Artificial Intelligence 216 (2014) 179–205

Contents lists available at ScienceDirect

Artificial Intelligence

www.elsevier.com/locate/artint

Depth-based short-sighted stochastic shortest path problems

Felipe W. Trevizan *, Manuela M. Veloso

School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213, USA

ARTICLE INFO

Article history:
Received 12 November 2012
Received in revised form 13 June 2014
Accepted 6 July 2014
Available online 14 July 2014

Keywords:
Probabilistic planning
Stochastic Shortest Path Problems
Markov Decision Processes

ABSTRACT

Stochastic Shortest Path Problems (SSPs) are a common representation for probabilistic planning problems. Two approaches can be used to solve SSPs: (i) consider all probabilistically reachable states and (ii) plan only for a subset of these reachable states. Closed policies, the solutions obtained in the former approach, require significant computational effort, and they do not require replanning, i.e., the planner is never re-invoked. The second approach, employed by replanners, computes open policies, i.e., policies for a subset of the probabilistically reachable states. Therefore, when a state is reached in which the open policy is not defined, the replanner is reinvoked to compute a new open policy. In this article, we introduce a special case of SSPs, the depth-based short-sighted SSPs, in which every state has a nonzero probability of being reached using at most t actions. We also introduce the novel algorithm Short-Sighted Probabilistic Planner (SSiPP), which solves SSPs through depth-based short-sighted SSPs and guarantees that at least t actions can be executed without replanning. Therefore, SSiPP can compute both open and closed policies: as t increases, the returned policy approaches the behavior of a closed policy, and for t large enough, the returned policy is closed. Moreover, we present two extensions to SSiPP: Labeled-SSiPP and SSiPP-FF. The former extension incorporates a labeling mechanism to avoid revisiting states that have already converged. The latter extension combines SSiPP and determinizations to improve the performance of SSiPP in problems without dead ends. We also performed an extensive empirical evaluation of SSiPP and its extensions in several problems against state-of-the-art planners. The results show that (i) Labeled-SSiPP outperforms SSiPP and the considered planners in the task of finding the optimal solution when the problems have a low percentage of relevant states; and (ii) SSiPP-FF outperforms SSiPP in the task of quickly finding suboptimal solutions to problems without dead ends while performing similarly in problems with dead ends.

1. Introduction

Stochastic Shortest Path Problems (SSP) [3] provide a convenient framework for modeling fully observable probabilistic planning problems. A solution to an SSP is a policy – a mapping from states to actions – that is guaranteed to reach a goal state when executed from the initial state of the given SSP. In this article, we address the question of how to improve the scalability of probabilistic planners when searching for (i) an optimal policy and (ii) a suboptimal policy given a time deadline.

* Corresponding author.
E-mail addresses: fwt@cs.cmu.edu (F.W. Trevizan), mmv@cs.cmu.edu (M.M. Veloso).

http://dx.doi.org/10.1016/j.artint.2014.07.001
0004-3702/

One approach to computing optimal solutions to SSPs is to use value iteration and policy iteration algorithms, which are optimal [4]. Planners based on these algorithms return a closed policy, i.e., a policy that is defined at least over all the probabilistically reachable states of the given SSP. Assuming the model correctly captures the cost and uncertainty of the actions in the environment, closed policies are extremely powerful as their execution never "fails"; therefore, the planner is never reinvoked. Unfortunately, the computation of such policies is prohibitive in complexity as problems scale up. The efficiency of value-iteration-based probabilistic planners can be improved by combining asynchronous updates and heuristic search (e.g., Labeled RTDP [5]), resulting in optimal algorithms with convergence bounds. Although these techniques allow planners to compute compact policies, in the worst case, these policies are still linear in the size of the state space, which itself can be exponential in the size of the state or goals.

Different approaches have been proposed to efficiently find nonoptimal solutions to SSPs based on replanning. Replanners do not invest the computational effort to generate a closed policy, and instead compute an *open policy*, i.e., a policy that does not address all the probabilistically reachable states. Different methods can be employed to generate open policies, e.g., determinization [30,31], sampling [11,26], and finite horizon search [21,17]. During the execution, if a state not included in the open policy is reached, the replanner is reinvoked to compute a new open policy starting from the unpredicted state.

In this work, we introduce a new model, the depth-based short-sighted Stochastic Shortest Path Problems (short-sighted SSPs), a special case of SSPs in which every state has a nonzero probability of being reached using at most t actions. We also introduce the novel algorithm Short-Sighted Probabilistic Planner (SSiPP), which solves SSPs using short-sighted SSPs to represent subproblems of the original problem. We prove that the policies computed by SSiPP can be executed for at least t time steps without replanning; therefore, by varying the parameter t of the short-sighted SSPs, SSiPP can behave as either a probabilistic planner or a replanner: for small values of t, the SSiPP returns open policies and less replanning is necessary as t increases; and for t large enough, SSiPP returns closed policies. We provide an upper bound for t in which SSiPP is guaranteed to return closed policies.

We also present two extensions of SSiPP: Labeled-SSiPP and SSiPP-FF. Labeled-SSiPP improves the performance of SSiPP when searching for the optimal solution of SSPs by not revisiting states that have already converged. SSiPP-FF combines SSiPP and determinization to improve the efficiency of SSiPP when searching for a suboptimal solution under *small* time constraints, e.g., the International Probabilistic Planning Competition (IPPC) [33,7,8] rules.

Lastly, we extensively compare SSiPP, Labeled-SSiPP, and SSiPP-FF in different domains against the state-of-the-art probabilistic planners. Our results show that in the task of finding the optimal solution for an SSP, Labeled-SSiPP represents an improvement of SSiPP, and Labeled-SSiPP outperforms the other considered planners when the optimal policy encompasses a small fraction of the state space. For the task of quickly finding a suboptimal solution to an SSP, our results indicate that SSiPP-FF successfully combines the behavior of SSiPP and FF-Replan: for problems without dead ends, SSiPP-FF performance is similar to FF-Replan performance (the best planner for problems without dead ends); and for problems with dead ends, SSiPP-FF performs better than FF-Replan and similarly to SSiPP. Moreover, the solutions obtained by SSiPP-FF in problems without dead ends have better quality, i.e., lower average cost, than the solutions obtained by FF-Replan.

This article is organized as follows: Section 2 reviews the basic background on SSPs and the related work. Section 3 defines formally our novel model, the depth-based short-sighted SSPs, as well as its properties. Section 4 presents our main algorithms, namely SSiPP, Labeled-SSiPP, and SSiPP-FF, and their theoretical guarantees. Section 5 empirically evaluates SSiPP (and its extensions) against the state-of-the-art planners in two settings: search for the optimal solution (Section 5.2) and search for a solution using the IPPC rules (Section 5.3). Section 6 concludes the article.

2. Background and related work

This section introduces the basic concepts and notation used in this article (Section 2.1) and reviews the related work in probabilistic planning (Section 2.2).

2.1. Stochastic shortest path problem

A Stochastic Shortest Path Problem (SSP) [3] 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 nonempty set of goal 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
- $C(s,a,s') \in (0, +\infty)$ is the immediate cost incurred when state s' is reached after applying action a in state s. This function is required to be defined for all s, a, and s' in which $P(s'|s,a) > 0$.

In SSPs, an agent executes actions $a \in \mathsf{A}$ in discrete time steps at a state $s \in \mathsf{S}$. The chosen action a changes state s to state s' with probability $P(s'|s,a)$ and the cost $C(s,a,s')$ is incurred. If a goal state $s_G \in \mathsf{G}$ is reached, the problem finishes, i.e., no more actions need to be executed. The sequence of states $\mathcal{T} = \langle s_0, s_1, s_2, \ldots \rangle$ visited by the agent is called a trajectory,

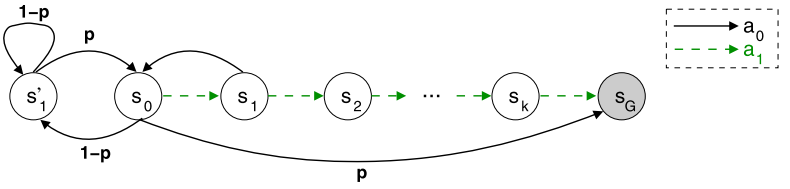

Fig. 1. Example of a Stochastic Shortest Path Problem (SSP). The initial state is s_0, the goal set is $\mathsf{G} = \{s_\mathsf{G}\}$ and $C(s, a, s') = 1, \forall s \in \mathsf{S}, a \in \mathsf{A}$ and $s' \in \mathsf{S}$.

and the state s_i is the state of the environment at time step i. Thus, for every trajectory $\mathcal{T}$, there exists at least one sequence of actions $\langle a_0, a_1, a_2, \ldots\rangle$ such that a_i is executed in state s_i and $P(\mathcal{T}|\langle a_0, a_1, a_2, \ldots\rangle) = \prod_{i\in\{0,1,\ldots\}} P(s_{i+1}|s_i, a_i) > 0$.

The horizon is the maximum number of actions the agent is allowed to execute in the environment, and therefore the maximum size of $\mathcal{T}$. For SSPs, the horizon is indefinite because under certain conditions discussed later in this section, a goal state can be reached using a finite, yet unbounded, number of actions. If the horizon is set to $t_{\max}$, then the obtained model is known as a finite-horizon Markov Decision Process (MDP) [22]. Alternatively, if no goal states are given, then the horizon becomes infinite as no stop condition is given to the agent. In order to guarantee that the total accumulated cost is finite in such models, the cost incurred at time step t is discounted by γ^t, for $\gamma \in (0, 1)$. The obtained model, known as discounted infinite-horizon MDPs [22], and the finite-horizon MDPs are special cases of SSPs [4].

A solution to an SSP is a policy π, i.e., a mapping from states to actions. A policy defined over all the states S is know as a complete policy because it is a complete mapping from S to A. Similarly, a policy π defined only for a subset of S is known as a partial policy. Given a policy π, the set of all the states reachable when following π from s_0 is denoted as $\mathsf{S}^\pi \subseteq \mathsf{S}$ and the set of states in which replanning is necessary as R^π. Formally, $\mathsf{R}^\pi = \{s \in \mathsf{S} \setminus \mathsf{G} | \pi \text{ is not defined for } s\}$. A policy π can also be classified according to S^π and R^π. If π can be followed from s_0 without replanning, i.e., $\mathsf{R}^\pi \cap \mathsf{S}^\pi = \emptyset$, then π is a closed policy. Therefore, every complete policy π is also a closed policy since $\mathsf{R}^\pi = \emptyset$. If a policy π is not closed, then $\mathsf{R}^\pi \cap \mathsf{S}^\pi \neq \emptyset$ and π is known as an open policy. For any open policy π, replanning has a nonzero probability of happening because every state $s \in \mathsf{R}^\pi \cap \mathsf{S}^\pi$ has a nonzero probability of being reached when following π from s_0. Thus, every open policy π is also partial as π is not defined over the states $s \in \mathsf{R}^\pi \cap \mathsf{S}^\pi$.

Policies can be further classified according to their termination guarantee. If it is inevitable to reach a goal state when following the policy π from s_0, then π is a proper policy. Formally,

Definition 1 *(Proper policy).* A policy π is proper if, for all $s \in \mathsf{S}^\pi$, there exists a trajectory $\mathcal{T} = \langle s, s_1, \ldots, s_k\rangle$ generated by π such that $s_k \in \mathsf{G}$ and $k \leq |\mathsf{S}|$.

A policy that is not proper is said to be improper. A common assumption used in the theoretical results for SSPs is as follows:

Assumption 1. There exists at least one policy that is proper independent of the initial state s_0 of the SSP.

This assumption is equivalent to Assumption 2.1 of [4] and implies that a goal state is always reachable from every state $s \in \mathsf{S}$.

By definition, every proper policy is closed, and every open policy is improper; however, not all closed policies are proper. To illustrate this relationship between closed and proper policies, consider the SSP depicted in Fig. 1: $\pi_0 = \{(s_0, a_0), (s'_1, a_0)\}$ is a proper policy and $\mathsf{S}^{\pi_0} = \{s_0, s'_1, s_\mathsf{G}\}$; $\pi_1 = \{(s_0, a_1), (s_1, a_1)\}$ is an open policy because $\pi_1(s_2)$ is not defined; and $\pi_2 = \{(s_0, a_1), (s_1, a_0)\}$ is a closed and improper policy as no goal state is reachable from s_0 when following π_2, and π_2 is defined for $\mathsf{S}^{\pi_2} = \{s_0, s_1\}$.

Given a closed policy π, $V^\pi(s)$ is the expected accumulated cost to reach a goal state from state $s \in S^\pi$. The function V^π, defined at least over S^π, is called the value function for π and is the fixed-point solution for the following system of equations:

$$V^\pi(s) = \begin{cases} 0 & \text{if } s \in \mathsf{G} \\ E[C(s, a, s') + V^\pi(s')|s, a = \pi(s)] & \text{otherwise,} \end{cases} \quad \forall s \in \mathsf{S}^\pi \tag{1}$$

where $E[C(s, a, s') + V^\pi(s')|s, a] = \sum_{s'\in\mathsf{S}} P(s'|s, a)[C(s, a, s') + V^\pi(s')]$. Another common assumption for SSPs is as follows:

Assumption 2. For every closed and improper policy π, there exists at least one state $s \in \mathsf{S}^\pi$ such that $V^\pi(s)$ is infinite.

This assumption is already true in our definition of SSPs as the cost function $C(s, a, s')$ is strictly positive. For instance, for the SSP depicted in Fig. 1, the trajectories generated by the closed and improper policy $\pi_2 = \{(s_0, a_1), (s_1, a_0)\}$ have infinite size, and at each time step, a strictly positive immediate cost is incurred; therefore, $V^{\pi_2}(s_0) = V^{\pi_2}(s_1) = \infty$.

An optimal policy π^* is any proper policy that minimizes, over all proper policies, the expected cost of reaching a goal state from s_0, i.e., $V^{\pi^*}(s_0) \leq \min_{\pi \text{ s.t. } \pi \text{ is closed}} V^\pi(s_0)$. For a given SSP, π^* might not be unique; however, the optimal

value function V^*, representing for each state s the minimal expected accumulated cost to reach a goal state over all policies, exists and is unique [4]. For all optimal policies π^* and $s \in \mathsf{S}^{\pi^*}$, we have that $V^*(s) = V^{\pi^*}(s)$; formally, V^* is the fixed-point solution for the *Bellman Equation(s)*:

$$V^*(s) = \begin{cases} 0 & \text{if } s \in \mathsf{G} \\ \min_{a \in \mathsf{A}} E[C(s,a,s') + V^*(s') | s, a] & \text{otherwise,} \end{cases} \quad \forall s \in \mathsf{S}. \tag{2}$$

Every optimal policy π^* can be obtained by replacing min by argmin in (2), i.e., π^* is a *greedy policy* of V^*:

Definition 2 *(Greedy policy).* Given a value function V, a greedy policy π^V is such that $\pi^V(s) = \text{argmin}_{a \in \mathsf{A}} E[C(s,a,s') + V(s') | s, a]$ for all $s \in \mathsf{S} \setminus \mathsf{G}$. For the states s in which V is not defined, $V(s) = \infty$ is assumed.

A possible approach to computing V^* is the value iteration algorithm (VI) [14]: given an initial guess V^0 for V^*, the sequence $\langle V^0, V^1, \ldots, V^k \rangle$ is computed where V^{t+1} is obtained by performing a *Bellman backup* in V^t, that is, applying the operator B in the value function V^t for all $s \in \mathsf{S}$:

$$V^{t+1}(s) = \left(BV^t\right)(s) = \begin{cases} 0 & \text{if } s \in \mathsf{G} \\ \min_{a \in \mathsf{A}} E[C(s,a,s') + V^t(s') | s, a] & \text{otherwise.} \end{cases}$$

We denote by B^k the composition of the operator B: $(B^k V)(s) = (B(B^{k-1}V))(s)$ for all $s \in \mathsf{S}$; thus, $V^t = B^t V^0$. Given a value function V, $B^t V$ represents the optimal solution for the SSP in which the horizon is limited to t and the extra cost $V(s)$ is incurred when agent reaches state $s \in \mathsf{S} \setminus \mathsf{G}$ after applying t actions. $(B^t V)(s)$ is known as t-look-ahead value of state s according to V.

For SSPs in which Assumption 1 holds, V^k converges to V^* as $k \to \infty$ and $0 \leq V^*(s) < \infty$ for all $s \in \mathsf{S}$ [2]. Because convergence to V^* is not feasible in the general case, one solution is to find a value function $\hat{V}$ that is at most ϵ away from V^*, i.e., $|\hat{V}(s) - V^*(s)| \leq \epsilon$ for all $s \in \mathsf{S}$ [2,12]. Unfortunately, the computation of this error bound for $\hat{V}$ is too expensive to be used as stop criterion; thus, in practice, we are interested in the problem of finding an ϵ-consistent value function V:

Definition 3 *(ϵ-consistent).* Given an SSP $\mathbb{S}$, a value function V for $\mathbb{S}$ is ϵ-consistent if

$$R(\mathbb{S}, V) = \max_{s \in \mathsf{S}'} R(s, V) = \max_{s \in \mathsf{S}'} \left| V(s) - (BV)(s) \right| \leq \epsilon,$$

where $\mathsf{S}' = \mathsf{S}^{\pi^V}$, i.e., the states reachable from s_0 when following a greedy policy π^V. The functions $R(s, V)$ and $R(\mathbb{S}, V)$ are known as the Bellman residual w.r.t. V of the state s and the SSP $\mathbb{S}$, respectively.

By (2), if V is 0-consistent, then V equals V^*.

Any initial guess V^0 for V^* can be used in VI and if V^0 is a lower bound of V^*, i.e., $V^0(s) \leq V^*(s)$ for all $s \in \mathsf{S}$, then V^0 is also known as an admissible heuristic. For any two value functions V and V', we write $V \leq V'$ if $V(s) \leq V'(s)$ for all $s \in \mathsf{S}$; thus, V^0 is an admissible heuristic if $V^0 \leq V^*$. Another important definition regarding value functions is *monotonicity*:

Definition 4 *(Monotonic value function).* A value function V is monotonic if $V \leq BV$.

The following well-known result is necessary in most of our proofs in this article.

Theorem 1. *Given an SSP $\mathbb{S}$ in which Assumption 1 holds, the operator B preserves [4, Lemma 2.1]:*

- *admissibility: if $V \leq V^*$, then $B^k V \leq V^*$ for $k \in \mathbb{N}^*$; and*
- *monotonicity: if $V \leq BV$, then $V \leq B^k V$ for $k \in \mathbb{N}^*$.*

Another important concept for probabilistic planning is *determinization*, a relaxation of a given probabilistic problem into a deterministic problem $\mathcal{D} = \langle \mathsf{S}, s_0, \mathsf{G}, \mathsf{A}' \rangle$. The set A' contains only deterministic actions represented as $a = s \to s'$, i.e., a deterministically transforms s into s'. Two common determinization procedures are (i) *most-likely outcome*, in which $\mathsf{A}' = \{s \to s' | \exists a \in \mathsf{A} \text{ s.t. } s' = \text{argmax}_{s''} P(s'' | s, a)\}$ (breaking ties randomly); and (ii) *all-outcomes* determinization, where $\mathsf{A}' = \{s \to s' | \exists a \in \mathsf{A} \text{ s.t. } P(s' | s, a) > 0\}$.

2.2. Related work

One direct extension of Value Iteration (VI) is Topological Value Iteration (TVI) [9]. TVI preprocesses the given SSP by performing a topological analysis of S. The result of this analysis is a set of the strongly connected components (SCCs), and TVI solves the SSP by applying VI on each SCC in reversed topological order. This decomposition can speed up the search for

optimal solutions when the original SSP can be decomposed into several close-to-equal-size SCCs. In the worst case, when the SSP has just one SCC, TVI performs worse than VI due to the overhead imposed by the topological analysis.

To increase the chances that a problem will be decomposed in several close-to-equal-size SCCs, Focused Topological Value Iteration (FTVI) [10] was introduced. FTVI performs a best-first forward search in which a lower bound $\underline{V}$ for V^* is iteratively improved and actions that are provably suboptimal are removed from the original SSP. Once the $R(\mathbb{S}, V)$ is small, the search is stopped and the resulting SSP is solved using TVI and $\underline{V}$. Because the removed actions are always suboptimal, FTVI returns an optimal solution. In the worst case, FTVI is equivalent to TVI as there is no guarantee that any action will be removed from the original SSP.

Another extension of VI is Real Time Dynamic Programming (RTDP) [1]. RTDP extends the asynchronous version of VI by using greedy search and sampling to find the next state to perform a Bellman backup. In order to avoid being trapped in loops and to find an optimal solution, RTDP updates its lower bound $\underline{V}(s)$ of $V^*(s)$ on every state s visited during the search. If Assumption 1 holds for the given SSP, then RTDP always finds an optimal solution after several search iterations (possibly infinitely many); that is, RTDP is asymptotically optimal. Unlike VI, TVI, and FTVI that compute complete policies, RTDP returns a closed policy (i.e., the returned policy might be partial).

Several extensions of RTDP have been proposed, and the first one is Labeled RTDP (LRTDP) [5]. LRTDP introduces a labeling mechanism to find states that have already converged and avoids exploring these converged states again. With this technique, LRTDP provides an upper bound on the number of iterations necessary to find an ϵ-consistent solution.

The following three algorithms also extend RTDP by maintaining a lower and an upper bound $\overline{V}$ on V^* and providing different methods to direct the exploration of the state space: Bounded RTDP (BRTDP) [20], Focused RTDP (FRTDP) [25] and, Value of Perfect Information RTDP (VPI-RTDP) [24]. The advantage of keeping an upper bound is that the exploration of the state space can be biased toward states s in which the uncertainty about $V^*(s)$ is large, e.g., the gap between $\overline{V}(s)$ and $\underline{V}(s)$ is large. This improved criterion to guide the search decreases the number of Bellman backups required to find an ϵ-consistent solution; however, each iteration of the search is considerably more expensive due to the maintenance of the upper bound $\overline{V}$. Although no clear dominance exists between RTDP and its extensions, empirically it has been shown that in most problems, (i) RTDP is outperformed by all its extensions, and (ii) VPI-RTDP outperforms BRTDP and FRTDP.

The extensions of RTDP mentioned so far are concerned with improving the convergence of RTDP to an ϵ-consistent solution, and ReTrASE [18] extends RTDP to improve its scalability. ReTrASE achieves this by projecting $\underline{V}$ into a lower dimensional space. The set of basis functions used by ReTrASE is obtained by solving the all-outcomes determinization of the original problem. Due to the lower dimensional representation, ReTrASE is nonoptimal.

A different approach for finding optimal solutions is Policy Iteration (PI) [14]. PI performs search in the policy space and iteratively improves the current policy until no further improvement is possible, i.e., an optimal policy is found. Because PI was originally designed for infinite-horizon MDPs, it returns a complete policy; therefore, when applied to SSPs, PI does not take advantage of the initial state s_0 to prune its search. LAO* [12] can be seen as a version of PI that takes advantage of s_0 and computes optimal closed policies that are potentially not complete. Precisely, LAO* computes an optimal closed policy for the sequence $\mathsf{S}_0 \subseteq \mathsf{S}_1 \subseteq \ldots \subseteq \mathsf{S}_k \subseteq \mathsf{S}$, where $\mathsf{S}_0 = \{s_0\}$ and S_i is generated by greedily expanding S_{i-1}. LAO* stops when $\mathsf{S}^{\pi^*} \subseteq \mathsf{S}_i$; therefore, the optimal closed policy for S^{π^*} is also optimal for the original problem.

Improved LAO* (ILAO*) [12] enhances LAO* performance by (i) increasing how many states are added to S_{i-1} to generate S_i and (ii) performing single Bellman Backups in a depth-first postorder traversal of S_i instead of using PI or VI to compute optimal solutions to S_i. Learning Depth-First Search (LDFS) [6], when applied to SSPs, improves ILAO* by incorporating a labeling mechanism. This labeling mechanism is similar to the mechanism employed by LRTDP and it labels states already converged to avoid revisiting them during the search.

Another technique to solve probabilistic planning problems is replanning. One of the simplest, yet powerful, replanners is FF-Replan [30]. Given a state s (initially, s equals s_0), FF-Replan generates the all-outcomes determinization $\mathcal{D}$ of the problem and uses the deterministic planner FF [13] to solve $\mathcal{D}$ from state s. If and when the execution of the solution for $\mathcal{D}$ fails in the probabilistic environment, FF is reinvoked to plan again from the failed state. FF-Replan was the winner of the first International Probabilistic Planning Competition (IPPC) [33], in which it outperformed the probabilistic planners due to their poor scalability. Despite its major success, FF-Replan is nonoptimal and oblivious to probabilities and dead ends, leading to poor performance in probabilistic interesting problems [19], e.g., the triangle tire domain.

FF-Hindsight [31] is a nonoptimal replanner that generalizes FF-Replan based on hindsight optimization. Given a state s, FF-Hindsight performs the following three steps: (i) it randomly generates a set of nonstationary deterministic problems D starting from s; (ii) it uses FF to solve them; and (iii) it combines the cost of their solutions to estimate the true cost of reaching a goal state from s. Each deterministic problem in D has a fixed horizon and is generated by sampling one outcome of each probabilistic action for each time step. This process reveals two major drawbacks of FF-Hindsight: (i) a bound in the horizon size of the problem is needed in order to produce the relaxed problems; and (ii) rare effects of actions might be ignored by the sampling procedure. While the first drawback is intrinsic to the algorithm, a workaround to the second one is proposed [32] by always adding the all-outcomes determinization of the problem to D and, therefore, ensuring that every effect of an action appears at least in one deterministic problem in D.

Based on solution refinement, two other nonoptimal replanners were proposed: Envelope Propagation (EP) [11] and Robust FF (RFF) [26]. In general terms, EP and RFF compute an initial partial policy π and iteratively expand it to avoid replanning. EP prunes the state space S and represents the removed states by a special meta-state `out` and the appropriate meta-actions to represent the transitions from and to `out`. At each iteration, EP refines its approximation S' of S by

expanding and then repruning S'. Repruning is necessary to avoid the convergence of S' to S. This repruning step is the main drawback of EP because low probability states are pruned and, therefore ignored, and they can represent states that need to be avoided, e.g., high cost states and dead ends. RFF, the winner of the third IPPC [8], uses a different approach: an initial partial policy π is computed by solving the most-likely outcome determinization of the original problem using FF and then the *robustness* of π is iteratively improved. For RFF, robustness is defined as the probability of replanning: given $\rho \in [0, 1]$, RFF computes π such that the probability of replanning when executing π from s_0 is at most ρ.

An orthogonal direction from all other approaches mentioned so far is applied by t-look-ahead [21,23], Upper Confidence bound for Trees (UCT) [17], and UCT-based planner (e.g., Prost [15]): they modify the problems' horizon from indeterminate to finite and choose actions greedily according to the solution of this new relaxed problem. t-look-ahead fixes the horizon of the relaxed problem to t time steps and solves it using dynamic programming. UCT approximates t-look-ahead by solving a series of *multi-armed bandits* problems where each *arm* represents an action and a finite-horizon of t actions are considered. Sparse sampling techniques are employed to efficiently solve this new problem and avoid actions whose cost is far from the best one found so far.

In the context of motion planning, another relevant approach is Variable Level-of-detail Motion Planner [34], in which poorly predictable physical interactions are ignored (pruned) in the far future.

3. Short-sighted stochastic shortest path problems

Depth-based short-sighted Stochastic Shortest Path Problems [28] are a special case of SSPs in which the original problem is transformed into a smaller one by (i) pruning the states that have zero probability of being reached using at most t actions; (ii) adding artificial goal states; and (iii) incrementing the cost of reaching artificial goals by a heuristic value to guide the search toward the goals of the original problem. In this article, we refer to depth-based short-sighted Stochastic Shortest Path Problems as short-sighted SSPs. Short-sighted SSPs are defined based on the action-distance between states (Definition 5) and are formalized in Definition 6.

Definition 5 *($\delta(s, s')$).* The nonsymmetric distance $\delta(s, s')$ between two states s and s' is

$$\delta(s, s') = \begin{cases} 0 & \text{if } s = s' \\ 1 + \min_{a \in \mathsf{A}} \min_{\hat{s}: P(\hat{s}|s,a) > 0} \delta(\hat{s}, s') & \text{otherwise.} \end{cases}$$

$\delta(s, s')$ is equivalent to the minimum number of actions necessary to reach s' from s in the all-outcomes determinization.

Definition 6 *(Short-sighted SSP).* Given an SSP $\mathbb{S} = \langle \mathsf{S}, s_0, \mathsf{G}, \mathsf{A}, P, C \rangle$, a state $s \in \mathsf{S}$, $t \in \mathbb{N}^*$, and a heuristic H, the (s, t)-short-sighted SSP $\mathbb{S}_{s,t} = \langle \mathsf{S}_{s,t}, s, \mathsf{G}_{s,t}, \mathsf{A}, P, C_{s,t} \rangle$ associated with $\mathbb{S}$ is defined as

- $\mathsf{S}_{s,t} = \{ s' \in \mathsf{S} \mid \delta(s, s') \leq t \}$;
- $\mathsf{G}_{s,t} = \{ s' \in \mathsf{S} \mid \delta(s, s') = t \} \cup (\mathsf{G} \cap \mathsf{S}_{s,t})$;
- $C_{s,t}(s', a, s'') = \begin{cases} C(s', a, s'') + H(s'') & \text{if } s'' \in \mathsf{G}_{s,t} \setminus \mathsf{G} \\ C(s', a, s'') & \text{otherwise,} \end{cases} \quad \forall s' \in \mathsf{S}_{s,t}, s'' \in \mathsf{S}_{s,t}, a \in \mathsf{A}$

For simplicity, when the heuristic H is not clear by context nor explicit then $H(s) = 0$ for all $s \in \mathsf{S}$.

Fig. 2(a) shows the $(s_0, 2)$-short-sighted SSP associated with the example in Fig. 1. The state space $\mathsf{S}_{s,t}$ of (s, t)-short-sighted SSPs is a subset of the original state space in which any state $s' \in \mathsf{S}_{s,t}$ is reachable from s using at most t actions. Given a short-sighted SSP $\mathbb{S}_{s,t}$, we refer to the states $s' \in \mathsf{G}_{s,t} \setminus \mathsf{G}$ as artificial goals, and we denote the set of artificial goals by G_a; thus, $\mathsf{G}_a = \mathsf{G}_{s,t} \setminus \mathsf{G}$. The key property of short-sighted SSPs that allows them to be used for solving SSPs is given by the definition of $C_{s,t}$: every artificial goal state $s_a \in \mathsf{G}_a$ has its heuristic value $H(s_a)$ added to the cost of reaching s_a. Therefore, the search for a solution to short-sighted SSPs is guided toward the goal states of the original SSP, even if such states are not in $\mathsf{S}_{s,t}$.

Since short-sighted SSPs are also SSPs, the optimal value function for $\mathbb{S}_{s,t}$, denoted as $V^*_{\mathbb{S}_{s,t}}$, is defined by (2). Although related, the $V^*_{\mathbb{S}_{s,t}}(s)$ and $(B^t H)(s)$, i.e., the t-look-ahead value of s w.r.t. H, are not the same. Before we formally prove their differences, consider the example depicted in Fig. 1 for depth $t = 2$, $p = 0.5$, and the zero-heuristic as H:

- The 2-look-ahead search from s_0 – $(B^2 H)(s_0)$ – represents the minimum expected cost of executing 2 actions in a row, so only trajectories of size 2 are considered. The resulting value is $(B^2 H)(s_0) = 1.5$ and is obtained by applying action a_0 in both s_0 and s'_1. The search space considered to compute $(B^2 H)(s_0)$ in this example is depicted in Fig. 2(b).
- The optimal value function for $\mathbb{S}_{s_0,2}$ on s_0 – $V^*_{\mathbb{S}_{s_0,2}}(s_0)$ – is defined as the minimum expected cost to reach a goal state in $\mathbb{S}_{s_0,2}$ (Fig. 2(a)) from s_0. So all possible trajectories in $\mathbb{S}_{s_0,2}$ are considered, and the maximum length of these trajectories is unbounded due to the loops generated by the policy in which action a_0 is applied in states s_0 and s'_1. In this example, $V^*_{\mathbb{S}_{s_0,2}}(s_0) = 2$ and the greedy policy w.r.t. $V^*_{\mathbb{S}_{s_0,2}}$ is $\langle (s_0, a_1), (s_1, a_1) \rangle$.

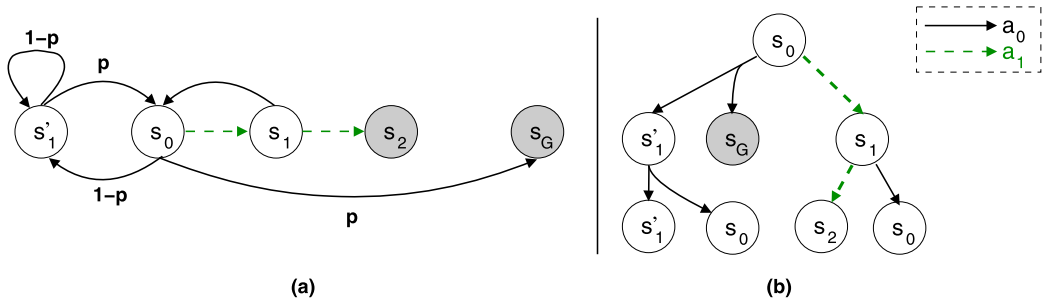

Fig. 2. (a) The $(s_0, 2)$-short sighted SSP associated to the SSP in Fig. 1. The set of goal states $\mathsf{G}_{s_0,2}$ is $\{s_\mathsf{G}, s_2\}$ and s_2 is an artificial goal. The cost function $C_{s_0,2}$ is the same as in the original SSP, except by $C_{s_0,2}(s_1, a_1, s_2) = 1 + H(s_2)$. (b) The 2-step look-ahead search from s_0 in the SSP from Fig. 1.

Precisely, the difference between the look-ahead and short-sighted SSPs is in how the original SSP is relaxed: look-ahead changes the indefinite horizon of the original SSP to a finite horizon; and short-sighted SSPs prune the state space of the original SSP without changing the horizon.

To formally prove the relationship between $V^*_{\mathbb{S}_{s,t}}(s)$ and $(B^t H)(s)$, we introduce $B_{s,t}$, the Bellman operator B applied to the short-sighted SSP $\mathbb{S}_{s,t}$. To simplify our proofs, we define $(B_{s,t}V)(\hat{s})$ to be equal to 0 if $\hat{s} \in \mathsf{G}_{s,t}$ and

$$(B_{s,t}V)(\hat{s}) = \min_{a \in \mathsf{A}} \sum_{s' \in \mathsf{S}_{s,t} \setminus \mathsf{G}_a} P(s' \mid \hat{s}, a)\big[C_{s,t}(\hat{s}, a, s') + V(s')\big] + \sum_{s' \in \mathsf{G}_a} P(s' \mid \hat{s}, a) C_{s,t}(\hat{s}, a, s')$$

for all $\hat{s} \in \mathsf{S}_{s,t} \setminus \mathsf{G}_{s,t}$. The only difference between the definitions of B and $B_{s,t}$ is the explicit treatment of the states $s_a \in \mathsf{G}_a$ in the summation by $B_{s,t}$: $V(s_a)$ is not considered because s_a is an artificial goal of $\mathbb{S}_{s,t}$. If $V(s_a) = 0$ for all $s_a \in \mathsf{G}_a$, then $BV = B_{s,t}V$ for $\mathbb{S}_{s,t}$. Lemmas 2 and 3 relate the operator B applied to an SSP $\mathbb{S}$ with operator $B_{s,t}$ applied to the (s,t)-short-sighted SSP $\mathbb{S}_{s,t}$ associated with $\mathbb{S}$.

Lemma 2. *Given an SSP $\mathbb{S} = \langle \mathsf{S}, s_0, \mathsf{G}, \mathsf{A}, P, C \rangle$ that satisfies Assumption 1, $s \in \mathsf{S}$, $t \in \mathbb{N}^*$ and a monotonic value function V for $\mathbb{S}$, then $(B^k_{s,t}V)(\hat{s}) = (B^k V)(\hat{s})$ for all $\hat{s} \in \mathsf{S}_{s,t} \setminus \mathsf{G}_a$ s.t. $\min_{s_a \in \mathsf{G}_a} \delta(\hat{s}, s_a) \geq k$, where B and $B_{s,t}$ represent, respectively, the Bellman operator applied to $\mathbb{S}$ and $\mathbb{S}_{s,t}$ using V as heuristic.*

Proof. See Appendix A. □

Lemma 3. *Under the conditions of Lemma 2, $(B^k_{s,t}V)(s) \leq (B^k V)(s)$ for all $k \in \mathbb{N}^*$ and $\hat{s} \in \mathsf{S}_{s,t}$, where B and $B_{s,t}$ represent, respectively, the Bellman operator applied to $\mathbb{S}$ and $\mathbb{S}_{s,t}$ using V as heuristic.*

Proof. See Appendix A. □

In Theorem 4, we prove that $V^*_{\mathbb{S}_{s,t}}(s)$ is a lower bound for $V^*(s)$ at least as tight as $(B^t H)(s)$ if H is a monotonic lower bound on V^* and Assumption 1 holds for $\mathbb{S}$. Corollary 5 shows that $V^*_{\mathbb{S}_{s,t}}(s)$ is always a tighter lower bound than $(B^t H)(s)$ if $\mathbb{S}$ has unavoidable loops (Definition 7).

Theorem 4. *Given an SSP $\mathbb{S} = \langle \mathsf{S}, s_0, \mathsf{G}, \mathsf{A}, P, C \rangle$ that satisfies Assumption 1, $s \in \mathsf{S}$, $t \in \mathbb{N}^*$ and a monotonic lower bound H for V^*, then*

$$\big(B^t H\big)(s) \leq V^*_{\mathbb{S}_{s,t}}(s) \leq V^*(s).$$

Proof. By the definition of $\mathbb{S}_{s,t}$, $\min_{s_a \in \mathsf{G}_a} \delta(s, s_a) = t$, thus by Lemma 2, we have that $(B^t H)(s) = (B^t_{s,t} H)(s)$. Because H is monotonic and $V^*_{\mathbb{S}_{s,t}}(s) = (\lim_{k \to \infty} B^k_{s,t} H)(s)$, then $(B^t H)(s) \leq V^*_{\mathbb{S}_{s,t}}(s)$. By Lemma 3, we have that $V^*_{\mathbb{S}_{s,t}}(s) \leq V^*(s)$. □

Definition 7 *(Unavoidable loops).* An SSP $\mathbb{S} = \langle \mathsf{S}, s_0, \mathsf{G}, \mathsf{A}, P, C \rangle$ that satisfies Assumption 1 has unavoidable loops if, for every optimal policy π^* of $\mathbb{S}$, the directed graph $G = (\mathsf{S}^{\pi^*}, E)$ contains at least one cycle, where $E = \{(s, s') | P(s'|s, \pi^*(s)) > 0\}$.

Corollary 5. *In Theorem 4, if the (s,t)-short-sighted SSP $\mathbb{S}_{s,t}$ has unavoidable loops (Definition 7), then $(B^t H)(s) < V^*_{\mathbb{S}_{s,t}}(s)$.*

Proof. By definition, $(B^t H)(s)$ considers only trajectories of size at most t from s. Since $V^*_{\mathbb{S}_{s,t}}(s) = \lim_{k \to \infty}(B^k_{s,t} H)(s)$, then all possible trajectories on $\mathbb{S}_{s,t}$ are considered by $V^*_{\mathbb{S}_{s,t}}$. By assumption, $\mathbb{S}_{s,t}$ has unavoidable loops, so the maximum size of a

trajectory generated by $\pi^*_{s,t}$ is unbounded. As every trajectory has nonzero probability and nonzero cost by definition, then $(B^t_{s,t}H)(s) < V^*_{\mathbb{S}_{s,t}}(s)$, and by Lemma 2, we have that $(B^tH)(s) < V^*_{\mathbb{S}_{s,t}}(s)$. □

Another important relationship between SSPs and short-sighted SSPs is through their policies. To formalize this relationship, we first define the concept of *t-closed policy w.r.t. s*, i.e., policies that can be executed from s independent of the probabilistic outcome of actions for at least t actions without replanning:

Definition 8 *(t-closed policy).* A policy π for an SSP $\mathbb{S} = \langle \mathsf{S}, s_0, \mathsf{G}, \mathsf{A}, P, C\rangle$ is t-closed w.r.t. a state $s \in \mathsf{S}$ if, for all $s' \in \mathsf{R}^\pi \cap \mathsf{S}^\pi$, $\delta(s, s') \geq t$.

All the replanners reviewed on Section 2.2 compute 1-closed policies w.r.t. the current state, i.e., there is no guarantee that the partial policy computed by them can be executed for more than one action without replanning. Notice that when $t \to \infty$, t-closed policies w.r.t. s_0 are equivalent to closed policies; Proposition 6 gives an upper bound on t for when a t-closed policy w.r.t. s_0 becomes a closed policy.

Proposition 6. *Given an SSP $\mathbb{S} = \langle \mathsf{S}, s_0, \mathsf{G}, \mathsf{A}, P, C\rangle$, for $t \geq |\mathsf{S}|$, every t-closed policy w.r.t. s_0 for $\mathbb{S}$ is also a closed policy for $\mathbb{S}$.*

Proof. Since π is t-closed w.r.t. s_0 for $t \geq |\mathsf{S}|$, then for all $s' \in \mathsf{R}^\pi \cap \mathsf{S}^\pi$, $\delta(s_0, s') \geq |\mathsf{S}|$. By the definition of S^π, we have that all $s' \in \mathsf{S}^\pi$ is reachable from s_0 when following π. Thus, $\delta(s_0, s') < |\mathsf{S}|$ because there exists a trajectory from s_0 to s' that visits each state at most once, i.e., that uses at most $|\mathsf{S}| - 1$ actions. Therefore, $\nexists s' \in \mathsf{S}^\pi$ such that $\delta(s_0, s') \geq |\mathsf{S}|$ and $\mathsf{R}^\pi \cap \mathsf{S}^\pi = \emptyset$. □

Policies for SSPs and policies for their associated (s, t)-short-sighted SSPs are related through the concept of t-closed policies w.r.t. s as follows:

Proposition 7. *Given an SSP $\mathbb{S} = \langle \mathsf{S}, s_0, \mathsf{G}, \mathsf{A}, P, C\rangle$, a state $s \in \mathsf{S}$, and $t \in \mathbb{N}^*$, then π is a closed policy for $\mathbb{S}_{s,t}$ if and only if π is a t-closed policy w.r.t. s for $\mathbb{S}$.*

Proof. We assume that π is a closed policy for $\mathbb{S}_{s,t}$, i.e., $\mathsf{R}^\pi_{s,t} \cap \mathsf{S}^\pi_{s,t} = \emptyset$. For contradiction purposes, suppose that there exists $s' \in \mathsf{R}^\pi \cap \mathsf{S}^\pi$ such that $\delta(s, s') < t$. Because $\delta(s, s') < t$, then $s' \in \mathsf{S}_{s,t}$; thus, $s' \in \mathsf{S}^\pi_{s,t} \subseteq \mathsf{S}^\pi$ and $s' \in \mathsf{R}^\pi_{s,t} \subseteq \mathsf{R}^\pi$. This is a contradiction because $\mathsf{R}^\pi_{s,t} \cap \mathsf{S}^\pi_{s,t} = \emptyset$; therefore, for all $s' \in \mathsf{R}^\pi \cap \mathsf{S}^\pi$, $\delta(s, s') \geq t$, i.e., π is t-closed w.r.t. s for $\mathbb{S}$.

Now, we assume that π is t-closed w.r.t. s for $\mathbb{S}$, i.e., for all $s' \in \mathsf{R}^\pi \cap \mathsf{S}^\pi$, $\delta(s, s') \geq t$. By the definition of $\mathbb{S}_{s,t}$, for all $s' \in \mathsf{S}_{s,t}$, $\delta(s, s') \leq t$. Thus, if $s' \in (\mathsf{R}^\pi \cap \mathsf{S}^\pi) \cap \mathsf{S}_{s,t}$, then $\delta(s, s') = t$, i.e., $s' \in \mathsf{G}_{s,t} \setminus \mathsf{G}$. Because $\mathsf{R}^\pi_{s,t} \cap \mathsf{G}_{s,t} = \emptyset$ by the definition of R^π and $\mathsf{S}^\pi_{s,t} = \mathsf{S}^\pi \cap \mathsf{S}_{s,t}$, then $\mathsf{R}^\pi_{s,t} \cap \mathsf{S}^\pi_{s,t} = \emptyset$, i.e., π is closed for $\mathbb{S}_{s,t}$. □

4. Short-Sighted Probabilistic Planner

The Short-Sighted Probabilistic Planner (SSiPP) [28] is an algorithm that solves SSPs based on short-sighted SSPs. SSiPP is presented in Algorithm 1 and consists of iteratively generating and solving short-sighted SSPs of the given SSP. Due to the reduced size of the short-sighted problems, SSiPP computes the optimal solution for each of them by calling the external procedure Optimal-SSP-Solver (line 7).[1] Therefore, SSiPP obtains a "fail-proof" solution, i.e., a closed policy for each short-sighted SSP generated. Due to Proposition 7, each policy $\pi^*_{\mathbb{S}_{s,t}}$ obtained in line 7 of Algorithm 1 is a t-closed policy w.r.t. the current state s for original SSP $\mathbb{S}$; therefore, $\pi^*_{\mathbb{S}_{s,t}}$ can be simulated or directly executed in the environment (line 11) for at least t steps before replanning is needed, i.e., before another short-sighted SSP is generated and solved.

To illustrate the execution of SSiPP, consider as input the SSP $\mathbb{S}$ in Fig. 1 for $t = 2$. The first short-sighted SSP built by the algorithm is $\mathbb{S}_{s_0,2}$ (Fig. 2(a)), and there are only two possible 2-closed policies for $\mathbb{S}_{s_0,2}$ that are proper: (i) $\pi_0 = \{(s_0, a_0), (s'_1, a_0)\}$; and (ii) $\pi_1 = \{(s_0, a_1), (s_1, a_1)\}$. Depending on the value of k (the length of the chain in $\mathbb{S}$) and the heuristic H used, Optimal-SSP-Solver can return either π_0 or π_1. If the former is returned, then the original SSP $\mathbb{S}$ is solved, because π_0 is a closed policy for $\mathbb{S}$. Instead, if π_1 is returned, then $\lceil \frac{k+1}{2} \rceil$ short-sighted SSPs, representing the 3-states subchains of $s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_1} \ldots \xrightarrow{a_1} s_{\mathsf{G}}$, are generated and solved.

SSiPP, as shown in Algorithm 1, only samples one trajectory of the SSP being solved and is not guaranteed to obtain an optimal solution. Nonetheless, SSiPP can be used for computing the optimal solutions by iteratively improving the current lower bound $\underline{V}$ until convergence is reached. This approach is formalized by Run-SSiPP-Until-Convergence (Algorithm 2).

In the remainder of this section, we prove that SSiPP performs Bellman backups (Theorem 8); SSiPP terminates (Theorem 9); and Algorithm 2 is asymptotically optimal (Theorem 10), that is, if the same problem is solved sufficiently many times by SSiPP, then an optimal policy is found.

[1] In practice, Optimal-SSP-Solver stops when an ϵ-consistent value function is found.

```
1  SSiPP(SSP 𝕊 = ⟨S, s_0, G, A, P, C⟩, t ∈ ℕ*, H a heuristic for V* and ε > 0)
2  begin
3      V ← Value function for S with default value given by H
4      s ← s_0
5      while s ∉ G do
6          𝕊_{s,t} ← Generate-Short-Sighted-SSP(𝕊, s, V, t)
7          (π*_{𝕊_{s,t}}, V*_{𝕊_{s,t}}) ← Optimal-SSP-Solver(𝕊_{s,t}, V, ε)
8          forall the s' ∈ S^{π*_{𝕊_{s,t}}} \ G_{s,t} do
9              V(s') ← V*_{𝕊_{s,t}}(s')
10         while s ∉ G_{s,t} do
11             s ← execute-action(π*_{𝕊_{s,t}}(s))
12     return V
```

Algorithm 1: SSiPP algorithm [28]. Any SSP optimal solver can be used as Optimal-SSP-Solver, e.g., value iteration and RTDP and, in practice, Optimal-SSP-Solver returns an ϵ-consistent solution. Notice that $V^*_{\mathbb{S}_{s,t}}$ needs to be defined only for the states reachable from s when following $\pi^*_{\mathbb{S}_{s,t}}$, i.e., for $s' \in \mathsf{S}^{\pi^*_{\mathbb{S}_{s,t}}}$.

```
1  Run-SSiPP-Until-Convergence(SSP 𝕊 = ⟨S, s_0, G, A, P, C⟩, t ∈ ℕ*, H a heuristic for V* and ε > 0)
2  begin
3      V ← Value function for S with default value given by H
4      while R(𝕊, V) > ε do
5          V ← SSiPP(𝕊, t, V, ε)
6      return V
```

Algorithm 2: Algorithm to compute an ϵ-consistent value function using SSiPP (Algorithm 1).

Theorem 8. *Given an SSP $\mathbb{S} = \langle \mathsf{S}, s_0, \mathsf{G}, \mathsf{A}, P, C\rangle$ such that Assumption 1 holds and a monotonic lower bound H for V^*, then the loop in line 8 of SSiPP (Algorithm 1) is equivalent to applying at least one Bellman backup on $\underline{V}$ for every state $s' \in \mathsf{S}^{\pi^*_{\mathbb{S}_{s,t}}} \setminus \mathsf{G}_{s,t}$.*

Proof. Let $\hat{\mathsf{S}}$ denote $\mathsf{S}^{\pi^*_{\mathbb{S}_{s,t}}} \setminus \mathsf{G}_{s,t}$. After the loop in line 8 is executed, for all $s' \in \hat{\mathsf{S}}$, $\underline{V}(s')$ equals $V^*_{\mathbb{S}_{s,t}}(s')$ for all $s' \in \hat{\mathsf{S}}$. Thus, we need to prove that $(B\underline{V})(s') \leq V^*_{\mathbb{S}_{s,t}}(s')\ \forall s' \in \hat{\mathsf{S}}$ since $\underline{V}$ is monotonic and admissible (Theorem 1). By the definition of short-sighted SSP (Definition 6), every state $s' \in \hat{\mathsf{S}}$ is such that $\{s'' \in \mathsf{S} | P(s''|s', a) > 0, \forall a \in \mathsf{A}\} \subseteq \mathsf{S}_{s,t}$, i.e., the states reached after applying an action in a state $s' \in \hat{\mathsf{S}}$ belong to $\mathsf{S}_{s,t}$. Therefore, $(B\underline{V})(s') = (B_{s,t}\underline{V})(s')\ \forall s' \in \hat{\mathsf{S}}$, where $B_{s,t}$ is the Bellman operator B for $\mathbb{S}_{s,t}$. Since $\underline{V}$ is monotonic and admissible, $(B_{s,t}\underline{V})(s') \leq V^*_{\mathbb{S}_{s,t}}(s')$. Therefore, $(B\underline{V})(s') \leq V^*_{\mathbb{S}_{s,t}}(s')\ \forall s' \in \hat{\mathsf{S}}$. □

Theorem 9. *SSiPP always terminates under the conditions of Theorem 8.*

Proof. Suppose SSiPP does not terminate. Then there exists a trajectory $\mathcal{T}$ of infinite size that can be generated by SSiPP, so there must be an infinite loop in $\mathcal{T}$ because S is finite by definition. Since $\pi^*_{\mathbb{S}_{s,t}}$ is proper for $\mathbb{S}_{s,t}$, the loop encompassed by lines 10 and 11 always terminates. Therefore the main loop (lines 5 to 11) is executed infinitely many times and $\underline{V}(s)$ diverges. Because Assumption 1 holds for $\mathbb{S}$, we have that $V^*(s) < \infty$ for all $s \in \mathsf{S}$. This is a contradiction since SSiPP maintains $\underline{V}$, initialized as H, admissible and monotonic (Theorems 4 and 8), i.e., $\underline{V}(s) \leq V^*(s)$ for all $s \in \mathsf{S}$. □

Theorem 10. *Given an SSP $\mathbb{S} = \langle \mathsf{S}, s_0, \mathsf{G}, \mathsf{A}, P, C\rangle$ such that Assumption 1 holds, a monotonic lower bound H for V^* and $t \in \mathbb{N}^*$, then the sequence $\langle \underline{V}^0, \underline{V}^1, \cdots, \underline{V}^k\rangle$, where $\underline{V}^0 = H$ and $\underline{V}^i = \text{SSiPP}(\mathbb{S}, t, \underline{V}^{i-1})$, converges to V^* as $k \to \infty$ for all $s \in \mathsf{S}^{\pi^*}$.*

Proof. Let the sequence of states $\mathcal{H}_k = \langle s_0, s_1, s_2, \ldots\rangle$ be the concatenation of the trajectories $\mathcal{T}_0, \cdots, \mathcal{T}_k$ of states visited by SSiPP when $\underline{V}^i$ is computed for $i \in \{0, \ldots, k\}$. By Theorem 9, each $\mathcal{T}_i$ has finite size, so $|\mathcal{H}_k|$ is finite for $k \in \mathbb{N}$. Because Assumption 1 holds for $\mathbb{S}$, and H is admissible and monotonic, when $k \to \infty$, we can construct an SSP $\mathbb{S}_\infty = \langle \mathsf{S}_\infty, s_0, \mathsf{G}_\infty, A_\infty, P, C\rangle$ such that [1, Theorem 3, p. 132]: $\mathsf{S}_\infty \subseteq \mathsf{S}$ is the nonempty set of states that appear infinitely often in $\lim_{k\to\infty} \mathcal{H}_k$; $\mathsf{G}_\infty \subseteq \mathsf{G}$ is the nonempty set of goal states that appear infinitely often in $\lim_{k\to\infty} \mathcal{H}_k$; and $A_\infty \subseteq A$ is the set of actions a such that $P(s'|s, a) = 0$ for all $s \in \mathsf{S}_\infty$ and $s' \in \mathsf{S} \setminus \mathsf{S}_\infty$. Therefore, there is a finite time step T such that the sequence $\mathcal{H}'$ of states visited after time step T contains only states in S_∞. By Theorem 8, we know that at least one Bellman backup is applied to s_j for any time step j. Thus, after time step T, the sequence of Bellman backups applied by SSiPP is equivalent to asynchronous value iteration on $\mathbb{S}_\infty$, and $\underline{V}^k(s)$ converges to $V^*(s)$ for all $s \in \mathsf{S}_\infty$ as $k \to \infty$ [4, Proposition 2.2, p. 27]. Furthermore, $\mathsf{S}^{\pi^*} \subseteq \mathsf{S}_\infty$ [1, Theorem 3]. □

```
1  CheckSolved(SSP 𝕊 = ⟨S, s_0, G, A, P, C⟩, state s ∈ S, value function V, set of states solved and ε > 0)
2  begin
3      conv ← true
4      open ← Empty-Stack
5      closed ← Empty-Stack
6      if s ∉ solved then open.Push(s)
7      while not open.isEmpty() do
8          s ← open.Pop()
9          closed.Push(s)
10         if s ∈ (G ∪ solved) then Continue
11         if R(s, V) > ε then
12             conv ← false
13             Continue
14         forall the s' s.t. P(s'|s, π^V(s)) > 0 do
15             if s' ∉ (solved ∪ open ∪ closed) then
16                 open.Push(s')
17     if conv = true then
18         forall the s' ∈ closed do
19             solved ← solved ∪ {s'}
20     else
21         while not closed.isEmpty() do
22             s ← closed.Pop()
23             V(s) ← (BV)(s)
24     return (solved, V)
```

Algorithm 3: CheckSolved from [5].

4.1. Improving the convergence of SSiPP

SSiPP obtains the next state s' from the current state s by either executing or sampling one outcome of the optimal policy $\pi^*_{\mathbb{S}_{s,t}}$ of the current short-sighted SSP (Algorithm 1 line 11). This procedure is repeated until s' is a goal state, either from the original SSP or an artificial goal.

Real Time Dynamic Programming (RTDP) [1] employs a similar technique: the next state s' is obtained by sampling an outcome of $\pi^V(s)$. The advantage of this unbiased sampling procedure is that more likely states are updated more often. As noticed in [5], the convergence of a given state s depends on all its reachable successors, so unlikely successors should also be visited. As a result, for a given state s, unbiased sampling might not update unlikely successors of s frequently, thus delaying the overall convergence of V.

Labeled RTDP (LRTDP) [5] extends RTDP by tracking the states that the estimate V of V^* has already converged and not visiting these states again. To find and label the converged states, the procedure CheckSolved (Algorithm 3) is introduced. Given a state s, CheckSolved searches for states s' reachable from s when following a greedy policy π^V such that $R(s', V) > \epsilon$. If no such state s' is found, then s and all the states in S^{π^V} reachable from s have converged, and they are labeled as solved. Alternatively, if there exists s' reachable from s when following a greedy policy π^V such that $R(s', V) > \epsilon$, then a Bellman backup is applied on at least $V(s')$. A key property of CheckSolved is that if V has not converged, then a call to CheckSolved either improves V or labels a new state as solved:

Theorem 11. *(See [5, Theorem 4].) Given an SSP $\mathbb{S} = \langle \mathsf{S}, s_0, \mathsf{G}, \mathsf{A}, P, C \rangle$ that satisfies Assumption 1, $\epsilon > 0$, and a monotonic lower bound V for V^*, then a* CheckSolved$(\mathbb{S}, s, V, solved, \epsilon)$ *call for $s \notin solved$ that returns $(solved', V')$ either: labels a state as solved (i.e., $|solved'| > |solved|$) or there exists $s' \in \mathsf{S}$ such that $V'(s') - V(s') > \epsilon$.*

Using the solved labels, the sampling procedure of LRTDP can be seen as a case of rejection sampling: if the sampled successor s' of s is marked as solved, restart the procedure from the initial state s_0; otherwise, use s'. This new sampling procedure gives LRTDP both a better anytime performance and a faster convergence when compared to RTDP.

Labeled-SSiPP (Algorithm 4) is an extension of Run-SSiPP-Until-Convergence (Algorithm 2) that incorporates the labeling mechanism of LRTDP and uses the CheckSolved procedure. Since the states marked as solved have already converged, there is no need to further explore and update them; therefore, the solved states are also considered as artificial goals for generated short-sighted SSPs (Algorithm 4 line 10). By adding the solved states to the goal set of the generated short-sighted SSPs, any algorithm used as Optimal-SSP-Solver (line 13) will implicitly take advantage of the labeling mechanism, i.e., the search is stopped once a solved state is reached.

The simulation of the current short-sighted SSP (Algorithm 4 line 16) for Labeled-SSiPP finishes when the state s is either (i) a goal state of the original problem, (ii) a solved state, or (iii) an artificial goal. Only in the last case, the algorithm continues to generate short-sighted SSPs. Thus, Labeled-SSiPP (as LRTDP) also employs rejection sampling: if a solved state is sampled, then the search restarts from the initial s_0.

```
1  LABELED-SSIPP(SSP 𝕊 = ⟨S, s_0, G, A, P, C⟩, t ∈ ℕ*, H a heuristic for V* and ε > 0)
2  begin
3      V ← Value function for S with default value given by H
4      solved ← ∅
5      while s_0 ∉ solved do
6          s ← s_0
7          visited ← EMPTY-STACK
8          while s ∉ (G ∪ solved) do
9              𝕊_{s,t} ← GENERATE-SHORT-SIGHTED-SSP(𝕊, s, V, t)
10             forall the s' ∈ S_{s,t} do
11                 if s' ∈ solved then
12                     G_{s,t} ← G_{s,t} ∪ {s'}
13             (π*_{𝕊_{s,t}}, V*_{𝕊_{s,t}}) ← OPTIMAL-SSP-SOLVER(𝕊_{s,t}, V, ε)
14             forall the s' ∈ S^{π*_{𝕊_{s,t}}} \ G_{s,t} do
15                 V(s') ← V*_{𝕊_{s,t}}(s')
16             while s ∉ G_{s,t} do
17                 visited.PUSH(s)
18                 s ← simulate-action(π*_{𝕊_{s,t}}(s))
19         while not visited.ISEMPTY() do
20             s ← visited.POP()
21             (solved, V) ← CHECKSOLVED(𝕊, s, V, solved, ε)
22             if s ∉ solved then
23                 break
24     return V
```

Algorithm 4: LABELED-SSIPP: Extension of SSiPP that incorporates the LRTDP labeling mechanism. In practice, OPTIMAL-SSP-SOLVER returns an ϵ-consistent solution. CHECKSOLVED is presented in Algorithm 3.

Besides the empirical advantage of LRTDP over RTDP shown in [5], the labeling mechanism also allows to upper bound the maximum number of iterations necessary for LRTDP to converge to the ϵ-consistent solution. This same upper bound holds for Labeled-SSiPP:

Corollary 12. *Given an SSP $\mathbb{S} = \langle \mathsf{S}, s_0, \mathsf{G}, \mathsf{A}, P, C\rangle$ that satisfies Assumption 1, $\epsilon > 0$, $t \in \mathbb{N}^*$ and a monotonic lower bound H for V^*, then Labeled-SSiPP (Algorithm 4) finds an ϵ-consistent lower bound $\underline{V}$ after at most $\epsilon^{-1}\sum_{s\in\mathsf{S}} V^*(s) - H(s)$ iterations of the loop in line 5.*

Proof. In each iteration of the loop in line 5 of Algorithm 4, CHECKSOLVED is called for at least one state $\hat{s} \notin solved$, since $s_0 \notin solved$. By Theorem 11, after CHECKSOLVED is called for $\hat{s}$, either (i) $\hat{s} \in solved$; or (ii) there exists $s' \notin solved$ reachable from $\hat{s}$ when following a greedy policy $\pi^{\underline{V}}$ such that $\underline{V}(s') - \underline{V}^{\text{old}}(s') > \epsilon$, where $\underline{V}^{\text{old}}$ denotes $\underline{V}$ before the CHECKSOLVED call. Thus, in the worst case, each CHECKSOLVED call improves $\underline{V}$ for exactly one state $s' \notin solved$. Therefore, CHECKSOLVED is called at most $\epsilon^{-1}\sum_{s\in\mathsf{S}} V^*(s) - H(s)$ times before $s_0 \in solved$, which is the termination condition for the loop in line 5. □

4.2. Combining SSiPP and FF

As shown in [28], SSiPP has good performance in probabilistic interesting domains; however, SSiPP does not scale up as well as determinization-based replanners such as FF-Replan for problems without dead ends in which many nonoptimal solutions are available, such as blocks world. In this section, we show how to combine the SSiPP and determinization to improve the scalability of SSiPP in such domains while dropping SSiPP's optimality guarantee.

Algorithm 5 shows SSSiPP-FF, an extension of SSiPP, that combines t-closed policies and plans obtained through the determinization of the original SSP. The rationale behind SSiPP-FF is to generate and solve short-sighted SSPs only in regions of the SSP in which FF-Replan chooses a poor solution. These regions are found when the plan computed by FF-Replan fails, e.g., due to its low probability of success.

Formally, after reaching an artificial goal s, SSiPP-FF computes a determinization $\mathcal{D}$ of the original SSP; runs FF to solve $\mathcal{D}$ using s as initial state; and executes the returned plan until failure (lines 12 to 17 in Algorithm 5). Any determinization can be used by SSiPP-FF (line 13), and if the chosen determinization is stationary (e.g., all-outcomes and most-likely determinization) then the deterministic representation of $\mathbb{S}$ can be precomputed and reused in every iteration to generate $\mathcal{D}$. Since SSiPP-FF does not assume any specific behavior of FF, any deterministic planner can be used for solving $\mathcal{D}$ in line 14 instead of FF.

Besides taking advantage of potential nonoptimal solutions, SSiPP-FF also improves the behavior of FF-Replan by not reaching avoidable dead ends in the generated short-sighted SSPs. Formally, suppose that a short-sighted SSP $\mathbb{S}_{s,t}$ generated in line 6 of Algorithm 5 has an avoidable dead end (i.e., there exists at least one proper policy π for $\mathbb{S}_{s,t}$) thus $s_d \notin \mathsf{S}^{\pi}_{s,t}$

```
1  SSiPP-FF(SSP 𝕊 = ⟨S, s_0, G, A, P, C⟩, t ∈ ℕ*, H a heuristic for V*, ϵ > 0)
2  begin
3      V ← Value function for S with default value given by H
4      s ← s_0
5      while s ∉ G do
6          𝕊_{s,t} ← Generate-Short-Sighted-SSP(𝕊, s, V, t)
7          (π*_{𝕊_{s,t}}, V*_{𝕊_{s,t}}) ← Optimal-SSP-Solver(𝕊_{s,t}, V, ϵ)
8          forall the s' ∈ S^{π*_{𝕊_{s,t}}} \ G_{s,t} do
9              V(s') ← V*_{𝕊_{s,t}}(s')
10         while s ∉ G_{s,t} do
11             s ← execute-action(π*_{𝕊_{s,t}}(s))
12         if s ∉ G then
13             𝒟 ← Determinize(𝕊)
14             ⟨s_1, a_1, s_2, ..., a_{k-1}, s_k⟩ ← CallFF(𝒟, s)
15             for i ∈ {1, ..., k − 1} do
16                 if s ≠ s_i then break
17                 s ← execute-action(a_i)
18     return V
```

Algorithm 5: SSiPP-FF: Extension of SSiPP that incorporates determinizations to efficiently obtain a nonoptimal solution.

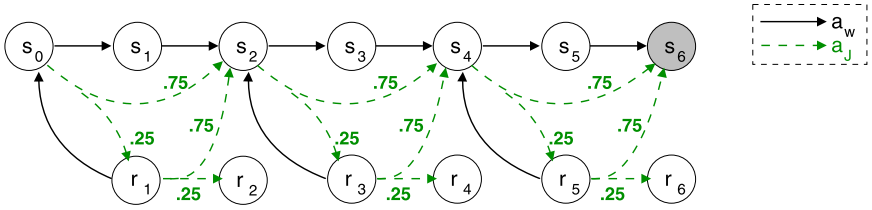

Fig. 3. Representation of the jumping chain problem (Example 1) for $k = 3$. The initial state is s_0, the goal set is $\mathsf{G} = \{s_6\}$. Actions a_W and a_J have cost 1 and 3 respectively.

for all dead end states $s_d \in \mathsf{S}_{s,t}$. Since an optimal policy $\pi^*_{\mathbb{S}_{s,t}}$ is computed for $\mathbb{S}_{s,t}$ (line 7), then $\pi^*_{\mathbb{S}_{s,t}}$ is one of the existing proper policies by the definition of optimal policies; therefore, the avoidable dead ends are not reached by executing $\pi^*_{\mathbb{S}_{s,t}}$. The guarantee of not reaching avoidable dead ends is not particular from SSiPP-FF; instead, this guarantee is inherited from SSiPP.

We finish this section by showing a series of problems in which SSiPP-FF avoids all dead ends while determinization approaches based on the shortest distance to the goal (e.g., FF-Replan) reach a dead end with probability exponentially large in the problem size.

Example 1 *(Jumping chain).* For $k \in \mathbb{N}^*$, the k-th jumping chain problem has $3k + 1$ states: $\mathsf{S} = \{s_0, s_1, \cdots, s_{2k}, r_1, r_2, \cdots, r_k\}$. The initial state is s_0 and the goal set is $\mathsf{G} = \{s_{2k}\}$. Two actions are available, a_W (*walk*) and a_J (*jump*), and their costs are, respectively, 1 and 3, independent of the current and resulting state. The *walk* action is deterministic: $P(s_{i+1}|s_i, a_W) = 1$ for all i, $P(s_{i-1}|r_i, a_W) = 1$ for i odd; and $P(r_i|r_i, a_W) = 1$ for i even. When the *jump* action is applied to s_i, for i even, the resulting state is s_{i+2} with probability 0.75 and r_{i+1} with probability 0.25; if i is odd, then a_J does not change the current state, i.e., $P(s_i|s_i, a_J) = 1$. For the states r_i, a_J is such that $P(r_i|r_i, s_J) = 1$ for even i, and for odd i, $P(s_{i+1}|r_i, s_J) = 0.75$ and $P(r_{i+1}|r_i, s_J) = 0.25$. For all i even, r_i is a dead end. The jumping chain problem for $k = 3$ is depicted in Fig. 3.

In the jumping chain problems, FF-Replan using both the most-likely outcomes and all-outcomes determinization are equivalent because the low probability effect of *jump*, i.e., move to a state r_i, is less helpful than its most-likely effect. When in a state r_i, for i odd, FF-Replan never chooses the action *walk* because (i) *walk* results in a state further away from the goal, and (ii) *jump* has a nonzero probability to reach a state in which the goal is still achievable. Therefore, the solutions obtained by FF-Replan have a nonzero probability of reaching a dead end, i.e., a state r_i for i even. Formally, the probability of FF-Replan reaching the goal for the k-th jumping chain problem is $(2p - p^2)^k$ for $p = 0.75$.

Alternatively, SSiPP-FF always reaches the goal for $t \in \mathbb{N}^*$ and the following trivial extension of the zero-heuristic: $h_d(s) = \infty$ if $P(s|s, a) = 1$ for all $a \in \mathsf{A}$ and $h_d(s) = 0$ otherwise. Formally, a dead end r_i (for i even) can only be reached when a_J is applied in r_{i-1}, and to show that SSiPP-FF never reaches r_i, we need to show that (i) $\pi^*_{\mathbb{S}_{s,t}}$ generated on line 7 never applies a_J on r_i; and (ii) if $r_i \in \mathsf{G}_{s,t}$, then $\pi^*_{\mathbb{S}_{s,t}}$ does not reach r_i since the determinization part of SSiPP-FF (line 14) would apply a_J. The former case is true since $\pi^*_{\mathbb{S}_{s,t}}$ is an optimal policy and $h_d(s_{i-1}) = 0 < h_d(r_{i+1}) = \infty$; therefore, $\pi^*_{\mathbb{S}_{s,t}}(r_i) = a_W$. In the latter case, if $r_i \in \mathsf{G}_{s,t}$, then $\{s_i, s_{i+1}\} \subset \mathsf{G}_{s,t}$. Since $h_d(r_i) = h_d(s_i) = h_d(s_{i+1}) = 0$ and $C(s_{i-1}, a_W, s_i) = 1 < C(s_{i-1}, a_J, \cdot) = 3$, then $\pi^*_{\mathbb{S}_{s,t}}(s_{i-1}) = a_W$ and the value of s in line 14 of SSiPP-FF is s_{i+1}. Therefore,

Table 1
Number of blocks and the cost of actions *pick-up* and *pick-up-from-table* for each of the 15 problems considered from the probabilistic blocks world.

Problem #	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Number of blocks	5				10				14				18		
Cost of *pick-up*	1	2	2	3	1	2	2	4	1	2	2	4	1	2	4
Cost of *pick-up-from-table*	1	2	3	2	1	2	3	3	1	2	3	3	1	2	3

SSiPP-FF using h_d always reaches the goal for $t \in \mathbb{N}^*$. SSiPP-FF can obtain a speedup over SSiPP in the jumping chain problems if the determinization solution can be efficiently obtained.

5. Experiments

We present two sets of experiments to compare (i) the convergence time of SSiPP (using Algorithm 2) and Labeled-SSiPP (Section 5.2); and (ii) the performance of SSiPP, Labeled-SSiPP, and SSiPP-FF in the settings of the International Probabilistic Planning Competition (IPPC) [33,7,8] (Section 5.3). The domains used in both experiments are described in Section 5.1.

For all experiments, SSiPP, Labeled-SSiPP, and SSiPP-FF use LRTDP [5] as Optimal-SSP-Solver to find 10^{-4}-consistent solutions for the short-sighted SSPs.

5.1. Domains and problems

We present the four domains from IPPC'08 [8], which we use in our experiments.[2] The first two domains, probabilistic blocks world (Section 5.1.1) and zeno travel (Section 5.1.2), are probabilistic extensions of their deterministic counterparts. Triangle tire world (Section 5.1.3) and exploding blocks world (Section 5.1.4) are probabilistic interesting problems [19], i.e., problems in which approaches that oversimplify the probabilistic structure of the actions perform poorly.

5.1.1. Probabilistic blocks world

The probabilistic blocks world is an extension of the well-known blocks world in which the actions *pick-up* and *put-on-block* can fail with probability 1/4. If and when these actions fail, the target block is dropped on the table. For instance, *pick-up* A from B results in block A being on the table with probability 1/4. The action *pick-up-from-table* also fails with probability 1/4, in which case nothing happens, i.e., the target block remains on the table. Lastly, the action *put-down* deterministically puts the block being held on the table.

This probabilistic version of blocks world also contains three new actions that allow towers of two blocks to be manipulated: *pick-tower*, *put-tower-on-block*, and *put-tower-down*. While action *put-tower-down* deterministically puts the tower still assembled on the table, the other two actions are probabilistic and fail with probability 9/10. The current state is not changed when *pick-tower* fails and *put-tower-on-block* fails by dropping the tower on the table (the dropped tower remains built).

Since every action in the probabilistic blocks world is reversible, the goal is always reachable from any state; therefore, Assumption 1 holds for all problems in this domain. The actions *put-on-block*, *put-down*, *pick-tower*, *put-tower-on-block*, and *put-tower-down* have cost 1. In order to explore the trade-offs between putting a block on top of other blocks versus putting a block on the table and picking up a single block versus a tower of blocks, the cost of *pick-up* and *pick-up-from-table* actions is different for each problem. Table 1 shows the total number of blocks and the cost of both *pick-up* and *pick-up-from-table* actions for the 15 problems considered. In all the considered problems, the goal statement contains all the blocks. In the remainder of this article, we refer to the probabilistic blocks worlds as blocks world.

5.1.2. Zeno travel

The zeno travel domain is a logistic domain in which a given number of people need to be transported from their initial locations to their destinations using a fleet of airplanes. Moreover, the level of fuel of each airplane is also modeled, so there is a need to plan to refuel.

The available actions in this domain are boarding, debarking, refueling, flying (at regular speed), and zooming (flying at a faster speed). Each action has a random duration modeled by a geometrically distributed random variable with probability p; the expected duration of each action (i.e., the average number of time steps necessary to succeed) is $1/p$. To ensure the geometric duration of the available actions, they are represented by a two-step procedure, e.g., *start-boarding* and *finish-boarding*, in which the first step is always deterministic and the second step succeeds with probability p. The value of p is 1/2, 1/4, 1/7, 1/25, and 1/15 for boarding, debarking, refueling, flying, and zooming, respectively.

The cost of all actions is 1 except for flying and zooming, which have costs 10 and 25, respectively. Although the fuel requirement for flying and zooming is the same, their expected costs differ due to their different costs and success probabilities: 250 for flying and 375 for zooming.

[2] All problems from IPPC'08 are available at http://ippc-2008.loria.fr/wiki/index.php/Results.html.

Table 2
Number of cities, persons and airplanes for each of the 15 problems considered of the zeno travel domain.

Problem #	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Cities	4	5	5	6	6	7	7	8	9	10	11	13	14	15	20
Persons	2	2	5	2	5	10	5	5	10	5	10	5	10	10	10
Airplanes	2	2	3	2	3	6	3	3	6	3	6	3	6	6	6

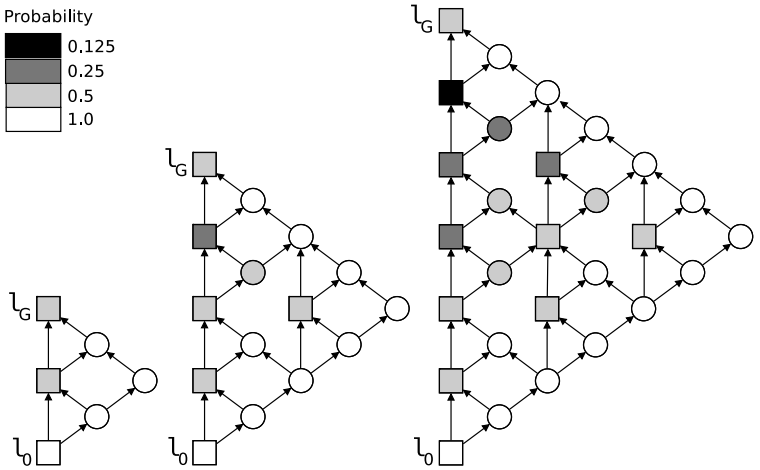

Fig. 4. Roadmap of the triangle tire world problems of size 1, 2 and 3. Circles (squares) represent locations in which there is one (no) spare tire available. In the initial state the car is at l_0 and the tire is not flat; the goal is to reach location l_G. The shades of gray represent, for each location l, $\max_\pi P$ (car reaches l and the tire is not flat when following the policy π from s_0).

As in the blocks world domain, Assumption 1 holds for all problems in the zeno travel domain. Table 2 shows the number of persons, cities, and airplanes for each of the 15 problems considered. In all the considered problems, the fuel level of each airplane is discretized into 5 categories: empty, 1/4, 1/2, 3/4, and full.

5.1.3. Triangle tire world

The triangle tire world [19] is a probabilistically interesting domain in which a car has to travel between locations to reach a goal location from its initial location. Every time the car moves between locations, a flat tire happens with probability 0.5. The car carries only one spare tire, which can be used at any time to fix a flat tire. Once the spare tire is used, a new one can be loaded into the car; however, only some locations have an available new tire to be loaded. The actions *load-tire* and *change-tire* are deterministic.

The roads between locations are one-way only, and the roadmap is represented as a directed graph in the shape of an equilateral triangle. Each problem in the triangle tire world is represented by a number $n \in \mathbb{N}^*$ corresponding to the roadmap size. Fig. 4 illustrates the roadmap for the problems 1, 2, and 3 of the triangle tire world. The initial and goal locations, l_0 and l_G, respectively, are in two different vertices of the roadmap, and their configuration is such that (i) the shortest path policy from l_0 and l_G has probability 0.5^{2n-1} of reaching the goal; and (ii) the only proper policy (and therefore the optimal policy) is the policy that takes the longest path. Assumption 1 does not hold for the triangle tire world problems: there exist states that do not have a proper policy starting from them (e.g., states in which the car is in the shortest path from l_0 to l_G). Since there exists one proper policy from s_0, then the triangle tire world problems contain avoidable dead ends.

In the IPPC'08, problems 11 to 15 are instances of a similar domain, the *rectangle tire world*. For our experiments, we only consider the triangle tire world, and the problem number corresponds to the parameter n of the triangle tire world problem.

5.1.4. Exploding blocks world

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, it is possible to reach dead ends in the exploding blocks world. Moreover, not all problems in the exploding blocks world domain have a proper policy, i.e., these problems might have unavoidable dead ends.

All actions available in the exploding blocks world, *pick-up*, *pick-up-from-table*, *put-down*, and *put-on-block*, have the same effects as their counterparts in the deterministic blocks world. *Pick-up* and *pick-up-from-table* have the extra precondition that the block being picked up is not destroyed. Actions *put-down* and *put-on-block* have the probabilistic side effect of

Table 3
Number of blocks and blocks in the goal statement for each of the 15 problems considered from the exploding blocks world.

Problem #	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Number of blocks	5		6		7	8	9	10	11	12	13	14	15	16	17
Blocks in the goal	2	3	3	4	5	6	7	8	9	10	11	12	13	14	15

detonating the block being held and destroying the table or the block below with probability 2/5 and 1/10, respectively. Once a block is detonated, it can be safely moved, i.e., a denoted block cannot destroy other blocks or the table.

The IPPC'08 encoding of the exploding blocks world has a flaw in which a block can be placed on top of itself [19]. This flaw allows planners to safely discard blocks not needed in the goal because after placing a block B on top of itself, (i) no block is being held (the planner is free to pick up another block), and (ii) only B might be destroyed, thus preserving the other blocks and the table. We consider the fixed version of the IPPC'08 exploding blocks world, in which the action *put-on-block* has the additional precondition that the destination block is not the same as the block being held; precisely, we added the precondition `(not (= ?b1 ?b2))` to `put-on-block(?b1 ?b2)`.

Table 3 shows the total number of blocks and blocks in the goal statement for the 15 problems considered from the exploding blocks world domain. In the considered problems, all actions have cost 1.

5.2. Convergence experiments

In the following experiments, we compare the time necessary for LRTDP [5], Focused Topological Value Iteration (FTVI) [10], SSiPP, and Labeled-SSiPP to compute ϵ-consistent solutions. We use Algorithm 2 to obtain ϵ-consistent solutions using SSiPP. SSiPP-FF is not considered because it has no convergence guarantee. For the experiments in Section 5.2.1, we use the domains from IPPC'08 (reviewed in Section 5.1), and in Section 5.2.2, we use the *racetrack* domain, a common domain to compare optimal probabilistic planners.

5.2.1. Problems from the international probabilistic planning competition

In this experiment, we compare the time to converge to an ϵ-consistent solution for the problems in the IPPC'08. Although Assumption 1 does not hold for the triangle tire world (Section 5.1.3), all problems in this domain are such that (i) they have avoidable dead ends, and (ii) these dead ends are states in which no action is available. Therefore, all the considered planners can trivially detect when a dead end s_d is reached, in which case $V(s_d)$ is updated to infinity, and the search is restarted. For this experiment, the value assigned to $V(s_d)$ is 10^5; this value is large enough because $V^*(s_0) < 12n$ for the triangle tire world problem of size n. Problems from the exploding blocks world domain are not considered because there is no guarantee they have a proper policy.

This experiment was conducted on a 2.4-GHz machine with 16 cores running a 64-bit version of Linux. The time and memory limit enforced for each planner was 2 hours and 5 GB, respectively. For SSiPP and Labeled-SSiPP, we considered $t \in \{2, 4, 8, 16, 32\}$. The admissible heuristic used by all the planners is the classical planning heuristic $h_{\max}$ applied to the all-outcomes determinization [27].

Table 4 presents the results of this experiment as the average and 95% confidence interval of the convergence time for 50 runs of each planner parametrization. From the 15 problems of each domain, we only present the results in which at least one planner converged to an ϵ-consistent solution. The problems 5′ to 8′ for blocks world are problems with 8 blocks obtained by removing blocks $b9$ and $b10$ from the original IPPC'08 problems 5 to 8. We generated these problems since no planner converged for problems 5 to 8 and problems 1 to 4 are too small (convergence is reached in about 1 second).

The performance difference between SSiPP and Labeled-SSiPP is not significant for small problems: blocks world 1 to 4, triangle tire world problems 1 and 2, and zeno travel problem 1 and 2. For the triangle tire world problems 3 and 4, $t = 32$ is large enough that an ϵ-consistent solution is found using a single short-sighted SSP, so the performance of SSiPP and Labeled-SSiPP for $t = 32$ is equivalent to the LRTDP performance. For the same problems, when $t < 32$, Labeled-SSiPP converges using between 6% to 32% of the convergence time of SSiPP for the value of t.

In the triangle tire world, the best parametrization of Labeled-SSiPP is not able to outperform LRTDP (the best planner in this domain) due to the overhead of building the short-sighted SSPs. This issue is specific to the triangle tire domain, since there is only one proper policy; therefore, a planner that prunes improper policies can efficiently focus its search on the single optimal policy of the triangle tire world problems. For instance, the $(s_0, 16)$-short-sighted SSP $\mathbb{S}_{s_0,16}$ associated with problem 4 of the triangle tire world contains 124 436 states, and $\mathbb{S}_{s_0,16}$ is generated and solved on every iteration of line 5 of Labeled-SSiPP (Algorithm 4), even after inferring that $\mathbb{S}_{s_0,16}$ also contains only one proper policy. This issue can be overcome by using *trajectory-based* short-sighted SSPs [29].

For the larger problems of the blocks world (5′ to 8′), Labeled-SSiPP obtains a major improvement over the considered planners and converged in at most 0.93, 0.80, and 0.26 of the time necessary for SSiPP, LRTDP, and FTVI to converge, respectively. Lastly, in the zeno travel domain, SSiPP and Labeled-SSiPP obtain a similar performance in the small problems (problems 1 and 2) and converge in at most 0.06 of the LRTDP convergence time. FTVI fails to converge in all the problems from the zeno travel domain, and Labeled SSiPP for $t = 32$ is the only planner able to converge for problem 4 of the zeno travel domain.

Table 4
Average and 95% confidence interval of the time, in seconds, to converge to an ϵ-consistent solution using $\epsilon = 10^{-4}$. If convergence is not reached, then '–' is shown. Best performance over all planners (column) is shown in bold font. $h_{\max}$ heuristic was used by all planners. Problems 5′ to 8′ of blocks world are the IPPC'08 problems 5 to 8 without blocks $b9$ and $b10$.

Problem		Blocks world							
		1	2	3	4	5′	6′	7′	8′
SSiPP	2	1.11 ± 0.0	1.23 ± 0.1	1.44 ± 0.1	1.15 ± 0.0	2689.2 ± 66.9	3018.8 ± 39.6	3278.5 ± 60.3	3758.2 ± 72.3
	4	0.78 ± 0.0	0.82 ± 0.0	1.01 ± 0.0	0.84 ± 0.0	2525.8 ± 54.8	2883.7 ± 49.7	3044.5 ± 46.1	3237.7 ± 50.4
	8	0.37 ± 0.0	0.53 ± 0.0	0.88 ± 0.1	0.46 ± 0.0	3243.3 ± 69.1	3550.1 ± 66.8	3586.9 ± 77.3	3597.1 ± 51.4
	16	0.22 ± 0.0	0.29 ± 0.0	0.29 ± 0.0	0.27 ± 0.0	581.6 ± 7.4	665.0 ± 8.3	701.6 ± 9.1	726.7 ± 11.1
	32	0.23 ± 0.0	**0.26 ± 0.0**	**0.28 ± 0.0**	0.27 ± 0.0	699.4 ± 11.1	840.8 ± 11.1	877.5 ± 14.2	968.5 ± 16.8
L-SSiPP	2	1.03 ± 0.0	1.15 ± 0.0	1.30 ± 0.0	1.12 ± 0.0	2385.9 ± 18.8	2776.7 ± 21.9	2973.7 ± 29.2	3275.1 ± 28.9
	4	0.82 ± 0.0	0.88 ± 0.0	1.00 ± 0.0	0.85 ± 0.0	2346.8 ± 23.1	2617.2 ± 20.8	2847.3 ± 27.2	2972.5 ± 28.2
	8	0.39 ± 0.0	0.54 ± 0.0	0.77 ± 0.0	0.41 ± 0.0	2893.6 ± 29.3	3380.7 ± 46.1	3051.6 ± 54.8	3332.1 ± 48.8
	16	0.26 ± 0.0	0.30 ± 0.0	0.28 ± 0.0	0.27 ± 0.0	**508.7 ± 4.2**	**590.6 ± 3.3**	**650.0 ± 5.9**	**672.3 ± 6.0**
	32	**0.22 ± 0.0**	0.27 ± 0.0	0.30 ± 0.0	**0.26 ± 0.0**	691.0 ± 5.4	808.7 ± 4.0	851.8 ± 7.7	970.9 ± 9.7
LRTDP		0.23 ± 0.0	0.28 ± 0.0	0.30 ± 0.0	0.26 ± 0.0	639.6 ± 4.9	758.7 ± 4.4	813.1 ± 6.5	915.0 ± 7.5
FTVI		0.71 ± 0.0	0.88 ± 0.0	1.02 ± 0.0	0.97 ± 0.0	–	2302.7 ± 21.0	2553.9 ± 32.0	3081.0 ± 34.1

Problem		Triangle tire world					Zeno travel		
		1	2	3	4	5	1	2	4
SSiPP	2	0.03 ± 0.0	0.82 ± 0.1	28.6 ± 5.3	705.7 ± 49.1	–	1325.8 ± 51.6	1367.3 ± 31.1	–
	4	0.03 ± 0.0	0.48 ± 0.1	23.5 ± 2.1	467.7 ± 35.6	–	559.8 ± 24.0	837.6 ± 27.5	–
	8	**0.02 ± 0.0**	0.56 ± 0.1	30.3 ± 5.0	601.6 ± 64.3	–	197.9 ± 8.7	303.3 ± 22.2	–
	16	0.02 ± 0.0	0.04 ± 0.0	64.3 ± 7.6	–	–	34.0 ± 1.2	45.7 ± 0.8	–
	32	0.02 ± 0.0	0.04 ± 0.0	0.51 ± 0.0	13.2 ± 0.4	–	217.7 ± 7.6	255.5 ± 5.2	–
L-SSiPP	2	0.03 ± 0.0	0.33 ± 0.0	6.77 ± 0.2	227.6 ± 7.8	–	1589.7 ± 45.9	1921.7 ± 41.6	–
	4	0.02 ± 0.0	0.15 ± 0.0	2.77 ± 0.1	114.3 ± 8.3	–	571.4 ± 24.0	826.0 ± 21.2	–
	8	0.02 ± 0.0	0.14 ± 0.0	2.08 ± 0.2	79.1 ± 8.8	–	210.8 ± 12.3	295.4 ± 17.8	–
	16	0.02 ± 0.0	0.05 ± 0.0	3.68 ± 0.1	297.2 ± 18.2	3780.0 ± 50.4	**33.5 ± 0.6**	**45.4 ± 1.6**	–
	32	0.02 ± 0.0	0.05 ± 0.0	0.50 ± 0.0	12.4 ± 0.1	–	210.6 ± 7.2	247.8 ± 4.4	**5370.28 ± 93.2**
LRTDP		0.02 ± 0.0	**0.04 ± 0.0**	**0.33 ± 0.0**	**8.45 ± 0.0**	**391.2 ± 4.4**	591.9 ± 15.1	1391.8 ± 19.6	–
FTVI		0.03 ± 0.0	0.11 ± 0.0	2.31 ± 0.1	69.7 ± 0.4	3014.7 ± 38.8	–	–	–

5.2.2. Racetrack problems

The goal of a problem in the racetrack domain [1,5] is to move a car from its initial location to one of the goal locations while minimizing the expected cost of travel. A state in the racetrack domain is the tuple (x, y, v_x, v_y, b) in which

- x and y are the positions of the car in the given 2-D grid (track);
- v_x and v_y are the velocities in each dimension; and
- b is a binary variable that is true if the car is broken.

At each time step, the position (x, y) of the car is updated by adding its current speed (v_x, v_y) to the respective dimension. Acceleration actions, represented by pairs $(a_x, a_y) \in \{-1, 0, 1\}^2$ and denoting the instantaneous acceleration in each direction, are available to control the car's velocity. The acceleration actions can fail with probability 0.1, in which case the car's velocity is not changed.

If the car attempts to leave the race track, then it is placed in the last valid position before exiting the track, its velocity in both directions is set to zero, and it is marked as broken, i.e., b is set to true. The special action *fix-car* is used to fix the car (i.e., set b to false). The cost of *fix-car* is 50 while the acceleration actions have cost 1.

We consider six racetracks in this experiment: ring-small, ring-large, square-small, square-large, y-small, and y-large. The shape of each track is depicted in Fig. 5, and Table 5 presents their corresponding $|\mathsf{S}|$, ratio of relevant states ($\mathsf{S}^{\pi^*}/\mathsf{S}$), largest value of t, $t_{\max}$, such that $\pi^*_{s_0,t_{\max}}$ is not closed for the original SSP, and $V^*(s_0)$.

The admissible heuristic used by all the planners is the min–min heuristic $h_{\min}$, and $h_{\min}(s)$ equals the cost of the optimal plan for reaching a goal state from s in the all-outcomes determinization. Therefore, $h_{\min}$ can be computed by the following fixed-point equations:

$$h_{\min}(s) = \begin{cases} 0 & \text{if } s \in \mathsf{G} \\ \min_{a \in \mathsf{A}} \min_{s' : P(s'|s,a) > 0} [C(s, a, s') + h_{\min}(s')] & \text{otherwise,} \end{cases} \quad \forall s \in \mathsf{S}.$$

This experiment was conducted on a 3.07-GHz machine with 4 cores running a 32-bit version of Linux. A time limit of 2 hours and 4 GB of memory were applied to each planner. For SSiPP and Labeled-SSiPP, we considered $t \in \{4, 8, 16, \ldots, 1024\}$. FTVI is not considered in this experiment because the implementation of FTVI we had access to

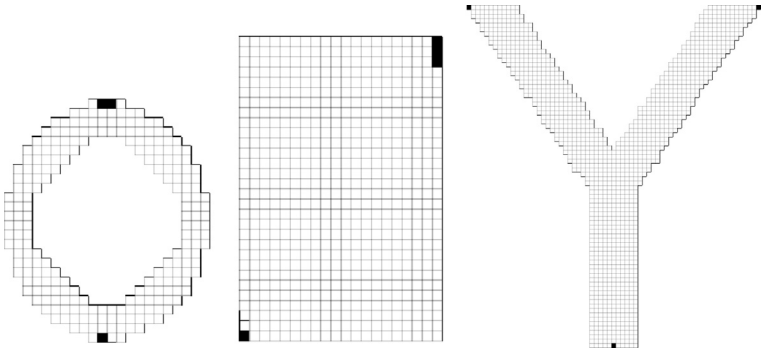

Fig. 5. Shape of the racetracks used. Each cell represents a possible position of the car. The initial position and the goal positions are, respectively, the marked cells in the bottom and top of each track.

Table 5
Description of each racetrack used: size, ratio S^{π^*}/S, t_{max}, $V^*(s_0)$, value of the min–min heuristic for s_0 ($h_{min}(s_0)$) and time in seconds to compute $h_{min}(s_0)$.

Problem	$\|S\|$	% rel.	t_{max}	$V^*(s_0)$	$h_{min}(s_0)$	Time $h_{min}(s_0)$
ring-s	4776	12.91	74	21.85	12.00	0.451
ring-l	75364	14.34	869	36.23	24.00	32.056
square-s	42396	2.01	71	18.26	11.00	14.209
square-l	193756	0.75	272	22.26	13.00	145.616
y-small	101481	10.57	114	29.01	18.00	32.367
y-large	300460	9.42	155	32.81	21.00	211.891

Table 6
Average and 95% confidence interval of the time, in seconds, to converge to an ϵ-consistent solution using $\epsilon = 10^{-4}$. If convergence is not reached, then '–' is shown. Best performance over all planners (column) is shown in bold font. The min-min heuristic was used by all planners.

	t	Ring-Small	Ring-Large	Square-Small	Square-Large	Y-Small	Y-Large
SSiPP	4	23.50 ± 8.53	2559.16 ± 849.09	27.44 ± 2.61	799.11 ± 32.47	1762.18 ± 74.32	4086.53 ± 235.65
	8	7.39 ± 2.73	745.52 ± 341.51	30.26 ± 3.71	844.18 ± 53.86	777.99 ± 58.12	3848.75 ± 227.40
	16	0.64 ± 0.02	612.99 ± 261.53	18.66 ± 2.22	811.27 ± 59.39	861.17 ± 94.57	3517.13 ± 215.14
	32	0.60 ± 0.02	64.10 ± 10.47	17.88 ± 1.99	693.18 ± 19.56	57.03 ± 1.29	2987.81 ± 207.47
	64	$\mathbf{0.59 \pm 0.02}$	62.86 ± 6.88	17.56 ± 0.55	642.28 ± 12.60	57.25 ± 1.84	320.75 ± 9.57
	128	0.61 ± 0.01	63.05 ± 7.15	17.44 ± 0.59	631.89 ± 31.89	55.46 ± 1.06	315.68 ± 9.29
	256	0.61 ± 0.01	64.25 ± 0.89	17.65 ± 0.66	639.51 ± 15.21	55.86 ± 1.49	319.14 ± 9.10
	512	0.61 ± 0.02	61.42 ± 2.09	18.35 ± 0.58	690.73 ± 12.25	55.79 ± 2.78	321.48 ± 9.90
	1024	0.61 ± 0.01	58.39 ± 2.40	18.08 ± 0.49	698.33 ± 16.90	55.78 ± 1.98	320.68 ± 8.22
Labeled SSiPP	4	1.85 ± 0.08	363.65 ± 11.57	24.63 ± 1.02	763.56 ± 47.25	425.42 ± 67.67	2810.27 ± 77.66
	8	1.89 ± 0.13	463.31 ± 38.67	25.87 ± 2.29	810.44 ± 94.35	368.43 ± 74.90	2858.96 ± 68.47
	16	0.65 ± 0.03	429.95 ± 28.96	18.66 ± 0.50	737.46 ± 96.43	302.97 ± 35.40	2700.81 ± 69.44
	32	0.60 ± 0.02	60.89 ± 3.67	17.84 ± 0.36	654.30 ± 40.81	56.65 ± 2.22	319.30 ± 9.74
	64	0.61 ± 0.01	60.08 ± 3.12	$\mathbf{16.15 \pm 0.25}$	631.78 ± 39.42	$\mathbf{51.61 \pm 2.68}$	311.44 ± 8.26
	128	0.61 ± 0.02	59.89 ± 3.14	16.72 ± 0.35	$\mathbf{612.78 \pm 30.44}$	55.60 ± 2.17	$\mathbf{307.45 \pm 5.66}$
	256	0.61 ± 0.02	58.05 ± 3.23	17.97 ± 0.72	623.85 ± 12.58	56.58 ± 2.34	316.56 ± 7.55
	512	0.61 ± 0.01	57.20 ± 3.49	18.65 ± 0.50	703.21 ± 10.46	55.99 ± 2.50	319.94 ± 7.29
	1024	0.61 ± 0.01	58.74 ± 3.88	18.98 ± 0.53	701.63 ± 11.95	55.66 ± 1.93	319.71 ± 8.62
LRTDP		$\mathbf{0.59 \pm 0.02}$	$\mathbf{55.81 \pm 2.92}$	18.60 ± 0.84	702.42 ± 12.82	54.00 ± 2.20	319.08 ± 8.35

is not compatible with the encoding of the racetrack problems. Table 6 presents the results as the average and 95% confidence interval for 10 runs of each planner parametrization.

The performance of SSiPP, Labeled-SSiPP, and LRTDP is similar for $t > t_{max}$ in all problems because LRTDP is used as Optimal-SSP-Solver, and t_{max} is such that $\mathbb{S}_{s_0,t}$ contains all the states necessary to find the optimal solution. The performance improvement of Labeled-SSiPP over SSiPP is more evident for smaller values of t, and as t approaches t_{max}, it decreases until both Labeled-SSiPP and SSiPP converge to the LRTDP performance.

For the square and y tracks, the best performance is obtained by Labeled-SSiPP for t, either 64 (small tracks) or 128 (large tracks), both values smaller than t_{max} for their respective problems. While this improvement obtained by Labeled-SSiPP is in the intersection of the 95% confidence interval for the y tracks, it is statistically significant for the square tracks, especially for the large instance: 612.78 ± 30.44 (Labeled-SSiPP) versus 702.42 ± 12.82 (LRTDP). This difference in performance is because an optimal policy in the square-large track reaches only 0.75% of the state space (Table 5). Therefore, both SSiPP

and Labeled-SSiPP take advantage of the short-sighted search to prune useless states earlier in the search, resulting in a better performance than LRTDP for $t \in \{32, 64, 128, 256\}$.

5.3. International probabilistic planning competition

In this section, we compare the performance of the following planners to obtain (suboptimal) solutions under a 20-minute time limit:

- FF-Replan [30] (winner of IPPC'04).
- Robust-FF [26] (winner of IPPC'08).
- HMDPP [16].
- ReTrASE [18].
- SSiPP.
- Labeled-SSiPP.
- SSiPP-FF.

For this experiment, we use the 15 problems from IPPC'08 of each domain as described in Section 5.1. We present the methodology used in this experiment in Section 5.3.1 and how to choose the value of t and heuristic for SSiPP, Labeled-SSiPP, and SSiPP-FF in Section 5.3.2. Section 5.3.3 presents the results of this experiment.

5.3.1. Methodology

We use a methodology similar to that used for the IPPCs, in which there is a time limit for each individual problem: a planner has 20 minutes to compute a policy and simulate the computed policy 50 times from the initial state s_0. A round is each simulation from s_0 of the same problem, and rounds are simulated in a client/server approach using MDPSIM [33], an SSP (and MDP) simulator. Planners send actions to be simulated to MDPSIM, which internally simulates the received actions and returns the resulting state. Every round terminates when either (i) the goal is reached; (ii) an invalid action (e.g., not applicable in the current state) is sent to MDPSIM; (iii) 2000 actions have been submitted to MDPSIM; or (iv) the planner explicitly gives up from the round (e.g., because the planner is trapped in a dead end). A round is considered successful if the goal is reached; otherwise, it is declared to be a failed round. Notice that planners are allowed to change their policies at any time, i.e., during a round or in between rounds. Therefore, the knowledge obtained from one round (e.g., the lower bound on $V^*(s_0)$) can be used to solve subsequent rounds.

A run is the sequence of rounds simulated by a planner for a given problem, and the previous IPPCs evaluate planners based on a single run per problem. Due to the stochastic nature of SSPs, the outcome of a single run depends on the random seed used in the initialization of both the planner and MDPSIM. To evaluate planners more accurately, we execute 50 runs for each problem and planner. Therefore, in this section, the performance of a planner in a given problem is estimated by 2500 rounds generated by 50 potentially different policies computed by the same planner. Our approach (50 runs of 50 rounds each) is not equivalent to the execution of one run of 2500 rounds because a planner might be guided toward *bad* decisions due to the outcomes of the probabilistic actions and not have enough time to revise such decisions. Alternatively, by simulating several runs, there is a small probability that this misguidance will happen in all runs.

To respect the 20-minute time limit, SSiPP, Labeled-SSiPP, and SSiPP-FF *solve rounds internally* for 15 minutes and then start solving rounds through MDPSIM. For SSiPP and SSiPP-FF, a round is solved internally by calling Algorithms 1 and 5, respectively, and the obtained lower bound $\underline{V}$ in round r is used as the heuristic for round $r+1$. The same effect is obtained for Labeled-SSiPP by adding a 15-minute time limit in line 5 of Algorithm 4.

The IPPCs also enforce that planners should not have free parameters, i.e., the only input for each planner is the problem to be solved. Therefore, all parameters of a planner, such as the value of t and heuristic for SSiPP, must be fixed a priori or automatically derived. Because of this rule, all the non-SSiPP planners considered do not have parameters. In the IPPC'08, two different parametrizations were fixed for Robust-FF, but we consider only the RFF-PG parametrization as it obtained the best performance in IPPC'08 for the considered problems [8]. Section 5.3.2 describes the two different methods we employed to obtain the parametrizations for SSiPP, Labeled-SSiPP, and SSiPP-FF.

5.3.2. Choosing the value of t and heuristic for SSiPP-based planners

To choose a fixed parametrization for SSiPP, Labeled-SSiPP, and SSiPP-FF, i.e., a value of t and a heuristic, we perform a *round-robin tournament* between different parametrizations of each planner. The round-robin tournament consists of comparing the performance of different parametrizations of a planner in the 15 final problems from IPPC'06 for blocks world, zeno travel, and exploding blocks world. No problem from the triangle tire world is used for training as they are deterministically generated, i.e., any triangle tire world problem of size 1 to 15 would be exactly the same as the problems in the main experiment. We refer to these 45 problems as the set of training problems J.

Formally, given a planner X and a set of parametrizations $\mathsf{K} = \{k_1, \ldots, k_n\}$ for X, we solve all problems in J using the same methodology as described in Section 5.3.1. We denote as $c(k_i, p)$ the number of rounds of the problem $p \in \mathsf{J}$

Table 7
Number of problems per domain in which a given planner has the best coverage. ReTrASE does not support the zeno travel problems (*n.a.*).

	Blocks world	Zeno travel	Triangle tire W.	Exploding blocks W.
FF-Replan	13	15	0	1
Robust-FF	8	0	4	1
HMDPP	4	2	13	1
ReTrASE	8	*n.a.*	4	2
SSiPP$_t$	4	0	1	2
SSiPP$_r$	4	2	2	9
L-SSiPP$_t$	5	2	2	2
L-SSiPP$_r$	5	2	2	3
SSiPP-FF$_t$	8	11	0	2
SSiPP-FF$_r$	8	13	0	7

in which X, using parametrization $k_i \in \mathsf{K}$, reached the goal. The function $m(k_i, k_j)$ represents the comparison (or match) between k_i and k_j, and $m(k_i, k_j)$ equals 1 if

$$\left|\left\{p \in \mathsf{P} \;\middle|\; c(k_i, p) > c(k_j, p)\right\}\right| > \left|\left\{p \in \mathsf{P} \;\middle|\; c(k_i, p) < c(k_j, p)\right\}\right|$$

(i.e., if k_i outperforms k_j in most problems) and 0 otherwise. The tournament winner is the parametrization k that outperforms the majority of other parametrizations in $\mathsf{K} : k = \operatorname{argmax}_{k_i \in \mathsf{K}} \sum_{k_j \neq k_i} m(k_i, k_j)$.

For SSiPP and Labeled-SSiPP, the set of considered parametrizations K is the cross product of $\mathsf{T} = \{2, 3, 4, \ldots, 10\}$ and the following set H of heuristics:

- zero-heuristic: $h_0(s) = 0$ for all $s \in \mathsf{S}$,
- FF-heuristic: $h_{\text{ff}}(s)$ equals the cost of the plan returned by the deterministic planner FF [13] to reach a goal state from the current state s in the all-outcomes determinization,
- $h_{\max}$ and h_{add} applied to the all-outcomes determinization of the original problem [27].

For SSiPP-FF, the determinization type is also a parameter, and its set of considered parametrizations K equals $\mathsf{T} \times \mathsf{H} \times \{\text{most-likely outcome, all-outcomes}\}$. The parametrization that won the round-robin tournament for each SSiPP-based planner in their respective set of considered parameters K is $t = 3$ and h_{add} for SSiPP; $t = 6$ and h_{add} for Labeled-SSiPP; and $t = 3$, h_{add} and the all-outcomes determinization for SSiPP-FF. We refer to these parametrizations as SSiPP$_t$, Labeled-SSiPP$_t$, and SSiPP-FF$_t$.

We also consider an approach in which the value of t is randomly selected for SSiPP, Labeled-SSiPP, and SSiPP-FF. Formally, before calling GENERATE-SHORT-SIGHTED-SSP in Algorithms 1, 4, and 5, we select t at random from $\{2, 3, 4, \ldots, 10\}$. Therefore, different values of t might be used for solving a given problem. For this approach, we use h_{add} as a heuristic for all the SSiPP-based planners and the all-outcomes determinization for SSiPP-FF. And to avoid generating large short-sighted SSPs, we stop GENERATE-SHORT-SIGHTED-SSP after 15 seconds or if $|\mathsf{S}_{s,t}| > 10^5$. When GENERATE-SHORT-SIGHTED-SSP is interrupted, the states that could not be explored are marked as artificial goals. We refer to these parametrizations as SSiPP$_r$, Labeled-SSiPP$_r$, and SSiPP-FF$_r$.

5.3.3. Results

The experiment in this section was conducted on a 2.4-GHz machine with 16 cores running a 64-bit version of Linux. Table 7 presents the summary of the results as the number of problems in which a given planner has the best coverage, i.e., it solved more rounds than the other planners. The detailed results are presented in Tables 8 and 9 as the coverage and 95% confidence interval obtained by each planner in each problem, and in Tables 10 and 11 as the average and 95% confidence interval for the obtained cost over the successful rounds for each problem.

SSiPP-FF$_t$ and SSiPP-FF$_r$ successfully take advantage of determinizations and improve the coverage obtained by SSiPP and Labeled-SSiPP in the domains without dead ends, i.e., blocks world and zeno travel. In particular, both parametrizations of SSiPP-FF, together with FF-Replan, are the only planners able to solve the medium and large problems of the zeno travel domain. SSiPP-FF also improves the performance of FF-Replan for problems with dead ends. In the triangle tire world, a problem designed to penalize determinization approaches, FF-Replan, SSiPP-FF$_t$, and SSiPP-FF$_r$ solve instances up to numbers 5, 7, and 9, respectively; moreover, the coverage of SSiPP-FF$_r$ is more than the double of the coverage of FF-Replan for problems 1 to 5. In the exploding blocks world, the combination of SSiPP and determinizations is especially useful for large instances: SSiPP-FF$_r$ is the planner with the best coverage for the 5 largest problems in this domain. The solution quality of FF-Replan is also improved by SSiPP-FF. For instance, in zeno travel problems in which the SSiPP-FF obtained coverage 1, the solutions found by SSiPP-FF$_r$ and SSiPP-FF$_t$ have an average cost between 0.80 and 0.92 of the FF-Replan solution's average cost.

Labeled-SSiPP performs well on small problems, obtaining good coverage and solutions with a low average cost; however it fails to scale up to large problems. Possible reasons for not scaling up include (i) the bias for exploration over exploitation employed by Labeled-SSiPP to speed up the convergence and (ii) the overhead added by the procedure CHECKSOLVED.

Table 8
Rounded coverage and 95% confidence internal for the Blocks World and Zeno World in the experiment of Section 5.3. Best coverage for each problem (row) is shown in bold. If no round is solved, i.e., zero coverage, then '–' is shown. ReTrASE does not support the zeno travel problems (*n.a.*).

Problem		FFReplan	RFF	HMDPP	ReTrASE	SSiPP$_t$	SSiPP$_r$	L-SSiPP$_t$	L-SSiPP$_r$	SSiPP-FF$_t$	SSiPP-FF$_r$
Blocks world	1	**1.000 ± 0.000**	**1.000 ± 0.000**	**1.000 ± 0.000**	**1.000 ± 0.000**	**1.000 ± 0.000**	**1.000 ± 0.000**	**1.000 ± 0.000**	**1.000 ± 0.000**	**1.000 ± 0.000**	**1.000 ± 0.000**
	2	**1.000 ± 0.000**	**1.000 ± 0.000**	**1.000 ± 0.000**	**1.000 ± 0.000**	**1.000 ± 0.000**	**1.000 ± 0.000**	**1.000 ± 0.000**	**1.000 ± 0.000**	**1.000 ± 0.000**	**1.000 ± 0.000**
	3	**1.000 ± 0.000**	**1.000 ± 0.000**	**1.000 ± 0.000**	**1.000 ± 0.000**	**1.000 ± 0.000**	**1.000 ± 0.000**	**1.000 ± 0.000**	**1.000 ± 0.000**	**1.000 ± 0.000**	**1.000 ± 0.000**
	4	**1.000 ± 0.000**	**1.000 ± 0.000**	**1.000 ± 0.000**	**1.000 ± 0.000**	**1.000 ± 0.000**	**1.000 ± 0.000**	**1.000 ± 0.000**	**1.000 ± 0.000**	**1.000 ± 0.000**	**1.000 ± 0.000**
	5	**1.000 ± 0.000**	0.992 ± 0.003	–	**1.000 ± 0.000**	0.957 ± 0.007	0.649 ± 0.021	**1.000 ± 0.000**	**1.000 ± 0.000**	**1.000 ± 0.000**	**1.000 ± 0.000**
	6	**1.000 ± 0.000**	**1.000 ± 0.000**	–	**1.000 ± 0.000**	0.964 ± 0.007	0.286 ± 0.020	0.342 ± 0.016	0.295 ± 0.029	**1.000 ± 0.000**	**1.000 ± 0.000**
	7	**1.000 ± 0.000**	0.984 ± 0.004	–	**1.000 ± 0.000**	0.059 ± 0.009	0.074 ± 0.018	0.117 ± 0.011	0.099 ± 0.020	**1.000 ± 0.000**	**1.000 ± 0.000**
	8	**1.000 ± 0.000**	**1.000 ± 0.000**	–	**1.000 ± 0.000**	0.132 ± 0.012	0.204 ± 0.025	0.173 ± 0.013	0.127 ± 0.022	**1.000 ± 0.000**	**1.000 ± 0.000**
	9	**0.999 ± 0.002**	0.987 ± 0.004	–	0.870 ± 0.013	0.443 ± 0.018	0.186 ± 0.025	0.061 ± 0.008	0.040 ± 0.013	0.761 ± 0.015	0.507 ± 0.020
	10	0.999 ± 0.001	**1.000 ± 0.000**	–	0.883 ± 0.013	0.002 ± 0.001	–	0.003 ± 0.002	0.001 ± 0.001	0.761 ± 0.015	0.533 ± 0.021
	11	**0.998 ± 0.002**	0.995 ± 0.002	–	0.881 ± 0.013	–	–	–	–	0.764 ± 0.014	0.475 ± 0.016
	12	0.998 ± 0.002	**1.000 ± 0.000**	–	0.925 ± 0.010	0.003 ± 0.002	0.006 ± 0.005	–	–	0.821 ± 0.013	0.519 ± 0.021
	13	**0.847 ± 0.016**	–	–	–	–	–	–	–	0.067 ± 0.009	0.008 ± 0.006
	14	**0.867 ± 0.015**	–	–	–	–	–	–	–	0.089 ± 0.010	0.010 ± 0.006
	15	**0.886 ± 0.014**	–	–	–	–	–	–	–	0.062 ± 0.008	0.007 ± 0.005
Zeno travel	1	**1.000 ± 0.000**	0.175 ± 0.013	**1.000 ± 0.000**	*n.a.*	0.017 ± 0.005	**1.000 ± 0.000**	**1.000 ± 0.000**	**1.000 ± 0.000**	**1.000 ± 0.000**	**1.000 ± 0.000**
	2	**1.000 ± 0.000**	0.081 ± 0.010	**1.000 ± 0.000**	*n.a.*	0.100 ± 0.011	**1.000 ± 0.000**	**1.000 ± 0.000**	**1.000 ± 0.000**	**1.000 ± 0.000**	**1.000 ± 0.000**
	3	**1.000 ± 0.000**	–	–	*n.a.*	–	–	–	–	**1.000 ± 0.000**	**1.000 ± 0.000**
	4	**1.000 ± 0.000**	–	–	*n.a.*	–	–	–	–	**1.000 ± 0.000**	**1.000 ± 0.000**
	5	**1.000 ± 0.000**	–	–	*n.a.*	–	–	–	–	**1.000 ± 0.000**	**1.000 ± 0.000**
	6	**1.000 ± 0.000**	–	–	*n.a.*	–	–	–	–	**1.000 ± 0.000**	**1.000 ± 0.000**
	7	**1.000 ± 0.000**	–	–	*n.a.*	–	–	–	–	**1.000 ± 0.000**	**1.000 ± 0.000**
	8	**1.000 ± 0.000**	–	–	*n.a.*	–	–	–	–	**1.000 ± 0.000**	**1.000 ± 0.000**
	9	**1.000 ± 0.000**	–	–	*n.a.*	–	–	–	–	**1.000 ± 0.000**	**1.000 ± 0.000**
	10	**1.000 ± 0.000**	–	–	*n.a.*	–	–	–	–	**1.000 ± 0.000**	**1.000 ± 0.000**
	11	**1.000 ± 0.000**	–	–	*n.a.*	–	–	–	–	0.261 ± 0.015	0.469 ± 0.028
	12	**1.000 ± 0.000**	–	–	*n.a.*	–	–	–	–	**1.000 ± 0.000**	**1.000 ± 0.000**
	13	**1.000 ± 0.000**	–	–	*n.a.*	–	–	–	–	0.443 ± 0.018	**1.000 ± 0.000**
	14	**1.000 ± 0.000**	–	–	*n.a.*	–	–	–	–	0.269 ± 0.015	**1.000 ± 0.000**
	15	**1.000 ± 0.000**	–	–	*n.a.*	–	–	–	–	–	0.740 ± 0.028

Table 9
Rounded coverage and 95% confidence internal for the Triangle Tire World and Exploding Blocks World in the experiment of Section 5.3. Best coverage for each problem (row) is shown in bold. If no round is solved, i.e., zero coverage, then '–' is shown.

Problem		FFReplan	RFF	HMDPP	ReTrASE	SSiPP$_t$	SSiPP$_r$	L-SSiPP$_t$	L-SSiPP$_r$	SSiPP-FF$_t$	SSiPP-FF$_r$
Triangle tire world	1	0.480 ± 0.019	$\mathbf{1.000 \pm 0.000}$	$\mathbf{1.000 \pm 0.000}$	$\mathbf{1.000 \pm 0.000}$	$\mathbf{1.000 \pm 0.000}$	$\mathbf{1.000 \pm 0.000}$	$\mathbf{1.000 \pm 0.000}$	$\mathbf{1.000 \pm 0.000}$	0.747 ± 0.015	0.969 ± 0.011
	2	0.122 ± 0.012	$\mathbf{1.000 \pm 0.000}$	$\mathbf{1.000 \pm 0.000}$	$\mathbf{1.000 \pm 0.000}$	0.857 ± 0.012	$\mathbf{1.000 \pm 0.000}$	$\mathbf{1.000 \pm 0.000}$	$\mathbf{1.000 \pm 0.000}$	0.120 ± 0.012	0.774 ± 0.027
	3	0.036 ± 0.007	$\mathbf{1.000 \pm 0.000}$	$\mathbf{1.000 \pm 0.000}$	0.975 ± 0.006	0.564 ± 0.018	0.653 ± 0.032	0.815 ± 0.014	0.817 ± 0.025	0.036 ± 0.006	0.322 ± 0.030
	4	0.010 ± 0.004	$\mathbf{1.000 \pm 0.000}$	$\mathbf{1.000 \pm 0.000}$	0.964 ± 0.007	0.280 ± 0.016	0.287 ± 0.031	0.661 ± 0.016	0.659 ± 0.030	0.011 ± 0.004	0.141 ± 0.022
	5	0.001 ± 0.001	0.936 ± 0.009	$\mathbf{1.000 \pm 0.000}$	0.895 ± 0.012	0.159 ± 0.013	0.114 ± 0.022	0.479 ± 0.017	0.525 ± 0.034	0.001 ± 0.001	0.063 ± 0.015
	6	–	0.857 ± 0.015	$\mathbf{1.000 \pm 0.000}$	0.901 ± 0.012	0.127 ± 0.012	0.084 ± 0.017	0.146 ± 0.013	0.101 ± 0.020	0.001 ± 0.001	0.019 ± 0.009
	7	–	0.319 ± 0.016	$\mathbf{1.000 \pm 0.000}$	0.866 ± 0.013	0.116 ± 0.011	0.073 ± 0.017	0.033 ± 0.006	0.029 ± 0.011	0.001 ± 0.001	0.001 ± 0.001
	8	–	0.129 ± 0.012	$\mathbf{1.000 \pm 0.000}$	0.880 ± 0.013	0.062 ± 0.009	0.046 ± 0.015	0.021 ± 0.005	0.022 ± 0.010	–	0.002 ± 0.001
	9	–	0.058 ± 0.008	$\mathbf{1.000 \pm 0.000}$	0.800 ± 0.016	0.023 ± 0.005	0.025 ± 0.011	0.012 ± 0.004	0.014 ± 0.008	–	0.001 ± 0.001
	10	–	0.054 ± 0.008	$\mathbf{1.000 \pm 0.000}$	0.731 ± 0.017	0.011 ± 0.004	0.013 ± 0.007	0.005 ± 0.002	0.004 ± 0.002	–	–
	11	–	0.015 ± 0.004	$\mathbf{1.000 \pm 0.000}$	0.775 ± 0.016	0.006 ± 0.003	0.003 ± 0.002	0.001 ± 0.001	–	–	–
	12	–	0.003 ± 0.002	$\mathbf{1.000 \pm 0.000}$	0.510 ± 0.020	0.003 ± 0.001	0.004 ± 0.002	–	–	–	–
	13	–	0.010 ± 0.004	$\mathbf{0.663 \pm 0.019}$	0.348 ± 0.019	0.001 ± 0.001	0.001 ± 0.001	–	–	–	–
	14	–	0.004 ± 0.002	–	$\mathbf{0.367 \pm 0.019}$	0.001 ± 0.001	0.001 ± 0.001	–	–	–	–
	15	–	0.009 ± 0.003	–	$\mathbf{0.260 \pm 0.017}$	–	–	–	–	–	–
Exploding blocks world	1	0.358 ± 0.018	0.580 ± 0.017	0.599 ± 0.019	0.904 ± 0.012	0.907 ± 0.010	0.893 ± 0.019	0.901 ± 0.011	$\mathbf{0.909 \pm 0.019}$	0.896 ± 0.011	0.891 ± 0.020
	2	0.218 ± 0.016	0.217 ± 0.014	0.358 ± 0.019	0.359 ± 0.019	0.378 ± 0.017	$\mathbf{0.383 \pm 0.032}$	0.351 ± 0.017	0.376 ± 0.032	0.220 ± 0.014	0.283 ± 0.029
	3	0.359 ± 0.018	0.363 ± 0.017	0.365 ± 0.019	0.388 ± 0.019	–	$\mathbf{0.467 \pm 0.032}$	0.401 ± 0.016	0.412 ± 0.032	0.347 ± 0.018	0.346 ± 0.031
	4	0.534 ± 0.019	0.533 ± 0.018	0.363 ± 0.019	0.402 ± 0.019	0.534 ± 0.018	$\mathbf{0.562 \pm 0.035}$	0.484 ± 0.017	0.481 ± 0.034	0.341 ± 0.017	0.328 ± 0.031
	5	$\mathbf{1.000 \pm 0.000}$	$\mathbf{1.000 \pm 0.000}$	$\mathbf{1.000 \pm 0.000}$	$\mathbf{1.000 \pm 0.000}$	$\mathbf{1.000 \pm 0.000}$	$\mathbf{1.000 \pm 0.000}$	$\mathbf{1.000 \pm 0.000}$	$\mathbf{1.000 \pm 0.000}$	$\mathbf{1.000 \pm 0.000}$	$\mathbf{1.000 \pm 0.000}$
	6	0.898 ± 0.011	0.904 ± 0.010	0.173 ± 0.015	0.532 ± 0.020	0.920 ± 0.010	$\mathbf{0.940 \pm 0.016}$	0.918 ± 0.010	0.911 ± 0.019	0.898 ± 0.010	0.926 ± 0.017
	7	0.996 ± 0.002	0.608 ± 0.017	–	$\mathbf{1.000 \pm 0.000}$	$\mathbf{1.000 \pm 0.000}$	$\mathbf{1.000 \pm 0.000}$	$\mathbf{1.000 \pm 0.000}$	$\mathbf{1.000 \pm 0.000}$	$\mathbf{1.000 \pm 0.000}$	$\mathbf{1.000 \pm 0.000}$
	8	0.131 ± 0.013	0.133 ± 0.012	0.001 ± 0.001	0.223 ± 0.016	0.453 ± 0.018	$\mathbf{0.455 \pm 0.035}$	0.412 ± 0.017	0.373 ± 0.025	0.199 ± 0.013	0.195 ± 0.013
	9	0.073 ± 0.010	0.101 ± 0.011	–	0.194 ± 0.015	0.797 ± 0.014	$\mathbf{0.827 \pm 0.023}$	0.795 ± 0.014	0.816 ± 0.026	0.129 ± 0.012	0.159 ± 0.023
	10	0.008 ± 0.003	0.007 ± 0.003	–	0.039 ± 0.008	0.116 ± 0.021	$\mathbf{0.189 \pm 0.014}$	0.012 ± 0.004	–	0.012 ± 0.004	0.020 ± 0.009
	11	0.079 ± 0.010	0.059 ± 0.008	–	0.018 ± 0.005	–	–	–	–	0.044 ± 0.007	$\mathbf{0.086 \pm 0.017}$
	12	0.008 ± 0.003	0.014 ± 0.004	–	–	–	–	–	–	0.008 ± 0.003	$\mathbf{0.021 \pm 0.006}$
	13	0.110 ± 0.012	0.121 ± 0.011	–	0.059 ± 0.009	0.182 ± 0.014	0.079 ± 0.017	0.007 ± 0.003	0.006 ± 0.005	0.038 ± 0.007	$\mathbf{0.205 \pm 0.012}$
	14	0.026 ± 0.006	0.026 ± 0.006	–	–	–	–	–	–	0.029 ± 0.006	$\mathbf{0.047 \pm 0.014}$
	15	0.129 ± 0.013	0.058 ± 0.008	–	0.007 ± 0.003	–	–	–	–	0.076 ± 0.009	$\mathbf{0.201 \pm 0.025}$

Table 10
Rounded average and 95% confidence interval for the obtained cost over successful rounds for the Blocks World and Zeno World in the experiment of Section 5.3. If no round is solved, then '–' is shown; if exactly one round is solved, then *inf* is shown in the 95% confidence interval. ReTrASE does not support the zeno travel problems (*n.a.*).

Problem		FF-Replan	RFF	HMDPP	ReTrASE	SSiPP$_t$	SSiPP$_r$	L-SSiPP$_t$	L-SSiPP$_r$	SSiPP-FF$_t$	SSiPP-FF$_r$
Blocks world	1	21.5 ± 0.2	21.8 ± 0.1	17.0 ± 0.1	17.0 ± 0.1	17.0 ± 0.1	17.0 ± 0.2	16.9 ± 0.1	16.8 ± 0.2	31.8 ± 4.0	17.1 ± 0.2
	2	11.6 ± 0.1	11.8 ± 0.1	9.1 ± 0.1	9.1 ± 0.1	9.1 ± 0.1	9.0 ± 0.2	9.1 ± 0.1	9.3 ± 0.2	21.6 ± 1.5	9.2 ± 0.1
	3	18.7 ± 0.2	20.4 ± 0.2	15.6 ± 0.2	16.4 ± 0.2	15.6 ± 0.2	15.6 ± 0.3	15.6 ± 0.2	15.8 ± 0.3	30.6 ± 2.6	16.0 ± 0.3
	4	16.1 ± 0.2	15.5 ± 0.1	11.1 ± 0.1	11.1 ± 0.1	11.1 ± 0.1	11.1 ± 0.1	11.1 ± 0.1	11.1 ± 0.2	33.9 ± 1.9	11.2 ± 0.1
	5	87.3 ± 0.9	48.5 ± 0.2	–	50.8 ± 0.6	35.5 ± 0.3	38.4 ± 0.9	35.9 ± 0.3	36.8 ± 0.6	81.1 ± 2.8	51.1 ± 1.7
	6	29.1 ± 0.2	27.0 ± 0.1	–	19.2 ± 0.1	18.5 ± 0.1	23.1 ± 0.7	25.2 ± 0.5	24.8 ± 0.9	27.6 ± 0.5	26.6 ± 0.7
	7	45.0 ± 0.4	40.4 ± 0.3	–	31.9 ± 0.3	38.5 ± 1.8	56.5 ± 7.6	46.6 ± 2.7	48.6 ± 4.1	51.2 ± 1.3	48.1 ± 1.7
	8	72.9 ± 0.6	68.5 ± 0.3	–	45.8 ± 0.3	57.4 ± 1.5	61.0 ± 2.6	64.2 ± 2.7	71.0 ± 6.4	62.8 ± 1.1	71.3 ± 2.2
	9	327.3 ± 36.3	38.4 ± 0.2	–	53.7 ± 1.1	46.4 ± 0.5	44.6 ± 1.1	44.2 ± 1.3	52.2 ± 3.5	256.1 ± 13.3	165.0 ± 29.8
	10	85.1 ± 7.5	21.4 ± 0.1	–	23.3 ± 0.2	21.8 ± 1.9	–	23.1 ± 2.3	25.0 ± *inf*	76.6 ± 2.7	48.8 ± 6.2
	11	147.0 ± 14.1	31.9 ± 0.3	–	37.1 ± 0.4	–	–	–	–	141.5 ± 4.6	102.9 ± 16.8
	12	235.7 ± 21.2	53.8 ± 0.3	–	56.2 ± 0.6	59.5 ± 13.7	78.0 ± 7.6	–	–	153.6 ± 4.9	125.2 ± 21.8
	13	1744.7 ± 92.0	–	–	–	–	–	–	–	423.3 ± 46.4	164.2 ± 22.6
	14	435.0 ± 21.7	–	–	–	–	–	–	–	104.1 ± 7.6	60.5 ± 11.4
	15	1125.4 ± 55.5	–	–	–	–	–	–	–	288.8 ± 24.5	147.9 ± 27.7
Zeno travel	1	792.1 ± 22.8	4642.6 ± 311.8	505.4 ± 13.7	*n.a.*	498.6 ± 96.5	521.0 ± 30.3	503.4 ± 11.6	507.1 ± 24.2	642.2 ± 15.8	700.4 ± 30.7
	2	865.2 ± 21.4	3556.9 ± 187.2	490.2 ± 13.2	*n.a.*	496.1 ± 41.6	614.5 ± 39.2	494.1 ± 11.3	496.2 ± 23.3	748.9 ± 17.1	751.9 ± 30.5
	3	1615.1 ± 32.6	–	–	*n.a.*	–	–	–	–	1427.0 ± 24.5	1490.2 ± 47.7
	4	1072.3 ± 24.8	–	–	*n.a.*	–	–	–	–	976.7 ± 19.1	991.1 ± 36.6
	5	1796.5 ± 31.7	–	–	*n.a.*	–	–	–	–	1446.6 ± 25.1	1438.0 ± 44.9
	6	3449.8 ± 48.5	–	–	*n.a.*	–	–	–	–	2946.7 ± 34.4	2984.2 ± 60.7
	7	2132.0 ± 35.7	–	–	*n.a.*	–	–	–	–	1822.6 ± 26.0	1956.3 ± 50.3
	8	1973.0 ± 35.1	–	–	*n.a.*	–	–	–	–	1797.6 ± 27.2	1609.4 ± 50.1
	9	3282.0 ± 48.7	–	–	*n.a.*	–	–	–	–	2827.6 ± 38.6	2626.7 ± 57.8
	10	1761.6 ± 34.6	–	–	*n.a.*	–	–	–	–	1452.9 ± 23.4	1458.5 ± 44.8
	11	3391.7 ± 48.7	–	–	*n.a.*	–	–	–	–	4490.4 ± 88.4	4527.3 ± 88.3
	12	2283.9 ± 36.0	–	–	*n.a.*	–	–	–	–	1977.9 ± 28.4	2037.3 ± 64.3
	13	4159.6 ± 55.6	–	–	*n.a.*	–	–	–	–	4496.0 ± 68.9	3441.4 ± 70.9
	14	3997.1 ± 50.5	–	–	*n.a.*	–	–	–	–	3728.2 ± 75.3	3243.7 ± 67.3
	15	4350.2 ± 55.0	–	–	*n.a.*	–	–	–	–	–	5049.6 ± 98.0

Table 11
Rounded average and 95% confidence interval for the obtained cost over successful rounds for the Triangle Tire World and Exploding Blocks World in the experiment of Section 5.3. If no round is solved, then '–' is shown; if exactly one round is solved, then *inf* is shown in the 95% confidence interval.

Problem		FF-Replan	RFF	HMDPP	ReTrASE	SSiPP$_t$	SSiPP$_r$	L-SSiPP$_t$	L-SSiPP$_r$	SSiPP-FF$_t$	SSiPP-FF$_r$
Triangle tire world	1	2.0±0.0	6.3±0.1	6.2±0.1	6.7±0.1	6.8±0.1	6.7±0.2	6.8±0.1	6.8±0.2	4.7±0.0	6.0±0.1
	2	4.0±0.0	11.8±0.1	11.8±0.1	13.8±0.1	12.5±0.1	12.8±0.2	12.9±0.1	13.1±0.2	7.0±0.0	11.2±0.2
	3	6.0±0.0	19.2±0.1	19.3±0.1	21.7±0.2	20.8±0.2	20.2±0.3	20.4±0.1	20.7±0.3	9.0±0.0	18.1±0.5
	4	8.0±0.0	27.0±0.1	27.1±0.1	28.6±0.2	29.8±0.3	29.7±0.5	28.2±0.2	28.2±0.3	11.0±0.0	23.4±0.7
	5	10.0±0.0	35.0±0.1	35.2±0.2	37.5±0.2	39.8±0.4	38.6±0.8	36.1±0.2	36.1±0.4	13.0±0.0	27.6±1.3
	6	–	45.4±0.3	42.9±0.2	45.3±0.2	50.4±0.4	48.7±1.0	45.7±0.4	45.5±1.0	15.0±0.0	28.8±2.5
	7	–	53.6±0.4	50.8±0.2	54.1±0.2	60.7±0.5	60.1±1.1	57.3±0.8	58.5±1.6	17.0±*inf*	24.0±*inf*
	8	–	63.8±0.6	59.2±0.2	61.9±0.3	75.4±0.8	68.8±2.0	70.2±1.0	70.4±1.9	–	42.0±11.8
	9	–	72.7±0.9	66.9±0.2	69.4±0.3	89.2±1.3	80.7±2.3	82.6±1.4	84.2±3.1	–	32.0±*inf*
	10	–	83.7±1.0	75.0±0.2	78.0±0.3	103.5±1.4	94.1±1.9	92.9±2.0	92.7±3.6	–	–
	11	–	93.4±1.9	82.9±0.2	86.0±0.3	117.3±2.1	108.7±3.6	102.5±1.0	–	–	–
	12	–	103.9±2.9	91.2±0.3	93.8±0.4	128.9±2.9	125.3±0.7	–	–	–	–
	13	–	113.6±2.6	99.1±0.3	101.9±0.5	137.8±5.7	128.0±*inf*	–	–	–	–
	14	–	121.5±5.2	–	110.0±0.5	155.0±1.1	157.0±1.1	–	–	–	–
	15	–	131.2±3.9	–	117.8±0.6	–	–	–	–	–	–
Exploding blocks world	1	8.0±0.0	8.0±0.0	10.2±0.0	10.0±0.0	10.0±0.0	10.0±0.0	10.0±0.0	10.0±0.0	10.0±0.0	10.0±0.0
	2	12.9±0.1	12.0±0.0	12.0±0.0	12.0±0.0	12.0±0.0	12.0±0.0	12.0±0.0	12.0±0.0	12.0±0.0	15.5±0.2
	3	10.0±0.0	10.0±0.0	10.0±0.0	28.5±0.9	–	30.6±2.3	30.4±1.1	30.2±1.7	12.0±0.0	21.0±1.5
	4	15.4±0.1	15.4±0.1	14.0±0.0	14.6±0.1	15.3±0.1	15.4±0.2	15.4±0.1	15.5±0.2	15.4±0.1	15.4±0.2
	5	6.8±0.0	6.0±0.0	6.0±0.0	6.0±0.0	6.0±0.0	6.0±0.0	6.0±0.0	6.0±0.0	6.2±0.0	6.0±0.0
	6	13.9±0.1	14.0±0.1	14.7±0.1	13.2±0.1	13.4±0.1	13.5±0.1	13.4±0.1	13.3±0.1	14.9±0.1	13.8±0.1
	7	15.8±0.0	12.0±0.0	–	12.6±0.0	12.0±0.0	12.4±0.1	12.0±0.0	12.2±0.0	13.2±0.0	14.0±0.2
	8	27.2±0.4	24.0±0.0	34.0±0.0	48.0±1.7	28.1±0.0	28.1±0.1	28.7±0.1	39.0±0.6	32.9±0.5	40.5±3.4
	9	26.0±0.0	27.5±0.3	–	64.0±2.4	44.1±0.8	44.8±1.4	43.7±0.7	44.4±1.4	50.1±2.0	48.4±2.7
	10	35.0±0.5	36.0±0.0	–	78.1±6.3	62.5±3.9	60.8±1.9	63.9±6.2	–	34.8±0.3	42.8±4.9
	11	30.0±0.0	32.1±0.1	–	57.6±5.4	–	–	–	–	47.5±1.8	44.5±1.9
	12	38.8±0.4	38.0±0.0	–	–	–	–	–	–	44.3±1.8	44.0±1.0
	13	44.6±0.7	47.3±0.6	–	77.6±3.9	53.3±1.3	44.8±1.5	46.7±1.6	54.0±7.5	81.4±4.2	100.4±8.5
	14	37.0±0.3	51.0±0.9	–	–	–	–	–	–	40.9±0.6	67.2±6.2
	15	42.7±0.4	40.9±0.6	–	113.0±16.3	–	–	–	–	53.8±1.5	70.8±3.9

Table 12
Rounded coverage and 95% confidence interval for the triangle tire world using $t = 8$ and the zero-heuristic for SSiPP, Labeled-SSiPP and SSiPP-FF. For SSiPP-FF, the all-outcomes determinization is used. Best coverage for each problem (row), with respect to the results in Tables 8 and 9, are shown in bold. If no round is solved, then '–' is shown.

Problem		SSiPP	L-SSiPP	SSiPP-FF
Triangle tire world	1	$\mathbf{1.000 \pm 0.000}$	$\mathbf{1.000 \pm 0.000}$	$\mathbf{1.000 \pm 0.000}$
	2	$\mathbf{1.000 \pm 0.000}$	$\mathbf{1.000 \pm 0.000}$	$\mathbf{1.000 \pm 0.000}$
	3	0.997 ± 0.004	$\mathbf{1.000 \pm 0.000}$	0.533 ± 0.010
	4	0.977 ± 0.012	$\mathbf{1.000 \pm 0.000}$	0.162 ± 0.019
	5	0.963 ± 0.015	$\mathbf{1.000 \pm 0.000}$	0.082 ± 0.012
	6	0.950 ± 0.017	$\mathbf{1.000 \pm 0.000}$	0.049 ± 0.017
	7	0.913 ± 0.013	$\mathbf{1.000 \pm 0.000}$	0.023 ± 0.012
	8	0.870 ± 0.017	0.868 ± 0.013	0.015 ± 0.010
	9	0.882 ± 0.016	0.798 ± 0.017	0.003 ± 0.003
	10	0.842 ± 0.015	0.767 ± 0.012	–
	11	0.773 ± 0.015	0.717 ± 0.012	–
	12	0.738 ± 0.016	0.633 ± 0.013	–
	13	$\mathbf{0.717 \pm 0.018}$	0.595 ± 0.014	–
	14	$\mathbf{0.685 \pm 0.016}$	0.518 ± 0.016	–
	15	$\mathbf{0.617 \pm 0.016}$	0.422 ± 0.019	–

All SSiPP-based planners perform well in the exploding blocks world: SSiPP_r had the best coverage in 9 of the problems; SSiPP-FF_r had the best coverage in the 5 largest problems; and, for all the considered problems in the exploding blocks world, an SSiPP-based planner had the best coverage.

The performance in the triangle tire world problems is dominated by HMDPP. In this domain, the chosen parametrizations of SSiPP, Labeled-SSiPP, and SSiPP-FF do not perform as well as HMDPP or ReTrASE because their parametrizations use h_{add} as a heuristic. h_{add} in the triangle tire world guides the planners toward dead ends, and the SSiPP-based planners manage to avoid only the dead ends visible inside the short-sighted SSPs. As shown in [28], SSiPP performs the best in the triangle tire domain when the zero-heuristic is used. Table 12 shows the performance of SSiPP, Labeled-SSiPP, and SSiPP-FF using the parametrization $t = 8$ and the zero-heuristic (for SSiPP-FF, the all-outcomes determinization is used). For these parametrizations, the coverage obtained by SSiPP, Labeled-SSiPP, and SSiPP-FF is significantly improved: Labeled-SSiPP solved all rounds for problems 1 to 7, and SSiPP had the best coverage for the 3 largest problems compared to all considered planners.

6. Conclusion

In this article, we introduced the concept of short-sighted probabilistic planning through depth-based short-sighted SSPs, a special case of SSPs in which every state has a nonzero probability of being reached using at most t actions. Short-sighted probabilistic planning is a general technique that is used directly by the Short-Sighted Probabilistic Planner (SSiPP) algorithm. We also presented how to combine short-sighted probabilistic planning with two other techniques for probabilistic planning – labeling and determinizations – resulting in two new probabilistic planners, Labeled-SSiPP and SSiPP-FF, respectively.

The goal of Labeled-SSiPP is to improve the convergence time of SSiPP to the ϵ-consistent solution. This improvement is achieved by adding the labeling mechanism used by LRTDP. Moreover, Labeled-SSiPP takes advantage of states labeled as solved to prune the generated short-sighted SSPs as an ϵ-consistent solution from these labeled states is already known. For Labeled-SSiPP, we proved an upper bound on the number of iterations necessary to converge to an ϵ-consistent solution. On the empirical side, we showed that Labeled-SSiPP, using LRTDP as an underlying SSP solver, outperforms SSiPP, LRTDP, and FTVI on problems from the International Probabilistic Planning Competition (IPPC) and in control problems with a low percentage of relevant states.

We also introduced another probabilistic planner, SSiPP-FF, which combines short-sighted probabilistic planning and determinizations to compute suboptimal solutions more efficiently. The empirical results obtained show that SSiPP-FF successfully combines the behavior of SSiPP and FF-Replan through high coverage in problems without dead ends and a significant improvement the coverage of FF-Replan in problems with dead ends. The results also show that SSiPP and SSiPP-FF consistently outperform the other planners in all the problems of the exploding blocks world, a probabilistic interesting domain.

Our empirical results also show that SSiPP and its extensions have a state-of-the-art performance when the value of t is randomly selected for each depth-based short-sighted SSP. Because depth-based short-sighted SSPs can exploit the underlying structure of the problem through the parameter t, the performance of SSiPP and its extensions can be further improved by optimizing the value of t for the domain of the problem or for the problem being solved.

In the future, we plan to explore different methods to automatically adapt the value of t for a given problem, e.g., to model the problem of finding the best t as a multi-armed bandit problem. Our ongoing research agenda also includes (i) exploring different underlying SSP solvers for SSiPP (e.g., ILAO* [12]); (ii) combining SSiPP and Robust-FF; and (iii) exploring efficient methods to prune the generated short-sighted SSPs.

Acknowledgements

We thank the reviewers for their helpful comments. The first author was partly supported by São Paulo Research Foundation (FAPESP) under grant 2013/11724-0. This research was further sponsored by DARPA under agreement FA8750-12-2-0291, and by the National Science Foundation under award NSF IIS-1012733. The views and conclusions contained herein are those of the authors only.

Appendix A. Proof of Lemmas 2 and 3

Lemma 2. *Given an SSP* $\mathbb{S} = \langle \mathsf{S}, s_0, \mathsf{G}, \mathsf{A}, P, C\rangle$ *that satisfies Assumption 1,* $s \in \mathsf{S}$, $t \in \mathbb{N}^*$ *and a monotonic value function* V *for* $\mathbb{S}$*, then* $(B^k_{s,t}V)(\hat{s}) = (B^k V)(\hat{s})$ *for all* $\hat{s} \in \mathsf{S}_{s,t} \setminus \mathsf{G}_a$ *s.t.* $\min_{s_a \in \mathsf{G}_a} \delta(\hat{s}, s_a) \geq k$*, where* B *and* $B_{s,t}$ *represent, respectively, the Bellman operator applied to* $\mathbb{S}$ *and* $\mathbb{S}_{s,t}$ *using* V *as heuristic.*

Proof. If $\hat{s} \in \mathsf{S}_{s,t} \cap \mathsf{G}$, then $(B^k_{s,t}V)(\hat{s}) = (B^k V)(\hat{s}) = 0$ for all $k \in \mathbb{N}^*$ by the definitions of B and $B_{s,t}$. Otherwise, $\hat{s} \in \mathsf{S}_{s,t} \setminus \mathsf{G}_{s,t}$, therefore $1 \leq k \leq t$. We prove this case by induction on k:

- If $k = 1$, then by the definition of short-sighted SSPs (Definition 6), we can replace $C_{s,t}$ by C in $(B_{s,t}V)(\hat{s})$ as follows:

$$\begin{aligned}(B_{s,t}V)(\hat{s}) &= \min_a \sum_{s' \in \mathsf{S}_{s,t} \setminus \mathsf{G}_a} P(s' \mid \hat{s}, a)\big[C_{s,t}(\hat{s}, a, s') + V(s')\big] + \sum_{s' \in \mathsf{G}_a} P(s' \mid \hat{s}, a) C_{s,t}(\hat{s}, a, s') \\ &= \min_a \sum_{s' \in \mathsf{S}_{s,t} \setminus \mathsf{G}_a} P(s' \mid \hat{s}, a)\big[C(\hat{s}, a, s') + V(s')\big] \\ &\quad + \sum_{s' \in \mathsf{G}_a} P(s' \mid \hat{s}, a)\big[C(\hat{s}, a, s') + V(s')\big] \\ &= \min_a \sum_{s' \in \mathsf{S}_{s,t}} P(s' \mid \hat{s}, a)\big[C(\hat{s}, a, s') + V(s')\big].\end{aligned}$$

  Since $\min_{s_a \in \mathsf{G}_a} \delta(\hat{s}, s_a) \geq 1$, then $\{s' \in \mathsf{S} | P(s'|\hat{s}, a) > 0, \ \forall a \in A\} \subseteq \mathsf{S}_{s,t}$ and the previous sum over $\mathsf{S}_{s,t}$ equals the same sum over S. Therefore $(B_{s,t}V)(\hat{s}) = (BV)(\hat{s})$.
- Assume, as induction step, that this Lemma holds for $k \in \{1, \cdots, c\}$ where $c < t$. For $k = c + 1$, since $\min_{s_a \in \mathsf{G}_a} \delta(\hat{s}, s_a) \geq c + 1 > 1$, then $\{s' \in \mathsf{G}_a | P(s'|\hat{s}, a) > 0, \ \forall a \in A\} = \emptyset$. Thus,

$$\begin{aligned}\big(B_{s,t}(B^c V)\big)(\hat{s}) &= \min_a \sum_{s' \in \mathsf{S}_{s,t}} P(s' \mid \hat{s}, a)\big[C_{s,t}(\hat{s}, a, s') + (B^c V)(s')\big] \\ &= \min_a \sum_{s' \in \mathsf{S}_{s,t}} P(s' \mid \hat{s}, a)\big[C(\hat{s}, a, s') + (B^c V)(s')\big].\end{aligned}$$

  Since $c + 1 \leq t$ and $\hat{s} \in \mathsf{S}_{s,t} \setminus \mathsf{G}_{s,t}$, then $\{s' \in \mathsf{S} | P(s'|\hat{s}, a) > 0, \ \forall a \in A\} \subseteq \mathsf{S}_{s,t}$ and we can expand the previous sum from $s' \in \mathsf{S}_{s,t}$ to $s' \in \mathsf{S}$, i.e.,

$$\sum_{s' \in \mathsf{S}_{s,t}} P(s' \mid \hat{s}, a)\big[C(\hat{s}, a, s') + (B^c V)(s')\big] = \sum_{s' \in \mathsf{S}} P(s' \mid \hat{s}, a)\big[C(\hat{s}, a, s') + (B^c V)(s')\big].$$

  Therefore $(B^{c+1}_{s,t}V)(\hat{s}) = (B_{s,t}(B^c V))(\hat{s}) = (B^{c+1}V)(\hat{s})$. □

Lemma 3. *Under the conditions of Lemma 2,* $(B^k_{s,t}V)(s) \leq (B^k V)(s)$ *for all* $k \in \mathbb{N}^*$ *and* $\hat{s} \in \mathsf{S}_{s,t}$*, where* B *and* $B_{s,t}$ *represent, respectively, the Bellman operator applied to* $\mathbb{S}$ *and* $\mathbb{S}_{s,t}$ *using* V *as heuristic.*

Proof. By the definitions of B and $B_{s,t}$, we have the following trivial cases: (i) if $\hat{s} \in \mathsf{S}_{s,t} \cap \mathsf{G}$, then $(B^k_{s,t}V)(\hat{s}) = (B^k V)(\hat{s}) = 0$; and (ii) if $\hat{s} \in \mathsf{G}_a$, then $(B^k_{s,t}V)(\hat{s}) = 0 \leq (B^k V)(\hat{s})$. Thus, for the rest of this proof, we consider that $\hat{s} \in \mathsf{S}_{s,t} \setminus \mathsf{G}_{s,t}$.

Let m denote $\min_{s_a \in \mathsf{G}_a} \delta(\hat{s}, s_a)$. If $m \geq k$, then $(B^k_{s,t}V)(s) = (B^k V)(s)$ by Lemma 2. We prove the other case, i.e., $m < k$, by induction on $i = k - m$:

- If $i = 1$, then $(B^k_{s,t}V)(\hat{s}) = (B_{s,t}(B^{k-1}_{s,t}V))(\hat{s}) = (B_{s,t}(B^m_{s,t}V))(\hat{s})$ thus, by Lemma 2,

$$\begin{aligned}\big(B^k_{s,t}V\big)(\hat{s}) &= \big(B_{s,t}\big(B^m V\big)\big)(\hat{s}) \\ &= \min_a \sum_{s' \in \mathsf{S}_{s,t} \setminus \mathsf{G}_a} P(s' \mid \hat{s}, a)\big[C(\hat{s}, a, s') + (B^m V)(s')\big]\end{aligned}$$

$$+ \sum_{s' \in \mathsf{G}_a} P(s' \mid \hat{s}, a)\big[C(\hat{s}, a, s') + V(s')\big]$$
$$\leq \min_a \sum_{s' \in \mathsf{S}_{s,t}} P(s' \mid \hat{s}, a)\big[C(\hat{s}, a, s') + (B^m V)(s')\big],$$

where the last derivation is valid because V is monotonic by assumption. Since $\hat{s} \in \mathsf{S}_{s,t} \setminus \mathsf{G}_{s,t}$, then $\{s' \in \mathsf{S} | P(s'|\hat{s}, a) > 0, \forall a \in A\} \subseteq \mathsf{S}_{s,t}$ and we can expand the last sum over S. Therefore, $(B^k_{s,t} V)(\hat{s}) = (B_{s,t}(B^m V))(\hat{s}) \leq (B^k V)(\hat{s})$.

- Assume, as induction step, that it holds for $i \in \{1, \ldots, c\}$. Then, for $i = c + 1$, i.e., $k = m + c + 1$, we have that

$$\begin{aligned}(B^k_{s,t} V)(\hat{s}) &= \big(B_{s,t}(B^{m+c}_{s,t} V)\big)(\hat{s}) \\ &= \min_a \sum_{s' \in \mathsf{S}_{s,t} \setminus \mathsf{G}_a} P(s' \mid \hat{s}, a)\big[C(\hat{s}, a, s') + (B^{m+c}_{s,t} V)(s')\big] \\ &\quad + \sum_{s' \in \mathsf{G}_a} P(s' \mid \hat{s}, a)\big[C(\hat{s}, a, s') + V(s')\big].\end{aligned}$$

Since V is monotonic, we have that $V(s') \leq (B^{k+1} V)(s')$ for all $s' \in \mathsf{S}$. Also, by the induction assumption, $(B^{m+c}_{s,t} V)(s') \leq (B^{m+c} V)(s')$. Thus,

$$\begin{aligned}(B^k_{s,t} V)(\hat{s}) &\leq \min_a \sum_{s' \in \mathsf{S}_{s,t}} P(s' \mid \hat{s}, a)\big[C(\hat{s}, a, s') + (B^{m+c} V)(s')\big] \\ &= \min_a \sum_{s' \in \mathsf{S}} P(s' \mid \hat{s}, a)\big[C(\hat{s}, a, s') + (B^{m+c} V)(s')\big],\end{aligned}$$

because $\{s' \in \mathsf{S} | P(s'|\hat{s}, a) > 0, \ \forall a \in A\} \subseteq \mathsf{S}_{s,t}$. Therefore $(B^k_{s,t} V)(\hat{s}) \leq (B^k V)(\hat{s})$. □

References

[1] A. Barto, S. Bradtke, S. Singh, Learning to act using real-time dynamic programming, Artif. Intell. 72 (1–2) (1995) 81–138.
[2] D. Bertsekas, Dynamic Programming and Optimal Control, Athena Scientific, 1995.
[3] D. Bertsekas, J. Tsitsiklis, An analysis of stochastic shortest path problems, Math. Oper. Res. 16 (3) (1991) 580–595.
[4] D. Bertsekas, J.N. Tsitsiklis, Neuro-Dynamic Programming, Athena Scientific, 1996.
[5] B. Bonet, H. Geffner, Labeled RTDP: improving the convergence of real-time dynamic programming, in: Proceedings of the 13th International Conference on Automated Planning and Scheduling (ICAPS'03), 2003.
[6] B. Bonet, H. Geffner, Learning Depth-First Search: a unified approach to heuristic search in deterministic and non-deterministic settings, and its application to MDPs, in: Proceedings of the 16th International Conference on Automated Planning and Scheduling (ICAPS'06), 2006.
[7] B. Bonet, R. Givan, 2th International Probabilistic Planning Competition (IPPC-ICAPS'06), http://www.ldc.usb.ve/~bonet/ipc5/, 2007 (accessed on Dec 13, 2011).
[8] D. Bryce, O. Buffet, 6th International Planning Competition: uncertainty track, in: 3rd International Probabilistic Planning Competition (IPPC-ICAPS'08), 2008.
[9] P. Dai, J. Goldsmith, Topological value iteration algorithm for Markov decision processes, in: 20th International Joint Conference on Artificial Intelligence, IJCAI, 2007.
[10] P. Dai, Mausam, D.S. Weld, Focused topological value iteration, in: Int. Conf. on Automated Planning and Scheduling (ICAPS), 2009.
[11] T. Dean, L. Kaelbling, J. Kirman, A. Nicholson, Planning under time constraints in stochastic domains, Artif. Intell. 76 (1–2) (1995) 35–74.
[12] E. Hansen, S. Zilberstein, LAO*: a heuristic search algorithm that finds solutions with loops, Artif. Intell. 129 (1) (2001) 35–62.
[13] J. Hoffmann, B. Nebel, The FF planning system: fast plan generation through heuristic search, J. Artif. Intell. Res. 14 (1) (2001) 253–302.
[14] R. Howard, Dynamic Programming and Markov Processes, MIT Press, 1960.
[15] T. Keller, P. Eyerich, Probabilistic planning based on UCT, in: Proc. of the 22nd Int. Joint Conference on Automated Planning and Scheduling (ICAPS), 2012.
[16] E. Keyder, H. Geffner, The HMDP planner for planning with probabilities, in: 3rd International Probabilistic Planning Competition (IPPC-ICAPS'08), 2008, p. 8.
[17] L. Kocsis, C. Szepesvári, Bandit based Monte-Carlo planning, in: Proceedings of the European Conference on Machine Learning (ECML'06), 2006.
[18] A. Kolobov, Mausam, D.S. Weld, ReTrASE: integrating paradigms for approximate probabilistic planning, in: Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI'09), 2009.
[19] I. Little, S. Thiébaux, Probabilistic planning vs replanning, in: Proceedings of ICAPS Workshop on IPC: Past, Present and Future, 2007.
[20] H. McMahan, M. Likhachev, G. Gordon, Bounded real-time dynamic programming: RTDP with monotone upper bounds and performance guarantees, in: Proceedings of the 22nd International Conference on Machine Learning (ICML'05), 2005.
[21] J. Pearl, Heuristics: Intelligent Search Strategies for Computer Problem Solving, Addison-Wesley, Menlo Park, California, 1985.
[22] M. Puterman, Markov Decision Processes: Discrete Stochastic Dynamic Programming, John Wiley & Sons, Inc., 1994.
[23] S.J. Russel, P. Norvig, Artificial Intelligence – a Modern Approach. Prentice Hall Series in Artificial Intelligence, second edition, Prentice Hall, 2003.
[24] S. Sanner, R. Goetschalckx, K. Driessens, G. Shani, Bayesian real-time dynamic programming, in: Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI'09), 2009.
[25] T. Smith, R.G. Simmons, Focused real-time dynamic programming for MDPs: squeezing more out of a heuristic, in: Proceedings of the 21st National Conference on Artificial Intelligence (AAAI'06), 2006.
[26] F. Teichteil-Koenigsbuch, G. Infantes, U. Kuter, RFF: a robust, FF-based MDP planning algorithm for generating policies with low probability of failure, in: 3rd International Planning Competition (IPPC-ICAPS'08), 2008.

[27] F. Teichteil-Königsbuch, V. Vidal, G. Infantes, Extending classical planning heuristics to probabilistic planning with dead-ends, in: Proc. of the 26th Nat. Conf. on Artificial Intelligence (AAAI), 2011.
[28] F.W. Trevizan, M.M. Veloso, Short-sighted stochastic shortest path problems, in: Twenty-Second International Conference on Automated Planning and Scheduling (ICAPS), 2012.
[29] F.W. Trevizan, M.M. Veloso, Trajectory-based short-sighted probabilistic planning, in: Advances in Neural Information Processing Systems (NIPS), 2012.
[30] S. Yoon, A. Fern, R. Givan, FF-Replan: a baseline for probabilistic planning, in: Proceedings of the 17th International Conference on Automated Planning and Scheduling (ICAPS'07), 2007.
[31] S. Yoon, A. Fern, R. Givan, S. Kambhampati, Probabilistic planning via determinization in hindsight, in: Proceedings of the 23rd National Conference on Artificial Intelligence (AAAI'08), 2008.
[32] S. Yoon, W. Ruml, J. Benton, M.B. Do, Improving determinization in hindsight for online probabilistic planning, in: Proceedings of the 20th International Conference on Automated Planning and Scheduling (ICAPS'10), 2010.
[33] H. Younes, M. Littman, D. Weissman, J. Asmuth, The first probabilistic track of the international planning competition, J. Artif. Intell. Res. 24 (1) (2005) 851–887.
[34] S. Zickler, M. Veloso, Variable level-of-detail motion planning in environments with poorly predictable bodies, in: Proceedings of the 19th European Conference on Artificial Intelligence (ECAI'10), 2010.