Trajectory Constraint Heuristics for Optimal Probabilistic Planning

John R. Peterson[1], **Anagha Kulkarni**[2], **Emil Keyder**[2], **Joseph Kim**[2], **Shlomo Zilberstein**[1]

[1] University of Massachusetts Amherst, MA, USA
[2] Invitae Corporation, San Francisco, CA, USA
{jrpeterson, shlomo}@cs.umass.edu, {emil.keyder, anagha.kulkarni, joseph.kim}@invitae.com

Abstract

Search algorithms such as LAO* and LRTDP coupled with admissible heuristics are widely used methods for optimal probabilistic planning. Their effectiveness depends on the degree to which heuristics are able to approximate the optimal cost of a state. Many common domain-independent heuristics, however, rely on determinization, and ignore the probabilities associated with different effects of actions. Here, we present a method for decomposing a probabilistic planning problem into subproblems by constraining possible action outcomes. Admissible heuristics evaluated for each subproblem can then be combined via a weighted sum to obtain an admissible heuristic for the original problem that takes into account a limited amount of probabilistic information. We use this approach to derive new admissible heuristics for probabilistic planning, and show that for some problems they are significantly more informative than existing heuristics, giving up to an order of magnitude speedup in the time to converge to an optimal policy.

1 Introduction

In optimal probabilistic planning, planners must explore the state space of a problem and find a policy with minimum expected cost. The most effective techniques for solving such problems use admissible heuristics in conjunction with optimal search algorithms such as LAO* (Hansen and Zilberstein 2001) or LRTDP (Bonet and Geffner 2003b). Such heuristics can also benefit fast approximate solvers such as FLARES (Pineda, Wray, and Zilberstein 2017). While a number of different methods for obtaining admissible domain-independent heuristics have been proposed, these methods often use a form of *determinization*, which corresponds to an assumption that the agent can freely pick between the possible outcomes of an action instead of these occurring probabilistically. Once problems have been determinized, heuristics can be obtained by computing shortest paths in the directed graph representing the state space (Bonet and Geffner 2003b), or using other techniques from the classical planning literature (Bonet and Geffner 2005; Teichteil-Königsbuch, Vidal, and Infantes 2011).

While determinization provides a convenient way to modify a problem so that it is easier to compute admissible heuristics, estimates obtained this way can be uninformative. In addition to any inherent limitations – such as ignoring delete effects – heuristics may assume that an outcome with arbitrarily low probability occurs when an action is applied. Taking this to an extreme, the addition of a 0-cost action that can be applied in the initial state and leads to the goal with probability ϵ but a dead end with probability $1 - \epsilon$ will lead to a heuristic estimate of 0 on the determinized problem.

A less extreme version of this phenomenon can be observed in a family of problems that is of particular interest here, referred to as *information-gathering domains*. In these domains, an agent must gather various pieces of information about the world, each of which has a known multinomial distribution over its possible values, and then choose among strategies of differing cost that are enabled or ruled out by the discovered information. In this setting, heuristics based on determinization always assume the values of information variables that allow the cheapest possible course of action, and therefore underestimate the true cost (which is closer to a weighted average of the costs of the strategies, with the weight for each given by the likelihood of the information enabling the strategy). Such domains motivate the heuristics developed in this paper, and are discussed in Section 6.

To address this limitation of determinization-based heuristics, we introduce the notion of *trajectory constraints*, which limit the outcomes of probabilistic actions to a subset of their outcomes in the original problem. We show that a carefully chosen set of such constraints induces a subproblem whose optimal cost reflects the cost of the original problem when the probabilistic outcomes of the constrained actions match those specified in the constraints. If a *set* of trajectory constraints is chosen such that its elements are *pairwise disjoint* and *exhaustive* (defined formally below), admissible estimates computed for each subproblem can be combined with an appropriate weighting to obtain a globally more informative, but still admissible, heuristic. The weighting for a subproblem is computed roughly as the probability of the imposed constraint occurring in the original problem. These estimates are more informative than the underlying base heuristic evaluated on the original problem, even when the base heuristic is itself based on determinization.

We now introduce the basic formalisms and notation used in this paper, and briefly discuss related work. We then formally define trajectory constraints, discuss their properties, and introduce our heuristic approach. Finally, we present the information-gathering domains that motivated this work, and conclude with experimental results, showing that heuristics with trajectory constraints can be more informative than their counterparts computed on the original problem. In our experiments, the approach yields up to an order-of-magnitude speedup in computing an optimal policy.

2 Background & Related Work

Problem. A Stochastic Shortest Path Problem (SSP) is a tuple $M = \langle S, A, T, C, s_0, S_g \rangle$, where S is a finite set of states, A a finite set of actions, T a transition function mapping $S \times A \times S \to [0, 1]$, with $T(s, a, s')$ the probability of transitioning to $s' \in S$ when $a \in A$ is applied in $s \in S$, C a cost function $S \times A \to \mathbb{R}^{\geq 0}$, where $C(s, a)$ is the immediate cost of taking action $a \in A$ in state $s \in S$, $s_0 \in S$ the initial state, and $S_g \subseteq S$ the set of goal states. We assume that $C(s, a) = \infty$ when a is inapplicable in s.

A solution to an SSP is a policy $\pi : S \to A$ indicating an action to be taken in each state, where an *optimal policy* π^* minimizes the expected cumulative cost to a goal from the initial state $V^\pi(s_0)$, where

$$V^\pi(s) = C(s, \pi(s)) + \sum_{s' \in S} T(s, \pi(s), s') V^\pi(s') \qquad (1)$$

if $s \notin S_g$ and 0 otherwise. A *proper* policy reaches some $s_g \in S_g$ with probability 1. In this work we assume that a proper policy exists for any SSP M. We denote an ordered sequence of transitions as $\lambda = \langle (s_1, a_1, s'_1), \ldots, (s_k, a_k, s'_k) \rangle$, where $\forall\, i$, $T(s_i, a, s'_i) > 0$, and write the subsequence of λ corresponding to applications of a particular action a as $\lambda^a = \langle (s, a, s') \mid (s, a, s') \in \lambda \rangle$. We call a sequence λ, where $\forall\, i, 1 < i \leq k$, $s'_{i-1} = s_i$ and $s'_k \in S_g$, a *trajectory*, and denote it with τ. We write $\pi \models \tau$ to denote that a trajectory τ is possible under π. We define the probability of a sequence of transitions λ (or trajectory τ) to be $p(\lambda) = \prod_{(s,a,s') \in \lambda} T(s, a, s')$, and its cost to be $\mathcal{C}(\lambda) = \sum_{(s,a,s') \in \lambda} C(s, a)$.

In this work, we assume a STRIPS-like representation of SSPs, given by $\Pi = \langle F, I, O, \mathcal{C}, G \rangle$, where F is a set of fluents, the full state set is a subset of the power set of F denoted $\mathcal{P}(F)$, with state $s \subseteq F$ described by the set of fluents true in s, $I \subseteq F$ is the initial state, $G \subseteq F$ is the goal, with $S_g = \{s \mid G \subseteq s\}$, O is the set of operators, where each operator $o \in O$ is given by a precondition $\mathsf{pre}(o)$ and a set of n probabilistic effects $\mathsf{eff}(o) = \{e^1_o, \ldots, e^n_o\}$ with respective probabilities $p^1_o, \ldots, p^n_o$ such that $\sum_{i=1}^n p^i_o = 1$[1], and each effect is of the form $e^i_o = \langle \mathsf{add}(e^i_o), \mathsf{del}(e^i_o) \rangle$, and $\mathcal{C}$ is a cost function $O \to \mathbb{R}^{\geq 0}$. An operator o is *applicable* in s if $\mathsf{pre}(o) \subseteq s$. We denote the result of an effect e^i_o in s with $s[e^i_o] = s \setminus \mathsf{del}(e^i_o) \cup \mathsf{add}(e^i_o)$. We denote the set of states $\{s \mid \mathsf{pre}(o) \subseteq s\}$ in which o is applicable with $S_{\mathsf{pre}(o)}$.

Heuristics. A heuristic function $h : S \to \mathbb{R}^{\geq 0}$ estimates $V^{\pi^*}(s)$, and h is said to be *admissible* if $h(s) \leq V^{\pi^*}(s)$ for all s. Admissible heuristics are typically computed as exact or lower bounds on the costs of *relaxed* versions of the original problem.

A variety of admissible domain-independent heuristics have been developed for both classical and probabilistic planning. In the deterministic setting, these include those based on the delete relaxation (Bonet and Geffner 2001), landmarks (Helmert and Domshlak 2009), merge-and-shrink abstractions (Helmert et al. 2014), and others. In the probabilistic setting, heuristics are often based on classical techniques applied to a determinized problem. For example, h_{min} (Bonet and Geffner 2003b) is obtained as the optimal cost of this problem, computed as the shortest path in the problem's state space. When the problem is described in languages such as PPDDL (Younes and Littman 2004) or RDDL (Sanner 2010), heuristics developed for deterministic planning that operate on problem descriptions in terms of fluents and operators have been successfully extended and applied to SSPs (Bonet and Geffner 2005; Teichteil-Königsbuch, Vidal, and Infantes 2011). Recently, *occupation measure* heuristics have been developed that directly incorporate probabilistic information (Trevizan, Thiébaux, and Haslum 2017), and that are similar to operator-counting heuristics (Pommerening et al. 2014) in deterministic planning. Other approaches have extended pattern database heuristics developed for classical planning to probabilistic planning (Klößner et al. 2021).

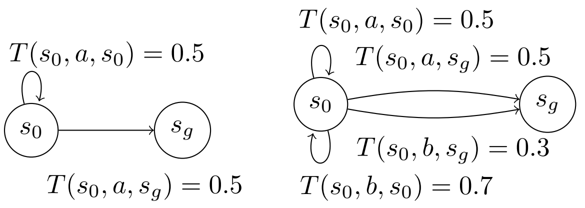
The heuristics introduced in this paper use h_{max}, an admissible classical planning heuristic that computes estimates in the problem relaxation in which delete effects are ignored (Bonet and Geffner 2001), as a base heuristic. To derive a polynomial approximation of the NP-hard optimal delete relaxation heuristic, h_{max} makes use of the independence assumption and computes estimates of the cost of a set of fluents as the cost of the most expensive fluent in that set.

There is also a rich literature of approximate solution techniques for SSPs, including approaches involving replanning (Yoon, Fern, and Givan 2007; Teichteil-Königsbuch, Kuter, and Infantes 2010; Yoon et al. 2010; Keyder and Geffner 2008b), short-sightedness (Bonet and Geffner 2003a; Pineda, Wray, and Zilberstein 2017), and sampling (Kearns, Mansour, and Ng 2002; Kocsis and Szepesvári 2006; Keller and Helmert 2013).

Search Algorithms. Optimal policies for SSPs are generally computed with heuristic search algorithms such as LAO* (Hansen and Zilberstein 2001), RTDP (Barto, Bradtke, and Singh 1995), and LRTDP (Bonet and Geffner 2003b). Used with admissible heuristics, these algorithms search for an optimal *partial* policy, in which the best action is computed only for states that are potentially reached under the policy. When a policy π is proper, Equation 1 is well-defined for the set of states reachable under π.

Notation. In the following sections, we use $\vec{c}$ to denote an ordered sequence, and $\vec{c}[i]$ its ith element. $\vec{c}[[i = c']]$ denotes $\vec{c}$ in which $\vec{c}[i]$ has been replaced with c', but all other values are unchanged. We sometimes abuse notation and write $v \in \vec{c}$ to mean $\exists i(\vec{c}[i] = v)$. $\vec{v}^k$ for a scalar value v denotes the sequence consisting of k repetitions of v.

[1] Languages such as PPDDL describe an operator by enumerating a set of independent probabilistic effects, here we assume a set of effects of which only one is triggered on action application.



(a) SSP 1, $A = \{a\}$ (b) SSP 2, $A = \{a, b\}$

Figure 1: Two simple SSPs, both with cost $h^*(s_0) = 2$.

3 Trajectory Constraints

While efficient, determinization-based admissible heuristics are commonly used to guide heuristic search for SSPs, they tend to provide poor heuristic estimates as they ignore outcome probabilities and may therefore dramatically underestimate the true expected cost to the goal. Trajectory constraints try to overcome these disadvantages by constructing *multiple* deterministic subproblems and limiting the action outcomes available in each for the first few applications of the constrained actions. Standard all-outcome determinizations can then be constructed for each subproblem separately. Heuristics evaluated on these subproblems are more informed in aggregate, as they are forced to consider a larger set of outcomes, instead of picking the most convenient one.

Consider the example shown in Figure 1a, where a is assumed to have unit cost $C(\cdot, a) = 1$. To solve this problem, we might consider two separate subproblems; one in which the first application of a deterministically results in the desired transition to s_g, and one in which it deterministically results in the self-loop transition to s_0. In both subproblems, the outcome of a is unconstrained after that initial application. Solving the determinized versions of these two subproblems gives optimal costs of 1 and 2 respectively, which can be combined by weighting each with the probability of the constraint for the associated subproblem. This gives a heuristic estimate of $(0.5 * 1) + (0.5 * 2) = 1.5$, improving over the $h^*(s_0) = 1$ estimate for the cost of the standard all-outcomes determinization. By constructing more subproblems that enforce constraints on additional applications of a, it is possible to obtain even more accurate estimates. When the first two applications of a both result in s_0, the cost is 3, if the first results in s_0 and the second in s_g, the cost is 2, and the cost of the other subproblem in which the first application of a results in s_g is unchanged at 1, leading to a heuristic estimate of $(0.25{*}3)+(0.25{*}2)+(0.5{*}1) = 1.75$.

To specify the construction of subproblems such as those in the example above, we now introduce the idea of *trajectory constraints*, which comprise all of the information necessary to describe a *single* subproblem. Informally, a trajectory constraint γ maps each action a in a subset $\gamma^a \subseteq A$ to a constraint χ_a, made up of a sequence of sets $\langle \sigma_{a1}, \dots, \sigma_{an} \rangle$ in which σ_{aj} describes the set of possible outcomes the jth time that a is applied. The possible outcomes are described as a set of tuples (s^q_{aj}, S^q_{aj}) for every state s^q_{aj} in which a is applicable (has non-infinite cost), with the only allowed transitions from s^q_{aj} under a being $s' \in S^q_{aj}$. More formally:

Definition 1 (Trajectory constraint)**.** *Given an SSP* $M = \langle S, A, T, C, s_0, S_g \rangle$*, a trajectory constraint* γ *consists of a set of pairs* $\{(a_1, \chi_{a_1}), \dots, (a_k, \chi_{a_k})\}$*, where each* χ_{a_i} *is an ordered sequence* $\langle \sigma_{a_i 1}, \dots, \sigma_{a_i n} \rangle$*, and each* $\sigma_{a_i j}$ *is a set of tuples* $\{(s^1_{a_i j}, S^1_{a_i j}), \dots, (s^q_{a_i j}, S^q_{a_i j})\}$*, where* $\forall i, l,\ a_i \neq a_l, \forall (s^r_{a_i j}, S^r_{a_i j}) \in \sigma_{a_i j},\ S^r_{a_i j} \neq \emptyset, \forall s' \in S^r_{a_i j},\ T(s^r_{a_i j}, a_i, s') > 0$*, and* $\{s \mid (s, S) \in \sigma_{aj}\} = \{s \mid C(s, a) \neq \infty\}$.

Example 1. *The subproblem in which* a *results in the self-loop* $s_0 \rightarrow s_0$ *the first two times it is applied is described by the trajectory constraint*

$$\gamma = \{(a, \langle \{(s_0, \{s_0\})\}, \{(s_0, \{s_0\})\} \rangle)\}$$

We refer to the set of actions $\{a \mid (a, \cdot) \in \gamma\}$ constrained by γ as γ^a. We say that a sequence of transitions λ^a *complies* with γ if $a \notin \gamma^a$ or $\forall j,\ 1 \leq j \leq \min(|\lambda^a|, |\chi_a|), \lambda^a[j] = (s_j, a, s'_j) \wedge \exists (s, S) \in \sigma_{aj},\ (s = s_j \wedge s'_j \in S)$. In other words, a sequence of transitions for a complies with γ if either a is unconstrained by γ, or if the transition at every application index j that is constrained by γ is one of the transitions listed in σ_{aj}. We say that a sequence λ complies with γ, denoted $\gamma \models \lambda$, if λ^a complies with γ for all $\{a \mid (s, a, s') \in \lambda\}$.

Given an SSP M and a trajectory constraint γ, we can now formulate an SSP M^γ that incorporates γ as follows:

Definition 2 (Trajectory constrained SSP)**.** *Given an SSP* $M = \langle S, A, T, C, s_0, S_g \rangle$ *and a trajectory constraint* $\gamma = \{\langle a_1, \chi_{a_1} \rangle, \dots, \langle a_k, \chi_{a_k} \rangle\}$*, the* trajectory-constrained problem M^γ *is given by* $\langle S^\gamma, A^\gamma, T^\gamma, C^\gamma, s^\gamma_0, S^\gamma_g \rangle$*, where*

- $S^\gamma = \{(s, \vec{\boldsymbol{c}}) \mid s \in S \wedge |\vec{\boldsymbol{c}}| = k \wedge$
  $\forall i,\ 1 \leq i \leq k,\ \vec{\boldsymbol{c}}[i] \leq |\chi_{a_i}| \wedge \vec{\boldsymbol{c}}[i] \in \mathbb{Z}^{\geq 0}$
- $A^\gamma = A$
- $T^\gamma((s, \vec{\boldsymbol{c}}), a, (s', \vec{\boldsymbol{c'}})) =$
  $$\begin{cases} T(s, a, s') & \textit{if } \vec{\boldsymbol{c}} = \vec{\boldsymbol{c'}} \\ & \quad \wedge (a \notin \gamma^a \vee\ (a = a_i \in \gamma^a \wedge \vec{\boldsymbol{c}}[i] = |\chi_i|)) \\ \frac{T(s,a,s')}{\alpha^j_{sa}} & \textit{if } a = a_i \in \gamma^a \wedge j = \vec{\boldsymbol{c}}[i] + 1 \wedge (s, S') \in \sigma_{a_i j} \\ & \quad \wedge s' \in S' \wedge \vec{\boldsymbol{c'}} = \vec{\boldsymbol{c}}[[i = j]] \\ 0 & \textit{otherwise} \end{cases}$$
  where $\alpha^j_{sa} = \sum_{\{s'' \mid (s, S') \in \sigma_{a_i j} \wedge s'' \in S'\}} T(s, a, s'')$.
- $C^\gamma((s, \vec{\boldsymbol{c}}), a) = C(s, a)$
- $s^\gamma_0 = (s_0, \vec{0}^k)$
- $S^\gamma_g = \{(s, \vec{\boldsymbol{c}}) \mid s \in S_g \wedge (s, \vec{\boldsymbol{c}}) \in S^\gamma\}$

In order to ensure that the constraint on the action outcomes is satisfied, the states of the original SSP are augmented with a sequence $\vec{\boldsymbol{c}}$ whose values are non-negative integers that track the number of times that each of the constrained actions $a \in \gamma^a$ has been applied. In the initial state s^γ_0, this value is 0 for all $a \in \gamma^a$. The transition function T^γ is identical to T for $a \notin \gamma^a$ as long as $\vec{\boldsymbol{c}}$ remains unchanged. When $a = a_i \in \gamma^a$ and its current application count is less than the number of constrained applications, $T^\gamma((s, \vec{\boldsymbol{c}}), a_i, (s', \vec{\boldsymbol{c'}}))$ is renormalized with a denominator that considers only the constrained outcomes available at state s for the jth application of a_i, and transitions are restricted to ensure that the counts vector $\vec{\boldsymbol{c}}$ is correctly updated by incrementing the corresponding entry. After a_i has been applied $|\chi_{a_i}|$ times, $\vec{c}[i] = |\chi_{a_i}|$ becomes true and the transition function is identical to that of the original problem. Goal states in M remain goal states in M^γ regardless of the value of $\vec{c}$, and the cost function is unchanged.

Example 2. *Incorporating γ from Example 1 into the SSP in Figure 1a, initial state $(s_0, \langle 0 \rangle)$ encodes that a has been applied 0 times. On the first application of a, the only allowed transition is to $(s_0, \langle 1 \rangle)$ with probability 1, since the numerator and denominator in the second case of T^γ are equal. The same is true for the second application of a in $(s_0, \langle 1 \rangle)$. Once state $(s_0, \langle 2 \rangle)$ is reached, there are no further constraints and a transitions to $(s_0, \langle 2 \rangle)$ or $(s_g, \langle 2 \rangle)$ (the only reachable goal state) with equal probability.*

4 Trajectory Constraint Heuristics

We now turn to the problem of how to use trajectory-constrained SSPs to obtain more informative heuristic estimates. Intuitively, for a *set* of trajectory constraints to be useful for the purposes of defining a heuristic, it must satisfy two conditions: (i) the trajectory constraints must not "overlap" with each other in a way that leads to overcounting of cost, and (ii) all possible transition sequences must comply with at least one of the trajectory constraints in the set so that possible solutions are not omitted. We formalize these properties as *disjointness* and *exhaustiveness* respectively:

Definition 3 (Disjoint trajectory constraints)**.** *Given an SSP $M = \langle S, A, T, C, s_0, S_g \rangle$, trajectory constraints $\gamma_i = \{(a_1^i, \chi_{a_1^i}), \ldots, (a_n^i, \chi_{a_n^i})\}$ and $\gamma_j = \{(a_1^j, \chi_{a_1^j}), \ldots, (a_m^j, \chi_{a_m^j})\}$ are* disjoint *iff there exists an action $a \in \gamma_i^a \cap \gamma_j^a$ such that no sequence of transitions λ^a with $|\lambda^a| = \max(|\chi_a^i|, |\chi_a^j|)$ that complies with both γ_i and γ_j exists.*

Example 3. *For the example shown in Figure 1b,*

$$\gamma_1 = \{(a, \langle \{(s_0, \{s_0\})\}, \{(s_0, \{s_0\})\} \rangle), (b, \langle \{(s_0, \{s_0\})\} \rangle)\},$$
$$\gamma_2 = \{(a, \langle \{(s_0, \{s_0\})\}, \{(s_0, \{s_g\})\} \rangle), (b, \langle \{(s_0, \{s_0\})\} \rangle)\}$$

can be seen to be disjoint, considering action a. However, neither γ_1 nor γ_2 is disjoint with

$$\gamma_3 = \{(a, \langle \{(s_0, \{s_0\})\} \rangle), (b, \langle \{(s_0, \{s_0\})\} \rangle)\}.$$

Definition 4 (Exhaustive trajectory constraints)**.** *Given an SSP $M = \langle S, A, T, C, s_0, S_g \rangle$, a set of trajectory constraints $\Gamma = \{\gamma_1, \ldots, \gamma_n\}$ is* exhaustive *iff for any sequence of transitions λ, there exists a trajectory constraint $\gamma_i \in \Gamma$ such that λ complies with γ_i.*

Example 4. *The constraint set $\Gamma = \{\gamma_1, \gamma_2, \gamma_3\}$ with*

$$\gamma_1 = \{(a, \langle \{(s_0, \{s_0\})\}, \{(s_0, \{s_0\})\} \rangle), (b, \langle \{(s_0, \{s_0\})\} \rangle)\},$$
$$\gamma_2 = \{(a, \langle \{(s_0, \{s_0\})\}, \{(s_0, \{s_g\})\} \rangle), (b, \langle \{(s_0, \{s_0\})\} \rangle)\},$$
$$\gamma_3 = \{(a, \langle \{(s_0, \{s_g\})\} \rangle)\}$$

is not exhaustive, as there is no $\gamma_i \in \Gamma$ such that $\lambda = \langle (s_0, a, s_0), (s_0, b, s_g) \rangle$ complies with γ_i. Adding $\gamma_4 = \{(b, \langle \{(s_0, \{s_g\})\} \rangle)\}$ to Γ would make it exhaustive. However, note that γ_3 and γ_4 are not disjoint as $\langle (s_0, a, s_g) \rangle$ and $\langle (s_0, b, s_g) \rangle$ comply with both. Adding $\gamma_5 = \{(a, \langle \{(s_0, \{s_0\})\} \rangle), (b, \langle \{(s_0, \{s_g\})\} \rangle)\}$ to Γ instead would render it both exhaustive and pairwise disjoint.

If a set of trajectory constraints Γ is both pairwise disjoint and exhaustive, any execution trajectory possible in the original problem M is permitted by at least one of the constrained problems M^γ for $\gamma \in \Gamma$, and all transition sequences of sufficient length are partitioned among the various M^γ, even if shorter trajectories may belong to multiple.

Finally, we introduce a restriction on trajectory constraints that makes their behavior and properties easier to reason about:

Definition 5 (Regular trajectory constraint)**.** *Given an SSP $M = \langle S, A, T, C, s_0, S_g \rangle$, a trajectory constraint γ is* regular *iff for every $a \in \gamma^a$ and $\sigma_{aj} \in \chi_a$, $\exists p_{aj}$ such that for every tuple $(s, S) \in \sigma_{aj}$, $\sum_{s' \in S} T(s, a, s') = p_{aj}$.*

Regularity ensures that for each constrained action a at application index j, the summed transition probabilities out of any constrained state s total to a unique value p_{aj}. While regularity may at first seem an overly restrictive property, note that for typical state spaces expressed in PPDDL or RDDL descriptions of SSPs, it is naturally satisfied by choosing a subset $E \subseteq \mathsf{eff}(o)$ of the probabilistic effects of an operator o to be active the jth time that the action is executed. The choice of E naturally induces a set of tuples $\sigma_{oj} = \{(s, S) \mid s \in S_{\mathsf{pre}(o)} \wedge S = \{s[e_o^i] \mid e_o^i \in E\}\}$.

Given a set of regular trajectory constraints Γ that is pairwise disjoint and exhaustive, it can be shown that a weighted sum of admissible heuristics computed for $\{M^{\gamma_i} \mid \gamma_i \in \Gamma\}$ is also admissible, where the weight for each is given by the probability of the associated γ_i:

Definition 6 (Probability of a regular trajectory constraint)**.** *Given an SSP $M = \langle S, A, T, C, s_0, S_g \rangle$ and a regular trajectory constraint $\gamma = \langle (a_1, \chi_{a_1}), \ldots, (a_k, \chi_{a_k}) \rangle$, the probability p^γ of γ is given by $\prod_{\{\chi_{a_i} | (a_i, \chi_{a_i}) \in \gamma\}} p(\chi_{a_i})$, where for $\chi_{a_i} = \langle \sigma_{a_i 1}, \ldots, \sigma_{a_i n} \rangle$, $p(\chi_{a_i}) = \prod_{\sigma_{a_i j} \in \chi_{a_i}} p(\sigma_{a_i j})$, where $p(\sigma_{a_i j}) = \sum_{s' \in S} T(s, a, s')$, in which $(s, S) \in \sigma_{a_i j}$.*

Note that the choice of the specific (s, S) in the last part of the definition does not matter as the summed transition probabilities are guaranteed to be equal due to regularity. Intuitively, the probability of a trajectory constraint p^γ is the probability of the transitions indicated in the constraint occurring in the original unconstrained problem M.

Lemma 1. *For a set of constraints Γ that is pairwise disjoint, exhaustive, and regular, $\sum_{\gamma \in \Gamma} p^\gamma = 1$.*

This can be seen by observing that for any action a that is constrained in $\gamma_i \in \Gamma$ but not in $\gamma_j \in \Gamma$, or that has a different number of constrained repetitions in γ_i than γ_j, adding additional dummy χ_a (if not present) or additional dummy σ_{aj} (if the number of repetitions is different) that list all possible transitions for a results in an equivalent Γ in which a is constrained for the same number of repetitions in each $\gamma \in \Gamma$. If the possible transitions for $|\chi_a|$ instances of each a from a state s in which a is applicable are enumerated, it can be seen that each sequence of transitions complies with exactly one γ_i (since Γ is disjoint and exhaustive), and that the summed probability of the transitions belonging to γ_i is equal to p^{γ_i} by construction. Since all sequences of transitions for γ^a for a particular state s are enumerated, their probabilities sum to 1, and therefore $\sum_{\gamma \in \Gamma} p^\gamma = 1$.

Example 5. *In the trajectory constraint set $\Gamma = \{\gamma_1, \gamma_2, \gamma_3, \gamma_5\}$ defined in Example 4, all constraints are trivially regular, since $|\sigma_{aj}| = 1$ for all a, j. The constraint probabilities for constraints in Γ are: $p^{\gamma_1} = p^{\gamma_2} = 0.5 * 0.5 * 0.7 = 0.175$, $p^{\gamma_3} = 0.5$, and $p^{\gamma_5} = 0.5 * 0.3 = 0.15$, with $\sum_{\gamma \in \Gamma} p^\gamma = 0.175 + 0.175 + 0.5 + 0.15 = 1$.*

We now introduce the *weighted trajectory constraint heuristic*:

Definition 7 (Weighted Trajectory Constraint Heuristic)**.** *Let M be an SSP, $\Gamma = \{\gamma_1, \dots \gamma_k\}$ a set of trajectory constraints, and h a heuristic. Denote the estimate given by h evaluated on state s in SSP M' as $h(s, M')$. The* weighted trajectory constraint heuristic *h_{tc} for base heuristic h and Γ is given by:*

$$h_{tc}[h, \Gamma](s, M) = \sum_{\gamma \in \Gamma} p^\gamma h((s, \vec{0}^k), M^\gamma)$$

Theorem 1. *Given an SSP M, an admissible heuristic h, and a set of trajectory constraints $\Gamma = \{\gamma_1, \dots, \gamma_k\}$ that is pairwise disjoint, exhaustive, and regular, $h_{tc}[h, \Gamma]$ is an admissible heuristic for M.*

Proof Sketch. We give a proof sketch for the case where $|\chi| \leq 1$ for all constraints, and sometimes omit the notation of applied action counts for simplicity. Let π^M be an optimal policy for M, $V^{*\gamma}$ the optimal value function for M^γ, and $V^{\pi^{M,\gamma}}$ the value function in M^γ following policy $\pi^{M,\gamma}((s, \vec{\boldsymbol{c}})) := \pi^M(s)$. Since π^M is proper in M, actions are constrained by γ for a finite number of applications, and successors in M^γ have nonzero probability in M, $\pi^{M,\gamma}$ is proper in M^γ. Since h is admissible and $\forall \gamma \in \Gamma$ $V^{*\gamma}((s, \vec{0}^k)) \leq V^{\pi^{M,\gamma}}((s, \vec{0}^k))$,

$$\begin{aligned} \sum_{\gamma \in \Gamma} p^\gamma h((s, \vec{0}^k), M^\gamma) &\leq \sum_{\gamma \in \Gamma} p^\gamma V^{\pi^{*\gamma}}((s, \vec{0}^k)) && \textit{(admissibility)} \\ &\leq \sum_{\gamma \in \Gamma} p^\gamma V^{\pi^{M,\gamma}}((s, \vec{0}^k)) && \textit{(def. of } V^*\textit{)} \end{aligned}$$

We argue that $\sum_{\gamma \in \Gamma} p^\gamma V^{\pi^{M,\gamma}}((s, \vec{0}^k)) = V^{\pi^M}(s)$. First observe that the (potentially infinite) set of trajectories possible under π^M and the union of the sets of trajectories possible under each $\pi^{M,\gamma}$ are identical, since π is not modified and every possible outcome of each action is captured in some γ_i due to exhaustivity. We characterize $V^{\pi^M}(s)$ as the sum of the (potentially infinite) series of the product of the probability of reaching the goal via trajectory τ while following π^M and the cost of τ:

$$V^{\pi^M}(s) = \sum_{\{\tau \mid \pi^M \models \tau\}} p_M(\tau) \cdot \mathcal{C}(\tau)$$

Since $\mathcal{C}(\tau)$ is unchanged between M and M^γ, and for every $\{\tau \mid \pi^M \models \tau\}$ $\exists \gamma$ s.t. $\pi^{M,\gamma} \models \tau$, we must show that the effective multiplier for $\mathcal{C}(\tau)$ summed over all γ is equal to its probability in M:

$$\begin{aligned} &\sum_{\gamma \in \Gamma \wedge \gamma \models \tau} p^\gamma \cdot p_{M^\gamma}(\tau) \\ &= \sum_{\gamma \in \Gamma \wedge \gamma \models \tau} p^\gamma \cdot \prod_{\{(s,a,s') \in \tau | a \notin \gamma^a\}} T(s, a, s') \prod_{\{(s,a,s') \in \tau | a \in \gamma^a\}} T^\gamma(s, a, s') \\ &= \sum_{\gamma \in \Gamma \wedge \gamma \models \tau} \prod_{a \in \gamma^a} \alpha^\gamma_{sa} \cdot \prod_{\{(s,a,s') \in \tau | a \notin \gamma^a\}} T(s, a, s') \cdot \\ &\qquad \prod_{\{(s,a,s') \in \tau | a \in \gamma^a\}} \frac{T(s, a, s')}{\alpha^\gamma_{sa}} \\ &= p_M(\tau) \cdot \sum_{\gamma \in \Gamma \wedge \gamma \models \tau} \prod_{a \in \gamma^a} \alpha^\gamma_{sa} \cdot \prod_{\{(s,a,s') \in \tau | a \in \gamma^a\}} \frac{1}{\alpha^\gamma_{sa}} \\ &= p_M(\tau) \cdot \sum_{\gamma \in \Gamma \wedge \gamma \models \tau} \prod_{a \in \gamma^a \wedge a \notin \tau} \alpha^\gamma_{sa} \end{aligned}$$

where α_{sa} is defined in Definition 2.[2] We construct $\hat{\Gamma} = \{\hat{\gamma} \mid \gamma \in \Gamma \wedge \gamma \models \tau\}$ where $\hat{\gamma} = \{(a, \chi) \mid (a, \chi) \in \gamma \wedge a \notin \tau\}$, and argue that it is pairwise disjoint, exhaustive, and regular. Then $\sum_{\gamma \in \Gamma \wedge \gamma \models \tau} \prod_{a \in \gamma^a \wedge a \notin \tau} \alpha^\gamma_{sa} = \sum_{\hat{\gamma} \in \hat{\Gamma}} p^{\hat{\gamma}} = 1$, by Lemma 1. $\hat{\Gamma}$ is regular, since all entries in $\hat{\gamma} \in \hat{\Gamma}$ appear in Γ which is regular by assumption. For disjointness, note that for $\hat{\gamma}_1, \hat{\gamma}_2 \in \hat{\Gamma}$, $\gamma_1, \gamma_2 \in \Gamma$ are disjoint by assumption, and there exist a, λ^a s.t. $\gamma_1 \models \lambda^a$, $\gamma_2 \not\models \lambda_a$. Since $\gamma_1, \gamma_2 \models \tau$ by construction of $\hat{\Gamma}$, $a \notin \tau$,[3] and $\hat{\gamma}_1, \hat{\gamma}_2$ contain the same entries for a as γ_1, γ_2, and are therefore also disjoint. For exhaustiveness, consider for a contradiction a minimal λ satisfying $\nexists \hat{\gamma} \in \hat{\Gamma}$ s.t. $\hat{\gamma} \models \lambda$.[4] Since λ is minimal, no action a appears in both λ and τ, since $\forall \hat{\gamma} \in \hat{\Gamma}, \hat{\gamma} \models \tau$. Consider the concatenation $\lambda' = \lambda \oplus \tau$. Since Γ is exhaustive by assumption, $\exists \gamma \in \Gamma$ s.t. $\gamma \models \lambda'$, and by definition, $\gamma \models \tau$ and $\gamma \models \lambda$. Then $\hat{\gamma} \in \hat{\Gamma}$ and $\hat{\gamma} \models \lambda$, a contradiction. □

For intuition regarding how the $|\chi| \leq 1$ assumption can be relaxed, note that given an MDP M, we can generate an equivalent MDP M' in which $a \in A$ is replaced with $a_1, \dots, a_k$ where each a_i is identical to a except in requiring that a_{i-1} has already been applied in order to enable it, and a copy of the original action a requires that a_k has been applied. Γ for M that constrains the first k applications of a is then equivalent to Γ' for M' that constrains only the first applications of $a_1, \dots, a_k$.

5 Building Trajectory Constraint Heuristics

In the previous section we formulated sufficient criteria for a set of trajectory constraints to define an admissible heuristic. We now consider the problem of choosing a specific Γ to maximize the informativeness of $h_{tc}[h, \Gamma]$ while keeping Γ reasonably small. Since our primary interest in this work is in showing the utility of the h_{tc} framework in large state spaces, we use the h_{max} heuristic, which is cheap to compute and scales with the number of fluents, rather than the size

[2] α_{sa} includes s in the subscript to match Definition 2. The particular s does not matter here due to the regularity assumption.

[3] This relies on the simplifying assumption of $|\chi| \leq 1$.

[4] This relies on the simplifying assumption of $|\chi| \leq 1$.

Algorithm 1: Selecting Γ for h_{tc}(n)

```
$\Gamma \leftarrow \{\emptyset\}$
FIXPOINT $\leftarrow$ FALSE
**while** $\neg$FIXPOINT **do**
    FIXPOINT $\leftarrow$ TRUE
    $\Gamma' \leftarrow \emptyset$
    **for** $\gamma \in \Gamma$ **do**
        RP $\leftarrow$ COMPUTE-RELAXED-PLAN$(M^\gamma)$
        RP-PROB $\leftarrow \{e^i_{o^\gamma} \mid e^i_{o^\gamma} \in \text{RP} \wedge |\mathsf{eff}(o^\gamma)| > 1\}$
        **if** RP-PROB $= \emptyset \vee |\gamma| = n$ **then**
            $\Gamma' \leftarrow \Gamma' \cup \{\gamma\}$
            **continue**
        **end if**
        $e^i_{o^\gamma} \leftarrow \arg\min_{e^i_{o^\gamma} \in \text{RP-PROB}} p^i_{o^\gamma}$
        $\gamma' \leftarrow \gamma \cup \{(o^\gamma, \langle\{(s, \{s[e^i_{o^\gamma}]\})\}\rangle) \mid s \in S_{\mathsf{pre}(o^\gamma)}\}$
        $\gamma'' \leftarrow \gamma \cup \{(o^\gamma, \langle\{(s, \{s[e^j_{o^\gamma}] \mid e^j_{o^\gamma} \in \mathsf{eff}(o^\gamma) \wedge$
                                                  $e^j_{o^\gamma} \neq e^i_{o^\gamma}\})\}\rangle)$
                                  $\mid s \in S_{\mathsf{pre}(o^\gamma)}\}$
        $\Gamma' \leftarrow \Gamma' \cup \{\gamma', \gamma''\}$
        FIXPOINT $\leftarrow$ FALSE
    **end for**
    $\Gamma \leftarrow \Gamma'$
**end while**
```

of the state space, as our base heuristic. It is therefore also a natural choice to inform the selection of a set of relevant constraints Γ in every state encountered during search. We note that the h_{tc} framework does not depend on the use of h_{max} specifically, and could use any other probabilistic planning heuristic as its base heuristic.

To obtain an informative Γ, we consider the relaxed plan computed using h_{max} best supporters.[5] We select the operators associated with low-probability determinized effects that are unlikely to occur in the original problem, but which the heuristic (and therefore the computed relaxed plan) relies on. We then build trajectory constraints that limit the possible effects of these operators in order to force the planner to consider the consequences when the desired effect is not obtained. This procedure is detailed in Algorithm 1. Note that the procedure can be performed for each unique state encountered during search in order to tailor the set of constraints to each state.

In words, the algorithm computes the h_{max} best supporters in each state, and uses these to extract a relaxed plan. Out of the determinized effects of probabilistic actions that are included in the relaxed plan RP, an effect $e^i_{o^\gamma}$ associated with a determinized instance of operator o^γ with lowest probability $p^i_{o^\gamma}$ in the current problem M^γ is selected, and the current trajectory constraint γ is extended to construct two new trajectory constraints γ' and γ''. In γ', the constraint is imposed that the low probability outcome is the *only* possible outcome, while in γ'' the constraint is imposed that only the other outcomes, currently unused in the relaxed plan, can occur. For the heuristic denoted h_{tc}(n), this process is iterated until all $\gamma \in \Gamma$ have size n or no further constraints can be added to any γ (e.g. because the relaxed plan does not make use of any probabilistic effects, or the problem has fewer than n probabilistic actions).

Restricting the Number of Action Applications. Heuristic estimates given by h_{tc} can be greatly improved by incorporating information about the maximum number of times an action can be applied. As an example, if an action can be proven to be applicable at most once, and its first application is constrained, the unconstrained version of the action that is applicable after the first application can be omitted from the problem entirely, often greatly (but admissibly) improving the heuristic estimate. In order to derive these conditions, we use the well-known method of computing the h^2 heuristic (Haslum and Geffner 2000) in the start state to identify a set of mutex fluent pairs. Given the set of mutex pairs, we can identify limitations on any (probabilistic) action a by checking whether the precondition for its ith application (which will contain fluents representing the fact that a has already been applied $i - 1$ times) contains mutex pairs. When this is the case, a is guaranteed to be applicable at most $i - 1$ times in any legal action sequence. This proves particularly useful in the information gathering domains discussed below, where actions used by the agent to determine an underlying fact are provably only applicable once, and their unconstrained versions can be completely omitted.

6 Information-Gathering Domains

One setting in which h_{tc} is especially useful is that of *information-gathering domains*, where an agent must first determine the values of a set of (probabilistic) information variables, and then take actions of varying cost depending on their values. Determinization-based heuristics give inaccurate estimates in such settings as they consider only the most advantageous values that give the lowest cost plans. In contrast, h_{tc} considers plan costs resulting from different values of information variables and computes estimates that take into consideration some of their non-optimal values.

The Medical Necessity Domain. A medical services provider must determine how to obtain reimbursement for a patient from their insurance provider. The agent must first establish *medical necessity* by either (a) using the known medical history of the patient or (b) querying for additional information, with cost dependent on the difficulty of obtaining that information. The combinations of facts that make a patient eligible for reimbursement at different levels are specified by the insurer. Omitting some details, the domain consists of the following sets of boolean variables:

- $V_{\textit{person-info}}$, the patient and relatives' medical information (e.g. $\textit{person-info}_{\textit{patient, has-cardiovascular-disease}}$),
- $V_{\textit{person-info-known}}$, whether a piece of information is known (e.g. $\textit{person-info-known}_{\textit{patient-sister, has-breast-cancer-diagnosis}}$),
- $V_{\textit{rule-selected}}$, whether a particular medical necessity rule was chosen (e.g. $\textit{rule-selected}_{\textit{rule1}}$),
- $V_{\textit{claim-submitted}}$, whether a billing claim was submitted,

and the following sets of actions:

[5] Details of relaxed plan heuristics are outside the scope of this paper. See Keyder and Geffner (2008a) for further information.

- $O_{\textit{discover-person-info(person, attribute)}}$ with precondition $\neg$*person-info-known*$_{\textit{person, attribute}}$, deterministic effect *person-info-known*$_{\textit{person, attribute}}$, probabilistic effect *person-info*$_{\textit{person, attribute}}$, with probability $p_{\textit{person, attribute}}$, and cost *attribute-discovery-cost*$_{\textit{attribute}}$,
- $O_{\textit{select-rule(rule)}}$ with precondition $\phi_{\textit{rule}}$ given for each rule by a conjunction over the $V_{\textit{person-info}}$ variables, effect *rule-selected*$_{\textit{rule}}$, and cost 0,
- $O_{\textit{submit-claim(claim)}}$ with disjunctive precondition over $V_{\textit{rule-selected}}$ for the rules that enable a particular claim, effect $V_{\textit{claim-submitted}}$, and reward equal to the expected reimbursement $r_{\textit{claim}}$ (or equivalently, cost equal to $K - r_{\textit{claim}}$ for some large constant K).

Some subset of the $V_{\textit{person-info}}$ and $V_{\textit{person-info-known}}$ variables are *true* in s_0. Given available rules, the agent must decide what attributes of the patient and relatives to query in what order and then choose a rule establishing medical necessity for the patient, while considering (a) the claims and reimbursements possible under different rules, (b) the probabilities of the different values in $V_{\textit{person-info}}$ and (c) the associated discovery costs. If it turns out to be too expensive or impossible to find a rule that allows a valid claim, the agent may instead declare that the account cannot be processed. The goal is to submit a claim or to indicate explicitly that a claim cannot be submitted.

The Discover-Key Domain. An agent must navigate in a gridworld to a goal location after obtaining one of several *keys* required to enter it. There are a known set of possible key locations, with independent prior probabilities on each. The agent must identify a location that has a key, navigate there to pick it up, and then navigate to the goal. Unit costs are accumulated at each time step. If no key is available after querying all locations, the agent must pay a large cost to bypass the locked door without a key.

We use two versions of this domain. In DK-remote, there is a monitor at a fixed location, and the agent must be at the location of the monitor to query for key availability. The monitor is the only way to identify whether a grid cell contains a key. In DK-local, the agent must visit a possible key location to check whether it contains a key.

The Canadian Traveller Problem (CTP). An agent must navigate to a goal location in a graph where edge presence is initially unknown. The version used here is based on Eyerich, Keller, and Helmert (2010) where there is a known prior on edge availability for each edge. At each step, the agent can either move from its location if an edge is known to be present (with cost dependent on the edge), or query the presence of an edge connected to its current location (with unit cost). The CTP is known to be intractable to solve optimally (Eyerich, Keller, and Helmert 2010; Bnaya, Felner, and Shimony 2009). We therefore focus on small instances for evaluation, generated as random Delaunay graphs with edge costs drawn uniformly from $[0, 50]$.

7 Experiments

We evaluate two versions of the weighted trajectory constraint heuristic against $h_{\textit{max}}$, $h_{\textit{min}}$, and a simple lookahead heuristic as baselines, using the procedure described in Algorithm 1 for $n \in \{1, 2\}$. We use *improved* LAO* (Hansen and Zilberstein 2001) and LRTDP (Bonet and Geffner 2003b) as search algorithms. We measure the total time for convergence to an optimal policy (including preprocessing and search), as well as the number of node expansions (for LAO*) or the number of Dynamic Programming (DP) updates required (for LRTDP).

Experiments were performed on a 3.6GHz AMD CPU with 8GB of memory. A time limit of 10 minutes was used for each configuration. Algorithms and heuristics are implemented in C++ as an extension of `mdp-lib` (Pineda and Zilberstein 2019), and will be released as an open-source fork at a future date.

Results are reported in Table 1. For the Medical Necessity Domain, 10 benchmark instances were used. On all problems solved by any configuration, $h_{tc}(2)$ gives the fastest time to convergence by at minimum an order of magnitude and requires the fewest node expansions and DP updates for both algorithms, with $h_{tc}(1)$ the second best.

For each Discover-Key domain, and each grid-size/key-location configuration, 10 randomly-generated instances were used. On both domains, $h_{tc}(2)$ gives the fastest time to convergence on all instances for LAO*, with $h_{tc}(1)$ the second best. On the majority of configurations for both domains, $h_{tc}(2)$ gives the fastest time to convergence for LRTDP. In DK-remote, $h_{tc}(2)$ either outperforms all heuristics or is competitive with $h_{\textit{min}}$ in number of node expansions for LAO*. In DK-local, $h_{tc}(2)$ outperforms all heuristics in number of node expansions. Finally, $h_{tc}(2)$ requires the fewest DP updates for all configurations on both domains. In general, $h_{\textit{min}}$ and $h_{tc}(2)$ are comparable in terms of informativeness as measured by node expansions for LAO*, as the use of multiple subproblems is able to compensate for the less informative base heuristic $h_{\textit{max}}$. However, $h_{tc}(2)$ tends to converge to a solution faster as it does not need to construct a significant proportion of the full state space to compute its estimates.

For the Canadian Traveller Problem, 3 randomly generated instances were used for each graph size. $h_{tc}(2)$ outperforms all other heuristics in time and node expansions on all instances solved within the time and memory limit.

For all domains, we additionally tested a single-step lookahead heuristic, computed as the $\min$ over the available actions of the transition probability-weighted sum of $h_{\textit{max}}$ values for successor states. This baseline performed similarly to or slightly worse than $h_{\textit{max}}$ in all cases. These results are omitted for space. We were unable to compare to several other baselines, including occupation measure heuristics (Trevizan, Thiébaux, and Haslum 2017) and probabilistic pattern database heuristics (Klößner et al. 2021), as open source implementations were not available.

We also considered several domains drawn from previous probabilistic planning competitions, such as BLOCKSWORLD and ELEVATORS (Bonet and Givan 2006). Information-gathering actions are absent in these problems, and the gain in informativeness with h_{tc} is minimal. Trajectory constraints increase heuristic estimates only slightly, as the constrained actions can be quickly "used up" to allow

		LAO*								LRTDP							
		$h_{tc}(1)$		$h_{tc}(2)$		h_{max}		h_{min}		$h_{tc}(1)$		$h_{tc}(2)$		h_{max}		h_{min}	
		t	n	t	n	t	n	t	n	t	n	t	n	t	n	t	n
Medical Necessity	1	6.92	609	**1.50**	**327**	16.90	912	23.40	909	1.75	41384	**0.59**	**5734**	4.08	130007	10.63	129342
	2	6.89	1623	**0.63**	**153**	25.92	3732	41.91	3651	3.98	66605	**0.83**	**2428**	10.31	255301	26.44	253987
	3	3.06	491	**0.60**	**24**	15.20	1996	32.23	2062	2.45	28386	**0.64**	**263**	8.85	176074	26.33	180160
	4	14.30	1060	**0.53**	**24**	36.05	1858	55.52	1858	6.50	117511	**0.51**	**56**	16.58	385644	35.69	385644
	5	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	6	237.63	17083	**40.22**	**7351**	-	-	-	-	75.27	1196590	**14.96**	**138860**	241.46	4114697	-	-
	7	46.07	3885	**3.18**	**329**	142.52	6731	203.26	6772	21.43	283489	**3.21**	**14413**	60.60	969820	121.03	966828
	8	-	-	-	-	-	-	*	*	-	-	-	-	-	-	*	*
	9	342.16	28156	**71.43**	**13849**	-	-	-	-	107.57	1645636	**34.06**	**316871**	-	-	-	-
	10	-	-	-	-	-	-	*	*	-	-	-	-	-	-	*	*
	$D\,k$																
DK-remote	5 2	0.66	106	**0.48**	**102**	2.14	151	2.53	104	0.54	2675	**0.49**	**590**	0.52	13356	1.08	12665
	4	11.27	818	**1.55**	**542**	22.98	938	24.90	545	2.42	30267	**1.63**	**4007**	2.67	57367	4.66	55253
	6	63.11	4139	**11.00**	2733	112.97	4399	107.61	**2014**	12.10	119611	**7.45**	**23594**	13.55	176336	26.34	167003
	6 2	1.97	150	**1.56**	141	4.65	203	5.12	**136**	1.63	4018	1.52	**1016**	**0.93**	18406	2.01	15288
	4	20.57	1149	**3.74**	721	44.72	1317	48.14	**654**	5.16	36304	**3.85**	**5302**	4.76	75533	8.34	65997
	6	94.00	5577	**20.32**	3726	177.97	5852	186.72	**2236**	23.39	172615	**15.76**	**35243**	25.45	275237	54.47	263097
	7 2	6.35	205	**5.11**	192	11.77	290	13.10	**181**	5.29	6575	5.04	**1356**	**1.96**	30376	3.93	28390
	4	49.81	1490	**10.19**	**856**	97.90	1707	106.18	979	12.84	63266	10.03	**8636**	**8.50**	115381	14.86	107245
	7 6	136.09(1)	5623(1)	**25.58**	3518	268.27(4)	5949(4)	270.33(4)	**2272**(4)	33.42	178760	**24.06**	**29409**	32.88	309234	61.26	273336
DK-local	5 2	0.81	58	**0.39**	**42**	3.00	98	3.23	90	0.55	4384	**0.40**	**182**	0.67	18148	1.22	17904
	4	6.40	317	**1.13**	**154**	12.63	424	14.41	382	2.19	28157	**1.21**	**2160**	2.47	55406	4.00	55709
	6	44.50	1492	**7.36**	**668**	77.44	1684	72.16	1542	9.35	121488	**4.46**	**21440**	11.23	182848	18.46	183436
	6 2	2.03	101	**1.58**	**87**	4.34	154	5.08	130	1.72	3499	1.59	**591**	**0.97**	18588	2.00	17827
	4	17.14	580	**3.63**	**257**	30.60	743	34.46	654	6.08	48855	**3.68**	**4142**	5.40	88123	8.33	86248
	6	72.36	1912	**11.99**	**846**	133.00	2151	126.02	1949	17.14	145323	**9.40**	**21130**	18.95	240689	31.68	233827
	7 2	6.11	98	**5.05**	**82**	10.25	172	11.72	156	5.30	5399	5.08	**640**	**1.99**	29624	3.84	28931
	4	38.78	762	**9.87**	**316**	68.44	978	74.26	871	13.91	71053	9.87	**5617**	**9.37**	128570	14.19	125016
	6	114.99	2406	**21.81**	**990**	211.84	2737	211.83	2325	31.55	186331	**19.05**	**25161**	26.63	277708	52.49	281047
CTP	4	0.81	977	**0.66**	**729**	1.23	1552	1.96	1552	0.87	3571	**0.75**	**2490**	0.90	9023	1.56	9023
	5	15.32	8666	**11.67**	**6734**	21.64(1)	12343(1)	42.08(1)	12343(1)	20.98	67791	**15.12**	**42265**	26.09	131398	45.97	131398
	6	-	-	-	-	-	-	*	*	-	-	-	-	-	-	*	*
Blocksworld	1	37.80	4035	71.03	3783	**4.33**	4179	11.46	**428**	-	-	-	-	59.25	457384	**16.70**	**26668**
	2	33.17	3072	57.30	2806	**3.24**	3098	11.17	**291**	-	-	-	-	55.32	330545	**14.63**	**15149**
	3	24.51	1747	38.39	1575	**2.04**	1853	9.72	**146**	-	-	-	-	46.99	240183	**14.26**	**9858**
	4	41.39	4552	78.52	4308	**4.54**	4612	11.93	**725**	-	-	-	-	62.69	453828	**17.47**	**30918**
	5	29.17	2688	48.58	2510	**2.75**	2736	9.48	**237**	342.26	289969	-	-	52.53	297313	**14.69**	**12881**
	6	-	-	-	-	-	-	*	*	-	-	-	-	-	-	*	*
Elevators	1	0.09	338	0.11	324	**0.06**	335	**0.06**	**27**	0.13	2448	0.17	2427	0.08	2559	**0.06**	**180**
	2	0.03	43	0.03	43	**0.02**	44	0.03	**8**	0.05	260	0.04	260	0.03	247	**0.02**	**8**
	3	0.06	252	0.08	251	**0.05**	252	0.06	**15**	0.11	2369	0.15	2354	0.09	2417	**0.06**	**15**
	4	0.05	182	0.05	181	**0.04**	182	**0.04**	**13**	0.10	1461	0.12	1453	0.07	1461	**0.04**	**13**
	5	0.08	174	0.10	171	0.05	192	**0.04**	**11**	0.16	1508	0.21	1468	0.09	1600	**0.04**	**11**
	6	2.61	2575	2.96	2568	1.53	2645	**1.21**	**142**	8.96	47597	11.38	46769	4.75	48779	**1.35**	**1042**

Table 1: *t* is average time over 3 trials in seconds. - and * indicate time/memory exhaustion, respectively. For Discover-Key and CTP, $(\cdot)$ is number of instances with timeouts, if any. n is number of node expansions for LAO* and number of DP updates for LRTDP. Best performers are **bold**. For Discover-Key, D indicates a $D \times D$ grid and k is number of possible key locations.

the application of the unconstrained, determinized versions that allow any desired outcome to be obtained. As a result, h_{tc} is outperformed by h_{max} and h_{min} due to their lower overhead and higher informativeness, respectively. These results show that determinization-based heuristics are better-suited to problems where the impact of individual probabilistic outcomes on expected policy cost is more limited. Other competition domains with similar results are omitted here.

8 Conclusion

We have introduced a new method for constructing domain-independent probabilistic planning heuristics. The method consists of decomposing a problem into multiple subproblems, each satisfying a different trajectory constraint. With this decomposition, heuristics based on the all-outcome determinization can be forced to consider a wider range of outcomes, and the resulting values can be weighted by the probability of the constraint to obtain more informative heuristics for the problem as a whole. On information-gathering domains, these heuristics improve over baseline heuristics in terms of both informativeness and search time.

In future work we plan to consider different base heuristics, such as heuristics that are sensitive to deletes and can therefore reason more effectively about different types of probabilistic effects. Additionally, we will investigate alternative approaches for constructing relevant and informative trajectory constraints, for instance by varying the number of constrained problems that are considered in each node.

References

Barto, A. G.; Bradtke, S. J.; and Singh, S. P. 1995. Learning to Act Using Real-Time Dynamic Programming. *Journal of Artificial Intelligence*, 72(1–2): 81–138.

Bnaya, Z.; Felner, A.; and Shimony, S. E. 2009. Canadian Traveler Problem with Remote Sensing. In *IJCAI 2009*, 437–442.

Bonet, B.; and Geffner, H. 2001. Planning as Heuristic Search. *Journal of Artificial Intelligence*, 129(1): 5–33.

Bonet, B.; and Geffner, H. 2003a. Faster Heuristic Search Algorithms for Planning with Uncertainty and Full Feedback. In *IJCAI 2003*, 1233–1238.

Bonet, B.; and Geffner, H. 2003b. Labeled RTDP: Improving the Convergence of Real-Time Dynamic Programming. In *ICAPS 2003*, 12–21.

Bonet, B.; and Geffner, H. 2005. mGPT: A Probabilistic Planner Based on Heuristic Search. *Journal of Artificial Intelligence*, 24: 933–944.

Bonet, B.; and Givan, R. 2006. The Fifth International Probabilistic Planning Competition.

Eyerich, P.; Keller, T.; and Helmert, M. 2010. High-Quality Policies for the Canadian Traveler's Problem. In *AAAI 2010*, 51–58.

Hansen, E. A.; and Zilberstein, S. 2001. LAO*: A Heuristic Search Algorithm that Finds Solutions with Loops. *Journal of Artificial Intelligence*, 129(1–2): 35–62.

Haslum, P.; and Geffner, H. 2000. Admissible Heuristics for Optimal Planning. In *AIPS 2000*, 140–149.

Helmert, M.; and Domshlak, C. 2009. Landmarks, Critical Paths and Abstractions: What's the Difference Anyway? In *ICAPS 2009*, 162–169.

Helmert, M.; Haslum, P.; Hoffmann, J.; and Nissim, R. 2014. Merge-and-Shrink Abstraction: A Method for Generating Lower Bounds in Factored State Spaces. *Journal of the ACM*, 61(3): 16:1–63.

Kearns, M. J.; Mansour, Y.; and Ng, A. Y. 2002. A Sparse Sampling Algorithm for Near-Optimal Planning in Large Markov Decision Processes. *Machine Learning*, 49(2-3): 193–208.

Keller, T.; and Helmert, M. 2013. Trial-based Heuristic Tree Search for Finite Horizon MDPs. In *ICAPS 2013*, 135–143.

Keyder, E.; and Geffner, H. 2008a. Heuristics for Planning with Action Costs Revisited. In *ECAI 2008*, 588–592.

Keyder, E.; and Geffner, H. 2008b. The HMDPP Planner for Planning with Probabilities. In *ICAPS 2008*.

Klößner, T.; Hoffmann, J.; Steinmetz, M.; and Torralba, A. 2021. Pattern Databases for Goal-Probability Maximization in Probabilistic Planning. In *ICAPS 2021*, 201–209.

Kocsis, L.; and Szepesvári, C. 2006. Bandit Based Monte-Carlo Planning. In *ECML 2006*, 282–293.

Pineda, L.; and Zilberstein, S. 2019. Probabilistic Planning with Reduced Models. *Journal of Artificial Intelligence Research*, 65: 271–306.

Pineda, L. E.; Wray, K. H.; and Zilberstein, S. 2017. Fast SSP Solvers Using Short-Sighted Labeling. In *AAAI 2017*, 3629–3635.

Pommerening, F.; Röger, G.; Helmert, M.; and Bonet, B. 2014. LP-based Heuristics for Cost-Optimal Planning. In *ICAPS 2014*, 226–234.

Sanner, S. 2010. Relational Dynamic Influence Diagram Language (RDDL): Language Description.

Teichteil-Königsbuch, F.; Kuter, U.; and Infantes, G. 2010. Incremental Plan Aggregation for Generating Policies in MDPs. In *AAMAS 2010*, 1231–1238.

Teichteil-Königsbuch, F.; Vidal, V.; and Infantes, G. 2011. Extending Classical Planning Heuristics to Probabilistic Planning with Dead-Ends. In *AAAI 2011*, 1017–1022.

Trevizan, F. W.; Thiébaux, S.; and Haslum, P. 2017. Occupation Measure Heuristics for Probabilistic Planning. In *ICAPS 2017*, 306–315.

Yoon, S.; Ruml, W.; Benton, J.; and Do, M. 2010. Improving Determinization in Hindsight for On-Line Probabilistic Planning. In *ICAPS 2010*, 209–216.

Yoon, S. W.; Fern, A.; and Givan, R. 2007. FF-Replan: A Baseline for Probabilistic Planning. In *ICAPS 2007*, 352–360.

Younes, H. L. S.; and Littman, M. L. 2004. PPDDL1.0: An Extension to PDDL for Expressing Planning Domains with Probabilistic Effects. Technical Report CMU-CS-04-167, Carnegie Mellon University, School of Computer Science.