25)
p04	6	0.16	0 (–)	30 (1466.1 ± 34.50)	30 (1153.1 ± 48.34)
p05	7	0.01	0 (–)	30 (2.3 ± 0.08)	30 (0.8 ± 0.03)
p06	8	0.30	0 (–)	0 (–)	0 (–)
p07 (r)	8	0.50	0 (–)	30 (1.8 ± 0.09)	30 (0.2 ± 0.01)
p08 (r)	8	0.63	0 (–)	30 (105.4 ± 6.15)	30 (16.3 ± 2.98)
p09 (r)	8	0.40	0 (–)	30 (301.2 ± 12.68)	30 (166.7 ± 6.53)

Table 3: Results of the exploding blocks world experiment presented as coverage and average cpu-time (with 95% confidence interval) in seconds. Problems marked with (r) were reduced by keeping only the first 8 blocks. Threshold is the constrain value over maximum expected number of exploded blocks allowed per problem.

	Problem	Planner	Avg. # of vis. states	Average % time computing Enc.	$\vec{H}$	x
SAR (n, r, d)	$4, 0.5, 4$	dual-lp	102,928	7.2%	–	92.4%
		h_{lmc},$h_{\max}$	4,129	1.6%	1.3%	96.3%
		h_{add},h_{add}	3,693	1.9%	0.4%	96.7%
	$5, 0.25, 4$	dual-lp	53,075	21.3%	–	77.9%
		h_{lmc},$h_{\max}$	1,459	3.1%	7.4%	88.2%
		h_{add},h_{add}	1,263	3.7%	2.5%	92.0%
	$5, 0.75, 3$	h_{lmc},$h_{\max}$	6,015	1.4%	1.8%	95.8%
		h_{add},h_{add}	4,604	1.9%	0.7%	96.3%
Elevators (e, h, w)	$1, 2, 2$	dual-lp	5,317	43.1%	–	52.5%
		$h_{\max}$,$h_{\max}$	1,478	5.4%	47.4%	43.9%
		h_{lmc},$h_{\max}$	1,255	0.4%	95.3%	3.7%
		h_{lmc},h_0	1,258	0.4%	95.9%	3.1%
		h_{add},h_0	529	6.5%	32.5%	55.2%
		h_{add},h_{add}	486	5.1%	60.2%	30.1%
	$2, 1, 2$	dual-lp	117,819	6.2%	–	93.5%
		$h_{\max}$,$h_{\max}$	10,641	2.9%	2.3%	93.4%
		h_{lmc},$h_{\max}$	9,240	2.3%	38.9%	57.5%
		h_{lmc},h_0	10,748	3.7%	41.6%	53.0%
		h_{add},h_0	1,034	5.4%	4.1%	88.2%
		h_{add},h_{add}	957	8.3%	15.4%	72.4%
	$3, 2, 2$	h_{add},h_0	4,347	3.6%	14.5%	80.7%
		h_{add},h_{add}	4,929	2.5%	14.7%	81.7%
Exploding BW	p01	dual-lp	342,650	2.5%	–	97.2%
		h_{lmc},$h_{\max}$	3,466	10.5%	4.0%	83.8%
		h_{add},h_{add}	3,388	10.8%	1.0%	86.4%
	p02	dual-lp	359,343	1.4%	–	98.4%
		h_{lmc},$h_{\max}$	19,730	1.7%	0.2%	97.5%
		h_{add},h_{add}	17,302	1.9%	0.0%	97.5%
	p04	h_{lmc},$h_{\max}$	30,689	1.7%	0.3%	97.4%
		h_{add},h_{add}	28,576	1.8%	0.1%	97.6%

Table 4: Average number of states explored and percentage of the total cpu-time spent encoding LP 3, computing the vector of heuristics $\vec{H}$, and solving LP 3 for selected problems in Tables 1 to 3. h_X, h_Y represents i-dual(h_X, h_Y). The dimension of $\vec{H}$ for the search and rescue (SAR), elevators, and exploding blocks world problems is 2, 2, and $1 + 2(w + h)$, respectively.

dual LP for the problems that the latter can solve (p01 and p02). As shown in Table 4, i-dual obtains this performance by using the heuristics to prune a large portion of the reachable state space. By extending the cpu-time cutoff to 2 hours, i-dual using h_{add} was able to solve p06 while i-dual using h_{lmc} was still unable to find the optimal solution.

5 Conclusion, Related and Future Work

The i-dual algorithm combines heuristic search and the dual LP formulation of SSPs to optimally solve constrained SSPs resulting, to the best of our knowledge, in the first heuristic search algorithm for constrained SSPs (C-SSPs). I-dual is able to efficiently encode and enforce secondary cost constraints, and to focus the search on regions that are relevant to the optimal stochastic policy, using (admissible) value function heuristics. This results in up to two orders of magnitude improvements in run-time and memory when compared to C-SSP algorithms based on linear programming alone. Furthermore, i-dual is able to trade-off optimality with scalability and quickly compute policies that still respect the constraints, an option not offered by the primal linear programming approach.

I-dual handles dead ends by assigning a penalty for reaching such states; nonetheless, i-dual can be easily adapted to use other approaches for handling dead ends in a similar spirit to S3P (Teichteil-Königsbuch 2012b) and iSSPUDE (Mausam and Kolobov 2012). This adaptation consists in solving two C-SSPs: (i) a C-SSP with the probability to reach the goal as primary cost and the constraints on k secondary costs; and (ii) a C-SSP with the desired primary cost as objective function and $k + 1$ constrained secondary costs, namely, the original k secondary costs and the probability of reaching the goal upper bounded by the result of (i). Note that no dead-end penalty is necessary for either C-SSPs and that the C-SSP in (i) is the constrained version of the *max-prob* criterion for SSPs (Mausam and Kolobov 2012).

In terms of related work, Altman (1999) was one of the first to research linear and dynamic programming algorithms for a range of constrained Markov Decision Processes. Dolgov and Durfee (2005) extend the work of Altman to the generation of deterministic policies for constrained MDPs, including MDPs with different discount factors for the various secondary costs, for which the generation of optimal stochastic policies is an open problem. By replacing the LP with a Mixed-Integer Linear Program, i-dual could also be used to generate optimal deterministic policies for C-SSPs, albeit more efficiently than (Dolgov and Durfee 2005). This is part of our future work agenda. Even though stochastic policies dominate deterministic ones for C-SSPs and computing the latter is known to be NP-complete (Feinberg 2000), deterministic policies can be useful in domains where policies need to be understandable and trusted by humans (Khan, Poupart, and Black 2009), or in multi-agent domains where randomisation can lead to miscoordination (Paruchuri et al. 2004).

Poupart and co-authors (2015) introduce an approximate linear programming algorithm for constrained partially observable MDPs that uses some of the ingredients of i-dual, notably the search in occupation measures space by using LPs. Their goal is to manage a potentially infinite belief state space using iterative refinement of finite set of beliefs. When applied to MDPs, their algorithm is equivalent to plain linear programming. I-dual differs from this approach by performing incremental envelope expansion instead of iterative refinement; furthermore, their algorithm does not make use of heuristics to guide the search. It would be interesting to combine the main advantages of their algorithm and i-dual to deal with POMDPs.

Santana, Thiébaux, Williams (2016) propose RAO*, an extension of AO* for finite-horizon POMDPs with *chance constraints*, i.e., bounds on the probability of a constraint being violated. In the particular case where their violation cause policy termination, chance constraints fall within the range of constraints over expectations handled by i-dual. Similarly to i-dual, RAO* uses heuristics on the chance constraints for early detection of constraints violations. Besides the difference in the constraints handled and the focus on partial observability, RAO* differs from i-dual by perform-

ing heuristic search in the primal space and considering only deterministic policies.

Trevizan and Veloso (2012) introduce an algorithm for *unconstrained* SSPs that shares with i-dual the usage of subproblems with artificial goals. Similar to i-dual, their algorithm uses heuristic values as the terminal costs of the artificial goals to guide the search towards the goals of the original problem. Their algorithm differs from i-dual by performing search in the primal space and by using the subproblems in an online planning and execution fashion, while still providing execution guarantees. Our future work agenda includes combining both algorithms to obtain a version of i-dual for planning and execution that could offer guarantees regarding constraint violation during execution.

Teichteil, Sprauel, and Kolobov (2012a, 2014) consider the generation of optimal stochastic policies for discounted MDPs with path constraints expressed in probabilistic computational tree logic (pCTL). They describe iterative linear and dynamic programming algorithms that improve on algorithms used in the probabilistic model-checking community (Kwiatkowska and Parker 2013, Baier et al. 2004). Having an efficient algorithm for C-SSPs which could be used or extended to deal with path constraints in probabilistic temporal logic is one of the motivation that lead to i-dual, and this extension is an item on our future work agenda.

Acknowledgements

This research was partially funded by AFOSR grant FA2386-15-1-4015 and NICTA. NICTA is funded by the Australian Government through the Department of Communications and the Australian Research Council through the ICT Centre of Excellence Program. We would also like to thank Patrik Haslum for fruitful discussions which helped to improve this paper, and the anonymous reviewers for their constructive and helpful comments.

References

Altman, E. 1999. *Constrained Markov Decision Processes*, volume 7. CRC Press.

Baier, C.; Größer, M.; Leucker, M.; Bollig, B.; and Ciesinski, F. 2004. Controller synthesis for probabilistic systems. In *Proc. Int. Conf. on Theoretical Computer Science (TCS)*.

Bertsekas, D., and Tsitsiklis, J. 1991. An Analysis of Stochastic Shortest Path Problems. *Mathematics of Operations Research* 16(3):580–595.

Bertsekas, D., and Tsitsiklis, J. N. 1996. *Neuro-Dynamic Programming*. Athena Scientific.

Bonet, B., and Geffner, H. 2003. Labeled RTDP: improving the convergence of real-time dynamic programming. In *Proc. Int. Conf. on Automated Planning and Scheduling*.

Bryce, D., and Buffet, O. 2008. 6th International Planning Competition: Uncertainty Track. In *3rd Int. Probabilistic Planning Competition (IPPC-ICAPS'08)*.

D'Epenoux, F. 1963. A probabilistic production and inventory problem. *Management Science* 10:98–108.

Dolgov, D. A., and Durfee, E. H. 2005. Stationary deterministic policies for constrained mdps with multiple rewards, costs, and discount factors. In *Proc. Int. Joint Conf. on Artificial Intelligence*.

Feinberg, E., and Shwarz, A. 1995. Constrained discounted dynamic programming. *Math. of Operations Research* 21:922–945.

Feinberg, E. A. 2000. Constrained discounted markov decision processes and hamiltonian cycles. *Math. Oper. Res.* 25(1):130–140.

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

Helmert, M., and Domshlak, C. 2009. Landmarks, critical paths and abstractions: What's the difference anyway? In *Proc. Int. Conf. on Automated Planning and Scheduling*.

Howard, R. 1960. *Dynamic Programming and Markov Processes*. MIT Press.

Jimenez, S.; Coles, A.; and Smith, A. 2006. Planning in probabilistic domains using a deterministic numeric planner. In *Proc. Workshop of the UK Planning and Scheduling Special Interest Group*.

Kallenberg, L. 1983. *Linear Programming and Finite Markovian Control Problems*. Math. Centrum, Amsterdam.

Khan, O. Z.; Poupart, P.; and Black, J. P. 2009. Minimal sufficient explanations for factored markov decision processes. In *Proc. Int. Conf. on Automated Planning and Scheduling*.

Kwiatkowska, M. Z., and Parker, D. 2013. Automated verification and strategy synthesis for probabilistic systems. In *Int. Symp. on Automated Technology for Verification and Analysis*.

Mausam, and Kolobov, A. 2012. *Planning with Markov Decision Processes*. Morgan & Claypool.

Paruchuri, P.; Tambe, M.; Ordóñez, F.; and Kraus, S. 2004. Towards a formalization of teamwork with resource constraints. In *Proc. Int.l Joint Conf. Autonomous Agents and Multiagent Systems*.

Poupart, P.; Malhotra, A.; Pei, P.; Kim, K.-E.; Goh, B.; and Bowling, M. 2015. Approximate linear programming for constrained partially observable markov decision processes. In *Proc. AAAI Conf. on Artificial Intelligence*.

Puterman, M. 1994. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc.

Santana, P.; Thiébaux, S.; and Williams, B. 2016. RAO*: an algorithm for chance constrained POMDPs. In *Proc. AAAI Conference on Artificial Intelligence*.

Sprauel, J.; Kolobov, A.; and Teichteil-Königsbuch, F. 2014. Saturated path-constrained MDP: planning under uncertainty and deterministic model-checking constraints. In *Proc. AAAI Conf. on Artificial Intelligence*.

Teichteil-Königsbuch, F.; Vidal, V.; and Infantes, G. 2011. Extending Classical Planning Heuristics to Probabilistic Planning with Dead-Ends. In *Proc. AAAI Conf. on Artificial Intelligence*.

Teichteil-Königsbuch, F. 2012a. Path-Constrained Markov Decision Processes: bridging the gap between probabilistic model-checking and decision-theoretic planning. In *Proc. European Conf. on Artificial Intelligence*.

Teichteil-Königsbuch, F. 2012b. Stochastic safest and shortest path problems. In *Proc. AAAI Conf. on Artificial Intelligence*.

Trevizan, F. W., and Veloso, M. M. 2012. Short-sighted stochastic shortest path problems. In *Proc. Int. Conf. on Automated Planning and Scheduling*.

Undurti, A., and How, J. P. 2010. An online algorithm for constrained pomdps. In *Proc. IEEE Int. Conf. on Robotics and Automation*.

Younes, H. L. S.; Littman, M. L.; Weissman, D.; and Asmuth, J. 2005. The first probabilistic track of the international planning competition. *J. Artif. Intell. Res. (JAIR)* 24:851–887.