Short-Sighted Probabilistic Planning

Felipe Trevizan

August 2013
CMU-ML-13-109

Carnegie Mellon

Short-Sighted Probabilistic Planning

Felipe W. Trevizan

August 2013
CMU-ML-13-109

Machine Learning Department
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Manuela M. Veloso, Chair
Emma Brunskill
Reid Simmons
Bart Selman (Cornell University)

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy.

This research was sponsored by the National Science Foundation under grant numbers CNS0509383, CNS0625518, and IIS0644225; the US Army Research Office under grant number W911NF0710287; the Pennsylvania Department of Community and Economic Development; and the Pittsburgh Life Sciences Greenhouse.

Keywords: probabilistic planning, short-sighted planning, planning under uncertainty, optimal planning

To my late grandfather Edgard and aunt Léa.

Abstract

Planning is an essential part of intelligent behavior and a ubiquitous task for both humans and rational agents. One framework for planning in the presence of uncertainty is probabilistic planning, in which actions are described by a probability distribution over their possible outcomes. Probabilistic planning has been applied to different real-world scenarios such as public health, sustainability and robotics; however, the usage of probabilistic planning in practice is limited due to the poor performance of existing planners.

In this thesis, we introduce a novel approach to effectively solve probabilistic planning problems by relaxing them into short-sighted problems. A short-sighted problem is a relaxed problem in which the state space of the original problem is pruned and artificial goals are added to heuristically estimate the cost of reaching an original goal from the pruned states. Differently from previously proposed relaxations, short-sighted problems maintain the original structure of actions and no restrictions are imposed in the maximum number of actions that can be executed. Therefore, the solutions for short-sighted problems take into consideration all the probabilistic outcomes of actions and their probabilities. In this thesis, we also study different criteria to generate short-sighted problems, i.e., how to prune the state space, and the relation between the obtained short-sighted models and previously proposed relaxation approaches.

We present different planning algorithms that use short-sighted problems in order to solve probabilistic planning problems. These algorithms iteratively generate and execute optimal policies for short-sighted problems until the goal of the original problem is reached. We also formally analyze the introduced algorithms, focusing on their optimality guarantees with respect to the original probabilistic problem. Finally, this thesis contributes a rich empirical comparison between our algorithms and state-of-the-art probabilistic planners.

Acknowledgments

There are many people, to whom I am deeply indebted, and without whom this work would never have been finished. First and foremost is my advisor, Manuela Veloso. She has provided invaluable insights, advice, guidance, and support on both my research and career.

I would like to thank my thesis committee members, Emma Brunskill, Reid Simmons and Bart Selman. Their valuable questions, insights and comments have greatly improved the content of this work. I would also like to express my gratitude to Professors Leliane Nunes de Barros, Fabio Cozman, Hector Geffner, Carlos Guestrin and Seth Goldstein for their guidance and support during my graduate studies.

I am grateful for being part of the CORAL group at CMU. Their feedback on all my practice talks greatly improved the presentation of my work. I would also like to thank Somchaya Liemhetcharat, who read this thesis and provided several useful comments and suggestions. I am also grateful for the administrative support given by Diane Stidle, Christina Contreras, Michelle Martin and Marilyn Walgora which has helped me focus in my research.

I would like to thank the friends I made along the way and which have been there for me through the ups and downs of graduate school: Bob Coblentz, Charalampos (Babis) Tsourakakis and Maria Tsiarli, Chiara Cosentino, Daniel Pickem, Harini Aiyer, Jan Pristavok and Maggie Hari, Lisa Storey and Kevin Peterson, Matt Schnall, Mladen Kolar and Gorana Smailagic, Neil Blumen, Polo Chau, Regina Schulte-Ladbeck, Sam Taggard, Stefan Kremser, and Toni Price.

I would also like to thank my friends back in Brazil for their support, encouragement and friendship: Ana Lygia Monteferrario, Carlos Cardonha, Cesar Augusto, Ednei Reis, Ettore Ligorio, Fabio Kacuta, George Silva, Giuliano Araujo, Gordana Manić, Heitor Marcos and Talita Cordeiro, Kim Samejima, Marcelo Hashimoto, Marcelo and Telma Amador, Maria Manuella Rocco, Mel Akemi, Paulo Lima, Paulo Salem, Regis Chinen, Ricardo Andrade, Thiago Abdalla, and William Moreira.

Most of all, I would like to thank my family, especially my parents, Marcos and Rosana, and my sister, Carolina, for always being supportive and encouraging. Without their boundless love, this thesis would not have been possible.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Planning is an essential part of intelligent behavior and a ubiquitous task for both humans and rational agents [Newell and Simon, 1963]. One framework for planning is probabilistic planning, in which actions are described by the probability distribution over their possible outcomes. Solutions to a probabilistic planning problem are policies, i.e., a mapping from states to actions.

In order to illustrate the trade-offs between different types of policies, consider the probabilistic problem of an agent navigating an environment to reach a goal location with two possible paths: (i) a maze; and (ii) a hallway with locked doors. The agent has all the necessary keys to open the doors in the hallway; however, assume that with non-zero probability, the key jams in the door lock, resulting in a door that cannot be unlocked. Figure 1.1 illustrates this probabilistic planning problem.

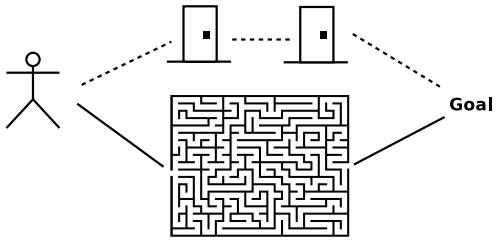


Figure 1.1: Example of a probabilistic planning problem. The agent has to reach the goal location from the initial location. The two doors in the hallway (top) are locked and, with non-zero probability, the key jams in the doors. A jammed key cannot open a door.

One possible solution for probabilistic planning problems is a policy that maps every state of the problem to an action. Solutions of this class, i.e., closed policies, are extremely powerful because they encompass all the possible probabilistic reachable states in the environment. Therefore, a closed policy for the example in Figure 1.1 encompasses the cases in which: the keys do not jam; the key jams in the first door; the key opens the first door and jams in the second; and

the complete solution of the maze. Suppose the probability of the key jamming is 0.01, then the probability of not reaching the goal through the hallway is $1 - 0.99^2 = 0.0199$. Thus, with probability 0.9801, the possibly large computational effort to find the maze's solution is wasted, since the maze would not be explored.

The second class of possible solutions is a policy that maps only a subset of states to actions. Such policies, i.e., partial policies, do not address all possible probabilistic reachable states in the environment. Therefore, a state not predicted by the partial policy might be reached and, when and if such a state is reached, a new partial policy has to be computed and executed. In the example of Figure 1.1, a possible partial policy is to reach the goal through the hallway and not consider the case in which a key jams. If a key jams, then a new partial policy in which the agent backtracks and solves the maze is returned. Note that this partial policy ignores the size of the maze, and it would be executed even if the probability of jamming the key is high or if the maze is small.

Algorithms to solve probabilistic planning problems can be classified according to the type of policy returned by them: probabilistic planners, e.g., [Barto et al., 1995], compute (optimal) closed policies; and replanners, e.g., [Yoon et al., 2007], return partial policies. Since probabilistic planners must consider all the probabilistic reachable states in order to compute a closed policy, their scalability is limited to small problems. Alternatively, replanners compute partial policies based on simplifications of the original problem and are able to scale up to large problems. A common simplification applied by replanners is to relax the probabilistic actions into deterministic actions [Yoon et al., 2007]. This action relaxation results in algorithms that are oblivious to probabilities. Therefore, replanners based on action simplification obtain good performance in some domain but poor performance in probabilistic interesting problems [Little and Thiébaux, 2007], i.e., problems in which probabilities cannot be ignored.

This thesis introduces a novel approach to solve probabilistic planning problems by relaxing them into *short-sighted problems*. A short-sighted problem is a relaxation in which the state space of the original problem is pruned and artificial goals are added to heuristically estimate the cost of reaching an original goal from the states pruned. Figure 1.2 shows an example of short-sighted problem for the probabilistic planning problem depicted in Figure 1.1. This short-sighted problem example perfectly represents the hallway path and prunes the maze path due to its large size. The locations $A_1, \cdots, A_4$ represent artificial goals, i.e., non-goal locations of the original problem which are goals for the short-sighted problem.

Differently from previously proposed relaxations, short-sighted problems maintain the original structure of actions and no restrictions are imposed in the maximum number of actions that can be executed. For instance, the probabilistic action to open a door is the same in both the orig-

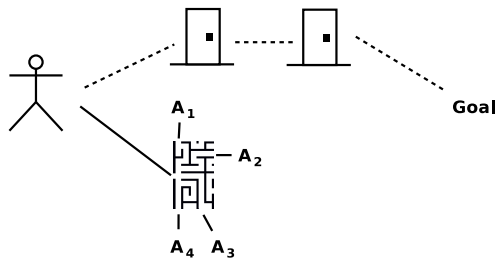
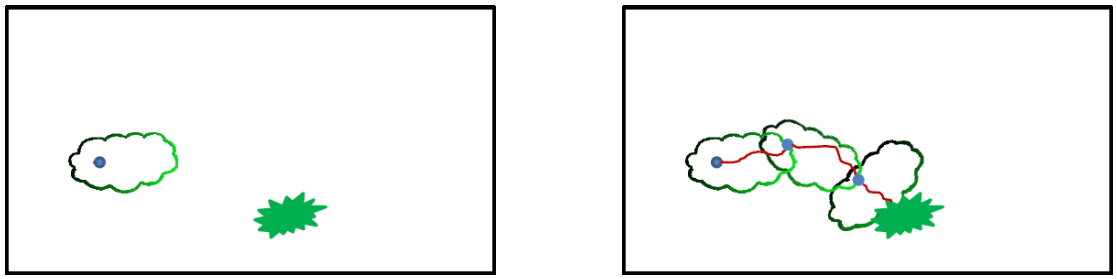


Figure 1.2: Example of short-sighted problem for the probabilistic planning problem in Figure 1.1. A_1 to A_4 represent artificial goals. A heuristic is used to estimate the cost of solving the maze from each location A_i.

inal problem of our example (Figure 1.1) and its short-sighted example (Figure 1.2). Therefore, the solutions for short-sighted problems take into consideration all the probabilistic outcomes of actions and their probabilities. In this thesis, we study different criteria to generate short-sighted problems, i.e., to how prune the state space, and the relation between the obtained short-sighted models and previously proposed relaxation approaches.

Another important aspect of short-sighted problems is the guidance towards the original goals offered by the artificial goals. For instance, in the short-sighted problem in Figure 1.2, the sum of Manhattan distances can be used as a heuristic to estimate the cost of solving the original problem starting from each artificial goal A_i. Therefore, an optimal closed policy for this short-sighted problem is able to heuristically approximate the trade-off between the two different paths in our running example: if the probability of a key jamming is small, the hallway path is preferred since it is the shortest path; alternatively, if the jamming probability is large, then solving the (large) maze is chosen due to the low probability of successfully opening both doors in the hallway path. Since short-sighted problems are small with respect to the original problem, the computation of an optimal closed policy for them is feasible.

This thesis then introduces different planning algorithms that use short-sighted problems and their optimal closed policies in order to solve probabilistic planning problems. These algorithms consist in iteratively generating and executing a closed policy for short-sighted problems until the goal state of the original problem is reached. Different methods of combining the solutions from short-sighted problems are studied, including sequential and parallel approaches. We formally analyze the introduced algorithms, focusing on their optimality guarantees with respect to the original probabilistic problem. Finally, this thesis also contributes a rich empirical comparison between the proposed algorithms and state-of-the-art probabilistic planners and replanners.



(a) (b)

Figure 1.3: Overview of the thesis approach. (a) Representation of the state space with one short-sighted problem and (b) a sequence of short-sighted problems. The initial state of the problem is represented by the blue dot, the goal states are represented by the green star. Each short-sighted problem is depicted as a cloud. States in the border of the cloud are artificial goals and the color gradient in the cloud contour represents the heuristic cost to reach a goal state: darker regions are more costly than lighter regions. The red line represents the states visited during the execution of a closed policy of the respective short-sighted problem.

1.1 Thesis Question and Approach

This thesis seeks to answer the question,

> How to plan for probabilistic environments such that it *scales up* while offering *formal guarantees* underlying the policy generation?

We answer this question by introducing new models to represent subproblems of probabilistic planning problems, developing new algorithms to exploit the proposed subproblems and analyzing, both theoretically and empirically, the proposed algorithms.

Precisely, we introduce different models to represent short-sighted problems, i.e., subproblems of the original problem with pruned state space and artificial goals to heuristically guide the search towards the original goals. Figure 1.3(a) depicts the state space of a probabilistic planning problem and the state space of a short-sighted problem. Each short-sighted model defines a criterion to prune the state space of the original problem and, pictorially, a short-sighted model governs the shape of the clouds in Figure 1.3. We formally show the relationship between the optimal solutions for short-sighted models and probabilistic planning problems, e.g., the former is a lower bound for the latter.

Based on the general definition of short-sighted problems, we design algorithms that iteratively generate and solve short-sighted problems of the original probabilistic planning problem. Due to the reduced size of the short-sighted problems, an optimal closed policy can be computed and these policies are combined in order to obtain a solution to the original probabilistic planning problem. Figure 1.3(b) depicts this process in which a closed policy is computed for

a short-sighted problem (cloud) and executed (red line) until a goal of the original problem is reached; if an artificial goal is reached (point in the cloud's border), then this process is repeated using the reached artificial goal as the new initial state.

We also prove the theoretical properties of the introduced algorithms, e.g., guarantee to always reach an original goal state and convergence to the optimal solution of the original problem. Finally, we empirically compare the proposed algorithms and short-sighted models to understand the different trade-offs between them.

1.2 Contributions

The key contributions of this thesis are:

- **Depth-based, Trajectory-based and Greedy Short-Sighted Probabilistic Problems.** We introduce three different short-sighted models based on different criteria to prune the state space: depth-based short-sighted problems, in which all the states are reachable using no more than a given number of actions; trajectory-based short-sighted problems, in which all states are reachable with probability greater or equal than a given threshold; and greedy short-sighted problems, in which the states have the best trade-off between probability of being reached and expected cost to reach the goal from them.
- **Short-Sighted Probabilistic Planner and extensions.** We introduce the Short-Sighted Probabilistic Planner (SSiPP) algorithm that solves probabilistic planning problems using short-sighted problems. We extend SSiPP in three different directions: Labeled SSiPP, which improves the convergence of SSiPP to the optimal solution; SSiPP-FF, which improves the efficiency of SSiPP for generating suboptimal solutions; and Parallel Labeled SSiPP, which solves multiple short-sighted problems in parallel to speedup the search for the optimal solution.
- **Theoretical and Empirical Analysis.** We prove the theoretical properties of our algorithms, e.g., termination (i.e., always reach a goal state) and optimality. We also provide a comprehensive empirical evaluation of the proposed algorithms under different scenarios: (i) finding the optimal solution; (ii) finding a solution with limited time to compute the next action to be executed; and (iii) finding a solution under the International Probabilistic Planning Competition [Younes et al., 2005, Bonet and Givan, 2007, Bryce and Buffet, 2008] rules.

1.3 Guide to the Thesis

Here we outline the chapters that follow.

- **Chapter 2 – Background.** We review the basics for Stochastic Shortest Path problems (SSPs), our chosen model to represent probabilistic planning problems. We also review the following algorithms necessary for the next chapters: Real-Time Dynamic Programming [Barto et al., 1995] and FF-Replan [Yoon et al., 2007].
- **Chapter 3 – Short-Sighted Probabilistic Planning.** We present depth-based short-sighted Stochastic Shortest Path problems, a novel model to represent subproblems of SSPs. We also introduce the Short-Sighted Probabilistic Planner (SSiPP) algorithm using the depth-based short-sighted SSPs as model for the subproblems generated by SSiPP. We prove the relations between the solutions of SSPs and their depth-based short-sighted SSPs and that SSiPP is optimal. We conclude by showing the effectiveness of SSiPP using depth-based short-sighted SSPs in a proposed series of problems.
- **Chapter 4 – General Short-Sighted Models.** This chapter extends the concept of depth-based short-sighted SSPs to a general model in which a function to prune the state space is given. Using this general formulation, we introduce two new models for short-sighted problems: trajectory-based short-sighted SSPs and greedy short-sighted SSPs.
- **Chapter 5 – Extending Short-Sighted Probabilistic Planner.** We present three extensions of SSiPP: Labeled SSiPP, SSiPP-FF and Parallel Labeled SSiPP. We also present the theoretical guarantees of each of these algorithms and demonstrate their effectiveness in different proposed domains.
- **Chapter 6 – Related Work.** We discuss the previous work in optimal and suboptimal probabilistic planning, and how they relate to this thesis.
- **Chapter 7 – Empirical Evaluation.** This chapter presents an extensive empirical evaluation of the proposed probabilistic planners against the state-of-the-art probabilistic planners.
- **Chapter 8 – A Real World Application: a Service Robot Searching for Objects.** We show how the problem of an autonomous agent moving in a known environment to find objects, while minimizing the search cost, can be solved by using short-sighted probabilistic planning. As a concrete example, we use the problem of a mobile service robot that moves in a building to find an object, whose location is not deterministically known, and to deliver it to a location.

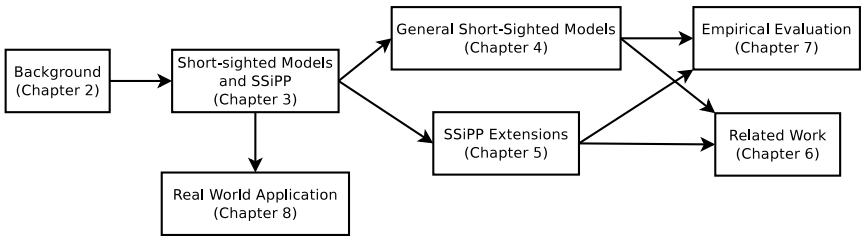


Figure 1.4: Organization of the chapters in this thesis.

- **Chapter 9 – Conclusion.** We conclude this dissertation with a summary of our contributions along with a discussion of future work for short-sighted planning.

Figure 1.4 illustrates the chapters' organization and the dependency between chapters of this dissertation. All readers should begin with Chapter 2, which provides the necessary mathematical background and defines the notation used in this dissertation.

Chapter 2

Background

This chapter introduces the *Stochastic Shortest Path Problems*, the probabilistic planning model used in this dissertation. We begin with a basic overview (Section 2.1) that follows the presentation in [Bertsekas, 1995] with small differences in notation to match the planning community notation. In Section 2.2, we present the high-level representation of probabilistic planning problems proposed by the planning community. Finally, Section 2.3 reviews two standard algorithms for probabilistic planning, *Real-Time Dynamic Programming* and *FF-Replan*, that are frequently referred in this dissertation.

2.1 Stochastic Shortest Path Problem

A Stochastic Shortest Path Problem (SSP) [Bertsekas and Tsitsiklis, 1991] is a tuple $\mathbb{S} = \langle \mathsf{S}, s_0, \mathsf{G}, \mathsf{A}, P, C \rangle$, in which:

- S is the finite set of states;
- $s_0 \in \mathsf{S}$ is the initial state;
- $\mathsf{G} \subseteq \mathsf{S}$ is the non-empty set of goal 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, 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 $\mathcal{T} = \langle s_0, s_1, s_2, \dots \rangle$ visited by the agent is called a trajectory 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 $\langle a_0, a_1, a_2, \dots \rangle$ such that a_i is executed in state s_i and $P(\mathcal{T}|\langle a_0, a_1, a_2, \dots \rangle) = \prod_{i \in \{0,1,\dots\}} P(s_{i+1}|s_i, a_i) > 0$.

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 since, 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) [Puterman, 1994]. Alternatively, if no goal states are given, then the horizon becomes infinite since 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 is known as discounted infinite-horizon MDPs [Puterman, 1994]. Both finite-horizon and discounted infinite-horizon MDPs and are special cases of SSPs [Bertsekas and Tsitsiklis, 1996].

A solution to an SSP is a policy π, i.e., a mapping from S to A. We denote all the states reachable from s_0 when following π 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 be classified according to S^π and R^π. If a policy 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. A special case of closed policies is the *complete policies*, i.e., policies that can be followed from any state $s \in \mathsf{S}$ without replanning. Thus, for any complete policy π, we have that $\mathsf{R}^\pi = \emptyset$. If a policy π is not closed, then $\mathsf{R}^\pi \cap \mathsf{S}^\pi \neq \emptyset$ and π is known as a partial policy. For any partial policy π, replanning has non-zero probability of happening, since every state $s \in \mathsf{R}^\pi \cap \mathsf{S}^\pi$ has non-zero probability of being reached when following π from s_0.

Policies can also be classified according to their termination guarantee. π is a proper policy if it is inevitable to reach a goal state when following the policy π from s_0. Formally:

Definition 2.1 (Proper policy)**.** A policy π is proper if, for all $s \in \mathsf{S}^\pi$, there exists a trajectory $\mathcal{T} = \langle s, s_1, \dots, s_k \rangle$ 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:

Assumption 2.1. There exists at least one policy that is both proper and complete.

By definition, every proper policy is closed and every partial policy is improper; however, not all closed policies are proper. To illustrate this relationship between closed and proper poli-

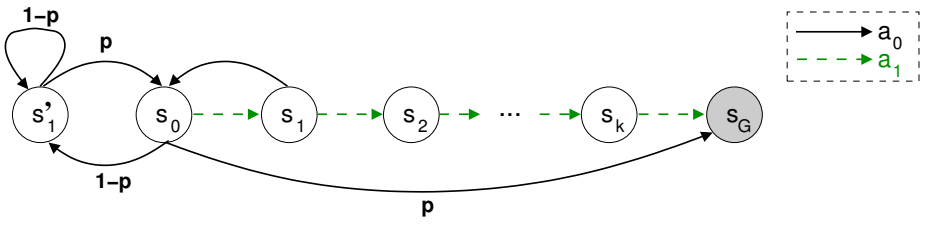


Figure 2.1: Example of an Stochastic Shortest Path Problem (SSP). The initial state is s_0, the goal set is $\mathsf{G} = \{s_\mathsf{G}\}$ and $C(s, a, s') = 1, \forall s \in \mathsf{S}, a \in \mathsf{A}$ and $s' \in \mathsf{S}$.

cies, consider the SSP depicted in Figure 2.1: $\pi_0 = \{(s_0, a_0), (s'_1, a_0)\}$ 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 a partial 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 since, 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:

$$V^\pi(s) = \begin{cases} 0 & \text{if } s \in \mathsf{G} \\ E\left[C(s, a, s') + V^\pi(s') | s, a = \pi(s)\right] & \text{otherwise} \end{cases}, \qquad \forall s \in \mathsf{S}^\pi. \tag{2.1}$$

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

Assumption 2.2. 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, since the cost function $C(s, a, s')$ is strictly positive. For instance, consider the SSP depicted in Figure 2.1; 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 closed 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 overall policies, exists and is

unique [Bertsekas and Tsitsiklis, 1996]. 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 Equations*:

$$V^*(s) = \begin{cases} 0 & \text{if } s \in \mathsf{G} \\ \min\limits_{a \in \mathsf{A}} E\left[C(s,a,s') + V^*(s')|s,a\right] & \text{otherwise} \end{cases}, \qquad \forall s \in \mathsf{S}. \tag{2.2}$$

Every optimal policy π^* can be obtained by replacing $\min$ by argmin in (2.2), i.e., π^* is a *greedy policy* of V^*:

Definition 2.2 (Greedy policy)**.** Given a value function V, the greedy policy π^V is such that $\pi^V(s) = \operatorname{argmin}_{a \in \mathsf{A}} E[C(s,a,s') + V(s')|s,a]$ 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) [Howard, 1960]: given an initial guess V^0 for V^*, compute the sequence $\langle V^0, V^1, \ldots, V^k \rangle$ 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 $s \in \mathsf{S}$:

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

We denote by B^k the composition of the operator B, i.e., $(B^k V)(s) = (B(B^{k-1})V)(s)$ 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 2.1 holds, V^k converges to V^* as $k \to \infty$ and $0 \leq V^*(s) < \infty$ for all $s \in \mathsf{S}^{\pi^*}$ [Bertsekas, 1995]. In practice, we are interested in the problem of finding ϵ-optimal solutions, i.e., given $\epsilon > 0$, to find a value function V that is no more that ϵ away from V^*:

Definition 2.3 (ϵ-optimality)**.** Given an SSP $\mathbb{S}$, a value function V for $\mathbb{S}$ is ϵ-optimal if

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

where $\mathsf{S}' = \mathsf{S}^{\pi^V}$, i.e., the states reachable from s_0 when following the greedy policy π^V; $R(s,V)$ 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.

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 referred 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 2.4 (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 dissertation:

Theorem 2.1. *Given an SSP $\mathbb{S}$ in which Assumption 2.1 holds, then the operator B preserves [Bertsekas and Tsitsiklis, 1996, 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 $k \in \mathbb{N}^*$.*

2.2 Factored Representation

In the previous section, we reviewed SSPs using their *enumerative* representation (also known as *explicit* representation). In the enumerative representation, the set of states S, the set of goal states G, the set of actions A and the transition probability distributions $P(\cdot|\cdot,\cdot)$ are represented explicitly by directly enumerating each element of them. This enumerative specification of $\mathbb{S}$ can be burdensome for large problems, especially the encoding of $P(\cdot|\cdot,a)$ as a matrix $\mathsf{S} \times \mathsf{S}$ for each action a. Also, in many cases it is advantageous, from both the computational and representational perspective, to define a set of states by their properties; for instance, the goal for a service robot navigating in a building could be compactly represented by a high-level statement such as "the robot is at a kitchen".

To compactly represent large SSPs and to use high-level statements to represent set of states, the *factored* representation is used [Boutilier et al., 1999]. In the factored representation, SSPs are encoded using *state variables*, i.e., variables f_i with domain D_i, and the set of state variables is denoted as $\mathsf{F} = \{f_1, \cdots, f_k\}$. The cross product $\times_{i=1}^{|\mathsf{F}|} \mathsf{D}_i$ represents the state space S, thus a state $s \in \mathsf{S}$ is the tuple $\langle v_0, \cdots, v_{|\mathsf{F}|} \rangle$ where $v_i \in \mathsf{D}_i$. For example, the SSP in Figure 2.2 can be factored using two binary state variables, x and y, such that state $\langle x, y \rangle$ equals the state s_i for $i = x + 2y$. For the rest of this dissertation, we assume the domain of each state variable $f \in \mathsf{F}$ to be binary, thus $|\mathsf{S}| = 2^{|\mathsf{F}|}$.

Another benefit of using state variables is a compact representation of the transition probabilities $P(\cdot|\cdot,a)$ using two-stages temporal Bayesian Networks [Boutilier et al., 1999]. To illustrate the space savings obtained by using the factored representation of actions, consider action a_0 of the SSP depicted in Figure 2.2. The enumerative representation of $P(\cdot|\cdot,a_0)$ is a 4-by-4 stochastic matrix, which is encoded with $4 \times 3 = 12$ numbers. For this example, a factored representation is $P(\langle x', y' \rangle | \langle x, y \rangle, a_0) = P(x'|x,a_0) \times P(y'|y,a_0)$ where $P(x' = 1 | x = 0, a_0) = 0.25$,

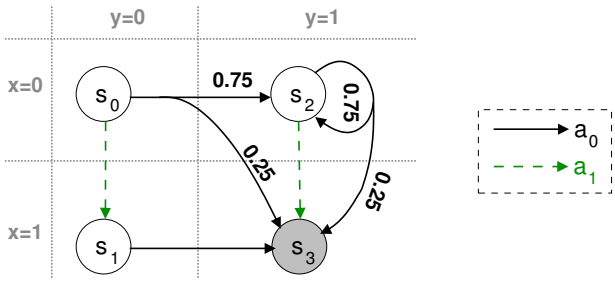


Figure 2.2: Example of a factored SSP. The initial state is s_0, the goal set $\mathsf{G} = \{s_3\}$ and $C(s, a, s') = 1$ for all $s \in \mathsf{S}, a \in \mathsf{A}, s' \in \mathsf{S}$. This SSP can be represented as a factored SSP with two binary state variables, x and y, such that the state $\langle x, y \rangle$ equals the state s_i for $i = x + 2y$.

```
(:action a0
 :effect (and (y) (prob 0.25 (x) 0.75 (not (x))))
)
(:action a1
 :precondition (not (x))
 :effect (x)
)
```

Figure 2.3: Example of PPDDL representation of the actions of the SSP in Figure 2.2. Note that only action `a1` has a precondition.

$P(x' = 1|x = 1, a_0) = 1$ and $P(y' = 1|y = 0, a_0) = P(y' = 1|y = 1, a_0) = 1$, which can be encoded with only 4 numbers.

The Probabilistic Planning Domain Description Language (PPDDL) [Younes and Littman, 2004] is a standard language to represent factored SSPs that is used in the international probabilistic planning competitions (IPPC) [Younes et al., 2005, Bonet and Givan, 2007, Bryce and Buffet, 2008]. PPDDL syntax is based on LISP and an action a consists of a precondition, that is, a formula over the state variables characterizing the states in which a is applicable, and an effect. The effect describes how the states variables change when a is applied. Any state variable not explicitly modified by a remains unchanged after executing a (frame assumption). Figure 2.3 contains the PPDDL representation of the actions a_0 and a_1 of the SSP represented in Figure 2.2.

PPDDL also features *predicates* and *action schemas*. These extensions use the concept of domain variables, i.e., class of finite objects. A predicate is mapping from a value assignment of one or more domain variables to a state variables. For instance, we can model a graph $G = \langle N, E \rangle$ by using a domain variable called `NODE` in which its domain is N and edges of the graph as the predicate `edge`(i, j) where i and j are domain variables of the type `NODE`; in this case, each possible instantiation of `edge`(i, j) represents one binary state variable. Therefore, if the planning problem defines three objects of the type `NODE`, namely n_1, n_2, n_3, then six state variables are instantiated

representing the edges $(n_1, n_2), (n_1, n_3), (n_2, n_1), \ldots, (n_3, n_2)$. Similarly to predicates, action schemas map value assignment of one or more domain variables to an action.

2.3 Relevant Probabilistic Planning Algorithms

In this section, we review two algorithms for solving SSPs: *Real-Time Dynamic Programming* (Section 2.3.1) that uses dynamic programming and sampling in order to compute optimal closed policies; and *FF-Replan* (Section 2.3.2) that relaxes SSPs into deterministic problems and returns partial policies.

2.3.1 Real-Time Dynamic Programming

Real-Time Dynamic Programming (RTDP) [Barto et al., 1995] is an extension of Learning-Real-Time-A* [Korf, 1990] for probabilistic planning problems. RTDP computes closed policies instead of complete policies, and, since a closed policy π is defined only for the states in $\mathsf{S}^\pi \subseteq \mathsf{S}$, RTDP converges to the ϵ-optimal solution faster than VI when $|\mathsf{S}^\pi| \ll |\mathsf{S}|$.

RTDP, presented in Algorithm 2.1, simulates the current greedy policy π^V (Line 13) to sample trajectories from the initial state s_0 to a goal state. Each trajectory is sampled by the procedure RTDP-TRIAL (Line 7): while the current state s is not a goal state, the greedy action a w.r.t. $V(s)$ is chosen; a Bellman backup is applied on s; and a resulting state of applying a on s is sampled (Lines 10 to 13). The value function V is initialized by the input heuristic H (Line 3) using a lazy approach, i.e., if the value $V(s)$ is requested and V is not defined on s, then $H(s)$ is computed (on demand) and assigned to $V(s)$.

Since the greedy selection of actions is interleaved with updates on V, RTDP-TRIAL cannot be trapped in loops and always reaches a goal state. Formally, if Assumption 2.1 holds for the given SSP, then RTDP-TRIAL always terminates. Moreover, if Assumption 2.1 holds and the heuristic H used is also admissible, then RTDP always converges to the optimal solution V^*, i.e., $R(\mathbb{S}, V) = 0$, after several calls of RTDP-TRIAL (possibly infinitely many) [Barto et al., 1995, Theorem 3, p. 132].

2.3.2 FF-Replan

FF-Replan [Yoon et al., 2007] is a replanner based on *determinization*, i.e., a relaxation of a given SSP $\mathbb{S} = \langle \mathsf{S}, s_0, \mathsf{G}, \mathsf{A}, P, C \rangle$ into a deterministic problem $\mathbb{D} = \langle \mathsf{S}, s_0, \mathsf{G}, \mathsf{A}' \rangle$. The set A' contains only deterministic actions represented as $a = s \to s'$, i.e., a deterministically transforms s into s'. Two common determinization techniques are:

```
1  RTDP(SSP 𝕊 = ⟨S, s_0, G, A, P, C⟩, H a heuristic for V*, 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 ← RTDP-TRIAL(𝕊, V)
6      return V

7  RTDP-TRIAL(SSP 𝕊 = ⟨S, s_0, G, A, P, C⟩, value function V)
8  begin
9      s ← s_0
10     while s ∉ G do
11         a ← π^V(s)
12         V(s) ← (BV)(s)
13         s ← APPLY-ACTION(a,s)
14     return V
```

Algorithm 2.1: Real-Time Dynamic Programming (RTDP) [Barto et al., 1995].

- *most-likely outcome*, in which $\mathsf{A}' = \{s \rightarrow s' | \exists a \in \mathsf{A} \text{ s.t. } s' = \text{argmax}_{\hat{s}} P(\hat{s}|s,a)\}$ (breaking ties randomly); and
- *all-outcomes*, where $\mathsf{A}' = \{s \rightarrow s' | \exists a \in \mathsf{A} \text{ s.t. } P(s'|s,a) > 0\}$.

The idea behind FF-Replan (Algorithm 2.2), is simple and powerful: relax the probabilistic problem into a deterministic problem $\mathbb{D}$ (Line 7) and use the deterministic planner FF [Hoffmann and Nebel, 2001] to solve $\mathbb{D}$ (Line 8). FF-Replan stores the obtained solution for $\mathbb{D}$ in the policy π (Line 10) and there is no guarantee that π is a closed policy for the original SSP $\mathbb{S}$, that is, π might be a partial policy for $\mathbb{S}$. The policy π is followed until failure (Line 11), i.e., π is not defined for the current state s; if and when π fails, FF is re-invoked to plan again from the failed state.

An earlier version of FF-Replan employed the most-likely outcome determinization [Yoon et al., 2007]; however this approach is not complete since the goal might not reachable in the most-likely determinization of $\mathbb{S}$ even when Assumption 2.1 holds for $\mathbb{S}$. Alternatively, if the all-outcomes determinization is used, then FF-Replan is complete when Assumption 2.1 holds, i.e., FF-Replan always reaches a goal state. In this dissertation, we consider the most recent version of FF-Replan, i.e., when the all-outcomes determinization is used.

FF-Replan is the winner of the first International Probabilistic Planning Competition (IPPC) [Younes et al., 2005] in which it outperformed the probabilistic planners due to their poor scalability. In general, FF-Replan can quickly reach a goal state and scales up to large problems when

```
1  FF-REPLAN(SSP 𝕊 = ⟨S, s_0, G, A, P, C⟩)
2  begin
3      π ← empty-policy
4      s ← s_0
5      while s ∉ G do
6          if π is not defined for s then
7              𝔻 ← ALL-OUTCOMES-DETERMINIZATION(𝕊)
8              (s_1, a_1, s_2, ..., a_{k-1}, s_k) ← FF(𝔻, s)
9              foreach i ∈ {1, ..., k − 1} do
10                 π(s_i) ← a_i
11         s ← APPLY-ACTION(π(s),s)
```

Algorithm 2.2: FF-Replan [Yoon et al., 2007]. On Line 8, the deterministic planner FF [Hoffmann and Nebel, 2001] is called to compute a sequence of states and actions starting from the current state s that reaches a goal state $s_k \in \mathsf{G}$. Different determinization approaches and deterministic planners can be used in Lines 7 and 8, respectively.

Assumption 2.1 holds. Despite its major success, FF-Replan is non-optimal and oblivious to probabilities and dead ends, leading to high-cost solutions and poor performance in probabilistic interesting problems [Little and Thiébaux, 2007], e.g., the triangle tire-domain.

2.4 Summary

In this chapter we described Stochastic Shortest Path Problems (SSPs), the framework used in this dissertation to represent probabilistic planning problems. We also presented the main definitions and results regarding the solutions of SSPs that are necessary for our proofs in this dissertation. We described how to compactly represents SSPs through the factored representation and the PPDDL language, a standard language from the planning community to represent probabilistic planning problems. Finally, we reviewed two main algorithms to solve SSPs: RTDP, an optimal probabilistic planner that returns closed policies; and FF-Replan, a replanner based on determinizations. In the next chapter, we exam how to combine the main features of both RTDP and FF-Replan, i.e., optimally and scalability.

Chapter 3

Short-Sighted Probabilistic Planning

In this chapter, we present the two main concepts for short-sighted probabilistic planning, i.e., how to generate short-sighted problems and how to plan using short-sighted problems [Trevizan and Veloso, 2012a, Trevizan and Veloso, 2013]. We begin by comparing RTDP and FF-Replan to motivate the definition of short-sighted problems. We then formally define the concept of short-sighted problems in Section 3.2 and prove its properties with respect to the original probabilistic planning problem in Section 3.2.1. In Section 3.3, we present the *Short-Sighted Probabilistic Planner* algorithm that solves probabilistic planning problems using the short-sighted problems defined previously. The properties of this algorithm, e.g., optimality, are proven in Section 3.3.1. We empirically demonstrate the benefits of *Short-Sighted Probabilistic Planner* against RTDP and FF-Replan in Section 3.4 using a proposed series of increasingly larger problems.

3.1 Motivation

In order to motivate the introduction of *short-sighted problems*, consider the problem of building a domino line. Precisely, given 3 dominoes, the goal is to build a straight line using all the 3 dominoes. A domino can be placed at position $l \in \{0, 1, 2\}$ of the line if l is empty, and, with probability 0.9, the new domino falls and drops all the other dominoes already in the line. The cost of action `place` is 1 independently of its outcome. Also, the special action `delegate`, which delegates the construction of the domino line to a more reliable agent, is available when the line is empty. `Delegate` deterministically builds the complete line of 3 dominoes at a cost of 9. Figure 3.1 depicts this problem.

Due to the all-outcomes determinization (Section 2.3.2), FF-Replan considers that it is possible to deterministically place each one of the dominoes and this relaxed action costs 1, since the original action `place` has cost 1. Therefore FF-Replan solves this problem by building the

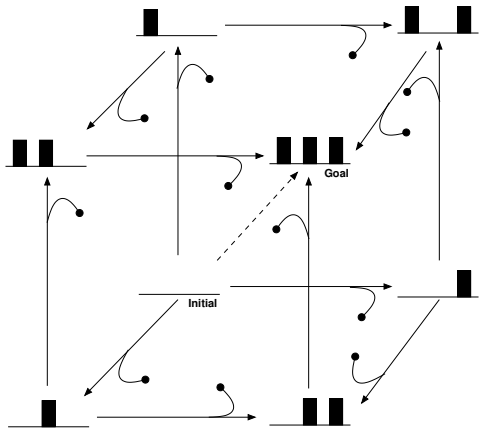


Figure 3.1: Domino line problem for $n = 3$. The initial state is the empty line and the goal is to build a line of 3 dominoes. The full-line arrows depict the action `place` that succeed with probability 0.1, otherwise (probability 0.9) all dominoes in the line are dropped (for ease of presentation, this side-effect is depicted as a ball-ended line). The action `delegate` is shown as a dashed arrow.

domino line piece-by-piece at a total relaxed cost of 3. However, in the original problem, the expected cost of this solution is 1110 (the formal analysis of the expected cost is provided in Section 3.4).

Alternatively, RTDP samples several trajectories from the initial state (empty dominoes line) to the goal (3-dominoes line). Initially, these sampled trajectories contains only the action `place`, since it costs 1 and `delegate` costs 9. After a large amount of samples, RTDP learns that building the dominoes line piece-by-piece is more expensive on expectation than using action `delegate`, i.e., expected cost of 1110 versus constant cost of 9, and selects `delegate`, the optimal solution for this problem.

Notice that RTDP is forced to explore the whole state space, i.e., all the combinations of 1, 2 and 3 dominoes placements before inferring that `delegate` is the optimal action. However, the expected cost of successfully placing the first domino is already larger than the cost of `delegate`. Precisely, the expected cost c of successfully placing the first domino is $c = 1 + 0.9c = 10$. Therefore, if we divide this problem of building a 3-dominoes lines into 3 subproblems, namely, building a line of 1 domino, then a line of 2, and finally a line of 3 dominoes, we would be able to infer that `delegate` is the optimal solution after solving only first subproblem, i.e., building a line of 1 domino.

In the remainder of this chapter, we introduce *short-sighted problems*, a novel definition of subproblems of probabilistic planning problems in which actions are not simplified, therefore the expected cost of each subproblem can be computed. We then show how to use *short-sighted*

problems in order to efficiently solve probabilistic planning problems. We also revisit the domino example in Section 3.4 to show the trade-offs of short-sighted planning.

3.2 Short-Sighted Stochastic Shortest Path Problems

In this section, we define depth-based short-sighted Stochastic Shortest Path Problems, a special case of Stochastic Shortest Path Problems (SSPs) in which the original problem is transformed into a smaller one by:

- pruning the states that have a zero probability of being reached using at most t actions;
- adding artificial goal states; and
- incrementing the cost of reaching artificial goals by a heuristic value in order to guide the search towards the goals of the original problem.

Throughout this chapter, we refer to depth-based short-sighted Stochastic Shortest Path Problems as short-sighted SSPs and before formally introduce them, we need to define the action-distance between states:

Definition 3.1 ($\delta(s, s')$)**.** The non-symmetric distance $\delta(s, s')$ between two states s and s' is:

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

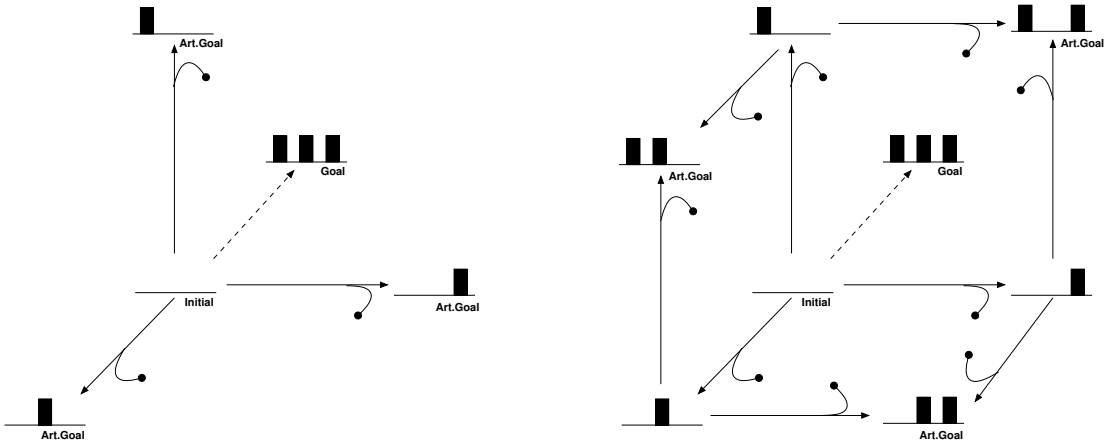
$\delta(s, s')$ is equivalent to the minimum number of actions necessary to reach s' from s in the all-outcomes determinization.

Using the action-distance function δ (Definition 3.1), the short-sighted SSP associated to an SSP is defined as:

Definition 3.2 (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:

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

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



(a) (b)

Figure 3.2: Example of (s,t)-depth-based short-sighted SSPs for the 3-line dominoes problem (Figure 3.1). For both (a) and (b) the parameter s equals the initial state (no dominoes) and t equals to 1 and 2 for (a) and (b) respectively. The action `delegate` is shown as a dashed arrow and ball-ended lines represent the side-effect of `place` in which all dominoes pieces are dropped, i.e., transition to the initial state.

Figure 3.2 shows the $(s_0, 1)$ and $(s_0, 2)$-short-sighted SSP associated with the 3-dominoes line example (Figure 3.1) where s_0 represents the initial state (i.e., no dominoes). 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 feature 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 towards the goal states of the original SSP, even if such states are not in $\mathsf{S}_{s,t}$.

3.2.1 Properties

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 (2.2). 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 (Section 2.1 p.12), are not the same. Before we formally prove their differences, consider the 3-dominoes line problem depicted in Figure 3.1, depth $t = 2$, and the zero-heuristic as H:

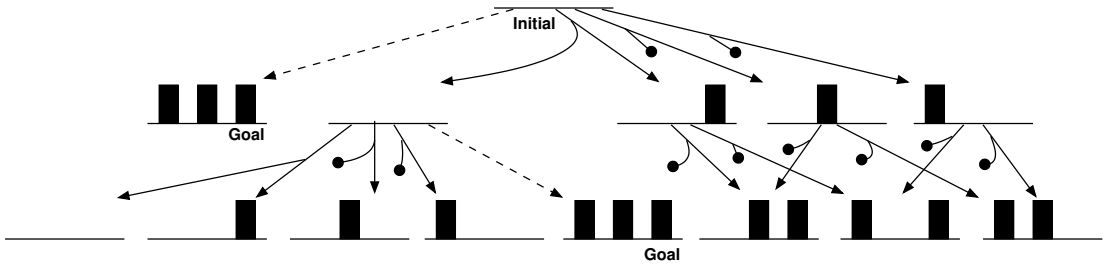


Figure 3.3: Example of look-ahead search tree for the 3-line dominoes problem (Figure 3.1). In this example, the root of the search tree is the initial state s_0 and the depth is $t = 2$. Ball-ended lines represent a transition to the state in which the line is empty in the next level of the tree; this transition happens with probability 0.9.

- The 2-look-ahead search from s_0, $(B^2H)(s_0)$, represents the minimum expected cost of executing 2 actions in a row, therefore only trajectories of size 2 are considered. Figure 3.3 shows the search tree associated with $(B^2H)(s_0)$. The resulting value is $(B^2H)(s_0) = 2$ that is obtained by applying any sequence of two `place` actions, since `delegate` has cost 9.
- 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 $\mathbb{S}_{s_0,2}$ (Figure 3.2(b)), i.e., a state in $\mathsf{G}_{s_0,2}$, from s_0. Thus all possible trajectories in $\mathbb{S}_{s_0,2}$ are considered and the maximum size of these trajectories is unbounded due to the loops generated by the policy in which the action `place` is applied. In this example, $V^*_{\mathbb{S}_{s_0,2}}(s_0) = 9$ and the closed greedy policy w.r.t. $V^*_{\mathbb{S}_{s_0,2}}$ is to apply `delegate` in the initial state.

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.

In order to formally prove the relation between $V^*_{\mathbb{S}_{s,t}}(s)$ and $(B^tH)(s)$, 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,

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

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 since 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 3.1

and 3.2 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 3.1. *Given an SSP $\mathbb{S} = \langle \mathsf{S}, s_0, \mathsf{G}, \mathsf{A}, P, C\rangle$ that satisfies the Assumption 2.1, $s \in \mathsf{S}$, $t \in \mathbb{N}^*$ and a monotonic value function V for $\mathbb{S}$, then $(B^k_{s,t}V)(\hat{s}) = (B^kV)(\hat{s})$ 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 $\mathbb{S}_{s,t}$.*

Proof. See Appendix. □

Lemma 3.2. *Under the same conditions of Lemma 3.1, $(B^k_{s,t}V)(s) \leq (B^kV)(s)$ 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 $\mathbb{S}_{s,t}$.*

Proof. See Appendix. □

In Theorem 3.3, we prove that $V^*_{\mathbb{S}_{s,t}}(s) \leq V^*(s)$ and that $V^*_{\mathbb{S}_{s,t}}(s)$ is a lower bound for $V^*(s)$ at least as tight as $(B^tH)(s)$ if H is a monotonic lower bound on V^* and Assumption 2.1 holds for $\mathbb{S}$. Corollary 3.4 shows that $V^*_{\mathbb{S}_{s,t}}(s)$ is always a tighter lower bound than $(B^tH)(s)$ if $\mathbb{S}$ has unavoidable loops (Definition 3.3).

Theorem 3.3. *Given an SSP $\mathbb{S} = \langle \mathsf{S}, s_0, \mathsf{G}, \mathsf{A}, P, C\rangle$ that satisfies the Assumption 2.1, $s \in \mathsf{S}$, $t \in \mathbb{N}^*$ and a monotonic lower bound H for V^*, then*

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

Proof. By the definition of $\mathbb{S}_{s,t}$, $\min_{s_a \in \mathsf{G}_a} \delta(s, s_a) = t$. Therefore $(B^tH)(s) = (B^t_{s,t}H)(s)$ by Lemma 3.1. Since H is a monotonic lower bound and $V^*_{\mathbb{S}_{s,t}}(s) = (\lim_{k\to\infty} B^k_{s,t}H)(s)$, we have that $(B^tH)(s) \leq V^*_{\mathbb{S}_{s,t}}(s)$. By Lemma 3.2, we have that $V^*_{\mathbb{S}_{s,t}}(s) \leq V^*(s)$.

□

Definition 3.3 (Unavoidable Loops)**.** An SSP $\mathbb{S} = \langle \mathsf{S}, s_0, \mathsf{G}, \mathsf{A}, P, C\rangle$ that satisfies the Assumption 2.1 has unavoidable loops if, for every optimal policy π^* of $\mathbb{S}$, the directed graph $G = (\mathsf{S}^{\pi^*}, E)$, where $E = \{(s, s') | P(s'|s, \pi^*(s)) > 0\}$, is not acyclic.

Corollary 3.4. *In Theorem 3.3, if the (s,t)-short-sighted SSP $\mathbb{S}_{s,t}$ has unavoidable loops (Definition 3.3), then $(B^tH)(s) < V^*_{\mathbb{S}_{s,t}}(s)$.*

Proof. By definition, $(B^tH)(s)$ consider only trajectories of size at most t from s. By definition, $V^*_{\mathbb{S}_{s,t}}(s) = \lim_{k\to\infty}(B^k_{s,t}H)(s)$, 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, therefore the maximum size of a trajectory generated

by $\pi^*_{s,t}$ is unbounded. Since every trajectory has non-zero probability and non-zero cost, then $(B^t H)(s) = (B^t_{s,t} H)(s) < V^*_{\mathbb{S}_{s,t}}(s)$. □

Another important relation 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, independently of the probabilistic outcome of actions, for at least t actions without replanning:

Definition 3.4 (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 $s' \in \mathsf{R}^\pi \cap \mathsf{S}^\pi$, $\delta(s, s') \geq t$.

FF-Replan and its extensions (see Chapter 6) compute 1-closed policies w.r.t. the current state, i.e., there is no guarantee that 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 3.5 gives an upper bound on t for when a t-closed policy w.r.t. s_0 becomes a closed policy.

Proposition 3.5. *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 $s' \in \mathsf{R}^\pi \cap \mathsf{S}^\pi$, $\delta(s, s') \geq |\mathsf{S}|$. By the definition of S^π, we have that all $s' \in \mathsf{S}^\pi$ is reachable from s_0 when following π. Thus $\delta(s_0, s') < |\mathsf{S}|$, since there exist 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 $\mathsf{R}^\pi \cap \mathsf{S}^\pi = \emptyset$, i.e., π is a closed policy, since there exists no $s' \in \mathsf{S}^\pi$ such that $\delta(s, s') \geq |\mathsf{S}|$. □

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:

Proposition 3.6. *Given an SSP $\mathbb{S} = \langle \mathsf{S}, s_0, \mathsf{G}, \mathsf{A}, P, C \rangle$ and a state $s \in \mathsf{S}$, 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 $\delta(s, s') < t$. Since $\delta(s, s') < t$, 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 $\mathbb{S}_{s,t}$, we have that, for all $s' \in \mathsf{S}_{s,t}$, $\delta(s, \hat{s}) \leq t$. Thus, if $s' \in (\mathsf{R}^\pi \cap \mathsf{S}^\pi) \cap \mathsf{S}_{s,t}$, then $\delta(s, s') = t$, i.e., $s' \in \mathsf{G}_{s,t} \setminus \mathsf{G}$. Since, by the definition of R^π, $\mathsf{R}^\pi_{s,t} \cap \mathsf{G}_{s,t} = \emptyset$ 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 a closed for $\mathbb{S}_{s,t}$

```
1 NON-LEARNING-PLANNER(SSP 𝕊 = ⟨S, s_0, G, A, P, C⟩, t ∈ ℕ*, H a heuristic for V*)
2 begin
3     s ← s_0
4     while s ∉ G do
5         𝕊_{s,t} ← GENERATE-SHORT-SIGHTED-SSP(𝕊, s, H, t)
6         π_{𝕊_{s,t}} ← SSP-SOLVER(𝕊_{s,t})
7         while s ∉ G_{s,t} do
8             s ← APPLY-ACTION(π_{𝕊_{s,t}}(s),s)
```

Algorithm 3.1: Non-learning algorithm to solve SSPs using short-sighted SSPs. Any probabilistic planner can be used as SSP-SOLVER, e.g., value iteration, FF-Replan, and RTDP.

□

3.3 Short-Sighted Probabilistic Planner

We present a step towards the definition of our main probabilistic planner by describing its basic non-learning version. This algorithm, NON-LEARNING-PLANNER (Algorithm 3.1), is the straightforward adaptation of Proposition 3.6: a short-sighted SSP is generated, solved and its solution is applied in the original SSP (Lines 5 to 8); if an artificial goal is reached, then the procedure is repeated.

NON-LEARNING-PLANNER makes no assumption about the algorithm used as SSP-SOLVER (Line 6) and its behavior is highly dependant on the chosen algorithm to solve each short-sighted SSP. For instance, consider the 3-dominoes line problem (Figure 3.1), FF-Replan as SSP-SOLVER and H_0 as heuristic, then, independently of the value of t and the current state s, the solution returned by FF-Replan to $\mathbb{S}_{s,t}$ is always a sequence of `place` actions. Therefore, NON-LEARNING-PLANNER using FF-Replan is unable to find the optimal solution for 3-dominoes line problem.

In order to illustrate the need for an algorithm that *learns*, i.e., improves the given heuristic as execution (or simulation) is performed, consider the 3-dominoes line problem with the cost of `delegate` changed from 9 to 11. If NON-LEARNING-PLANNER, using RTDP as SSP-SOLVER, $t = 1$ and H_0 as heuristic, is applied to this modification of the 3-dominoes line problem, then after $\mathbb{S}_{s_0,1}$ (Figure 3.2(a)) is solved, we have that $V^*_{\mathbb{S}_{s_0,1}}(s_0) = 10$ and a `place` action is chosen. Every time that the initial state s_0 is revisited, a high probability event since every `place` action results in s_0 with probability 0.9, the same bound is computed by RTDP, i.e., RTDP is reinvoked to solve $\mathbb{S}_{s_0,1}$ generated using H_0 as heuristic. Therefore, NON-LEARNING-PLANNER is unable to infer that `delegate` is the optimal solution and always chooses a `place` action on s_0.

1 SSiPP(SSP $\mathbb{S} = \langle \mathsf{S}, s_0, \mathsf{G}, \mathsf{A}, P, C \rangle$, $t \in \mathbb{N}^*$, H a heuristic for V^*, $\epsilon > 0$)
2 **begin**
3 $\quad \underline{V} \leftarrow$ Value function for S with default value given by H
4 $\quad s \leftarrow s_0$
5 $\quad$ **while** $s \notin \mathsf{G}$ **do**
6 $\quad\quad \mathbb{S}_{s,t} \leftarrow$ GENERATE-SHORT-SIGHTED-SSP($\mathbb{S}, s, \underline{V}, t$)
7 $\quad\quad (\pi^*_{\mathbb{S}_{s,t}}, V^*_{\mathbb{S}_{s,t}}) \leftarrow \epsilon$-OPTIMAL-SSP-SOLVER($\mathbb{S}_{s,t}, \underline{V}, \epsilon$)
8 $\quad\quad$ **foreach** $s' \in \mathsf{S}^{\pi^*_{\mathbb{S}_{s,t}}} \setminus \mathsf{G}_{s,t}$ **do**
9 $\quad\quad\quad \underline{V}(s') \leftarrow V^*_{\mathbb{S}_{s,t}}(s')$
10 $\quad\quad$ **while** $s \notin \mathsf{G}_{s,t}$ **do**
11 $\quad\quad\quad s \leftarrow$ APPLY-ACTION($\pi^*_{\mathbb{S}_{s,t}}(s)$,$s$)
12 $\quad$ **return** $\underline{V}$

Algorithm 3.2: Short-Sighted Probabilistic Planner (SSiPP). Any SSP ϵ-optimal solver can be used as ϵ-OPTIMAL-SSP-SOLVER, e.g., value iteration and RTDP. Notice that $V^*_{\mathbb{S}_{s,t}}$ returned by ϵ-OPTIMAL-SSP-SOLVER needs to be defined only for the states reachable from s when following $\pi^*_{\mathbb{S}_{s,t}}$, i.e., for $s' \in \mathsf{S}^{\pi^*_{\mathbb{S}_{s,t}}}$.

Short-Sighted Probabilistic Planner (SSiPP), presented in Algorithm 3.2, overcomes the drawbacks of NON-LEARNING-PLANNER by maintaining a lower bound $\underline{V}$ for V^* that is updated according to the optimal solution of the generated short-sighted SSPs (Lines 8 and 9). [1] The lower bound $\underline{V}$ is initialized by the input heuristic H (Line 3) using a lazy approach, i.e., if the value $\underline{V}(s)$ is requested and $\underline{V}$ is not defined on s, then $H(s)$ is computed (on demand) and assigned to $\underline{V}(s)$.

Due to the reduced state space of short-sighted SSPs, it possible to compute the ϵ-optimal solution of each $\mathbb{S}_{s,t}$ efficiently (Line 7) and the obtained policy $\pi^*_{\mathbb{S}_{s,t}}$ is a t-closed policy w.r.t. the current state s for original SSP $\mathbb{S}$ (Proposition 3.6). 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, let us revisit the modified 3-dominoes lines problem in which `delegate` has cost 11. For this example, consider SSiPP using RTDP as OPTIMAL-SOLVER, $t = 1$ and H_0 as heuristic, i.e., initially $\underline{V}(s) = 0$ for all $s \in \mathsf{S}$ (Line 3). The first short-sighted SSP generated and solved is $\mathbb{S}_{s_0,1}$ (Figure 3.2(a)) and, after Line 8 is executed, we have that $\underline{V}(s_0) = 10$. Since `delegate` costs 11, a `place` action is chosen and applied until the current state s changes from s_0 to a state $s' \neq s_0$. Denote this chosen action as a. Once s' is reached, $\mathbb{S}_{s',1}$ is generated, solved and $\underline{V}(s')$ is updated to a value greater than zero since s' is not a goal state of

[1] SSiPP is pronounced as the word "sip."

1 RUN-SSIPP-UNTIL-CONVERGENCE(SSP $\mathbb{S} = \langle \mathsf{S}, s_0, \mathsf{G}, \mathsf{A}, P, C \rangle$, $t \in \mathbb{N}^*$, H a heuristic for V^*, $\epsilon > 0$)
2 **begin**
3 $\quad \underline{V} \leftarrow$ Value function for S with default value given by H
4 $\quad$ **while** $R(\mathbb{S}, \underline{V}) > \epsilon$ **do**
5 $\quad\quad \underline{V} \leftarrow$ SSIPP($\mathbb{S}$, t,$\underline{V}$, ϵ)
6 $\quad$ **return** $\underline{V}$

Algorithm 3.3: Algorithm to compute an ϵ-approximation of V^* using SSiPP (Algorithm 3.2).

the original problem. When the state s_0 is revisited for the first time, the expected cost of applying a in $\mathbb{S}_{s_0,1}$ using $\underline{V}$ as heuristic equals $0.9(1 + \underline{V}(s_0)) + 0.1(1 + \underline{V}(s')) = 10 + 0.1\underline{V}(s') > 10$ since $\underline{V}(s') > 0$. Therefore action a is not chosen since the expected cost of applying any of the remaining two `place` actions in s_0 is 10. As we prove in the next section, this process continues and the optimal solution is found.

3.3.1 Guarantees

In this section, we prove that: SSiPP performs Bellman backups (Theorem 3.7); SSiPP terminates (Theorem 3.8); and Algorithm 3.3 is asymptotically optimal (Theorem 3.9), that is, if the same problem is solved sufficiently many times by SSiPP, then the optimal policy is found.

Theorem 3.7. *Given an SSP $\mathbb{S} = \langle \mathsf{S}, s_0, \mathsf{G}, \mathsf{A}, P, C \rangle$ such that the Assumption 2.1 holds, and a monotonic lower bound H for V^*, then the loop in Line 8 of SSiPP (Algorithm 3.2) 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, we have that, for all $s' \in \hat{\mathsf{S}}$, $\underline{V}(s')$ equals $V^*_{\mathbb{S}_{s,t}}(s')$. 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 2.1). By the definition of short-sighted SSP (Definition 3.2), 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 3.8. *SSiPP always terminates under the same conditions of Theorem 3.7.*

Proof. Suppose SSiPP does not terminate. Then, there exists a trajectory $\mathcal{T}$ of infinite size that can be generated by SSiPP. Since S is finite, then there must be an infinite loop in $\mathcal{T}$ and, for all states s in this loop, $\underline{V}(s)$ diverges as the execution continues. Because Assumption 2.1 holds

for $\mathbb{S}$, we have that $V^*(s) < \infty$ for all $s \in \mathsf{S}$. A contradiction since SSiPP maintains $\underline{V}$, initialized as H, admissible and monotonic (Theorems 3.3 and 3.7), i.e., $\underline{V}(s) \leq V^*(s)$ for all $s \in \mathsf{S}$. □

Theorem 3.9. *Given an SSP $\mathbb{S} = \langle \mathsf{S}, s_0, \mathsf{G}, \mathsf{A}, P, C \rangle$ such that the Assumption 2.1 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 $\underline{V}^i = SSiPP(\mathbb{S}, t, \underline{V}^{i-1})$, converges to V^* as $k \to \infty$ for all $s \in \mathsf{S}^{\pi^*}$.*

Proof. Let the sequence of states $\mathcal{H} = \langle s_0, s_1, s_2, \ldots \rangle$ be the concatenation of the trajectories $\mathcal{T}_i$ of states visited by SSiPP when $\underline{V}^i$ is computed. By Theorem 3.8, $\mathcal{T}_i$ has finite size, therefore $|\mathcal{H}|$ is finite. Since Assumption 2.1 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 [Barto et al., 1995, Theorem 3, p. 132]: $\mathsf{S}_\infty \subseteq \mathsf{S}$ is the non-empty set of states that appear infinitely often in $\mathcal{H}$; $\mathsf{G}_\infty \subseteq \mathsf{G}$ is the non-empty set of goal states that appear infinitely often in $\mathcal{H}$; 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 3.7, 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 $V^*(s)$ for all $s \in \mathsf{S}_\infty$ as $k \to \infty$ [Bertsekas and Tsitsiklis, 1996, Proposition 2.2, p. 27]. Furthermore, $\mathsf{S}^{\pi^*} \subseteq \mathsf{S}_\infty$ [Barto et al., 1995, Theorem 3]. □

3.4 The n-Dominoes Line Problem

In this section, we generalize the 3-dominoes line problem to any number of dominoes (Example 3.1). The obtained series of problems, the n-dominoes line problems, has characteristics that illustrate the benefits of short-sighted planning as the the parameters of the problem varies [Veloso and Blythe, 1994]. Precisely, we illustrate the trade-offs of short-sighted planning by analyzing how the cost of `delegate` and the failure probability of the actions `place` influence the solutions for the n-dominoes line problems. Then we present an empirical comparison between RTDP, FF-Replan and SSiPP in the n-dominoes line problems for different parameters.

Example 3.1 (n-dominoes line)**.** Informally, given n dominoes, the goal of this problem is to build a line using all the dominoes. The actions `place`(i), for $i \in \{0, \ldots, n-1\}$, represent placing a domino in the position i of the line being built. Every action `place`(i) has cost 1 and can fail with probability $1 - p$, in which case all the dominoes already placed are dropped and the line needs to be rebuild from scratch. If the domino line is empty, the action `delegate` can be applied. `Delegate` costs k and deterministically builds the n-dominoes line.

action: a_i
pre: $\neg l_i$
with probability p:
  add: l_i
  del: $\emptyset$
with probability $1-p$:
  add: $\emptyset$
  del: $l_0, \cdots, l_{n-1}$
cost: 1

(a)

action: d
pre: $\neg l_0, \cdots, \neg l_{n-1}$
add: $l_0, \cdots, l_{n-1}$
del: $\emptyset$
cost: k

(b)

Figure 3.4: Definition of the actions in the n-dominoes line problems. Actions a_i (place(i)) and d (delegate) are presented using probabilistic STRIPS in (a) and (b) respectively.

Formally, we represent the domino line using the binary state variables $l_0, \cdots, l_{n-1}$ where l_i is true if there is a domino at position i of the line. We denote the actions place(i) by a_i and delegate by d. Figure 3.4 shows the formal definition of a_i and d. In the initial state s_0, all the state variables are false and the goal set G equals $\{s_G\}$ where s_G is the state in which all state variables are true.

The n-dominoes line problem has $n! + 1$ closed policies: π_d, that selects action d on s_0; and the $n!$ policies representing the permutations of $\pi_a = (a_0, a_1, \cdots, a_{n-1})$, where a_{i+1} is applied when a_i succeeds, i.e., results in a state $s \neq s_0$. Notice that every permutation of π_a results in the same overall policy in which the dominoes line is built one piece at the time. Since every action a_i has the same probability p of succeeding, in which case l_i is changed from false to true, same probability of returning to the initial state and same cost, then every permutation of π_a has the same expected cost $V^{\pi_a}(s_0)$ to reach the goal from the initial state.

In order to compute $V^{\pi_a}(s_0)$, consider the recurrence $T(i)$ which represents the expected cost of solving the problem of size n by using π_a when there are only i dominoes missing in the line. Clearly, $T(0) = 0$ since the dominoes line is done when no domino is missing in the line. Moreover, we have $V^{\pi_a}(s_0) = T(n)$, because the domino line is empty in the initial state. Since the domino line is constructed by adding one domino at each time step, then, for $i \in \{1, \cdots, n\}$, we have that $T(i) = 1 + (1-p)T(n) + pT(i-1)$. Let $c_{x:y}$ denote $\sum_{j=x}^{y} p^j$. By unrolling $T(i)$, we get $T(i) = c_{0:i-1} + c_{0:i-1}(1-p)T(n)$ for $i \in \{1, \cdots, n\}$, therefore

$$V^{\pi_a}(s_0) = T(n) = c_{0:n-1} + c_{0:n-1}(1-p)V^{\pi_a}(s_0) = \frac{c_{0:n-1}}{p^n} = \sum_{j=0}^{n-1} p^{j-n}.$$

Since d is deterministic and has cost k, we have that $V^{\pi_d}(s_0) = k$, thus π_d is the only optimal policy for the n-dominoes line problem when $k < V^{\pi_a}(s_0) = \sum_{j=0}^{n-1} p^{j-n}$.

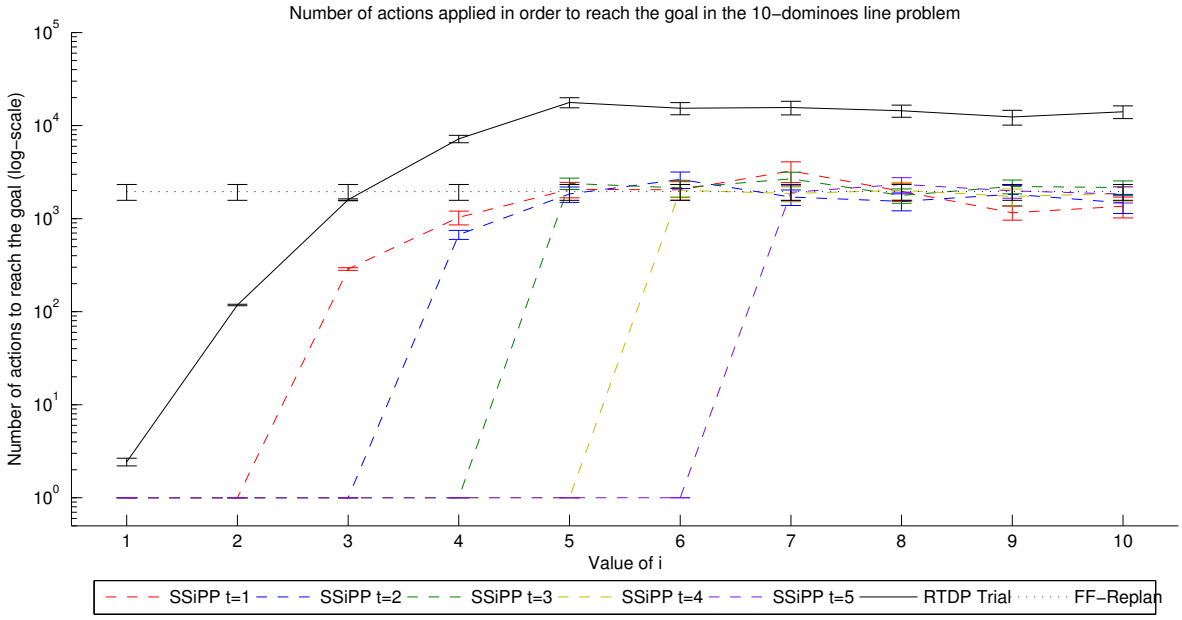


Figure 3.5: Average and 95% confidence interval for the number of actions to reach the goal of the 10-dominoes line problem. For this experiment, 100 samples were used and the parameter p of the dominoes line problem equals 0.5 Given a value of i in the x-axis, the cost of `delegate` equals $\sum_{j=0}^{i-1} 0.5^{j-i} - 1$, i.e., the expected cost of building a line of i dominoes decreased by 1. FF-Replan performance is constant because it always apply the same policy regardless the cost of `delegate`.

To demonstrate the trade-offs of SSiPP, consider the expected cost $V^{\pi_a}_{\mathbb{S}_{s_0,t}}(s_0)$ of π_a applied in the first short-sighted SSP solved by SSiPP, i.e., $\mathbb{S}_{s_0,t}$ using H_0 as heuristic. If $t \geq n$, then $\mathbb{S}_{s_0,t}$ equals the original problem, because all the states can be reached using at most n actions; thus $V^{\pi_a}_{\mathbb{S}_{s_0,t}}(s_0) = V^{\pi_a}(s_0)$. For $t < n$, every artificial goal of $\mathbb{S}_{s_0,t}$ represents a line of t dominoes and $V^{\pi_a}_{\mathbb{S}_{s_0,t}}(s_0) = T(t) = \sum_{j=0}^{t-1} p^{j-t}$ since $H_0(s) = 0$ for all $s \in \mathsf{G}_a$. Therefore, if $k < V^{\pi_a}_{\mathbb{S}_{s_0,t}}(s_0) = \sum_{j=0}^{t-1} p^{j-t}$, then SSiPP using the parameter t always selects π_d, which is also the optimal solution for the original problem; moreover, at most $|\mathsf{S}_{s_0,t}| = 1 + \sum_{i=0}^{t} \binom{n}{i}$ states are visited, i.e., all the states necessary to build a line of t dominoes using n dominoes plus the original goal state, while the original problem has 2^n states.

When $V^{\pi_a}_{\mathbb{S}_{s_0,t}}(s_0) \leq k < V^{\pi_a}(s_0)$, SSiPP can still infer that π_d is the optimal solution efficiently. We illustrate this case by empirically comparing RTDP-TRIAL (Algorithm 2.1 Line 7), FF-Replan and SSiPP for different values of t. Figure 3.5 shows the number of actions to reach the goal in the 10-dominoes line problem for $p = 0.5$ as a function of k. Each value of k considered equals $V^{\pi_a}_{\mathbb{S}_{s_0,i}}(s_0) - 1$, where i is the x-axis of Figure 3.5. Therefore, for $i \in \{1, \cdots, 10\}$, π_d is the optimal solution and SSiPP using $t \leq i$ always chooses π_d, i.e., solves the problem using only one action.

3.5 Summary

In this chapter, we described short-sighted Stochastic Shortest Path Problems (short-sighted SSPs), the main concept for short-sighted probabilistic planning. We then formally proved the properties of the solutions of short-sighted SSPs, in particular that, under common assumptions, it is a lower bound on the solution of the original problem. Moreover, we showed that a closed policy for an (s, t)-short-sighted SSP can be executed for at least t steps from s in the original SSP without replanning.

We also introduced Short-Sighted Probabilistic Planner (SSiPP), an algorithm that solves probabilistic planning problems by iteratively solving short-sighted SSPs and using their optimal solutions to update the lower bound on the optimal solution of the original SSP. We formally proved that, under common assumptions, SSiPP always reaches a goal state of the original problem and, if the same problem is solved sufficiently many times by SSiPP, then the optimal policy is found. Using the n-dominoes line problems introduced in this chapter, we illustrated how SSiPP is able to efficiently compute the solution of probabilistic planning problems.

In the next chapter, we extend the concept of short-sighted SSP by changing how states are pruned, e.g., we use the probability of reaching a state instead of the action-distance function δ. Chapter 5 presents extensions of SSiPP to incorporate other techniques from the probabilistic planning community, e.g., labeling of converged states and determinizations.

Chapter 4

General Short-Sighted Models

In this chapter, we extend the definition of depth-based short-sighted SSPs. We begin by introducing *trajectory-based* short-sighted SSPs, in which states that have low probability of being reached are pruned from the state space [Trevizan and Veloso, 2012b]. Next, in Section 4.2, we present *greedy* short-sighted that uses only the best k states according the current bound on V^* and their probability of being reached as state space. In Section 4.3, we prove a set of sufficient conditions under which SSiPP always terminates and is asymptotically optimal [Trevizan and Veloso, 2012b].

4.1 Trajectory-Based Short-Sighted SSPs

To motivate the definition of *trajectory-based* short-sighted SSPs, consider the SSP shown in Figure 4.1. In this example, there are two closed policies, $\pi_0 = \{(s_0, a_0), (s'_1, a_0), (s'_2, a_0), (s'_3, a_0)\}$ and $\pi_1 = \{(s_0, a_1), (s_1, a_1), (s_2, a_1), (s_3, a_1)\}$, representing, the bottom and top chains, respectively. Optimal policy π^* is π_0 because, both a_0 and a_1 have the same cost independently of their outcomes, the length of both chains is the same and a_0 has a lower self-loop probability, i.e., $P(s'_i|s'_i, a_0) < P(s_i|s_i, a_1)$.

Figure 4.2 depicts the (s_0, t)-depth-based short-sighted SSPs (Definition 3.2) associated with the example in Figure 4.1. For this example, the state space $\mathsf{S}_{s_0,t}$ of the (s_0, t)-depth-based short-sighted SSP contains t states of both chains because depth-based short-sighted SSPs ignore probabilities for the generation of $\mathsf{S}_{s_0,t}$. In the next section, we introduce *trajectory-based* short-sighted SSPs, a new class of short-sighted SSPs that prunes states based on their probability of being reached as opposed to their distance.

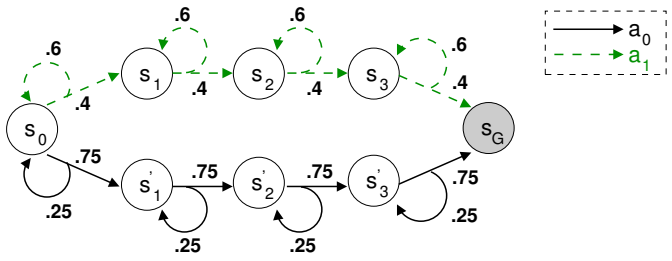
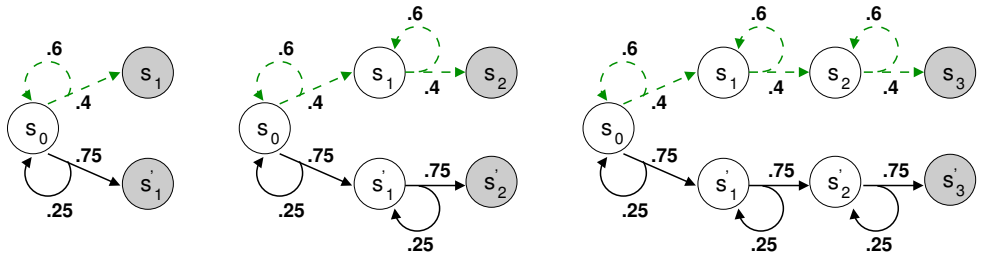


Figure 4.1: Example of SSP to motivate the definition of trajectory-based short-sighted SSPs. The initial state is s_0, the goal set is $\mathsf{G} = \{s_\mathsf{G}\}$, $C(s, a, s') = 1, \forall s \in \mathsf{S}$, $a \in \mathsf{A}$ and $s' \in \mathsf{S}$.



(a) $t = 1$ (b) $t = 2$ (c) $t = 3$

Figure 4.2: Examples of (s_0, t)-depth-based short-sighted SSPs for the SSP in Figure 4.1. For $t \geq 4$, the (s_0, t)-depth-based short-sighted SSP equals the original SSP.

4.1.1 Definition

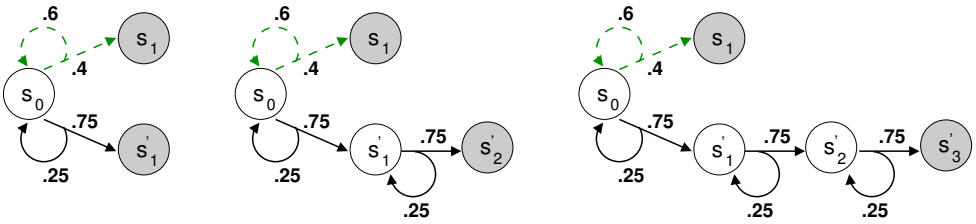
Trajectory-based short-sighted SSPs (Definition 4.2) address the issue of states with low probability of being reached by explicitly defining its state space $\mathsf{S}_{s,\rho}$ based on the maximum probability $P_{\max}(s, s')$ of a trajectory starting at s and stopping at s':

Definition 4.1 ($P_{\max}(s, s')$)**.** The maximum trajectory probability between two states s and s' is:

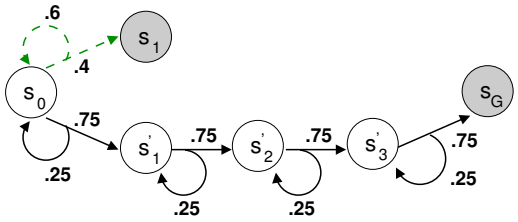
$$P_{\max}(s, s') = \begin{cases} 1 & \text{if } s = s' \\ 0 & \text{if } s \neq s' \text{ and } s \in \mathsf{G} \\ \max\limits_{\hat{s} \in \mathsf{S}, a \in \mathsf{A}} P(\hat{s}|s, a) P_{\max}(\hat{s}, s') & \text{otherwise} \end{cases}.$$

Definition 4.2 (Trajectory-Based Short-Sighted SSP)**.** Given an SSP $\mathbb{S} = \langle \mathsf{S}, s_0, \mathsf{G}, \mathsf{A}, P, C \rangle$, a state $s \in \mathsf{S}$, $\rho \in [0, 1]$ and a heuristic H, the (s, ρ)-trajectory-based short-sighted SSP $\mathbb{S}_{s,\rho} = \langle \mathsf{S}_{s,\rho}, s, \mathsf{G}_{s,\rho}, \mathsf{A}, P, C_{s,\rho} \rangle$ associated with $\mathbb{S}$ is defined as:

- $\mathsf{S}_{s,\rho} = \{s' \in \mathsf{S} | \exists \hat{s} \in \mathsf{S} \text{ and } a \in \mathsf{A} \text{ s.t. } P_{\max}(s, \hat{s}) \geq \rho \text{ and } P(s'|\hat{s}, a) > 0\}$;
- $\mathsf{G}_{s,\rho} = (\mathsf{G} \cap \mathsf{S}_{s,\rho}) \cup (\mathsf{S}_{s,\rho} \cap \{s' \in \mathsf{S} | P_{\max}(s, s') < \rho\})$;



(a) $\rho = 1.0$ (b) $\rho \in [0.75, 1.0)$ (c) $\rho \in [0.75^2, 0.75)$



(d) $\rho \in [0.75^3, 0.75^2)$

Figure 4.3: Examples of (s_0, ρ)-trajectory-based short-sighted SSPs for the SSP in Figure 4.1.

- $C_{s,\rho}(s', a, s'') = \begin{cases} C(s', a, s'') + H(s'') & \text{if } s'' \in \mathsf{G}_{s,\rho} \\ C(s', a, s'') & \text{otherwise} \end{cases}, \quad \forall s' \in \mathsf{S}_{s,\rho}, a \in \mathsf{A}, s'' \in \mathsf{S}_{s,\rho}$

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

Figure 4.3 shows, for values of $\rho \in [0.75^3, 1]$, the trajectory-based $\mathbb{S}_{s_0,\rho}$ for the SSP in Figure 4.1. For instance, if $\rho = 0.75^3$ (Figure 4.3(d)) then $\mathsf{S}_{s_0,0.75^3} = \{s_0, s_1, s'_1, s'_2, s'_3, s_G\}$ and $\mathsf{G}_{s_0,0.75} = \{s_1, s_G\}$. The case shows $\rho = 0.75^3$ how trajectory-based short-sighted SSP can be more efficient in managing uncertainty efficiently: $|\mathsf{S}_{s_0,\rho}| = 6$ and the goal of the original SSP s_G is already included in $\mathsf{S}_{s_0,\rho}$ while, for the depth-based short-sighted SSPs, $s_G \in \mathsf{S}_{s_0,t}$ only for $t \geq 4$ case in which $|\mathsf{S}_{s_0,t}| = |\mathsf{S}| = 8$.

Notice that the definition of $\mathsf{S}_{s,\rho}$ cannot be simplified to $\{\hat{s} \in \mathsf{S} | P_{\max}(s, \hat{s}) \geq \rho\}$ since not all the resulting states of actions would be included in $\mathsf{S}_{s,\rho}$. For example, consider the SSP in Figure 4.4(a); the set of states $\mathsf{S}' = \{\hat{s} \in \mathsf{S} | P_{\max}(s_0, \hat{s}) \geq \rho\} = \{s_0, s_H\}$ for all $\rho \in (0.1, 0.9]$. Therefore, if we use S' to generate a short-sighted SSP, an invalid SSP would be obtained (Figure 4.4(c)) because action a is included in the model and s_L, an effect of a with non-zero probability, is not in the state space S'.

4.1.2 Triangle Tire World

In this section, we use the *triangle tire world* [Little and Thiébaux, 2007] series of problems to show the advantage of trajectory-based short-sighted SSPs. In the triangle tire world problems,

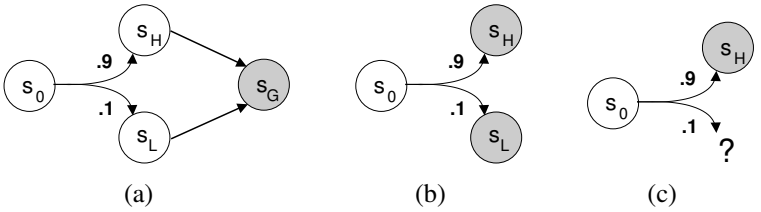


Figure 4.4: Example of why the definition of $\mathsf{S}_{s,\rho}$ cannot be simplified. (b) $(s_0, 0.8)$-trajectory-based short-sighted SSP associated with SSP in (a). (c) Ill-defined SSP obtained when $\mathsf{S}' = \{\hat{s} \in \mathsf{S} | P_{\max}(s_0, \hat{s}) \geq 0.8\} = \{s_0, s_H\}$: the state s_L is reachable however $s_L \notin \mathsf{S}'$.

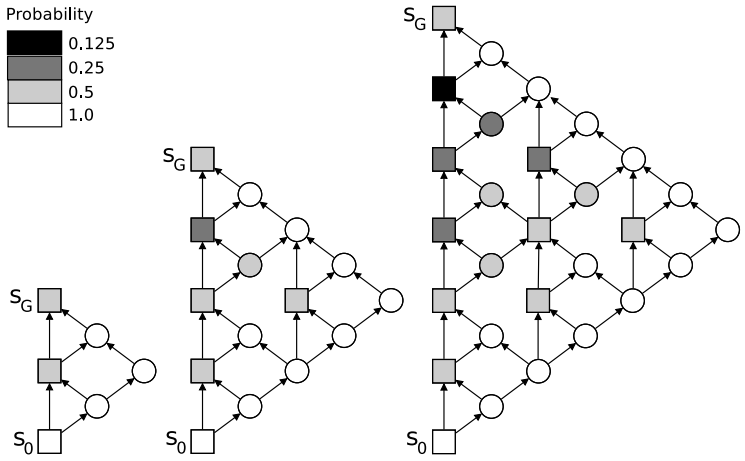
a car has to travel between locations in order to reach a goal location from its initial location. Every time the car moves between locations, a flat tire happens with probability 0.5. The car carries only one spare tire which can be used at anytime to fix a flat tire. Once the spare tire is used, a 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 a 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. Figure 4.5(a) 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:

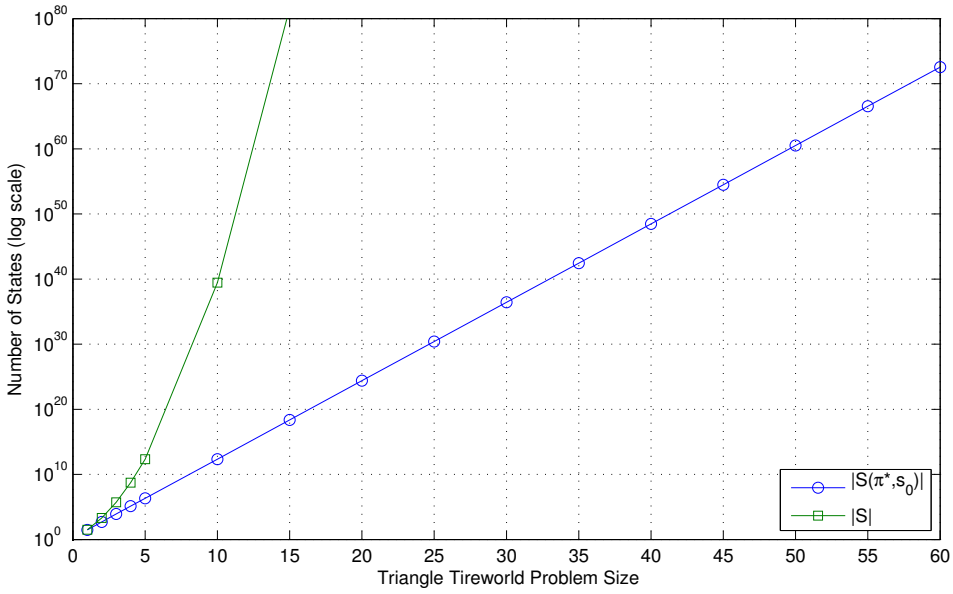
- the shortest path policy from l_0 and l_G has probability 0.5^{2n-1} of reaching the goal; and
- the only proper policy, and therefore the optimal policy, is the policy that takes the longest path.

Moreover, every triangle tire world problem is a probabilistic interesting problem [Little and Thiébaux, 2007] because only the optimal policy reaches the goal with probability 1. This property is illustrated by the shades of gray in Figure 4.5(a) that represents, for each location l, $\max_\pi P(\text{car reaches } l \text{ and the tire is not flat when following the policy } \pi \text{ from } s_0)$. Figure 4.5(b) shows the size of the state space S and $|\mathsf{S}^\pi|$, i.e., the number of states reachable from s_0 when following the optimal policy π^*, for problems up to $n = 60$.

Since the only proper policy is not complete, Assumption 2.1 does not hold for the triangle tire world problems, i.e., they contain avoidable dead ends. All dead ends of triangle tire world problems are states in which the tire is flat and there is not spare tire. Since the car cannot move when the tire is flat, these dead ends are states in which no action is available. Therefore, planners can trivially detect when a dead end s_d is reached, in which case $V(s_d)$ is updated to infinity. In



(a)



(b)

Figure 4.5: Map and state space statistics of the triangle tire world. (a) Roadmap of the triangle tire world for the sizes 1, 2 and 3. Circles (squares) represent locations in which there is one (no) spare tire. 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, $\max_\pi P(\text{car reaches } l \text{ and the tire is not flat when following the policy } \pi \text{ from } s_0)$. (b) Log-lin plot of the state space size ($|\mathsf{S}|$) and the size of the states reachable from s_0 when following the optimal policy π^* ($|\mathsf{S}^{\pi^*}|$) versus the number of the triangle tire world problem.

```
1 RUN-SSIPP(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     g ← 0
5     for i ∈ {1, ..., k} do
6         V ← SSIPP(𝕊, t, V, ε)
7         if SSiPP reached the goal then
8             g ← g + 1
9     return g
```

Algorithm 4.1: Algorithm to run SSiPP (Algorithm 3.2) k times reusing the inferred bound $\underline{V}$.

practice, the value assigned to $V(s_d)$ can be any value larger than $12n$ because $V^*(s_0) < 12n$ for the triangle tire world problem of size n.

Next, we compare SSiPP using depth-based and trajectory-based short-sighted SSPs in order to solve triangle tire world problems. Up to this point, we have not proved that SSiPP terminates (or converges) when trajectory-based short-sighted are used instead of depth-based short-sighted SSPs. In Section 4.3, we prove that SSiPP terminates and is optimal for a class of short-sighted SSPs that includes trajectory-based short-sighted SSPs.

Due to the large size of S^{π^*} (Figure 4.5(b)), it is infeasible to run SSiPP until ϵ-convergence (Algorithm 3.3). Thus, we evaluate depth-based and trajectory-based short-sighted SSPs using Algorithm 4.1 for $k = 50$, i.e., we run SSiPP 50 times reusing the inferred lower bound (Line 6). Our evaluation metric is the valued returned by Algorithm 4.1, i.e., the number of iterations of SSiPP that reached the goal. Because of the dead ends, not all executions of SSiPP might reach the goal, thus the performance of each planner is a number between 0 and 50.

Table 4.1 presents the average of 10 runs of Algorithm 4.1 for depth-based and trajectory-based short-sighted SSPs. We used the zero heuristic for both models, $t = 8$ for depth-based short-sighted SSPs, and $\rho \in \{0.125, 0.25, 0.5\}$ for trajectory-based short-sighted SSPs. For trajectory-based, we also considered an *exploration budget* approach, i.e., we fix the total number of states in $\mathsf{S}_{s,\rho}$ to be approximately the same as in the depth-based short-sighted SSP for $t = 8$ and state s. Formally, before $\mathbb{S}_{s,\rho}$ is computed in Algorithm 3.2 Line 6, we compute the state space $|\hat{\mathsf{S}}|$ of the $(s, 8)$-depth-based short-sighted SSP and choose $\rho = \text{argmax}_\rho\{|\mathsf{S}_{s,\rho}| \text{ s.t. } |\mathsf{S}_{s,\rho}| \leq |\hat{\mathsf{S}}|\}$. Since $\hat{\mathsf{S}}$ depends on the current state s, the value of ρ might differ for each $\mathbb{S}_{s,\rho}$ generated to solve a given SSP.

All the parametrizations of SSiPP using trajectory-based outperforms SSiPP using depth-based short-sighted SSPs. SSiPP using trajectory-based and $\rho \in \{0.5, 0.125\}$ is especially noteworthy because it achieves the perfect score in all problems, i.e., it reaches a goal state in all the 50 iterations in all the 10 runs for all the problems. This interesting behavior of SSiPP using

	Triangle Tireworld Problem Number											
Short-Sighted Model	5	10	15	20	25	30	35	40	45	50	55	60
Depth $t = 8$	44.6	43.3	43.1	43.3	43.7	43.7	42.9	42.5	42.1	37.8	16.3	-
Trajectory w. budget	**50.0**	**50.0**	**50.0**	**50.0**	**50.0**	**50.0**	**50.0**	**50.0**	**50.0**	**50.0**	**50.0**	**50.0**
Trajectory $\rho = 0.50$	**50.0**	**50.0**	**50.0**	**50.0**	**50.0**	**50.0**	**50.0**	**50.0**	**50.0**	**50.0**	**50.0**	**50.0**
Trajectory $\rho = 0.25$	48.6	47.3	45.4	44.6	44.6	45.1	44.1	44.9	44.2	43.9	43.8	43.4
Trajectory $\rho = 0.125$	**50.0**	**50.0**	**50.0**	**50.0**	**50.0**	**50.0**	**50.0**	**50.0**	**50.0**	**50.0**	**50.0**	**50.0**

Table 4.1: Performance comparison between depth-based and trajectory based short-sighted SSPs for the triangle tire world. Each value represents the average over 10 runs of Algorithm 4.1. For depth-based short-sighted SSPs, the parameter t equals 8; for trajectory-based short-sighted SSPs, different values of ρ and a budget approach are considered. The 95% confidence interval is less than 2.0 in all the obtained results, except for depth-based in problem 55, in which case it is 6.29. Best results shown in bold font.

trajectory-based short-sighted SSPs for the triangle tire world can be explained by the following theorem:

Theorem 4.1. *For the triangle tireworld, SSiPP using trajectory-based short-sighted SSPs and an admissible heuristic never falls in a dead-end for* $\rho \in (0.5^{i+1}, 0.5^i]$ *and* $i \in \{1, 3, 5, \dots\}$.

Proof. The optimal policy for the triangle tire world is to follow the longest path: move from the initial location l_0 to the goal location l_G passing through location l_c, where l_0, l_c and l_G are the vertices of the triangle formed by the problem's roadmap (Figure 4.5(a)). The path from l_c to l_G is unique, i.e., there is only one applicable move-car action for all the locations in this path. Therefore all the decision making to find the optimal policy happens between the locations l_0 and l_c. Each location l' in the path from l_0 to l_c has either two or three applicable move-car actions and we refer to the set of locations l' with three applicable move-car actions as N.

Every location $l' \in \mathsf{N}$ is reachable from l_0 by applying an even number of move-car actions and the three applicable move-car actions in l' are: (i) the optimal action a_c, i.e., move the car towards l_c; (ii) the action a_G that moves the car towards l_G; and (iii) the action a_p that moves the car parallel to the shortest-path from l_0 to l_G. The location reached by a_p does not have a spare tire, therefore a_p is never selected since it reaches a dead-end with probability 0.5. The locations reached by applying either a_c or a_G have a spare tire and the greedy choice between them depends on the admissible heuristic used, thus a_G might be selected instead of a_c. However, after applying a_G, only one move-car action a is available and it reaches a location that does not have a spare tire. Therefore, the greedy choice between a_c and a_G considering two or more move-car actions is optimal under any admissible heuristic: every sequence of actions $\langle a_G, a, \dots \rangle$ reaches a dead-end with probability at least 0.5 and at least one sequence of actions starting with a_c has probability 0 to reach a dead-end, e.g., the optimal solution.

Given ρ, we denote as $\mathsf{L}_{s,\rho}$ the set of all locations corresponding to states in $\mathsf{S}_{s,\rho}$ and as l_s the location corresponding to the state s. Thus, $\mathsf{L}_{s,\rho}$ contains all the locations reachable from l_s using up to $m = \lfloor \log_{0.5} \rho \rfloor + 1$ move-car actions. If m is even and $l_s \in \mathsf{N}$, then every location in $\mathsf{L}_{s,\rho} \cap \mathsf{N}$ represents a state either in $\mathsf{G}_{s,\rho}$ or at least two move-car actions away from any state in $\mathsf{G}_{s,\rho}$. Therefore the solution of the (s, ρ)-trajectory-based short-sighted SSP only chooses the action a_c to move the car. Also, since m is even, every state s used by SSiPP for generating (s, ρ)-trajectory-based short-sighted SSPs has $l_s \in \mathsf{N}$. Therefore, for even values of m, i.e., for $\rho \in (0.5^{i+1}, 0.5^i]$ and $i \in \{1, 3, 5, \dots\}$, SSiPP using trajectory-based short-sighted SSPs and ρ always chooses the actions a_c to move the car to l_c, thus avoiding all the dead-ends.

□

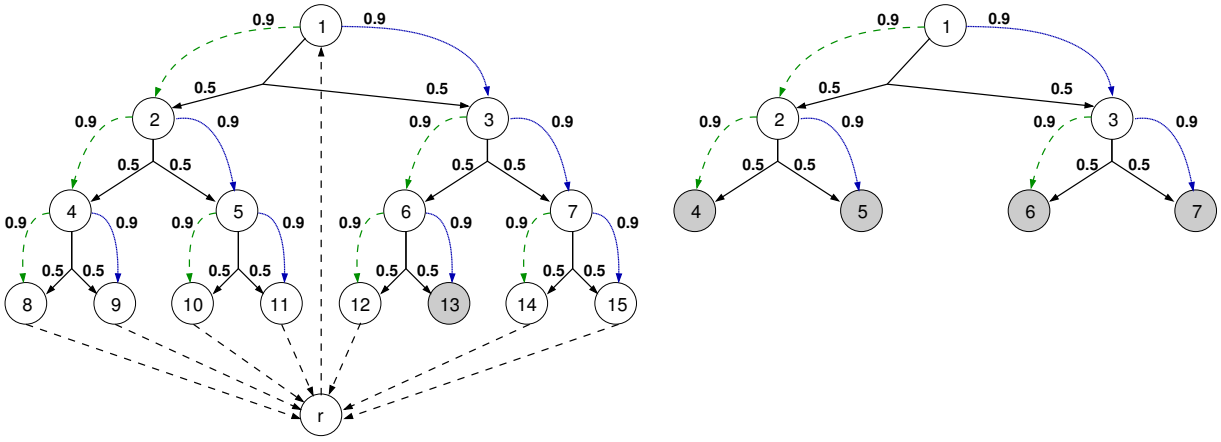
4.2 Greedy Short-Sighted SSPs

To motivate the definition of *greedy* short-sighted SSPs, consider the SSP shown in Figure 4.6(a). In this example, the state space represents a full binary tree of depth 3, with nodes labeled from 1 to 15, incremented with a special state r. The initial state is $s_0 = 1$, i.e., the root of the binary tree, and the goal is to reach the leaf represented by node 13, i.e., $\mathsf{G} = \{13\}$. Three actions are available in every non-leaf node of the binary tree: `left`, `right` and `random`. The action `random` has cost 1 and moves to the left (right) branch of the tree with probability 0.5. The action `left` (`right`) has cost 5 and moves to the left (right) branch of the tree with probability 0.9; with probability 0.1, `left` (`right`) fails and moves to the right (left) branch of the tree.

For all the leaves of the binary different of the goal leaf 13, the action `restart` is available and it deterministically transition to the state r. In state r, the action `restart` deterministically moves to the root node of the binary tree. `restart` has cost 1 when applied on a tree leaf or on r. Therefore, if the goal leaf 13 is not reached, the agent *restarts* the search from the root node 1; this process is repeated until the goal leaf is reached.

Figure 4.6(b) shows the $(s_0, 2)$-depth-based short-sighted SSP associated with the SSP in Figure 4.6(a); this depth-based short-sighted SSP is equivalent to the (s_0, ρ)-trajectory-based short-sighted SSP for $\rho \in (0.9^2, 0.9]$. Notice that state space and goal set of the short-sighted SSP in Figure 4.6(a) is the same independently of the heuristic H used as parameter, e.g., the zero heuristic.

The reason for the state space and goal set being independent of the heuristic H in depth-based and trajectory-based short-sighted SSPs is because H is used only for incrementing the cost of reaching artificial goals. In the next section, we introduce *greedy* short-sighted SSPs, a new short-sighted model that prunes states based on their heuristic cost of reaching the goal.



(a) (b)

Figure 4.6: Example of an SSP to motivate the definition of greedy short-sighted SSPs. (a) Example of an SSP. The initial state s_0 is the node 1, the goal set is $\mathsf{G} = \{13\}$. Actions `random` and `restart` cost 1 and are represented by solid black and dashed black arrows respectively. Actions `left` (green arrows) and `right` (blue arrows) cost 5 and succeed with probability 0.9. `left` (`right`) fails with probability 0.1 by moving to right (left) branch of the tree (this effect is omitted in the picture for ease of presentation). (b) $(s_0, 2)$-depth-based and (s_0, ρ)-trajectory-based, for $\rho \in (0.9^2, 0.9]$, short-sighted SSP associated with the SSP in (a).

4.2.1 Definition

Algorithm 4.2 presents our approach to generate a short-sighted state space that takes into account a given heuristic H. This algorithm performs a best-first search from the given state s using as node expansion criterion the fringe node s' that minimizes $H(s')/P_{\max}(s, s')$, i.e., the heuristic value of s' divided by the maximum trajectory probability between s and s' (Definition 4.1). The search fringe is stored in the priority queue Q, in which the next state to be popped minimizes the expansion criterion, and Q is initialized with the input state s (Lines 3 to 6). Once a state $\hat{s} \notin \mathsf{G}$ is popped from Q (Line 10), then: (i) $\hat{s}$ is removed from the short-sighted SSP goal set $\mathsf{G}_{s,k}$ (Line 14), i.e., $\hat{s}$ is not considered as an artificial goal anymore; and (ii) $\hat{s}$ is expanded (Lines 15 to 19), i.e., all states s' such that there exists $a \in \mathsf{A}$ and $P(s'|s, a) > 0$ are added to Q.

The search performed by Algorithm 4.2 terminates once $\mathsf{S}_{s,k}$ contains k or more states (Line 8). In order to guarantee that all effects of actions applied to states in $\mathsf{S}_{s,k} \setminus \mathsf{G}_{s,k}$ belong to $\mathsf{S}_{s,k}$ (Figure 4.4), Algorithm 4.2 might increase the size of $\mathsf{S}_{s,k}$ beyond k by adding more states, all of them as artificial goals. Therefore, we have that $|\mathsf{S}_{s,k} \setminus \mathsf{G}_{s,k}| < k$, since $|\mathsf{G}_{s,k}| > 0$.

Definition 4.3 formalizes the (s, k)-greedy short-sighted SSPs, where k is the size of the generated short-sighted state space.

```
GENERATE-GREEDY-SPACE(SSP 𝕊 = ⟨S, s_0, G, A, P, C⟩, s ∈ S, k ∈ ℕ*, H a heuristic for V*)
begin
    Q ← EMPTY-SMALLEST-FIRST-PRIORITY-QUEUE
    S_{s,k} ← {s}
    G_{s,k} ← {s}
    Q.INSERT(0, s)
    while not Q.ISEMPTY() do
        if |S_{s,k}| ≥ k then
            BREAK
        ŝ ← Q.POP()
        if ŝ ∈ G then
            CONTINUE
        else
            G_{s,k} ← G_{s,k} \ {ŝ}
        foreach a ∈ A and s' ∈ S s.t. P(s'|ŝ, a) > 0 do
            if s' ∉ Ŝ then
                S_{s,k} ← S_{s,k} ∪ {s'}
                G_{s,k} ← G_{s,k} ∪ {s'}
            Q.INSERT(H(s')/P_max(s, s'), s')
    return (S_{s,k}, G_{s,k})
```

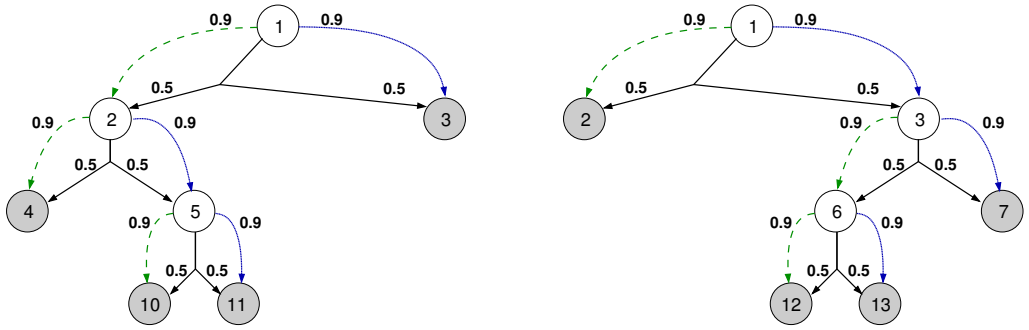
Algorithm 4.2: Algorithm to generate the state space and goal set for greedy short-sighted SSP.

Definition 4.3 (Greedy Short-Sighted SSP)**.** Given an SSP $\mathbb{S} = \langle \mathsf{S}, s_0, \mathsf{G}, \mathsf{A}, P, C \rangle$, a state $s \in \mathsf{S}$, $k \in \mathbb{N}^*$ and a heuristic H, the (s,k)-greedy short-sighted SSP $\mathbb{S}_{s,k} = \langle \mathsf{S}_{s,k}, s, \mathsf{G}_{s,k}, \mathsf{A}, P, C_{s,k} \rangle$ associated with $\mathbb{S}$ is defined as:

- $\mathsf{S}_{s,k}$ and $\mathsf{G}_{s,k}$ are the returned values of GENERATE-GREEDY-SPACE($\mathbb{S}, s, k, H$) (Algorithm 4.2); and
- $C_{s,k}(s', a, s'') = \begin{cases} C(s', a, s'') + H(s'') & \text{if } s'' \in \mathsf{G}_{s,k} \\ C(s', a, s'') & \text{otherwise} \end{cases}, \quad \forall s' \in \mathsf{S}_{s,k}, a \in \mathsf{A}, s'' \in \mathsf{S}_{s,k}$

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

Figure 4.7 shows two $(s_0, 7)$-greedy short-sighted SSPs associated with the SSP in Figure 4.6(a) when using the zero-heuristic. Due to ties in the zero-heuristic ($H_0(s) = 0$ for all $s \in \mathsf{S}$), five greedy short-sighted SSPs from s_0 using $k = 7$ are possible: one for each branch containing one pair of leaves, e.g., Figures 4.7(a) and 4.7(b), and the greedy short-sighted SSP



(a) (b)

Figure 4.7: Examples of $(s_0, 7)$-greedy short-sighted SSPs for the SSP in Figure 4.6. (a) and (b) are two of the five possible $(s_0, 7)$-greedy short-sighted SSPs when the zero heuristic is used. (b) is also the unique $(s_0, 7)$-greedy short-sighted SSPs obtained when the heuristic $H'(s) = \delta(s, 13)$ (Definition 3.1 p.21) is used.

equivalent to the depth-based short-sighted SSP for $t = 2$ (Figure 4.6(b)). Notice that the greedy short-sighted SSP in Figure 4.7(b) contains the original goal, i.e., the tree leaf labeled 13.

To further illustrate the advantages of greedy short-sighted SSPs, consider the heuristic H' defined as the minimum number of actions from s to the goal set. Formally, $H'(s) = \delta(s, 13)$ (Definition 3.1 p.21). Using H' as heuristic, the $(s_0, 7)$-greedy short-sighted SSPs associated with the SSP in Figure 4.6(a) is unique and is depicted in Figure 4.7(b). This example shows how greedy short-sighted SSPs can take advantage of informed heuristics to generate state spaces biased towards the goal set of the original problem.

4.2.2 The n-Binary Tree Problem

In this section, we generalize the binary search problem in Figure 4.6 to full binary trees any depth n (Example 4.1). Then we present an empirical comparison between SSiPP using depth-based and greedy short-sighted SSPs in the n-binary tree problems for different parameters and values n.

Example 4.1 (n-binary tree)**.** Given $n \in \mathbb{N}^*$, the n-binary tree problem contains 2^{n+1} states: $\mathsf{S} = \{1, 2, \cdots, 2^{n+1} - 1, r\}$. The initial state s_0 is 1 and the goal set G is the singleton set $\{s_G\}$, where

$$s_G = \begin{cases} 2^n + \sum_{i=0}^{\lfloor \frac{n}{2} \rfloor} 2^{2i} & \text{if n is odd} \\ 2^n + \sum_{i=1}^{\frac{n}{2}} 2^{2i-1} & \text{if n is even} \end{cases}.$$

For the states $i \in \{1, \cdots, 2^n - 1\}$, three actions are available `random`, `left` and `right`; the probability of reaching the state $2i$ is, respectively, 0.5, 0.9, and, 0.1 for `random`, `left` and

`right`; and with probability 0.5, 0.1, and, 0.9, the state $2i + 1$ is reached using `random`, `left` and `right`; respectively. In the states $i \in \{2^n, \cdots, 2^{n+1} - 1\} \setminus \mathsf{G}$, the only available action is `restart` and $P(r|i,\texttt{restart}) = 1$. `restart` is also the only available action in state r and $P(1|r,\texttt{restart}) = 1$. Actions `random` and `restart` have cost 1; and actions `left` and `right` have cost 5.

We empirically compare depth-based and greedy short-sighted SSPs by running SSiPP (Algorithm 3.2) using both definitions of short-sighted SSPs. The heuristic used in this experiment is the zero heuristic and for depth-based short-sighted SSPs, we use $t \in \{3, 4\}$. For greedy short-sighted SSPs, we choose the value of k based on the number of states used by the depth-based short-sighted SSPs. Formally, before $\mathbb{S}_{s,k}$ is computed in Algorithm 3.2 Line 6, we compute the state space $|\hat{\mathsf{S}}|$ of the (s,t)-depth-based short-sighted SSP and use $k = |\hat{\mathsf{S}}|$. We refer to this parametrization of greedy short-sighted SSPs as "budget t" and, in this experiment, we consider two budget parametrizations budget $t = 3$ and budget $t = 4$. Trajectory-based short-sighted SSPs are not considered because of the following equivalence between them and depth-based short-sighted SSPs in the n-binary tree problems: for all $\rho \in (0, 1]$, the trajectory-based short-sighted SSP using ρ as parameter is equivalent to the depth-based short-sighted SSP using $t = \lfloor \log_{0.9} \rho \rfloor + 1$.

Figure 4.8 presents the results of this experiment as average and 95% confidence interval over 100 samples for the accumulated cost to reach the goal in the n-binary tree problems. Both parametrizations of SSiPP using greedy short-sighted SSPs outperform SSiPP using depth-based SSPs. In special, the budget $t = 3$ parametrization of greedy short-sighted SSPs outperforms parametrization $t = 4$ of depth-based short-sighted SSPs.

Notice that the zero heuristic does not favor greedy short-sighted SSPs since this heuristic provides no information about the goal. However, SSiPP improves its current lower bound $\underline{V}$ every time a short-sighted SSP is solved and uses the improved $\underline{V}$ as heuristic for the subsequent short-sighted SSPs (Algorithm 3.2 Lines 6 and 8). Therefore, as the execution of SSiPP evolves, the greedy short-sighted SSPs are able to take advantage of the improved lower bound $\underline{V}$ in order to bias the short-sighted state space $\mathsf{S}_{s,k}$ towards the goals of the original problem.

4.3 Extending SSiPP to General Short-Sighted Models

In Section 3.3, we proved that SSiPP (Algorithm 3.2) always terminates and is asymptotically optimal for depth-based short-sighted SSPs. We generalize these results regarding SSiPP by: (i) providing the sufficient conditions for the generation of short-sighted problems (Algorithm 3.2 Line 6) in Definition 4.4; and (ii) proving that SSiPP implicitly performs Bellman backups (The-

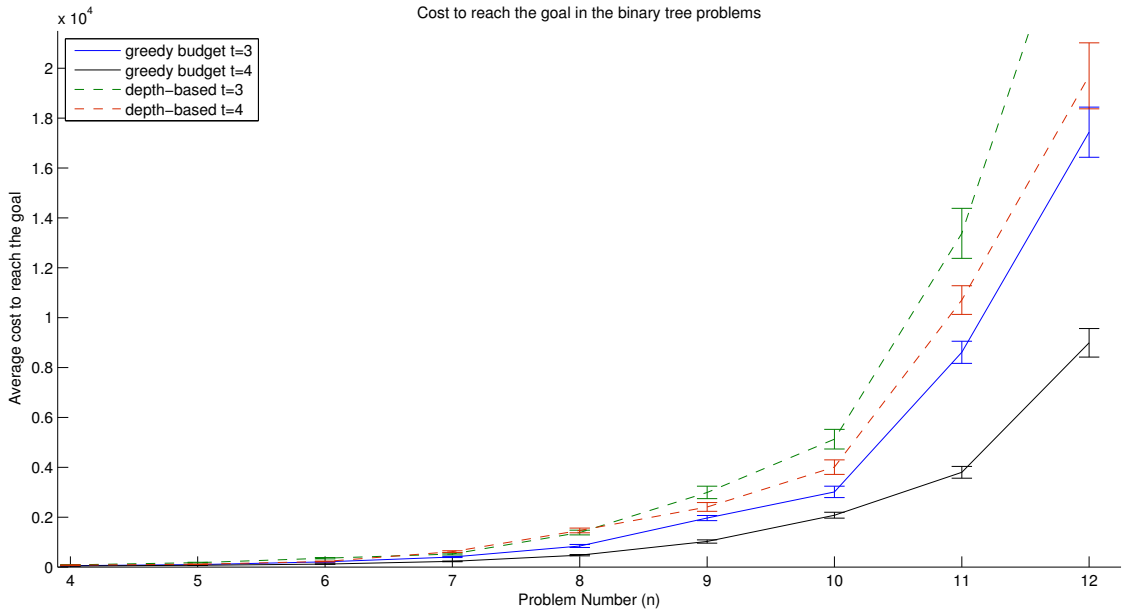


Figure 4.8: Results for the binary-tree domain experiment. Each point represents the average and 95% confidence interval over 100 samples for the accumulated cost to reach the goal in the n-binary tree problems.

orem 4.2), and always terminates (Theorem 4.3) when the short-sighted SSP generator respects Definition 4.4. The proof that SSiPP is asymptotically optimal (Theorem 3.9) automatically follows since it relies only on the fact that SSiPP terminates and performs Bellman updates.

Definition 4.4. Given an SSP $\langle \mathsf{S}, s_0, \mathsf{G}, \mathsf{A}, P, C\rangle$, the sufficient conditions on the short-sighted SSPs $\langle \mathsf{S}', \hat{s}, \mathsf{G}', \mathsf{A}, P', C'\rangle$ returned by the generator in Algorithm 3.2 Line 6 are:

1. $\mathsf{G} \cap \mathsf{S}' \subseteq \mathsf{G}'$;
2. $\hat{s} \notin \mathsf{G} \rightarrow \hat{s} \notin \mathsf{G}'$; and
3. for all $s \in \mathsf{S}' \backslash \mathsf{G}'$, $s' \in \mathsf{S}$ and $a \in \mathsf{A}$, if $P(s'|s,a) > 0$, then $s' \in \mathsf{S}'$ and $P'(s|s',a) = P(s|s',a)$.

Item 3 of Definition 4.4 guarantees that, if a state s is in the short-sighted SSP and is not a goal, i.e., $s \in \mathsf{S}' \setminus \mathsf{G}'$, then the resulting states of all applicable actions on s are also in S' (Figure 4.4) and they are reachable with the same probability as in the original SSP. Notice that, by definition, depth-based, trajectory-based and greedy short-sighted SSPs meet the sufficient conditions presented on Definition 4.4.

Theorem 4.2. *Given an SSP $\mathbb{S} = \langle \mathsf{S}, s_0, \mathsf{G}, \mathsf{A}, P, C\rangle$ such that the Assumption 2.1 holds, a monotonic lower bound H for V^*, and a short-sighted SSP generator that respects Definition 4.4, then the loop in Line 8 of SSiPP (Algorithm 3.2) is equivalent to applying at least one Bellman backup*

on $\underline{V}$ for every state $s' \in \mathsf{S}^{\pi^*_{\hat{\mathbb{S}}}} \setminus \hat{\mathsf{G}}$, where $\hat{\mathbb{S}} = \langle \hat{\mathsf{S}}, \hat{s}, \hat{\mathsf{G}}, \mathsf{A}, \hat{P}, \hat{C} \rangle$ is the generated short-sighted SSP on Line 6.

Proof. Let U denote $\mathsf{S}^{\pi^*_{\hat{\mathbb{S}}}} \setminus \hat{\mathsf{G}}$. After the loop in Line 8 of Algorithm 3.2 is executed, we have that, for all $s' \in \mathsf{U}$, $\underline{V}(s')$ equals $V^*_{\hat{\mathbb{S}}}(s')$. By item 1 of Definition 4.4, we have $\mathsf{U} \cap \mathsf{G} = \emptyset$, therefore $\underline{V}(s_G)$ remains equal to 0 for all $s_G \in \mathsf{G}$. Thus, we need to prove that, $(B\underline{V})(s') \leq V^*_{\hat{\mathbb{S}}}(s')$ for all $s' \in \mathsf{U}$, since $\underline{V}$ is monotonic and admissible (Theorem 2.1). By item 3 of Definition 4.4, every state $s' \in \mathsf{U}$ is such that $\{s'' \in \mathsf{S} | P(s''|s', a) > 0, \forall a \in \mathsf{A}\} \subseteq \hat{\mathsf{S}}$. Item 3 also guarantees that $\hat{P}(\cdot|s', a) = P(\cdot|s', a)$ for all $s' \in \mathsf{U}$ and $a \in A$, therefore $(B\underline{V})(s') = (\hat{B}\underline{V})(s')$ for all $s' \in \mathsf{U}$, where $\hat{B}$ is the Bellman operator B applied in the short-sighted SSP $\hat{\mathbb{S}}$. Since $\underline{V}$ is monotonic and admissible, $(B_{s,t}\underline{V})(s') \leq V^*_{\mathbb{S}_{s,t}}(s')$. Therefore, $(B\underline{V})(s') \leq V^*_{\mathbb{S}_{s,t}}(s')$ for all $s' \in \mathsf{U}$. □

Theorem 4.3. *SSiPP always terminates under the same conditions of Theorem 4.2.*

Proof. By Assumption 2.1 there is no dead ends in $\mathbb{S}$, thus ϵ-OPTIMAL-SSP-SOLVER always terminates. Since the short-sighted SSP $\hat{\mathbb{S}}$ is an SSP by definition, then a goal state $s_G \in \hat{G}$ of $\hat{\mathbb{S}}$ is always reached, therefore the loop in Line 11 of Algorithm 3.2 also always terminates. If s_G is a goal of the original SSP, i.e., $s_G \in \mathsf{G}$, then SSiPP terminates in this iteration. Otherwise, $s_G \in \hat{\mathsf{G}} \setminus \mathsf{G}$ and $s_G \neq s$ by item 2 of Definition 4.4, i.e., s_G differs from the state s used as initial state for the short-sighted SSP generation. Thus another iteration of SSiPP is performed using s_G as s in the generation of a new short-sighted SSP (Line 6). Suppose, for contradiction purpose, that every goal state reached during SSiPP execution is an artificial goal, i.e., SSiPP does not terminate. Then infinitely many short-sighted SSPs are solved. Since S is finite, then there exists $s \in \mathsf{S}$ that is updated infinitely often, therefore $\underline{V}(s) \to \infty$. However, $V^*(s) < \infty$ by Assumption 2.1. Since SSiPP performs Bellman updates (Theorem 4.2) then $\underline{V}(s) \leq V^*(s)$ by monotonicity of Bellman updates (Theorem 2.1) and admissibility of H, a contradiction. Thus every execution of SSiPP reaches a goal state $s_G \in \mathsf{G}$ and therefore terminates. □

4.4 Summary

In this chapter, we introduced trajectory-based short-sighted SSPs and greedy short-sighted SSPs. Trajectory-based short-sighted SSPs prune states in which every trajectory that reach them have probability less than δ. Greedy short-sighted SSPs perform a best-first search in the state space of the original problem using $H(s')/P_{\max}(s, s')$ as evaluation function, where $P_{\max}(s, s')$ is the maximum probability of a trajectory from s (the initial state of the short-sighted SSP) to s'; the search stops when the search tree contains k or more states and the visited states are used

as the short-sighted state space and the leaves as artificial goals. We also presented a set of sufficient conditions for any short-sighted SSP definition in which SSiPP always terminates and is asymptotically optimal.

Chapter 5

Extending SSiPP

In this chapter, we examine how to combine SSiPP (Section 3.3) with commonly used probabilistic planning techniques, e.g., labeling of converged states and determinizations [Trevizan and Veloso, 2013]. We begin by adding a labeling mechanism to SSiPP in order to keep track of states that already converged to their ϵ-optimal solution and avoid revisiting them. Next, in Section 5.2, we extend SSiPP to multi-core processing by generating and solving multiple short-sighted SSPs in parallel. In Section 5.3, we show how to combine SSiPP with determinizations in order to compute sub-optimal solutions more efficiently. The empirical comparison between SSiPP, the algorithms proposed in this chapter, and other state-of-the-art probabilistic planners is presented in Chapter 7.

5.1 Labeled SSiPP

As described in Section 3.3, SSiPP obtains the next state s' from the current state s by either executing or simulating the optimal policy $\pi^*_{\mathbb{S}_{s,t}}$ of the current short-sighted SSP $\mathbb{S}_{s,t}$ (Algorithm 3.2 Line 11). This procedure is repeated until s' is a goal state, either from the original SSP or an artificial goal of $\mathbb{S}_{s,t}$.

RTDP (Section 2.3.1) employs a similar technique: the next state s' is obtained by either executing or simulating π^V, i.e., the greedy action according to the current estimate V of V^*. This approach can be seen as an unbiased sampling of the next state; therefore, more likely successor states are updated more often. However, the ϵ-convergence of a given state s depends on all its reachable successors [Bonet and Geffner, 2003], thus 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 ϵ-convergence to V^*.

```
1  CHECKSOLVED(SSP 𝕊 = ⟨S, s_0, G, A, P, C⟩, state s ∈ S, value function V, solved ⊆ S,
   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         foreach s' s.t. P(s'|s, π^V(s)) > 0 do
15             if s' ∉ (solved ∪ open ∪ closed) then open.PUSH(s')
16     if conv = true then
17         foreach s' ∈ closed do
18             solved ← solved ∪ {s'}
19     else
20         while not closed.ISEMPTY() do
21             s ← closed.Pop()
22             V(s) ← (BV)(s)
23     return (solved, V)
```

Algorithm 5.1: CHECKSOLVED algorithm used by Labeled RTDP [Bonet and Geffner, 2003].

Labeled RTDP (LRTDP) [Bonet and Geffner, 2003] extends RTDP by tracking the states which the estimate V of V^* has already ϵ-converged and not visiting these states again. In order to find and label the ϵ-converged states, the procedure CHECKSOLVED (Algorithm 5.1) is introduced. Given a state s, CHECKSOLVED searches for states s' reachable from s when following the 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 ϵ-converged and they are labeled as solved. Alternatively, if there exists s' reachable from s when following the 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 ϵ-converged, then a call to CHECKSOLVED either improves V or labels a new state as solved; formally:

Theorem 5.1 ([Bonet and Geffner, 2003, Theorem 4])**.** *Given an SSP* $\mathbb{S} = \langle \mathsf{S}, s_0, \mathsf{G}, \mathsf{A}, P, C \rangle$ *that satisfies Assumption 2.1,* $\epsilon > 0$*, and a monotonic lower bound* V *for* V^**, then a call of*

CHECKSOLVED$(\mathbb{S}, s, V, solved, \epsilon)$ *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 to the ϵ-optimal solution when compared to RTDP.

Labeled-SSiPP (Algorithm 5.2) is an extension of SSiPP that incorporates the labeling mechanism of LRTDP and uses the CHECKSOLVED procedure. Since the states marked as solved have already ϵ-converged, there is no need to further explore and update them; therefore the solved states are also considered as artificial goals for the generated short-sighted SSPs (Algorithm 5.2 Line 10). By adding the solved states to the goal set of the generated short-sighted SSPs, any algorithm used as ϵ-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 5.2 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.

Besides the empirical advantage of LRTDP over RTDP [Bonet and Geffner, 2003], the labeling mechanism also allows to upper bound the maximum number of iterations necessary for LRTDP to converge to the ϵ-optimal solution. This same upper bound holds for Labeled-SSiPP:

Corollary 5.2. *Given an SSP* $\mathbb{S} = \langle \mathsf{S}, s_0, \mathsf{G}, \mathsf{A}, P, C \rangle$ *that satisfies Assumption 2.1,* $\epsilon > 0$*,* $t \in \mathbb{N}^*$ *and a monotonic heuristic* H *for* V^**, then Labeled-SSiPP (Algorithm 5.2) reaches* ϵ*-convergence 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 5.2, CHECKSOLVED is called for at least one state $\hat{s} \notin solved$, since $s_0 \notin solved$. By Theorem 5.1, 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 the greedy policy π^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. □

We empirically compare the ϵ-convergence time of Labeled-SSiPP and other state-of-the-art planners in Chapter 7. In the next section, we show how to extend Labeled-SSiPP to multi-core computing by generating and solving multiple short-sighted SSPs in parallel.

```
1  LABELED-SSIPP(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      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             foreach 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}}) ← ε-OPTIMAL-SSP-SOLVER(𝕊_{s,t}, V, ε)
14             foreach 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 ← APPLY-ACTION(π*_{𝕊_{s,t}}(s),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 5.2: Labeled SSiPP: version of SSiPP that incorporates the LRTDP labeling mechanism. CHECKSOLVED is presented in Algorithm 5.1.

5.2 Parallel Labeled SSiPP

Deterministic planners have benefited from parallelism to compute both optimal and suboptimal solutions. Different approaches have been proposed, e.g., search space abstraction [Burns et al., 2009, Burns et al., 2010, Zhou et al., 2010], hashing [Zhou and Hansen, 2007, Kishimoto et al., 2009, Kishimoto et al., 2010], and parallel successor generation [Vidal et al., 2010, Sulewski et al., 2011].

For discounted infinite-horizon MDPs (Section 2.1), i.e., probabilistic planning problems without goal states, parallel solvers have been proposed [Archibald et al., 1993, Archibald et al., 1995]. These planners extend *asynchronous* value iteration to perform updates in parallel. Although these approaches can be applied to SSPs, they do not exploit the problem's structure, e.g., the initial state and set of goals states. Therefore, parallel MDP solvers always explore the complete state space, including irrelevant states [Barto et al., 1995].

It is important to notice that finding the optimal solution of a deterministic and a probabilistic planning problem belong, respectively, to the NC and P-complete complexity classes in their enumerative representation [Papadimitriou and Tsitsiklis, 1987]. In other words, the optimal solution of a deterministic planning problem can be efficiently found using a parallel algorithm, while it is *unlikely* that optimal algorithms to solve probabilistic planning problem can be efficiently parallelized.[1] However, problems in which a given set of states L must be visited in order to reach the goal can take advantage of parallelization. To illustrate how parallelization can speedup some probabilistic planning problems, consider the *Hallway problem* example:

Example 5.1 (Hallway problem)**.** In the hallway problem, a robot has to navigate a grid composed by k rooms of size r each while avoiding the hazard locations and walls. The rooms form a line and each room is connect to the next by a single door. Figure 5.1 shows an example of grid for $k = 3$ and $r = 5$. Every time the robot enters a hazard location, it breaks with probability 0.9. Thus, the state space S for the hallway problem is composed by pairs $\langle l, b\rangle$, where l is a location in the grid and b is a boolean variable indicating if the robot is broken or not. The initial state s_0 is $\langle d_0, \text{false}\rangle$ and the goal is to reach the last door and be not broken, i.e., $\mathsf{G} = \{\langle d_k, \text{false}\rangle\}$. Five actions are available in the hallway problem: move north, south, east and west, and fix-robot. If the robot is not broken, the move actions succeeds with probability 0.9 and move the robot to the given direction; and fails with probability 0.1 by not moving the robot. When the robot is broken, the move actions do not change the current state. If the robot is broken, the fix-robot action deterministically fixes the robot and moves it to d_0, i.e., $P(s_0|\langle l, \text{true}\rangle, \text{fix-robot}) = 1$ for

[1] It is unlikely due to the unproven assumption that NC $\neq$ P.

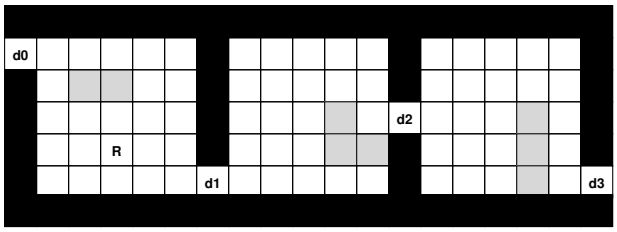


Figure 5.1: Grid of the hallway problem (Example 5.1) for $k = 3$ and $r = 5$. In this problem, a robot "R" has to navigate between rooms from location d_0 to location d_3 while avoiding the hazard locations (grey) and walls (black).

all locations l. When the robot is not broken, fix-robot does not change the current state. The cost of the move actions are 1 and the cost of fix-robot is 10.

Since there is a single door connecting each room, we can decompose a k-r-hallway problem into k instances of a 1-r-hallway problem, where the initial state and goal set of the i-th problem are, respectively, $\langle d_{i-1}, \text{false}\rangle$ and $\{\langle d_i, \text{false}\rangle\}$ for $i \in \{1, \cdots, k\}$. Therefore, we can compute an optimal policy for a k-r-hallway problem by combining an optimal policy for each one of its k subproblems.

In this section, we show how to extend Labeled-SSiPP (Algorithm 5.2) in order to exploit the structure of this problem by solving several short-sighted SSPs in parallel and then combine their solutions. We start by assuming that the list of states L that must be visited to reach the goal is given and introduce the new algorithm *Parallel Labeled-SSiPP* (Section 5.2.1). Then, in Section 5.2.2, we present a method based on landmarks to automatically generate the list of states L.

5.2.1 Algorithm

Parallel Labeled-SSiPP, shown in Algorithm 5.3, extends Labeled-SSiPP by solving multiple short-sighted SSPs in parallel and combining their solutions. Precisely, Parallel Labeled-SSiPP launches $n - 1$ new threads (Lines 10 to 12), each one with their own copy of the current lower bound $\underline{V}$, while the main thread solves the short-sighted SSP $\mathbb{S}_{s,t}$ associated with the state s (Line 13). Once the ϵ-optimal solution for $\mathbb{S}_{s,t}$ is obtained by the main thread, all the $n - 1$ threads are stopped and their current lower bound $\underline{V}_i$ is merged with $\underline{V}$ (Lines 14 to 16). Each thread selects a state $s' \in \mathsf{L}$ using a thread-safe procedure to avoid duplicates and solves the short-sighted SSP $\mathbb{S}_{s',t}$; if the ϵ-optimal solution for $\mathbb{S}_{s',t}$ is obtained before being stopped by the main thread, a new state $s' \in \mathsf{L}$ is selected (Lines 25 to 28).

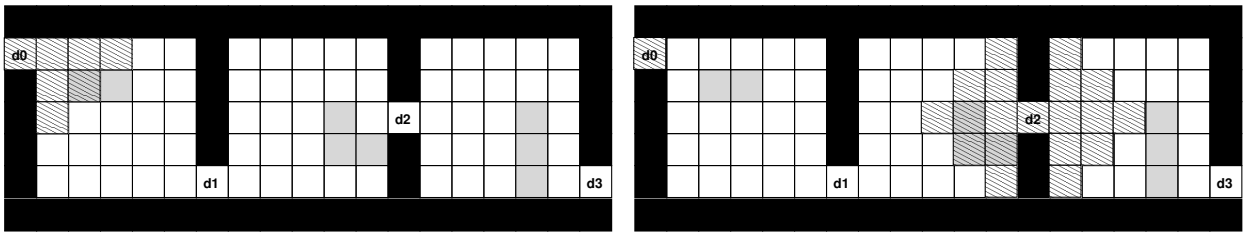
A copy of the lower bound $\underline{V}$ is given to each thread (Algorithm 5.3, line 10) in order to prevent interference between threads while computing the solutions of the short-sighted SSPs. To illustrate such interference, consider the $(s, 3)$-depth-based short-sighted SSPs $\mathbb{S}_0$ and $\mathbb{S}_2$ associ-

1 PARALLEL LABELED-SSIPP(SSP $\mathbb{S} = \langle \mathsf{S}, s_0, \mathsf{G}, \mathsf{A}, P, C \rangle$, $t \in \mathbb{N}^*$, H a heuristic for V^*, $\epsilon > 0$, $n \in \mathbb{N}^*$)
2 **begin**
3 $\quad \underline{V} \leftarrow$ Value function for S with default value given by H
4 $\quad solved \leftarrow \emptyset$
5 $\quad$ **while** $s_0 \notin solved$ **do**
6 $\quad\quad s \leftarrow s_0$
7 $\quad\quad visited \leftarrow$ EMPTY-STACK
8 $\quad\quad$ **while** $s \notin (\mathsf{G} \cup solved)$ **do**
9 $\quad\quad\quad \mathsf{L} \leftarrow$ COMPUTE-L(S, s)
10 $\quad\quad\quad$ **foreach** $i \in \{1, \ldots, n-1\}$ **do**
11 $\quad\quad\quad\quad \underline{V}_i \leftarrow$ MAKE-COPY($\underline{V}$)
12 $\quad\quad\quad\quad$ START-NEW-THREAD($\mathbb{S}, t, \underline{V}_i, \epsilon, \mathsf{L}$)
13 $\quad\quad\quad (\underline{V}, \hat{\pi}^*) \leftarrow$ SOLVE-SHORT-SIGHTED($\mathbb{S}, s, t, \underline{V}, \epsilon$)
14 $\quad\quad\quad$ STOP-ALL-THREADS()
15 $\quad\quad\quad$ **for** $s' \in \mathsf{S}$ **do in parallel**
16 $\quad\quad\quad\quad \underline{V}(s') \leftarrow \max\{\underline{V}(s'), \underline{V}_1(s'), \ldots, \underline{V}_{n-1}(s')\}$
17 $\quad\quad\quad$ **while** $s \notin \mathsf{G}_{s,t}$ **do**
18 $\quad\quad\quad\quad visited$.PUSH($s$)
19 $\quad\quad\quad\quad s \leftarrow$ APPLY-ACTION($\pi^*_{\mathbb{S}_{s,t}}(s)$,$s$)
20 $\quad\quad$ **while** ***not*** $visited$.ISEMPTY() **do**
21 $\quad\quad\quad s \leftarrow visited$.POP()
22 $\quad\quad\quad (solved, \underline{V}) \leftarrow$ CHECKSOLVED($\mathbb{S}, s, \underline{V}, solved, \epsilon$)
23 $\quad\quad\quad$ **if** $s \notin solved$ **then break**
24 $\quad$ **return** $\underline{V}$

25 START-NEW-THREAD(SSP $\mathbb{S}$, $t > 0$, $\underline{V}$ a lower bound for V^*, $\epsilon > 0$, L a list of states)
26 **begin**
27 $\quad$ **while** $(s \leftarrow$ THREAD-SAFE(L.POP())) **do**
28 $\quad\quad \underline{V} \leftarrow$ SOLVE-SHORT-SIGHTED($\mathbb{S}, s, t, \underline{V}, \epsilon$)

29 SOLVE-SHORT-SIGHTED(SSP $\mathbb{S}$, $s \in \mathsf{S}$, $t > 0$, $\underline{V}$ a lower bound for V^*, $\epsilon > 0$)
30 **begin**
31 $\quad \mathbb{S}_{s,t} \leftarrow$ GENERATE-SHORT-SIGHTED-SSP($\mathbb{S}, s, \underline{V}, t$)
32 $\quad \mathsf{G}_{s,t} \leftarrow \mathsf{G}_{s,t} \cup (solved \cap \mathsf{S}_{s,t})$
33 $\quad (\pi^*_{\mathbb{S}_{s,t}}, V^*_{\mathbb{S}_{s,t}}) \leftarrow \epsilon$-OPTIMAL-SSP-SOLVER($\mathbb{S}_{s,t}, \underline{V}, \epsilon$)
34 $\quad$ **foreach** $s' \in \mathsf{S}^{\pi^*_{\mathbb{S}_{\hat{s},t}}} \setminus \mathsf{G}_{\hat{s},t}$ **do**
35 $\quad\quad \underline{V}(s') \leftarrow V^*_{\mathbb{S}_{\hat{s},t}}(s')$
36 $\quad$ **return** $(\pi^*_{\mathbb{S}_{s,t}}, \underline{V})$

Algorithm 5.3: Parallel version of Labeled-SSiPP (Algorithm 5.2). STOP-ALL-THREADS (Line 14) cancels all the extra threads running and returns immediately to the main thread.



(a) (b)

Figure 5.2: Examples of (s,t)-depth-based short-sighted SSPs for the hallway problem in Figure 5.1. The patterned cells represent the locations included in each short-sighted SSP. For both short-sighted SSPs, $t = 3$, and s equals $\langle d_0, \text{false}\rangle$ and $\langle d_2, \text{false}\rangle$ for (a) and (b) respectively.

ated with the hallway problem example in Figure 5.1 for s equal to, respectively, $s_0 = \langle d_0, \text{false}\rangle$ and $\langle d_2, \text{false}\rangle$. As shown in Figure 5.2, the initial state s_0 belongs to the state space of both $\mathbb{S}_0$ and $\mathbb{S}_2$, also s_0 is an artificial goal of $\mathbb{S}_2$. Thus, if $\mathbb{S}_0$ and $\mathbb{S}_2$ are solved in parallel *sharing* the same lower bound $\underline{V}$, then the Bellman updates applied on $\underline{V}(s_0)$ when solving $\mathbb{S}_0$ affects the solution of $\mathbb{S}_2$, therefore there is no guarantee that solution computed by ϵ-OPTIMAL-SSP-SOLVER (Algorithm 5.3 Line 13) for $\mathbb{S}_2$ is ϵ-optimal since $\underline{V}(s_0)$ might have changed.

Another benefit of each thread manipulating their own copy $\underline{V}_i$ of $\underline{V}$ is that Theorem 3.7 guarantees that the monotonicity and admissibility of each $\underline{V}_i$ is preserved. Once the (partial) solutions from all threads are obtained, they are combined in parallel by keeping the maximum over all lower bounds on each state $s \in \mathsf{S}$ (Algorithm 5.3 Line 15). Clearly, the $\max$ operator preserves the admissibility of $\underline{V}$ and, in Lemma 5.3, we prove that the $\max$ operator also preserves the monotonicity of a value-function. Therefore, each iteration of Parallel Labeled-SSiPP maintains the lower bound $\underline{V}$ monotonic and admissible. Corollary 5.4 extends the convergence bound of Labeled-SSiPP (Corollary 5.2) to Parallel Labeled-SSiPP.

Lemma 5.3. *Given an SSP* $\mathbb{S} = \langle \mathsf{S}, s_0, \mathsf{G}, \mathsf{A}, P, C\rangle$ *and two monotonic value functions* V_1 *and* V_2 *for* $\mathbb{S}$*, then* V_m*, defined as* $V_m(s) = \max\{V_1(s), V_2(s)\}$*, is also a monotonic value function for* $\mathbb{S}$*.*

Proof. Suppose, for contradiction, that V_m is not monotonic, thus there exists $s \in \mathsf{S}$ such that $V_m(s) > (BV_m)(s)$. Without loss of generality assume $V_2(s) < V_1(s) = V_m(s)$. If there exist $s' \in \mathsf{S}$ and $a \in \mathsf{A}$ s.t. $P(s'|s,a) > 0$ and $V_2(s') > V_1(s')$, then either: (i) a equals $\text{argmin}_{a'} E\left[C(s,a',s') + V_m(s')|s,a'\right]$, and therefore $(BV_m)(s) \geq (BV_1)(s) \geq V_1(s) = V_m(s)$; otherwise (ii) $(BV_m)(s) = (BV_1)(s) \geq V_1(s) = V_m(s)$.

Alternatively, if there exists no such s', then $V_2(s') \leq V_1(s')$ for all $s' \in \mathsf{S}$ and $a \in \mathsf{A}$ s.t. $P(s'|s,a) > 0$; therefore (iii) $(BV_m)(s) = (BV_1)(s) \geq V_1(s) = V_m(s)$. By (i) – (iii), we have that $V_m(s) \leq (BV_m)(s)$, and a contradiction is obtained. □

Room Size (r)		5				10				15			
Number of Rooms (k)		5	10	15	20	5	10	15	20	5	10	15	20
Parallel L. SSiPP	$n = 2$	1.85	1.66	1.61	1.58	1.46	1.36	1.36	1.39	1.29	1.31	1.27	1.32
	$n = 4$	2.73	2.53	2.47	2.42	1.97	1.98	1.91	1.93	1.71	1.74	1.71	1.74
	$n = 8$	**2.95**	**3.23**	**3.17**	**3.01**	**2.05**	**2.33**	**2.31**	**2.31**	**1.85**	**2.17**	**2.16**	**2.11**

Table 5.1: Speedup of Parallel Labeled-SSiPP, for different number of parallel threads n, w.r.t. Labeled-SSiPP in the hallway robot domain. Results are averaged over 50 random problems for each combination of r and k. Best performance shown in bold.

Corollary 5.4. *Given an SSP $\mathbb{S} = \langle \mathsf{S}, s_0, \mathsf{G}, \mathsf{A}, P, C \rangle$ that satisfies Assumption 2.1, $\epsilon > 0$, $t \in \mathbb{N}^*$, $n \in \mathbb{N}^*$ and a monotonic heuristic H for V^*, then Parallel Labeled-SSiPP (Algorithm 5.3) reaches ϵ-convergence after at most $\epsilon^{-1} \sum_{s \in \mathsf{S}} [V^*(s) - H(s)]$ iterations of the loop in Line 5.*

Proof. Each iteration of the loop in Line 5 solves at least the short-sighted SSP associated with the current state s, i.e., if $n = 1$, Parallel Labeled-SSiPP and Labeled-SSiPP are equivalent. Since the $\max$ operator preserves the admissibility and monotonicity of $\underline{V}$ (Lemma 5.3), then this proof follows from Corollary 5.2. □

To illustrate the advantages of Parallel Labeled-SSiPP and (sequential) Labeled-SSiPP, we present an experiment comparing both of them in randomly generated k-r-hallway problems. For this experiment, both planners use the Manhattan distance as the heuristic H, LRTDP as the ϵ-optimal solver and $t = 5$ for the generation of depth-based short-sighted SSPs. The list L given to Parallel Labeled-SSiPP contains all states in which the robot is not broken and at one of the internal doors, precisely, $\mathsf{L} = \{\langle d_1, \text{false} \rangle, \langle d_2, \text{false} \rangle, \ldots, \langle d_{k-1}, \text{false} \rangle\}$.

We generated 50 random problems for each combination of $r \in \{5, 10, 15\}$ and $k \in \{5, 10, 15, 20\}$. Door locations are chosen uniformly at random and every location that is not a door, is marked as hazard with probability 0.15. Each planner is run until ϵ-convergence, for $\epsilon = 10^{-4}$, and we limit the runtime and memory to 1 hour and 4 GB, respectively. The experiments were conducted on a Linux machine with 8 cores running at 2.40 GHz.

Table 5.1 shows the results averaged over the 50 random problems for each parametrization. Parallel Labeled-SSiPP outperforms its sequential version in all the parametrizations. The obtained speedup varies from 1.85 to 3.23 when 8 threads are used. As expected, we see the diminishing returns effect: the obtained improvement decreases as more threads are added.

5.2.2 Choosing States for Parallel Labeled SSiPP

In this section, we present an algorithm to compute the list of states L used by Parallel Labeled-SSiPP to build short-sighted SSPs. Notice that L can be seen as a list of subgoals of the original problem,

1 COMPUTE-L(SSP $\mathbb{S} = \langle \mathsf{S}, s_0, \mathsf{G}, \mathsf{A}, P, C \rangle$, $s \in \mathsf{S}$)
2 **begin**
3 $\quad \mathbb{D} \leftarrow$ ALL-OUTCOMES-DETERMINIZATION($\mathbb{S}$)
4 $\quad \mathcal{G} \leftarrow$ FIND-LANDMARKS($\mathbb{D}, s$)
5 $\quad \mathsf{P} \leftarrow$ FIND-SHORTEST-PATH($\mathcal{G}, s, \mathsf{G}$)
6 $\quad \mathsf{L} \leftarrow$ INSTANTIATE-ALL-FORMULAS(P)
7 $\quad$ **return** L
8 **end**

Algorithm 5.4: Landmark approach to compute L for Parallel Labeled-SSiPP (Algorithm 5.3). $\mathcal{G}$ is a graph representing the landmarks of the deterministic problem $\mathbb{D}$. INSTANTIATE-ALL-FORMULAS generates all the states $s' \in \mathsf{S}$ such that at least on landmark in P is true in s'.

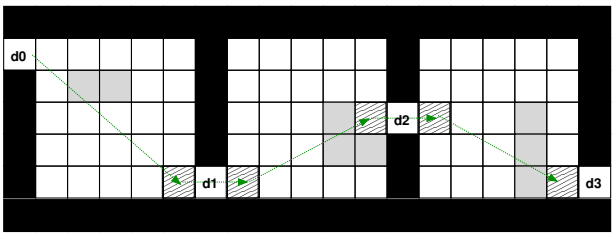


Figure 5.3: Example of states returned by Algorithm 5.4 from the initial state for the hallway problem in Figure 5.1. The patterned cells and green arrows represents, respectively, the vertices (landmarks) and arcs (ordering) of the path P (Algorithm 5.4 Line 5).

e.g., the states $\langle d_1, \text{false} \rangle$ and $\langle d_2, \text{false} \rangle$ in the hallway problem example (Figure 5.1). Parallel Labeled-SSiPP makes no assumption w.r.t. the states $s \in \mathsf{L}$ and any state s that is reachable from s_0 has potential to generate a speedup.

In deterministic planning, one approach to obtain subgoals is through *landmarks* [Hoffman et al., 2004]. A landmark is a formula over the problem's state variables (Section 2.2) that must be true at some point during the execution of every solution that reaches the goal. Two landmarks a and b can also be (partially) ordered according to different constraints, e.g., if a is true some time before b and if a is always true one step before b. Finding landmarks and ordering them is computationally expensive, for instance, deciding if a state variable is a landmark is PSPACE-complete [Hoffman et al., 2004]. Therefore, algorithms to automatically find (ordered) landmarks relies on approximations in order to be computationally feasible.

Our approach to generate L for a given $\mathbb{S}$ is to obtain partially ordered landmarks for the all-outcomes determinization of $\mathbb{S}$ (Section 2.3.2) and post-process them in order to remove landmarks that have already being met. Algorithm 5.4 describe our method to generate L and we use the Fast-Downward [Helmert, 2006] landmark identification algorithm as FIND-LANDMARKS in Line 4. Figure 5.3 shows the landmarks selected in Line 4 from the initial state in the hallway example in Figure 5.1.

Room Size (r)		5				10				15			
Number of Rooms (k)		5	10	15	20	5	10	15	20	5	10	15	20
Parallel L. SSiPP	$n = 2$	1.50	1.41	1.43	1.37	1.21	1.19	1.17	1.14	1.08	1.13	1.09	1.11
	$n = 4$	2.07	1.97	1.91	1.93	1.38	1.34	1.29	1.31	1.20	1.16	1.13	1.14
	$n = 8$	**2.33**	**2.17**	**2.13**	**2.06**	**1.52**	**1.43**	**1.44**	**1.42**	**1.29**	**1.22**	**1.21**	**1.21**

Table 5.2: Speedup of Parallel Labeled-SSiPP using Algorithm 5.4 to generate the list of state L in the hallway robot domain. Results are averaged over 50 random problems for each combination of r and k. Best performance shown in bold.

We repeated the series of random hallway problems experiments (Table 5.1) following the same methodology and using Algorithm 5.4 to generate the list L. Table 5.2 presents the results as average speedup with respect to (sequential) Labeled-SSiPP over the 50 random problems for each parametrization. Parallel Labeled-SSiPP using Algorithm 5.4 still outperforms its sequential version in all the parametrizations and the speedup varies from 1.21 to 2.33 when 8 threads are used. As expected, the speedup decreases with respect to Parallel Labeled-SSiPP using the list of doors as L, i.e., Table 5.1. There are two reasons for this decrease in performance, the extra overhead of computing the landmarks and the extra states returned by Algorithm 5.4. The latter is illustrated in Figure 5.3: Algorithm 5.4 returns the locations before and after each door location d_i because they are the only locations in which the robot can reach d_i.

5.3 SSiPP-FF

In this section, we show how to combine the SSiPP and determinizations in order to improve the scalability of SSiPP while dropping SSiPP's optimality guarantee. This extension of SSiPP, SSiPP-FF, is depicted in Algorithm 5.5. After reaching an artificial goal s, SSiPP-FF performs the following extra steps with respect to SSiPP (Algorithm 3.2): (i) compute a determinization $\mathbb{D}$ of the original SSP; (ii) runs FF to solve $\mathbb{D}$ using s as initial state; and (iii) executes the returned plan until failure (Lines 12 to 17 in Algorithm 5.5).

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 pre-computed and reused in every iteration to generate $\mathbb{D}$. Since SSiPP-FF does not assume any specific behavior of FF, any deterministic planner can be used for solving $\mathbb{D}$ in Line 14 instead of FF.

Besides taking advantage of potential non-optimal 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 5.5 has an

```
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}}) ← ε-OPTIMAL-SSP-SOLVER(𝕊_{s,t}, V, ε)
8          foreach 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 ← APPLY-ACTION(a_i, s)
18     return V
```

Algorithm 5.5: SSiPP-FF: version of SSiPP that incorporates determinizations to obtain a non-optimal solution efficiently.

avoidable dead end, i.e., there exist at least one proper policy for $\mathbb{S}_{s,t}$ and every proper policy for $\mathbb{S}_{s,t}$ is closed but not complete. Since an ϵ-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 $\pi^*_{\mathbb{S}_{s,t}}$.

Notice that the guarantee of not reaching avoidable dead ends that are included in the current short-sighted SSP is not due to SSiPP-FF. Instead, this guarantee is inherited from SSiPP. We finish this section by introducing and analyzing the *jumping chain* problems (Example 5.2), a series of problems in which SSiPP-FF avoids all dead ends while determinization approaches based on shortest distance to goal, e.g., FF-Replan, reach a dead end with probability exponentially large in the problem size.

Example 5.2 (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 independently 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 a_J 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

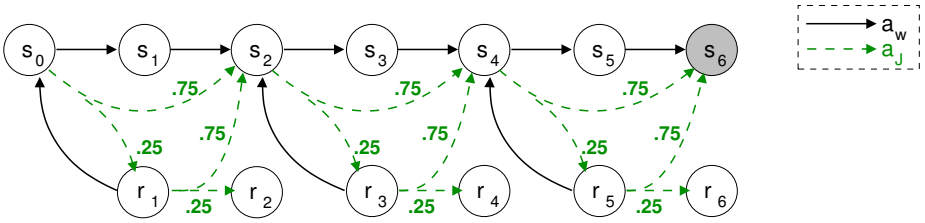


Figure 5.4: Representation of the jumping chain problem (Example 5.2) for $k = 3$. The initial state is s_0, the goal set is $\mathsf{G} = \{s_6\}$. Actions a_W and a_J have cost 1 and 3 respectively.

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 $P(r_{i+1}|r_i, s_J) = 0.25$. Notice that, for all i even, r_i is a dead end. Figure 5.4 shows the jumping chain problem for $k = 3$.

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 action `walk` because: (i) `walk` results in a state further away from the goal; and (ii) `jump` has a non-zero probability to reach a state in which the goal is still achievable. Therefore, the solutions obtained by FF-Replan have non-zero 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 $p = P(s_{i+2}|s_i, a_J)$.

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

5.4 Summary

In this chapter, we presented three extensions of SSiPP: Labeled-SSiPP, Parallel Labeled-SSiPP and SSiPP-FF. Labeled-SSiPP improves the convergence time of SSiPP to the ϵ-optimal solution by labeling states that have already ϵ-converged as solved. Solved states are not revisited by the during the search for the ϵ-optimal solution and are also pruned from the short-sighted SSP since an ϵ-optimal solution from these labeled states is already known. Parallel Labeled-SSiPP extends Labeled-SSiPP by generating and solving multiple short-sighted SSPs in parallel. For both Labeled-SSiPP and Parallel Labeled-SSiPP, we proved an upper bound on the number of iterations necessary for them to converge to the ϵ-optimal solution.

We also introduced SSiPP-FF, a planner that combines SSiPP with determinizations in order to compute sub-optimal solutions more efficiently. Besides improving the scalability of SSiPP, we show how SSiPP-FF can make FF-Replan safer by avoiding dead ends within the solved short-sighted SSPs.

In the next chapter, we present the previous work in optimal and suboptimal probabilistic planning and how they relate to SSiPP, Labeled-SSiPP, Parallel Labeled-SSiPP and SSiPP-FF. Then, in Chapter 7, we empirically compare our algorithms against the state-of-the-art probabilistic planners.

Chapter 6

Related Work

This chapter presents a review of related work in probabilistic planning. Probabilistic planners, i.e., algorithms that return closed policies, are reviewed in Sections 6.1 to 6.3; and replanners, algorithms that return partial policies, are reviewed in Section 6.4. Section 6.5 presents how this thesis fits with respect to the presented related work.

6.1 Extensions of Value Iteration

One direct extension of Value Iteration (VI), presented in Section 2.1, is Topological Value Iteration (TVI) [Dai and Goldsmith, 2007]. TVI pre-processes the given SSP by performing a topological analysis of the state space 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, i.e., from the goals to the initial state. This decomposition can speed up the search of ϵ-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 SSC, TVI performs worst 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) [Dai et al., 2009] 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 sub-optimal are removed from the original SSP. Once the $R(\mathbb{S}, \underline{V})$ is small, the search is stopped and the resulting SSP is solved using TVI and $\underline{V}$ as lower bound. Since the removed actions are always sub-optimal, FTVI returns an ϵ-optimal solution. In the worst, FTVI is equivalent to TVI since there is no guarantee that any action will be removed from the original SSP.

6.2 Real Time Dynamic Programming and Extensions

Another extension of VI is Real Time Dynamic Programming (RTDP) [Barto et al., 1995], presented in Section 2.3.1. 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 ϵ-optimal solution, RTDP updates its lower bound $\underline{V}(s)$ of $V^*(s)$ on every state s visited during the search. If Assumption 2.1 holds for the given SSP, then RTDP always finds an ϵ-optimal solution after several search iterations (possibly infinitely many), i.e., RTDP is asymptotically optimal. Differently from the VI, TVI and FTVI that compute complete policies, RTDP returns a closed policy if ϵ is small enough or a partial policy otherwise.

Several extensions of RTDP have been proposed and the first one is Labeled RTDP (LRTDP) [Bonet and Geffner, 2003]. LRTDP introduces a labeling mechanism to find states that have already ϵ-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-optimal 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) [McMahan et al., 2005], Focused RTDP (FRTDP) [Smith and Simmons, 2006] and Value of Perfect Information RTDP (VPI-RTDP) [Sanner et al., 2009]. The advantage of keeping an upper bound is that the exploration of the state space can be biased towards 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-optimal 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 of the 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 the ϵ-optimal solution, and ReTrASE [Kolobov et al., 2009] extends RTDP in order 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 (Section 2.3.2). Due to the lower dimensional representation, ReTrASE is non-optimal.

6.3 Policy Iteration and Extensions

A different approach for finding ϵ-optimal solutions is Policy Iteration (PI) [Howard, 1960]. 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. Since 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* [Hansen and Zilberstein, 2001] can be seen as a version of PI which takes advantage of s_0 and computes ϵ-optimal closed policies that are potentially not complete. Precisely, LAO* computes a closed ϵ-optimal policy for the sequence $\mathsf{S}_0 \subseteq \mathsf{S}_1 \subseteq \ldots \mathsf{S}_k \subseteq \mathsf{S}$, where $\mathsf{S}_0 = \{s_0\}$, i.e., S_0 contains only the initial state, S is the complete state space of the SSP and S_i is generated by greedily expanding S_{i-1}. LAO* stops when $\mathsf{S}^{\pi^*} \subseteq \mathsf{S}_i$; therefore the closed ϵ-optimal policy for S^{π^*} is also ϵ-optimal for the original problem. Improved LAO* (ILAO*) [Hansen and Zilberstein, 2001] enhance LAO* performance by increasing how many states are added to S_{i-1} to generate S_i and performing single Bellman Backups in a depth-first postorder traversal of S_i instead of using PI or VI to compute ϵ-optimal solutions to S_i.

6.4 Replanners

Another direction to solve probabilistic planning problem is replanning. One of the simplest, yet powerful, replanners is FF-Replan [Yoon et al., 2007], presented in Section 2.3.2. Given a state s (initially s equals s_0) FF-Replan generates the all-outcomes determinization $\mathbb{D}$ of the SSP $\mathbb{S}$ being solved and uses the deterministic planner FF [Hoffmann and Nebel, 2001] to solve $\mathbb{D}$ from state s. The solution π for $\mathbb{D}$ is then applied to the $\mathbb{S}$; if and when the execution of π fails in the probabilistic environment, FF is re-invoked to plan again from the failed state. FF-Replan was the winner of the first International Probabilistic Planning Competition (IPPC) [Younes et al., 2005] in which it outperformed the probabilistic planners due to their poor scalability. Despite its major success, FF-Replan is non-optimal and oblivious to probabilities and dead ends, leading to poor performance in probabilistic interesting problems [Little and Thiébaux, 2007], e.g., the triangle tire-domain (Section 4.1.2).

FF-Hindsight [Yoon et al., 2008] is a non-optimal replanner that generalizes FF-Replan based on hindsight optimization. Given a state s, FF-Hindsight performs the following three steps: (i) randomly generate a set of non-stationary deterministic problems D starting from s; (ii) use

FF to solve each problem in D; and (iii) combine 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 [Yoon et al., 2010] 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.

Another determinization-based replanner is HMDPP [Keyder and Geffner, 2008]. Instead of using the all-outcomes or the most-likely outcomes, HMDPP uses the *self-loop determinization*, a determinization approach that implicitly encode the probability of actions is in their costs. Formally, given an SSP $\mathbb{S} = \langle \mathsf{S}, s_0, \mathsf{G}, \mathsf{A}, P, C \rangle$, the self-loop determinization of $\mathbb{S}$ is the problem $\mathbb{D} = \langle \mathsf{S}, s_0, \mathsf{G}, \hat{A}, \hat{C} \rangle$, in which, for all $s \in \mathsf{S}$, $a \in A$ and $s' \in \mathsf{S}$ such that $P(s'|s,a) > 0$, $\hat{A}$ contains the action a' that deterministically transforms s into s' and its cost $\hat{C}(s,a',s')$ is $C(s,a',s')/P(s'|s,a)$. Therefore, solutions for $\mathbb{D}$ that use low probability effect of actions are penalized. HMDPP also pre-process the original SSP $\mathbb{S}$ using pattern databases [Haslum et al., 2007] for a fixed amount of time in order to obtain a set of partial policies π_{db} from some states in S to the goal. These two techniques are combined in HMDPP as follows: at a state s, if there is a pre-computed policy π_{db} from s to the goal, then $\pi_{db}(s)$ is applied; otherwise, a solution π_{det} for the self-loop determinization of $\mathbb{S}$ is computed from s and executed until a state s', in which π_{det} is not defined, is reached. This process is repeat until a goal state is reached.

Based on solution refinement, two other non-optimal replanners were proposed: Envelope Propagation (EP) [Dean et al., 1995] and Robust FF (RFF) [Teichteil-Koenigsbuch et al., 2008]. In general terms, EP and RFF compute an initial partial policy π and iteratively expand it in order to avoid replanning. EP performs state aggregation by selecting a set of states $\hat{\mathsf{S}}$ and replacing them by a meta state `out`. Set of states $\hat{\mathsf{S}}$ is obtained by finding states that have low probability of being reached and also have an expected cost larger than the current state. The obtained state space S' equals $\texttt{out} \cup (\mathsf{S} \setminus \hat{\mathsf{S}})$ and special actions are also added to the aggregated SSP to represent transitions between S' and the meta state `out`. At each iteration, EP refines its approximation S' of S by selecting states $\hat{s} \in \hat{\mathsf{S}}$ and adding them to S'. After S' is expanded, a new round of aggregation is performed in order to avoid the convergence of S' to S. If a state $\hat{s} \in \hat{\mathsf{S}}$ needs to be avoided, e.g., high cost states and dead ends, then EP is unable to take that signal into account to effectively avoid them.

RFF, the winner of the third IPPC [Bryce and Buffet, 2008], uses a different approach for solution refinement: 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, i.e., given $\rho \in [0, 1]$, RFF computes π such that the probability of replanning when following π from s_0 is at most ρ. Since computing the probability of replanning when following π is costly, RFF approximates it by performing Monte-Carlo simulations.

An orthogonal direction from all other approaches mentioned so far is applied by t-look-ahead [Pearl, 1985, Russel and Norvig, 2003] and Upper Confidence bound for Trees (UCT) [Kocsis and Szepesvri, 2006]. The approach employed by these algorithms is to relax SSPs into finite-horizon MDPs with goals, i.e., to modify horizon of the SSP from indeterminate to finite. T-look-ahead fixes the horizon of the relaxed problem to t time steps and solves it using dynamic programming (Chapter 2).

UCT is an approximation of the t-look-ahead obtained by using sparse sampling techniques. Formally, UCT iteratively builds a policy tree by expanding the best node according to a biased version of the Bellman equations (Equation (2.2) p.12) to ensure that promising actions are sampled more often. Notice that UCT, as t-look-ahead, builds a policy tree, i.e., a policy free of loops, since the horizon of the problem is relaxed from indefinite to finite of size t. While UCT does not require the search depth parameter t, it is governed by two other parameters: w the number of samples per decision step and c the weight of the bias term for choosing actions. UCT is the base of PROST [Keller and Eyerich, 2012], the winner of IPPC 2011 [Coles et al., 2012].

In the context of motion planning in dynamic environments, another relevant approach is *Variable Level-of-Detail* (VLOD) [Zickler and Veloso, 2010] planning and execution. VLOD computes a collision-free trajectory from an initial state to the goal by ignoring the physical interactions with poorly predictable dynamic objects in the far future. Formally, VLOD computes a plan in which: (i) all actions applicable from the initial state (time $t = 0$) until a given time threshold t_{LOD} consider the original model of the world; and (ii) all actions applicable at time $t > t_{LOD}$ consider a relaxed model $\hat{M}$ of the world. This relaxed model $\hat{M}$ simplifies the problem by ignoring the collisions between the agent and the other dynamic objects in the environment, e.g., the agent is able to pass through other moving agents and objects in $\hat{M}$. Therefore, VLOD can efficiently compute a plan that locally avoids collisions while still taking in consideration the goal set in order to be robust against local minima.

For planning with incomplete information, a relevant approach is *assumptive planning and execution* [Nourbakhsh and Genesereth, 1996]. In this approach, the uncertainty of execution is decreased by making simplifying assumptions, for instance, if the initial state s_0 is partially

defined, then one possible simplifying assumption is to instantiate some of the undefined state variables. Planning and execution is then interleaved through a replanning loop: (i) given the current a set of possible states b, a smaller set $\hat{\mathsf{b}}$ is obtained by making additional assumptions about b; (ii) a conditional plan C from $\hat{\mathsf{b}}$ to the goal G is computed; (iii) C is executed in the environment; and (iv) both b and $\hat{\mathsf{b}}$ are updated according to the actions applied in the environment. When and if $\hat{\mathsf{b}}$ is inconsistent, then replanning is applied using the new current incomplete state b. The authors also provide sufficient conditions over the simplifying assumptions to guarantee that this replanning approach is sound and complete.

6.5 How our Work Fits

Table 6.1 summarizes the related work and provides an overview of how this thesis fits with respected to the related work.

This thesis presents a novel relaxation technique for probabilistic planning, the short-sighted SSPs. Short-sighted SSPs relax probabilistic planning problems by pruning the state space and adding artificial goals to heuristically estimate the cost of reaching an original goal from the pruned states. The usage of artificial goals is the key difference between short-sighted SSPs and the state space aggregation performed by EP. Since a heuristic cost is incurred when an artificial goal is reached, the solutions of short-sighted SSPs can be effectively biased towards the original goals and away from high-costs areas of the state space.

Short-sighted SSPs also differ from determinizations because they do not change the action structure. Therefore all effects of actions are considered and their probabilities are not ignored. Similarly, short-sighted SSPs differ from VLOD and assumptive planning and execution because they neither simplify the model in the far future nor make additional assumptions to reduce uncertainty. Instead, short-sighted SSPs use a heuristic to estimate the cost of reaching the goal from the artificial goals and preserve the original action structure. These features allow SSiPP to iteratively improve the given heuristic until it ϵ-converges to the optimal solution of the SSP begin solved. Notice that the determinization approaches, VLOD, and assumptive planning are not able to compute ϵ-optimal solutions of SSPs.

Depth-based short-sighted SSPs, one formulation of short-sighted SSPs, also presents a novel property with respect to the previous work: closed policies for (s, t)-depth-based short-sighted SSPs can be applied to the original SSP for at least t steps without replanning. The replanners reviewed in Section 6.4 do not guarantee how many steps their partial policies can be applied before replanning is needed.

Planner	Optimal	Policy Computed is	Uses Heur.	Simplification applied	Approach
Value Iteration	yes	complete	yes	–	dynamic programming
Topological VI					dynamic programming and topological analysis
Focused Topological VI				action elimination	
Policy Iteration			no	–	dynamic programming
LAO*		closed	yes		dynamic programming and and/or graph search
ILAO*					dyn. prog., and/or graph search and topological analysis
RTDP					dynamic programming
Labeled RTDP					dynamic programming and labeling
Bounded RTDP					dynamic programing and upper bounds
Focused RTDP					
VPI-RTDP					
ReTrASE	no			state space projection	dynamic programming
Envelope Propagation		partial	no	state space	aggregation and solution refinement
FF-Replan				action space	determinization
FF-Hindsight					determinization and hindsight optimization
Robust FF					determinization and solution refinement
HMDPP			yes		determinization and pattern database
Look ahead				horizon	local search
UCT	yes				local search and sparse sampling
SSiPP	yes	closed	yes	state space	**short-sighted planning**
Labeled-SSiPP					**short-sighted planning** and labeling
Parallel Labeled-SSiPP					
SSiPP-FF	no	partial		state and action space	**short-sighted planning** and determinization

Table 6.1: Summary of the related work and how our work fits in. For each planner it is shown: if it is optimal or non-optimal; the type of policy computed; if the planner is able to use state space heuristics $H(s)$; the simplification applied to manage the uncertainty structure of the problems; and the overall approach employed by the planner.

Chapter 7

Empirical Evaluation

In this chapter, we present a rich empirical comparison between the proposed algorithms and state-of-the-art probabilistic planners and replanners. We begin by reviewing the domains and problems used in the experiments. Next, in Section 7.2, we present a series of experiments to evaluate the convergence time to the ϵ-optimal solution of SSiPP, Labeled-SSiPP, and other optimal planners. In Section 7.3, we simulate an International Probabilistic Planning Competition (IPPC) [Younes et al., 2005, Bonet and Givan, 2007, Bryce and Buffet, 2008] using SSiPP, Labeled-SSiPP, SSiPP-FF, previous IPPC winners and other state-of-the-art planners as contestants.

7.1 Domains and Problems

In this section, we present the four domains from IPPC'08 [Bryce and Buffet, 2008] which we use in our experiments.[1] The first two domains, probabilistic blocks world (Section 7.1.1) and zeno travel (Section 7.1.2), are probabilistic extensions of their deterministic counterparts. Triangle tire world (Section 7.1.3) and exploding blocks world (Section 7.1.4) are probabilistic interesting problems [Little and Thiébaux, 2007], i.e., problems in which approaches that oversimplify the probabilistic structure of the actions perform poorly.

7.1.1 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

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

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

Table 7.1: 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.

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 2.1 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: (i) putting a block on top of other blocks versus putting a block on the table; and (ii) 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 7.1 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. For the remainder of this chapter, we refer to the probabilistic blocks worlds as blocks world.

7.1.2 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 and therefore 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 number of time steps necessary to succeed, is $1/p$. In order to ensure the ge-

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

Table 7.2: Number of cities, persons and airplanes for each of the 15 problems considered of the zeno travel domain.

ometric 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 actions flying and zooming that have costs 10 and 25 respectively. Although the fuel requirement for flying and zooming is the same, their expected cost differ due to their different costs and success probabilities: 250 for flying and 375 for zooming.

As in the blocks world domain, Assumption 2.1 holds for all problems in zeno travel domain. Table 7.2 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.

7.1.3 Triangle Tire World

The triangle tire world, described in Section 4.1.2, is a probabilistically interesting domain with avoidable dead ends. In the experiments, the problem number corresponds to the parameter n of the triangle tire world problem.

7.1.4 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

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

Table 7.3: Number of blocks and blocks in the goal statement for each of the 15 problems considered from the exploding blocks world.

is not destroyed. Actions `put-down` and `put-on-block` have the probabilistic side-effect of 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 [Little and Thiébaux, 2007]. 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, i.e., 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 7.3 shows the total number of blocks and blocks in the goal statement for the 15 exploding blocks world problems considered. In the considered problems, all actions have cost 1.

7.2 Convergence to the Optimal Solution

In the following experiments, we compare the time necessary for LRTDP [Bonet and Geffner, 2003], Focused Topological Value Iteration (FTVI) [Dai et al., 2009], SSiPP and Labeled-SSiPP to ϵ-converge to the optimal solution. SSiPP-FF is not considered since it is not guaranteed to converge to an ϵ-optimal solution. For the experiments in Section 7.2.1, we use the domains from IPPC'08 (reviewed in Section 7.1) and, in Section 7.2.2, we use the *race-track* domain, a common domain to compare optimal probabilistic planners.

7.2.1 Problems from the International Probabilistic Planning Competition

In this experiment, we compare the time to converge to the ϵ-optimal solution for the problems in the IPPC'08 (Section 7.1). Although Assumption 2.1 does not hold for the triangle tire world (Section 7.1.3), all problems in this domain are such that: (i) there exists a proper (but not complete) policy; and (ii) the 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 since $V^*(s_0) < 12n$ for the triangle tire world problem of size n. The exploding blocks world problems are not considered because there is no guarantee they have a closed policy.

This experiment was conducted on a 2.4GHz machine with 16 cores running a 64-bit version of Linux. The time and memory cutoff enforced for each planner was 2 hours and 5GB respectively. For SSiPP and Labeled-SSiPP, we used LRTDP as ϵ-OPTIMAL-SSP-SOLVER and depth-based short-sighted SSPs for $t \in \{2, 4, 8, 16, 32\}$. The admissible heuristic used by all the planners is the classical planning heuristic $h_{\max}$ applied to the all-outcomes determinization [Teichteil-Königsbuch et al., 2011].

Table 7.4 presents the results of this experiment as the average and 95% confidence interval of the ϵ-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 ϵ-converged to the optimal 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 to the optimal solution for problems 5 to 8 and problems 1 to 4 are too small (ϵ-convergence is reached in about 1s).

The performance difference between SSiPP and Labeled-SSiPP is not significant for small problems, i.e., 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 the optimal solution is found using a single short-sighted SSP, therefore 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 reaches convergence using between 6% to 32% of the 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 problem 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 in 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 124436 states and $\mathbb{S}_{s_0,16}$ is generated and solved on every iteration of Line 5 of Labeled-SSiPP (Algorithm 5.2), even after inferring that $\mathbb{S}_{s_0,16}$ also contains only one proper policy. As shown in Section 4.1.2, trajectory-based short-sighted SSPs can be used in order to overcome this issue.

		Blocks World							
Problem		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

		Triangle Tire World					Zeno Travel		
Problem		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	**5370.2 ±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	-	-	-

Table 7.4: Results of the ϵ-convergence experiment for the IPPC domains. Each cell represents the average and 95% confidence interval of the time, in seconds, to converge to the ϵ-optimal solution using $\epsilon = 10^{-4}$. If ϵ-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 $b10$.

For the larger problems of the blocks world ($5'$ to $8'$), Labeled-SSiPP obtains a large 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, i.e., problems 1 and 2, and converge in at most 0.06 of LRTDP convergence time. Notice that FTVI fails to converge in all the zeno travel problems and Labeled SSiPP for $t = 32$ is the only planner able to converge for problem 4 of the zeno travel domain.

7.2.2 Race-track problems

The goal of a problem in the race-track domain [Barto et al., 1995, Bonet and Geffner, 2003] 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 race-track domain is the tuple (x, y, v_x, v_y, b) in which:

- x and y are the position 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) on their 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. An acceleration action (a_x, a_y) 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 in order 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 race-tracks in this experiment: ring-small, ring-large, square-small, square-large, y-small and y-large. The shape of each track is depicted in Figure 7.1 and Table 7.5 presents their corresponding state space size $|\mathsf{S}|$, ratio of relevant states (i.e., $|\mathsf{S}^{\pi^*}|/|\mathsf{S}|$), largest parameter t, $t_{\max}$, for depth-based short-sighted SSPs 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 deter-

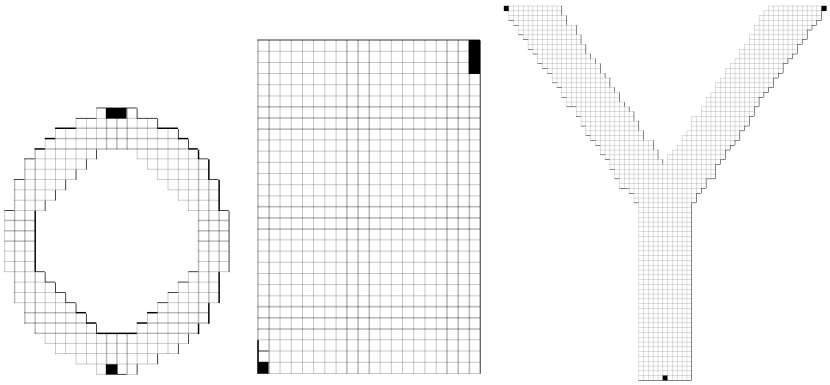


Figure 7.1: Shape of the race-tracks used in the ϵ-convergence experiment. Each cell represents a possible position of the car. The initial position and the goal positions are, respectively, the marked cells in the bottom and top of each track.

problem	$\|\mathsf{S}\|$	% 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 7.5: Description of each race-track used in the ϵ-convergence experiment. The columns represent: size of the state space $|\mathsf{S}|$, ratio $\mathsf{S}^{\pi^*}/\mathsf{S}$, $t_{\max}$, $V^*(s_0)$, value of the min-min heuristic for s_0 ($h_{\min}(s_0)$) and time in seconds to compute $h_{\min}(s_0)$.

minization. Therefore, $h_{\min}$ can be computed by the following fixed point equations:

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

This experiment was conducted on a 3.07GHz machine with 4 cores running a 32-bit version of Linux. A time cutoff of 2 hours and 4GB of memory was applied to each planner. For SSiPP and Labeled-SSiPP, we used LRTDP as ϵ-OPTIMAL-SSP-SOLVER and depth-based short-sighted SSPs for $t \in \{4, 8, 16, \ldots, 1024\}$. FTVI is not considered in this experiment because the implementation of FTVI we had access to is not compatible with the encoding of the racetrack problems. Table 7.6 presents the results as the average and 95% confidence interval for 10 runs of each planner parametrization.

	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	**0.59 ±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	**16.15 ±0.25**	631.78 ±39.42	**51.61 ±2.68**	311.44 ±8.26
	128	0.61 ±0.02	59.89 ±3.14	16.72 ±0.35	**612.78 ±30.44**	55.60 ±2.17	**307.45 ±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		**0.59 ±0.02**	**55.81 ±2.92**	18.60 ±0.84	702.42 ±12.82	54.00 ±2.20	319.08 ±8.35

Table 7.6: Results of the ϵ-convergence experiment for the racetrack domain. Each cell represents the average and 95% confidence interval of the time, in seconds, to converge to the optimal solution using $\epsilon = 10^{-4}$. Best performance over all planners (column) is shown in bold font. The min-min heuristic was used by all planners.

The performance of SSiPP, Labeled-SSiPP and LRTDP is similar for $t > t_{\max}$ in all the problems since LRTDP is used as ϵ-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 the optimal policy in the square-large track reaches only 0.75% of the state space (Table 7.5). Therefore both SSiPP and Labeled-SSiPP take advantage of the short-sighted search to prune useless states earlier in the search, resulting a better performance than LRTDP for $t \in \{32, 64, 128, 256\}$.

7.3 International Probabilistic Planning Competition

In this section, we compare the performance of the following planners to obtain (sub-optimal) solutions under a 20 minutes time cutoff:

- FF-Replan [Yoon et al., 2007] (winner of IPPC'04),
- Robust-FF [Teichteil-Koenigsbuch et al., 2008] (winner of IPPC'08),
- HMDPP [Keyder and Geffner, 2008],
- ReTrASE [Kolobov et al., 2009],
- SSiPP,
- Labeled-SSiPP, and
- SSiPP-FF.

The non-SSiPP planners are reviewed in Chapter 6 and, for these experiments, we use 15 problems from IPPC'08 of each domain described in Section 7.1. We present the methodology used in this experiment in Section 7.3.1. In Section 7.3.2, we describe heuristics to choose the parameters of SSiPP, Labeled-SSiPP and SSiPP-FF, i.e., the value of t for depth-based short-sighted SSPs. Section 7.3.3 presents the results of this experiment.

7.3.1 Methodology

We use a methodology similar to the IPPCs, in which there is a time cutoff 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 [Younes et al., 2005], an SSP (and MDP) simulator. Planners send actions to be simulated to MDPSIM and MDPSIM 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 it inferred that it is trapped in a dead end. A round is considered successful if the goal is reached, otherwise it is declared as 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. In order to evaluate planners more accurately, we execute 50 runs for each problem and planner, and no information is shared between the different runs, i.e., all the internal variables of the planners are reseted when a new run starts. Therefore, in this section, the performance of a planner in a given problem is estimated by 2500 rounds generated by potentially 50 different policies computed by the same planner. Notice that our approach (50 runs of 50 rounds each) is not equivalent to the execution of one run of 2500 rounds. In the latter case, a planner might be guided towards *bad* decisions by the outcomes of the probabilistic actions and not have enough time to revise such decisions. Alternatively, by simulating several runs, there is small probability that this guidance will happen in all the runs.

In order to respect the 20 minutes time cutoff, 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 3.2 and 5.5, respectively, and the obtained lower bound $\underline{V}$ in round r is used as heuristic for round $r + 1$. The same effect is obtained for Labeled-SSiPP by adding a 15 minutes time cutoff in Line 5 of Algorithm 5.2.

The IPPCs also enforce that planner must not have free parameters, i.e., the only input for each planner is the problem to be solved. Therefore, all parameters of a planner, e.g., 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 differ-

ent parametrization were fixed for Robust-FF and we consider only the RFF-PG parametrization, since it obtained the best performance in IPPC'08 for the considered problems [Bryce and Buffet, 2008]. Section 7.3.2 describes the two different methods we employed to obtain the parametrizations for SSiPP, Labeled-SSiPP and SSiPP-FF.

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

In order 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 in 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. While these three domains are the same between IPPC'06 and IPPC'08, their final problems are different. No problem from the triangle tire world is used for training, since they are deterministically generated, i.e., any triangle tire world of size $\{1, \ldots, 15\}$ 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 7.3.1. We denote as $c(k_i, p)$ the number of rounds of the problem $p \in \mathsf{J}$ in which X, using parametrization $k_i \in \mathsf{K}$, reached the goal. The function $m(k_i, k_j)$ represents the tournament bracket between k_i and k_j, and $m(k_i, k_j)$ equals 1 if

$$\Big|\{p \in \mathsf{P} | c(k_i, p) > c(k_j, p)\}\Big| > \Big|\{p \in \mathsf{P} | c(k_i, p) < c(k_j, p)\}\Big|,$$

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

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 [Hoffmann and Nebel, 2001] to reach a goal state from the current state s in the all-outcomes determinization; and
- h_{max} and h_{add} applied to the all-outcomes determinization of the original problem [Teichteil-Königsbuch et al., 2011].

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, we select t at random from $\{2, 3, 4, \ldots, 10\}$ before calling GENERATE-SHORT-SIGHTED-SSP in Algorithms 3.2, 5.2 and 5.5. Therefore, different values of t might be used for solving a given problem. For this approach, we use h_{add} as heuristic for all the SSiPP-based planners and the all-outcomes determinization for SSiPP-FF. Also, in order 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$_r$.

7.3.3 Results

This experiment was conducted on a 2.4GHz machine with 16 cores running a 64-bit version of Linux. We use coverage, i.e., the ratio between the number of successful rounds and 2500 (the total number of round), as performance metric. Table 7.7 presents the summary of the results as the number of problems in which a given planner has the best coverage. The detailed results are presented in Tables 7.8 and 7.9 as the coverage obtained by each planner in every problem, and in Tables 7.10 and 7.11 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 improved 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 number 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.

	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	1
SSiPP$_t$	4	0	1	2
SSiPP$_r$	4	2	2	8
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

Table 7.7: Summary of the IPPC experiment. Each cell represents the number of problems per domain in which a given planner has the best coverage. For each problem, more than one planner might obtain the best coverage, therefore the columns do not add up to 15. ReTrASE does not support the zeno travel problems (*n.a.*).

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 1 to 10 and 12, i.e., all the problems in which the SSiPP-FF obtained coverage 1, the solutions found by SSiPP-FF$_r$ and SSiPP-FF$_t$ have average cost between 0.80 and 0.92 of the FF-Replan solutions average costs.

Labeled-SSiPP performs well in the small problems, obtaining good coverage and solutions with small average cost; however Labeled-SSiPP fails to scale up to large problems. The reason for not scaling up is the bias for exploration over exploitation employed by Labeled-SSiPP in order to speedup the convergence to the ϵ-optimal solution.

All SSiPP-based planners perform well in the exploding blocks world: SSiPP$_t$ has the best coverage in 9 of the problems; SSiPP-FF$_r$ has the best coverage in the 5 largest problems; and, for all the considered problems in the exploding blocks world, a SSiPP-based planner has 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 heuristic. h_{add} in the triangle tire world guides the planners towards dead ends and the SSiPP-based planners manage to avoid only the dead ends visible inside the short-sighted SSPs. As shown in Section 4.1.2, SSiPP performs the best in the triangle tire domain when the zero-heuristic is used and Table 7.12 shows the

Prob.		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**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**
	2	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**
	3	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**
	4	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**
	5	**1.000**	0.992	-	**1.000**	0.957	0.649	**1.000**	**1.000**	**1.000**	**1.000**
	6	**1.000**	**1.000**	-	**1.000**	0.964	0.286	0.342	0.295	**1.000**	**1.000**
	7	**1.000**	0.984	-	**1.000**	0.059	0.074	0.117	0.099	**1.000**	**1.000**
	8	**1.000**	**1.000**	-	**1.000**	0.132	0.204	0.173	0.127	**1.000**	**1.000**
	9	**0.999**	0.987	-	0.870	0.443	0.186	0.061	0.040	0.761	0.507
	10	0.999	**1.000**	-	0.883	0.002	-	0.003	0.001	0.761	0.533
	11	**0.998**	0.995	-	0.881	-	-	-	-	0.764	0.475
	12	0.998	**1.000**	-	0.925	0.003	0.006	-	-	0.821	0.519
	13	**0.847**	-	-	-	-	-	-	-	0.067	0.008
	14	**0.867**	-	-	-	-	-	-	-	0.089	0.010
	15	**0.886**	-	-	-	-	-	-	-	0.062	0.007
Zeno Travel	1	**1.000**	0.175	**1.000**	*n.a.*	0.017	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**
	2	**1.000**	0.081	**1.000**	*n.a.*	0.100	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**
	3	**1.000**	-	-	*n.a.*	-	-	-	-	**1.000**	**1.000**
	4	**1.000**	-	-	*n.a.*	-	-	-	-	**1.000**	**1.000**
	5	**1.000**	-	-	*n.a.*	-	-	-	-	**1.000**	**1.000**
	6	**1.000**	-	-	*n.a.*	-	-	-	-	**1.000**	**1.000**
	7	**1.000**	-	-	*n.a.*	-	-	-	-	**1.000**	**1.000**
	8	**1.000**	-	-	*n.a.*	-	-	-	-	**1.000**	**1.000**
	9	**1.000**	-	-	*n.a.*	-	-	-	-	**1.000**	**1.000**
	10	**1.000**	-	-	*n.a.*	-	-	-	-	**1.000**	**1.000**
	11	**1.000**	-	-	*n.a.*	-	-	-	-	0.261	0.469
	12	**1.000**	-	-	*n.a.*	-	-	-	-	**1.000**	**1.000**
	13	**1.000**	-	-	*n.a.*	-	-	-	-	0.443	**1.000**
	14	**1.000**	-	-	*n.a.*	-	-	-	-	0.269	**1.000**
	15	**1.000**	-	-	*n.a.*	-	-	-	-	-	0.740

Table 7.8: Coverage for the blocks world and zeno travel domains in the IPPC experiment. 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.*).

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

Table 7.9: Coverage for the triangle tire world and exploding blocks world domains in the IPPC experiment. Best coverage for each problem (row) is shown in bold. If no round is solved, i.e., zero coverage, then '-' is shown.

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 ±∞	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 7.10: Cost of the solutions for the block world and zeno travel domains in the IPPC experiment. Each cell represents the average and 95% confidence interval for the obtained cost over the successful rounds. If no round is solved, then '-' is shown; if exactly one round is solved, then ∞ 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$_r$
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 ±∞	24.0 ±∞
	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 ±∞
	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 ±∞	-	-	-	-
	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 7.11: Cost of the solutions for the triangle tire world and exploding blocks domain in the IPPC experiment. Each cell represents the average and 95% confidence interval for the obtained cost over the successful rounds. If no round is solved, then '-' is shown; if exactly one round is solved, then ∞ is shown in the 95% confidence interval.

Problem		SSiPP	L-SSiPP	SSiPP-FF
Triangle Tire World	1	**1.000**	**1.000**	**1.000**
	2	**1.000**	**1.000**	**1.000**
	3	0.997	**1.000**	0.533
	4	0.977	**1.000**	0.162
	5	0.963	**1.000**	0.082
	6	0.950	**1.000**	0.049
	7	0.913	**1.000**	0.023
	8	0.870	0.868	0.015
	9	0.882	0.798	0.003
	10	0.842	0.767	-
	11	0.773	0.717	-
	12	0.738	0.633	-
	13	**0.717**	0.595	-
	14	**0.685**	0.518	-
	15	**0.617**	0.422	-

Table 7.12: Coverage of SSiPP-based planner in the triangle tire world using depth-based short-sighted SSPs and the zero-heuristic. For all planners, the parameter t equals 8 for all the planners and, for SSiPP-FF, the all-outcomes determinization is used. Best coverage for each problem (row), with respect to the results in Tables 7.8 and 7.9, are shown in bold. If no round is solved, then '-' is shown.

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 the rounds for the problems 1 to 7; and SSiPP has the best coverage for the 3 largest problems in comparison with all the considered planners.

7.4 Summary

In this chapter, we presented a rich empirical comparison between the proposed algorithms and other state-of-the-art algorithms in two tasks: finding an ϵ-optimal solution and finding a (sub-optimal) solution under the International Probabilistic Planning Competition (IPPC) rules, e.g., small time cutoff. The results from the ϵ-convergence experiments showed that Labeled-SSiPP, using LRTDP as underlying SSP solver, outperforms SSiPP, LRTDP and FTVI on problems from the IPPC and on control problems with low ratio of relevant states, i.e., $|\mathsf{S}^{\pi^*}|/|\mathsf{S}|$. The results obtained in the experiment following the IPPC rules show that SSiPP-FF successfully combines the behavior of SSiPP and FF-Replan by having a large coverage in problems without dead ends

and significantly improving the coverage of FF-Replan in problems with dead ends. These results also show that SSiPP and SSiPP-FF consistently outperforms the other planners in all the problems of the exploding blocks world, a probabilistic interesting domain.

Chapter 8

A Real World Application: a Service Robot Searching for Objects

In this chapter, we present how a mobile service robot moving in a building in order to find an object, whose location is not deterministically known, can use short-sighted planning to improve its performance. We begin by motivating the mobile service problem and, in Section 8.2, we formally present how to represent this problem as an SSP. In Section 8.3, we empirically evaluate different planners, including SSiPP, in different instances of the mobile service robot problem.

8.1 Motivation

The problem of an autonomous agent moving in an environment to find objects while minimizing the search cost is ubiquitous in the real world, e.g., a taxi driver looking for passengers and minimizing the usage of gas, a software agent finding information about a product on the web while minimizing the bandwidth usage, a service robot bringing objects to users minimizing distance traversed, and a robot collecting rocks for experiments while minimizing power consumption. In all these problems, we assume that the agent does not know where the exact objects are, and has some probabilistic model of the location of the objects.

For this chapter, our concrete motivation is the mobile service robot that moves in a building to find an object, e.g., *coffee*, and to deliver it to a location, e.g., office #171. We assume that the robot is given a map of the environment and that the object can be in more than one location. Also, we consider that the probability of the object being at a location type, e.g., offices, is given. Such prior distribution can be designed by an expert or automatically obtained, for example by querying the web (e.g., [Samadi et al., 2012]). In particular, we focus on the problem of finding the desired object, since the delivery problem can be cast as the problem of finding an object that

is deterministically present only in the delivery location. In the next section, we present how to represent the problem of finding a given object as an SSP.

8.2 Representing the Problem as an SSP

In this section we present our formulation of the problem of finding an object in a building as an SSP represented in PPDDL (Section 2.2). For this representation, we use one domain variable, LOCATION, that describes the locations the agent is allowed to visit and the following predicates defined over locations:

- `connected`(l_1, l_2): true when the agent can move from location l_1 to l_2;
- `at`(l): to represent the agent's current location;
- `objAt`(l): to denote that an instance of the object being searched for is at l;
- `searched`(l): to indicate that l has already being searched;
- and a set of predicates to denote the type of each location, e.g., `isOffice`(l) for office locations and `isKitchen`(l) for kitchens.

Also, we use the state variable `hasObject` to indicate that the agent has the desired object.

For each location type t, we use the binary random variable X_t to denote if the object is at the locations of type t and we assume that a prior probability $\bar{P}(X_t)$ is given. Note that $\sum_t \bar{P}(X_t = \text{true})$ is not required to sum up to 1. This feature is used for representing scenarios such as an object that can be found deterministically in more than one location type or an object that has a low probability to be found in any location type. To simplify notation, we denote $\bar{P}(X_t = \text{true})$ as p_t for every location type t.

We model the object finding through a pair of action schemas, `Search` and `PickUp`. The action `Search`(l), depicted in Figure 8.1, has the precondition that the agent is at location l and l has not been searched before. Its effect is `searched`(l), i.e., to mark l as searched, and, with probability p_t, where t is the location type of l, the object is found. With probability $1 - p_t$, the object is not found at l. Since `searched`(l) is true after the execution of `Search`(l), the agent cannot search the same location l more than once. We enforce this restriction because $(1 - p_t)^k \to 0$ as $k \to \infty$ for $p_t > 0$, i.e., if the agent were allowed to search the same location enough times it would always find the object there.

The action `PickUp`(l), depicted in Figure 8.2, represents the agent obtaining the object at location l if the object is there. This action can be easily extended to encompass more general scenarios, e.g., a robotic agent with grippers that can fail and the object might not be always

```
(:action Search
 :parameters (?l - location)
 :precondition (and (at ?l) (not (searched ?l)))
 :effect (and
    (searched ?l)
    (when (isBathroom ?l) (prob 0.08 (objAt ?l)))
    (when (isKitchen ?l)  (prob 0.18 (objAt ?l)))
    (when (isOffice ?l)   (prob 0.02 (objAt ?l)))
    (when (isPrinterR ?l) (prob 0.72 (objAt ?l))))
)
```

Figure 8.1: PPDDL code for the action $\texttt{Search}(l)$ of the service robot problem. For this action the prior used for the object being at a location l is 8%, 18%, 2% and 72% if l is, respectively, a bathroom, a kitchen, an office or a printer room.

```
(:action PickUp
 :parameters (?l - location)
 :precondition (and (at ?l) (objAt ?l))
 :effect (and
   (not (objAt ?loc))
   (hasObject))
)
```

Figure 8.2: PPDDL code for the action $\texttt{PickUp}(l)$ of the service robot problem..

obtained or a symbiotic autonomous agent that might ask people for help to manipulate the object [Rosenthal et al., 2010]. Such extensions can be modeled by converting $\texttt{PickUp}(l)$ into a probabilistic action or a chain of probabilistic actions.

We use the action schema `Move` to model the agent moving in the map represented by the predicate $\texttt{connected}(l_1, l_2)$. The action $\texttt{Move}(l_1, l_2)$ is probabilistic and with probability p the agent moves from l_1 to l_2 and with probability $1 - p$ the agent stays at l_1. For all the examples and experiments in this chapter, we use $p = 0.9$.

Initially, the value of the state variable `hasObject` is false and the goal of the agent is to reach any state in which `hasObject` is true. For easy of presentation, we define the cost of all actions to be 1, i.e., $C(s, a, s') = 1\ \forall s \in \mathsf{S}, a \in \mathsf{A}, s' \in \mathsf{S}$. Therefore the average cost of reaching the goal equals the average number of actions applied by the agent.

To illustrate our model, consider the map presented in Figure 8.3(a). In this map, the agent is at position 0 and there are two hallways that can be explored: (i) the right hallway of size k in which the last location is a kitchen; and (ii) the left hallway with $2r$ offices. Notice that Figure 8.3(a) represents only the map of the environment and not the search space. A fraction of the search space is depicted on Figure 8.3(b).

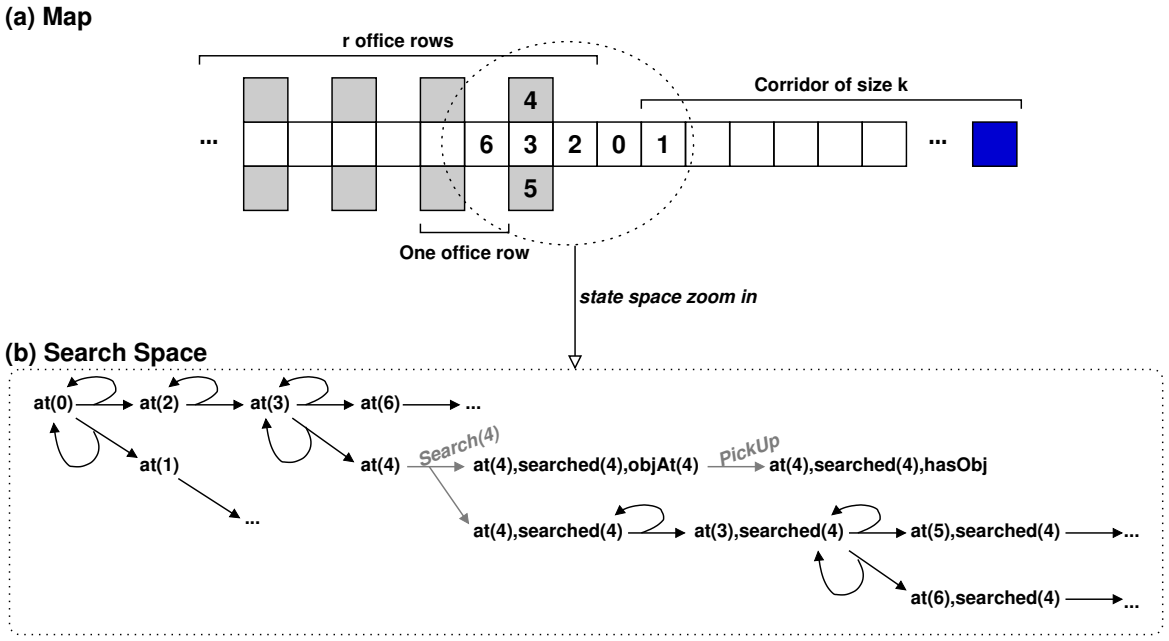


Figure 8.3: Example of map and state space of the service robot problem. (a) Example of map representing a building. The agent is initially at location 0. Gray cells represent offices, the dark blue cell represents the kitchen and white cells represent the hallways. (b) Visualization of the initial portion of the search space for the map on (a). Arrows depict actions: arrows with self-loop represent the action `Move`, gray arrows represent either `Search` or `PickUp`. closed-world assumption, any state variable not presented in (b) is considered false. State $\langle$`at`(4),`searched`(4),`hasObj`$\rangle$ is a goal state.

Object	Location			
	Bathroom	Kitchen	Office	Printer Room
coffee	0.08	**0.72**	0.18	0.02
cup	**0.42**	0.36	0.12	0.10
papers	0.00	0.13	**0.70**	0.17
pen	0.15	0.23	**0.35**	0.27
toner	0.05	0.02	0.06	**0.87**

Table 8.1: Prior probability used in our service robot experiments. These probabilities, obtained using ObjectEval [Samadi et al., 2012], represent the probability of the object being in a given location type. The mode of each prior is shown in bold.

In order to show the effects of each parameter in the solution of the problem, consider the policies π_j, for $j \in \{0, \ldots, r\}$, in which the agent explores the first j offices rows, then explores the kitchen and finally the remaining $r - j$ offices row. For all π_j, the exploration stops once the object is found. For instance, if $p_{\text{office}} = 1$, then the only policy that explores the kitchen is π_0 since no office is explored before the kitchen, and all other policies stop exploring after the first office is visited.

Figure 8.4 shows the average cost of following the policies π_j from the location 0 in the map from Figure 8.3(a). Each plot of Figure 8.4 varies either k, r, p_{kitchen} or p_{office} while fixing the other parameters to $k = 10$, $r = 10$, $p_{\text{kitchen}} = 0.9$, $p_{\text{office}} = 0.1$. Figure 8.4(c) shows that the average cost of π_j, which is exponential in r, since the cost depends on the probability of not finding the object in a sequence of i offices, i.e., $(1 - p_{\text{office}})^i$, which is exponential in r. Also, the optimal policy, i.e., the lowest π_j at any point of the plots, is either exploring the kitchen first (π_0) or all the offices first (π_r) for this example.

8.3 Experiments

We present five different experiments, each of them for a different object over the same map. The objects considered in the experiments are: coffee, cup, papers, pen and toner. The prior distribution of the objects for each location type (Table 8.1) is obtained using ObjectEval [Samadi et al., 2012], a system that infers this information using the web. Also, we consider that the object is never in the hallways, i.e., $p_{\text{hallway}} = 0$.

For all the experiments, we consider the map depicted in Figure 8.5. The graph representing this map contains 126 edges and 121 nodes, i.e., locations: 2 bathrooms, 2 kitchens, 59 offices, 1 printer room and 57 segments of hallway. Since there is no location in which any of the considered objects can be found with probability 1, then, with positive probability, the object

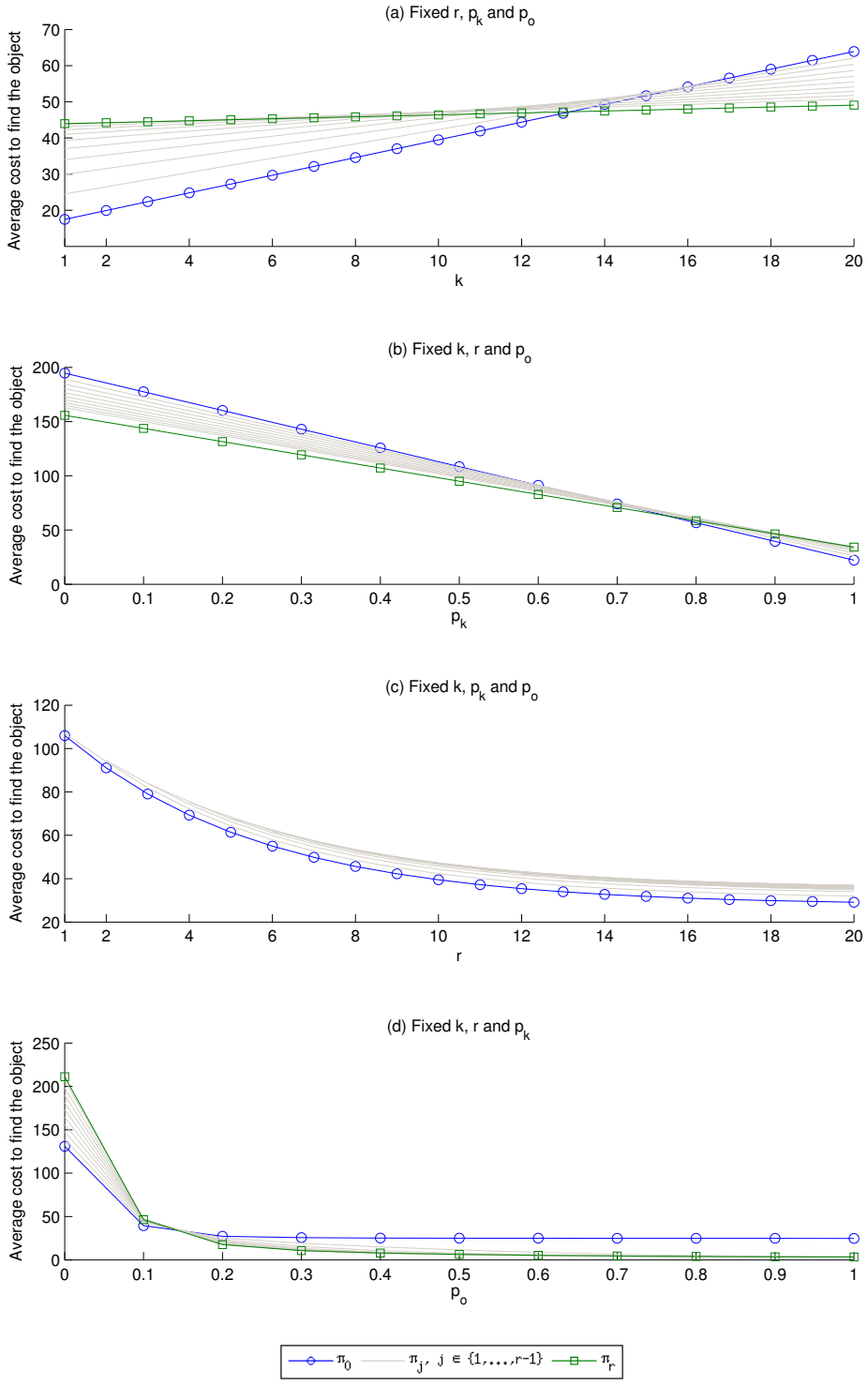


Figure 8.4: Average cost of the policies π_j in the map depicted in Figure 8.3(a). The parameters used are: $k = 10$, $r = 10$, $p_{\text{kitchen}} = 0.9$ and $p_{\text{office}} = 0.1$. In each plot, one of the four parameters is varied in the x-axis. In all plots, the best policy (bottom curve) is either π_0 (explore the kitchen and then the offices) or π_r (explore the offices and then the kitchen). In plot (c), the policy π_r varies as a function of r, the x-axis, and is depicted in gray for clarity.

Figure 8.5: Floor plan used in our service robot experiments. The embedded graph represents the map given to the planners. The initial location for the experiments are represented by the numbers 1,. . . ,10.

might not be found after visiting all locations. This probability is approximately 5×10^{-7}, 6×10^{-5}, 9×10^{-32}, 2×10^{-12} and 3×10^{-3} for *coffee*, *cup*, *papers*, *pen* and *toner*, respectively. The simulations in which this low probability event happens are ignored and rerun.

The planners considered in the experiments are FF-Replan (Algorithm 2.2), UCT (Section 6.4) and SSiPP (Algorithm 3.2). For the latter two, we use the FF-heuristic h_{ff}: for a given state s, $h_{\text{ff}}(s)$ equals the number of actions in the plan returned by FF using s as initial state and the all-outcomes determinization. For UCT, we considered 12 different parametrizations obtained by using the bias parameter $c \in \{1, 2, 4, 8\}$ and the number of samples per decision $w \in \{10, 100, 1000\}$. For SSiPP, we used LRTDP as ϵ-OPTIMAL-SSP-SOLVER and depth-based short-sighted SSPs for $t \in \{2, 4, 6, \cdots, 20\}$. The experiments were conducted in a 3.07GHz machine with 4 cores running a 32-bit version of Linux. A cutoff of 10 minutes of CPU time and 3GB of memory was applied to each planner.

The methodology for the experiments is as follows: each planner solves the same problem, i.e., find a giving object from a particular initial location, 100 times. Learning is not allowed, that is, SSiPP and UCT cannot use the bounds obtained in previous solutions of the same problem to improve their performance. Table 8.2 presents the results as the average and 95% confidence interval of number of actions performed in each problem; For ease of presentation, only the best 3 parametrizations of UCT and best 6 parametrizations of SSiPP are shown.

Overall, SSiPP performs better than the other planners in 55 problems out of 60 (approximately 92%) while the FF-Replan and UCT are the best planner in 3 and 4 problems respectively. Another clear trend is that as t increases for SSiPP, the better is its performance. This is expected since the behavior of SSiPP approaches the behavior of its underlying ϵ-optimal planner, in this case LRTDP, as t increases. However, this improvement in performance is obtained by increasing the search space and consequently the running time of SSiPP. This trade-off between performance and computational time is shown in Figure 8.6 where the run time of the planners is presented.

Looking at specific objects and their priors, we can categorize the objects into: abundant, uniformly distributed and rare. An example of an abundant object in the experiments is *papers* since its prior is 0.7 for office locations and offices represent 48% of the locations. Thus, the probability of not finding *papers* is the lowest between all the object considered: approximately 9×10^{-32}. Therefore, finding objects of this category is not a hard task and optimistic approaches, such as FF-Replan, perform well. This effect is illustrated by the results in third block of Table 8.2 in which the 95% confidence interval of every planner considerably overlaps. A similar phenomenon happens with uniformly distributed objects, i.e., objects in which their prior is *close* to an uniform distribution, represented in the experiments by *pen*.

A more challenging problem is posed by rare objects, i.e., objects in which their prior probability is concentrated in very few locations. In this experiment, *coffee*, *cup* and *toner* can be seen as rare objects. As expected, FF-Replan performs poorly for rare objects and extra reasoning is necessary in order to efficiently explore the state space. For instance, consider finding the object *cup* starting at position 7 (Figure 8.6). Both a kitchen and an office are 3 steps away from position 7. In the all-outcomes determinization used by FF-Replan, the planner will have access to a deterministic action that always finds *cup* in the office and in the kitchen, therefore FF-Replan will randomly break the tie between exploring the kitchen and the neighboring office from position 7. If the office is explored, then FF-Replan will explore all the other offices in the hallway between positions 7 and 3 because they will be the closest locations not explored yet. Since the prior for *cup* is 0.12 for offices, a better policy is to explore the kitchen (prior 0.36) and then the two bathrooms (prior 0.42) that are at distance 4 and 5 of the kitchen.

			UCT $w = 1000$			SSiPP					
	l_0	FF-Replan	$c = 2$	$c = 4$	$c = 8$	$t = 10$	$t = 12$	$t = 14$	$t = 16$	$t = 18$	$t = 20$
coffee	1	17.9 ±3	18.8 ±7	19.9 ±7	23.1 ±9	19.9 ±4	18.5 ±3	19.3 ±4	20.8 ±3	20.7 ±3	**17.2 ±3**
	2	19.4 ±3	18.0 ±7	23.3 ±8	22.2 ±9	14.2 ±3	13.3 ±2	14.1 ±3	13.6 ±3	**12.9 ±2**	13.0 ±2
	3	13.7 ±5	12.3 ±5	13.4 ±8	11.4 ±5	9.7 ±2	10.5 ±3	8.6 ±2	8.1 ±2	8.8 ±2	**8.0 ±2**
	4	18.5 ±4	17.2 ±9	18.1 ±10	16.2 ±4	12.2 ±3	13.1 ±3	12.2 ±3	11.9 ±2	**11.3 ±2**	12.1 ±2
	5	14.5 ±3	14.9 ±4	15.1 ±4	14.7 ±10	14.9 ±2	15.4 ±3	16.7 ±4	17.2 ±3	**13.6 ±2**	14.2 ±2
	6	21.6 ±3	22.7 ±8	24.4 ±10	23.2 ±11	19.1 ±4	21.7 ±4	19.5 ±4	19.1 ±4	18.7 ±2	**18.0 ±4**
	7	21.3 ±4	37.6 ±9	36.3 ±11	34.1 ±12	25.8 ±4	20.5 ±4	21.2 ±4	**20.0 ±3**	20.8 ±4	20.8 ±3
	8	17.3 ±4	27.7 ±9	22.1 ±9	31.5 ±8	13.9 ±3	**13.0 ±2**	14.5 ±4	14.4 ±3	12.9 ±3	15.0 ±3
	9	15.2 ±4	14.0 ±6	17.8 ±7	18.7 ±6	11.6 ±3	**10.0 ±3**	13.0 ±3	11.5 ±3	**10.0 ±3**	11.8 ±3
	10	20.5 ±5	17.3 ±8	25.9 ±9	23.4 ±7	20.8 ±4	16.4 ±3	15.1 ±3	15.9 ±3	**14.1 ±3**	14.3 ±3
cup	1	28.4 ±5	38.9 ±9	34.7 ±8	31.9 ±8	35.4 ±4	30.8 ±5	29.8 ±6	30.1 ±7	29.3 ±7	**27.6 ±5**
	2	33.5 ±6	31.8 ±9	27.0 ±9	26.6 ±8	30.7 ±6	25.6 ±5	26.4 ±6	27.7 ±6	**23.3 ±4**	24.0 ±5
	3	27.6 ±4	30.8 ±9	33.8 ±10	29.4 ±10	25.5 ±6	26.2 ±7	23.9 ±5	19.5 ±5	**17.7 ±3**	19.1 ±4
	4	34.0 ±6	41.6 ±10	48.6 ±9	35.5 ±9	25.9 ±5	23.0 ±5	23.8 ±5	**22.9 ±4**	24.5 ±5	24.5 ±5
	5	30.5 ±5	35.5 ±8	36.8 ±8	42.3 ±9	29.6 ±6	25.0 ±5	28.4 ±6	25.2 ±4	27.3 ±5	**24.3 ±4**
	6	30.3 ±6	41.5 ±9	37.1 ±9	33.7 ±8	34.3 ±6	28.7 ±5	24.1 ±5	26.2 ±5	**20.7 ±3**	21.1 ±4
	7	28.1 ±5	30.6 ±8	35.8 ±8	33.3 ±8	34.5 ±6	23.8 ±5	22.9 ±4	29.2 ±7	**21.6 ±5**	23.2 ±6
	8	35.4 ±7	20.7 ±9	24.5 ±10	21.9 ±11	24.1 ±6	21.9 ±4	19.9 ±6	20.9 ±5	**18.0 ±4**	20.8 ±5
	9	35.9 ±8	29.3 ±10	25.6 ±8	26.4 ±9	29.3 ±7	19.3 ±6	19.9 ±6	15.9 ±4	15.5 ±5	**15.3 ±3**
	10	31.4 ±6	37.4 ±10	23.7 ±10	27.6 ±8	23.3 ±4	27.6 ±6	22.4 ±4	24.0 ±5	20.9 ±4	**20.8 ±4**
papers	1	3.3 ±1	3.2 ±1	3.9 ±1	3.9 ±2	**3.2 ±0**	3.6 ±1	**3.2 ±0**	3.8 ±1	3.3 ±0	3.6 ±1
	2	3.7 ±1	3.7 ±1	3.1 ±1	4.4 ±1	4.0 ±1	3.7 ±1	4.2 ±1	3.5 ±1	3.8 ±1	**3.4 ±1**
	3	4.4 ±1	4.9 ±1	4.4 ±1	4.8 ±1	3.7 ±1	**3.5 ±1**	3.8 ±1	3.8 ±1	**3.5 ±1**	3.6 ±1
	4	4.4 ±1	4.3 ±1	4.7 ±1	4.9 ±3	3.6 ±1	3.7 ±1	**3.5 ±1**	**3.5 ±1**	3.6 ±1	3.7 ±1
	5	3.5 ±1	3.4 ±1	3.9 ±1	**3.3 ±1**	3.7 ±1	3.9 ±1	3.4 ±1	3.9 ±1	3.5 ±1	3.4 ±1
	6	3.6 ±1	3.7 ±1	3.9 ±1	3.8 ±1	3.5 ±1	3.5 ±1	3.9 ±1	3.6 ±1	**3.4 ±1**	3.6 ±1
	7	5.9 ±1	6.4 ±1	6.2 ±1	6.0 ±1	6.0 ±1	6.1 ±1	6.0 ±1	**5.8 ±1**	6.2 ±1	**5.8 ±1**
	8	4.7 ±1	3.9 ±1	**3.5 ±1**	3.8 ±1	4.4 ±1	**3.5 ±1**	3.9 ±1	3.6 ±1	3.6 ±1	3.7 ±1
	9	4.8 ±1	**3.5 ±1**	3.7 ±1	4.0 ±1	4.0 ±1	**3.5 ±1**	3.9 ±1	3.8 ±1	3.8 ±1	3.8 ±1
	10	3.4 ±0	3.3 ±1	4.1 ±2	3.5 ±1	**3.2 ±1**	3.3 ±0	3.5 ±1	3.4 ±1	3.7 ±1	3.5 ±1
pen	1	9.4 ±2	9.1 ±3	8.7 ±3	9.3 ±4	9.0 ±2	10.2 ±2	8.7 ±2	8.5 ±2	9.1 ±2	**8.4 ±1**
	2	8.8 ±2	8.9 ±4	9.0 ±2	8.7 ±3	9.8 ±2	9.2 ±2	9.8 ±2	**8.5 ±1**	8.9 ±2	8.9 ±2
	3	8.5 ±1	10.8 ±3	10.8 ±3	12.0 ±3	9.5 ±2	8.2 ±2	9.5 ±2	8.9 ±2	8.7 ±2	**7.8 ±1**
	4	**8.2 ±2**	9.6 ±3	10.4 ±3	9.1 ±3	9.2 ±2	8.3 ±2	9.0 ±2	8.7 ±3	9.0 ±2	8.5 ±2
	5	**8.7 ±2**	9.6 ±3	8.6 ±2	9.7 ±5	9.6 ±1	9.9 ±2	8.8 ±2	9.0 ±2	9.4 ±2	9.1 ±2
	6	11.1 ±3	11.0 ±3	11.7 ±2	10.8 ±3	11.0 ±2	10.7 ±1	10.6 ±2	**10.0 ±2**	10.1 ±2	**10.0 ±2**
	7	**10.9 ±2**	11.7 ±3	11.9 ±3	11.4 ±4	11.4 ±2	11.1 ±2	11.2 ±2	11.3 ±2	11.2 ±2	11.5 ±2
	8	10.7 ±2	10.4 ±3	10.9 ±2	10.5 ±3	**10.1 ±2**	11.8 ±2	8.6 ±2	10.8 ±2	10.4 ±2	10.2 ±2
	9	11.3 ±2	10.4 ±3	10.6 ±3	10.9 ±4	10.2 ±2	10.9 ±2	10.8 ±2	10.9 ±2	**10.0 ±2**	10.9 ±2
	10	9.7 ±2	**9.3 ±2**	9.9 ±2	9.7 ±2	9.4 ±2	9.8 ±2	9.5 ±2	9.6 ±2	9.9 ±2	9.5 ±2
toner	1	54.1 ±9	43.2 ±10	41.9 ±11	41.3 ±11	42.8 ±7	29.5 ±7	27.2 ±5	37.9 ±7	**27.1 ±6**	27.9 ±6
	2	56.8 ±9	41.9 ±10	45.7 ±12	40.3 ±11	41.5 ±5	19.0 ±5	**18.3 ±5**	18.7 ±5	18.5 ±6	**18.3 ±6**
	3	50.1 ±9	56.6 ±12	55.3 ±11	53.1 ±13	38.5 ±5	33.1 ±6	25.3 ±6	22.4 ±4	23.4 ±9	**21.2 ±5**
	4	61.3 ±9	59.3 ±10	58.0 ±12	42.2 ±11	30.2 ±9	20.7 ±6	20.5 ±6	**19.1 ±7**	21.3 ±7	19.3 ±7
	5	39.3 ±6	38.9 ±10	31.5 ±10	36.5 ±12	30.2 ±7	31.8 ±8	23.9 ±5	**23.2 ±6**	25.0 ±7	23.6 ±7
	6	53.3 ±6	37.5 ±11	29.8 ±7	23.1 ±6	18.6 ±6	19.6 ±4	19.0 ±5	18.9 ±6	**18.4 ±4**	18.6 ±6
	7	45.5 ±7	26.4 ±10	20.7 ±8	21.2 ±7	18.3 ±5	17.9 ±5	18.0 ±6	18.4 ±7	**17.6 ±7**	17.9 ±5
	8	33.9 ±8	21.5 ±10	19.8 ±12	18.7 ±9	23.4 ±10	19.7 ±9	18.8 ±6	16.7 ±8	**16.2 ±8**	17.1 ±7
	9	36.8 ±8	29.9 ±10	25.9 ±10	23.6 ±9	18.5 ±8	17.6 ±6	18.8 ±7	18.3 ±9	16.6 ±6	**16.2 ±5**
	10	54.5 ±8	31.5 ±9	29.5 ±7	27.6 ±10	27.8 ±6	25.1 ±6	23.0 ±6	24.1 ±7	22.6 ±7	**22.1 ±6**

Table 8.2: Performance of different planners in the service robot experiments. Each cell represents the average and 95% confidence interval of the number of actions applied to find the given object starting at location l_i (Figure 8.5). Bold font shows the best performance planner for the given problem, i.e., the combinations of objects and initial locations represented by each line of the table.

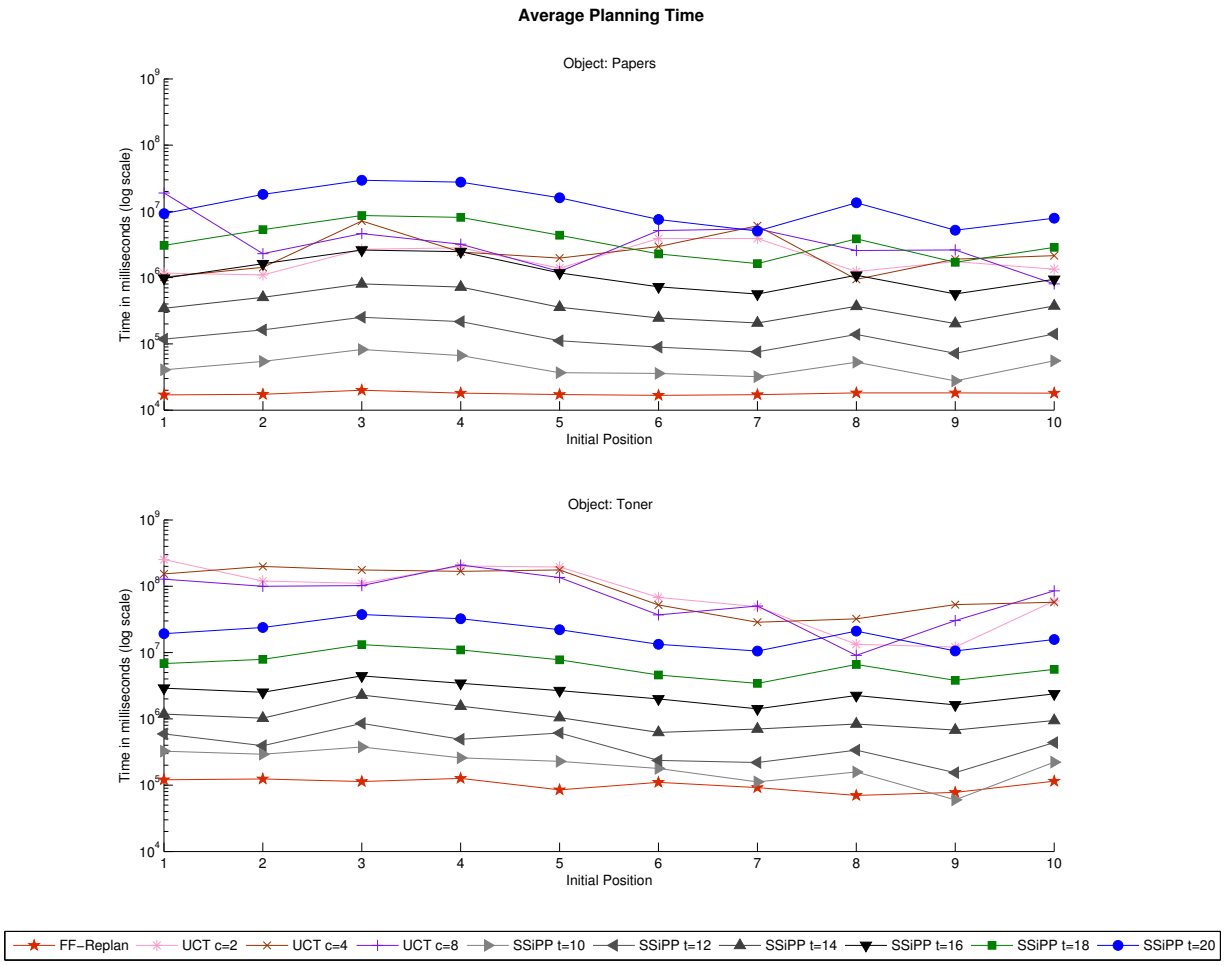


Figure 8.6: Average run time for the planners to find the objects *papers* and *toner* in our service robot problem. The y-axis is in log-scale and its unit is milliseconds. Error bars omitted for clarity. The plot for the other objects follows a similar pattern, with SSiPP for $t = 12$ always faster than UCT planners for $w = 1000$.

The improvement in performance over FF-Replan is remarkable for the rare object *toner*, that can be found with probability 0.87 in one single location, the printer room. For these problems, both UCT and SSiPP present better performance than FF-Replan and the average number of actions applied by SSiPP, for $t \geq 14$, is approximately half of the average number of actions applied by FF-Replan. Moreover, for the *toner* problems, the best SSiPP parametrization (i.e., $t = 20$) solves the problem using from 39.9% to 91.1% of the total actions used by the best parametrization of UCT ($w = 1000$ and $c = 8$).

8.4 Summary

In this chapter, we presented how to solve the problem of a software or robotic agent moving in a known environment in order to find an object using SSPs encoded in PPDDL, a standard probabilistic planning language. We empirically compared three different replanning approaches to solve the proposed problems: determinizations (FF-Replan), sampling (UCT) and short-sighted planning (SSiPP). The experiments showed that the simpler and optimistic approach used by FF-Replan suffices if the object can be found in most locations with high probability or nearly uniform across over all locations. Alternatively, if the probability of finding the object is concentrated in few locations, then SSiPP outperforms the other approaches and, for some parametrizations, SSiPP executes on average less than half of the actions executed by FF-Replan to find the desired object.

It is important to notice that all the planners compared in this chapter are domain-independent planners. Due to the strong geometric constraints in robotics applications, most real world robots use *domain-dependent* planners. This class of planners take advantage of domain specific knowledge to prune the search space and to employ more accurate heuristics. For these reasons, it is unlikely that SSiPP (or any other domain-independent planner) will be able to outperform domain-dependent planners in real world robotics problems. Nonetheless, the concept of short-sighted planning could be easily incorporated to domain-dependent planners to improve their performance in probabilistic environments, such as the finding object domain presented in this chapter.

Chapter 9

Conclusion

This dissertation sets out to address question,

> How to plan for probabilistic environments such that it scales up while offering formal guarantees underlying policy generation?

This final chapter summarizes the contributions we have presented to answer this question. We also describe some new directions for future work that this thesis raises.

9.1 Contributions

The contributions of this thesis can be grouped into four classes:

1. **Short-Sighted Models**
   We introduced the concept of short-sighted probabilistic planning problems, a special case of probabilistic planning problems in which the state space is pruned and actions are not simplified. Three short-sighted models, based on different criteria to prune the state space, were presented: depth-based short-sighted problems, in which all the states are reachable using no more than a given number of actions; trajectory-based short-sighted problems, in which all states are reachable with probability greater or equal than a given threshold; and greedy short-sighted problems, in which the states have the best trade-off between probability of being reached and expected cost to reach the goal from them.

2. **Short-Sighted Probabilistic Planners**
   We introduced the Short-Sighted Probabilistic Planner (SSiPP) algorithm that solves probabilistic planning problems by iteratively generating and solving short-sighted subproblems. We also presented three extensions of SSiPP: Labeled-SSiPP, which improves the convergence of SSiPP to the ϵ-optimal solution; Parallel Labeled-SSiPP, which solves mul-

tiple short-sighted problems in parallel to speedup the search for the ϵ-optimal solution; and SSiPP-FF, which improves the efficiency of SSiPP when a suboptimal solution is acceptable.

3. **Theoretical Analysis**
   We proved that the optimal solution of short-sighted subproblems are lower bounds for the original probabilistic planning problem associated with them. Moreover, we showed that solutions for depth-based short-sighted subproblems can be executed for at least t steps, where t is a parameter, in the original problem without replanning. We proved that SSiPP, Labeled-SSiPP and Parallel Labeled-SSiPP are asymptotically optimal and derived an upper bound on the number of iterations necessary for Labeled-SSiPP and Parallel Labeled-SSiPP to converge to the ϵ-optimal solution.

4. **Empirical Evaluation**
   We provided a rich empirical evaluation of the proposed algorithms for two different tasks: (i) to find an ϵ-optimal solutions, and (ii) to compute a solution under the International Probabilistic Planning Competition [Younes et al., 2005, Bonet and Givan, 2007, Bryce and Buffet, 2008] rules. Several domains were used in our empirical evaluation, including domains proposed in this thesis and benchmarks from the probabilistic planning community. We also empirically showed how a mobile service robot moving in a building in order to find an object can use short-sighted planning to improve its performance.

9.2 Directions for Future Work

This thesis opens up new interesting directions for further research in probabilistic planning. Moreover, short-sighted planning is a general concept that can be applied to any planning under uncertainty model. Next, we enumerate a number of directions for future work.

9.2.1 Automatically Choosing a Short-Sighted Model and its Parameters

Short-sighted SSPs can exploit the underlying structure of the problem through their different simplifications of the state space and parameters, e.g., the parameter t for depth-based short-sighted SSPs and ρ for trajectory-based short-sighted SSPs. Our experiments show that the performance of SSiPP and its extensions can be further improved by optimizing the choice of short-sighted model used and its parameters for each domain.

A future direction is to derive (heuristic) methods that automatically choose or adapt the short-sighted model and its parameters for the current SSP being solved. One approach to tackle

this problem is to model it as a multi-armed bandit problem in which the combinations of short-sighted models and their parameters are different arms.

A different approach is to perform automatic domain analysis. This technique has been successfully applied to automatically elicit knowledge implicitly embedded in the domain, e.g., generation of state constraint [Gerevini and Schubert, 1998, Gerevini and Schubert, 2000, Hoffmann, 2011] and removal of irrelevant fact and actions [Nebel et al., 1997, Haslum and Jonsson, 2000, Haslum, 2007]. It would be interesting to explore what features can be extracted from preprocessing the domain that can guide, or constraint, the choice of short-sighted model and its parameters.

9.2.2 Transfer Learning using Short-Sighted Problems

Transfer learning for probabilistic planning can be seen as the problem of solving an SSP $\mathbb{S}$ by reusing policies for similar SSPs. Formally, let $\pi^*_{\mathbb{S}}$ be an optimal policy for $\mathbb{S}$ and define a new SSP $\mathbb{S}$' in which only the set of goal states G differs between $\mathbb{S}$ and $\mathbb{S}$'. In this case, $\mathbb{S}$' has a different optimal value function $V^*_{\mathbb{S}'}$ that, most likely, yields to optimal policies $\pi^*_{\mathbb{S}'}$ different from $\pi^*_{\mathbb{S}}$. Transfer learning aims to use $\pi^*_{\mathbb{S}}$ to guide the learning of $V^*_{\mathbb{S}'}$ and thus speed up the search for $\pi^*_{\mathbb{S}'}$ [Fernández and Veloso, 2006].

Although $\pi^*_{\mathbb{S}}$ and $\pi^*_{\mathbb{S}'}$ can be different, the policy for some of the short-sighted SSPs used during the solution of both $\mathbb{S}$ and $\mathbb{S}'$ might still be the same. This is potentially interesting for problems that share states that must always be visited in both in order to compute their optimal solutions, e.g., the intermediary doors in the hallway problems (Example 5.1 on page 53).

It would be interesting to explore how the solutions of different short-sighted SSPs are affected when the goal of the original SSP is changed. Another step in this direction is the analysis of the necessary conditions of SSPs in order to be able to efficiently reuse the optimal policies of their associated short-sighted SSPs.

9.2.3 Short-Sighted Planning for Imprecise Probabilistic Problems

In many real-world problems, it is not possible to obtain a precise representation of the transition probabilities in order to use probabilistic planning models. This may occur for many reasons, including imprecise or conflicting elicitations from experts, insufficient data from which to estimate precise transition models, or non-stationary transition probabilities due to insufficient state information.

Several models were proposed [Satia and Lave Jr, 1973, Givan et al., 2000, Trevizan et al., 2007, Delgado et al., 2011] to handle this uncertainty in the transition probabilities and their

drawback is the increased computational complexity to find an optimal policy. Notice that the previously proposed problem relaxations for probabilistic planning do not obtain robust solutions for imprecise probabilistic problems. For instance, solutions obtained using determinizations completely ignore the extra information regarding the imprecise probabilities of actions and relax them to deterministic actions.

Alternatively, the extension of short-sighted planning to imprecise probabilistic planning problems has potential to efficiently compute robust solutions since the structure of actions are not simplified. Therefore, short-sighted models for imprecise probabilistic problems would be able to represent both loops in the states and, more importantly, the imprecision in the action representation, e.g., a probability interval for each effect. In order to extend short-sighted planning to imprecise probabilistic problems, two steps are necessary: to define short-sighted imprecise models, and to extend SSiPP to handle imprecise probabilistic problems.

9.2.4 Short-Sighted Decentralized SSPs with Sparse Interactions

One assumption of SSPs (and MDPs) is that there is only one single agent executing actions and thus modifying the environment. If more than one agent is modifying the environment, i.e., a multi-agent problem, then SSPs need to be generalized to encompass the interaction between agents. One possible approach to model such problems is to assume joint-observability, i.e., each agent is aware of the state and actions performed by all other agents, which seldom holds in practice. If joint-observability is completely ignored, then finding the optimal policy for even the case where agents share the same cost function is undecidable [Bernstein et al., 2002].

In practice, joint-observability is only required in specific parts of the environment, i.e., the interaction between agents is sparse [Melo and Veloso, 2009, Melo and Veloso, 2011]. One example of sparse interactions is two or more service robots navigating in a building (Figure 9.1). These robots coordinate their actions during navigation only when they need to pass through the same doors or a narrow hallway. More generally, coordination is required between agents only in regions of the state space in which: (i) there is a conflict of resource; or (ii) direct interaction is needed in order to achieve a goal.

One novel approach to solve sparse interaction problems would be to use short-sighted probabilistic planning. The benefits of using short-sighted models for this class of problems is that the local interactions can be perfectly modeled while future and unlikely interactions can be approximated. Besides extending SSiPP in order to handle multi-agent interactions, this novel approach also requires the proposal of new short-sighted models to remove and heuristically approximate unlikely interactions between agents.

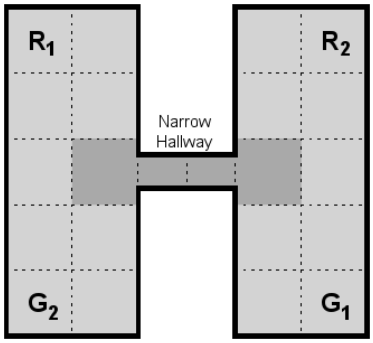


Figure 9.1: Example of sparse-interaction multi-agent planning problem. Two robots, R_1 and R_2, have to navigate in the depicted map to reach their goal locations, G_1 and G_2 respectively. Coordination between R_1 and R_2 is only need if and when both try to cross the narrow hallway at the same time. Figure adapted from [Melo and Veloso, 2009].

9.2.5 Short-Sighted Partially Observable Probabilistic Problems

Partially Observable MDPs (POMDPs) generalize MDPs (Section 2.1) by modeling agents that have incomplete state information [Sondik, 1971]. A common approach to solve POMDPs is to convert them to belief MDPs, i.e., an MDP in the belief space, and RTDP (Section 2.3.1) can be applied to solve the obtained belief MDPs [Bonet and Geffner, 2009]. This adaptation of RTDP, RTDP-Bel, handles the continuous state space of the belief MDPs by using function approximations [Bertsekas and Tsitsiklis, 1996], specifically by discretizing the belief space into a finite grid.

The main drawback of the representation using function approximations is that convergence is no longer guaranteed. However, in practice, RTDP-Bel performance is comparable with state-of-the-art POMDP solvers and outperforms them in domains such as RockSample and LifeSurvey [Bonet and Geffner, 2009]. An interesting future direction is to use RTDP-Bel as the optimal solver for SSiPP and apply the proposed short-sighted models in this thesis to model subproblems of belief MDPs. New definitions of short-sighted models that are specific for belief MDPs might be necessary in order to make this approach feasible.

9.3 Summary

This thesis contributes a number of techniques to effectively solve probabilistic planning problems. The cornerstone of the presented algorithms is the concept of short-sighted problems, a novel approach to relax probabilistic planning problems. We proved the relationship between solutions of short-sighted subproblems and the original probabilistic planning problem associated

with them, as well as, the main properties of our algorithms, e.g., optimality. We demonstrated the effectives of our presented algorithms and different short-sighted models in a rich empirical comparison against state-of-the-art probabilistic planners in several domains.

Appendix A

Proof of Lemmas 3.1 and 3.2

Proof of Lemma 3.1. If $\hat{s} \in \mathsf{S}_{s,t} \cap \mathsf{G}$, then $(B^k_{s,t}V)(\hat{s}) = (B^kV)(\hat{s}) = 0$ 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 3.2), we can replace $C_{s,t}$ by C in $(B_{s,t}V)(\hat{s})$ as follows:

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

  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,

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

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.,

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

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

□

Proof of Lemma 3.2. 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 $(B_{s,t}^k V)(\hat{s}) = (B^k V)(\hat{s}) = 0$; and (ii) if $\hat{s} \in \mathsf{G}_a$, then $(B_{s,t}^k V)(\hat{s}) = 0 \leq (B^k V)(\hat{s})$. 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 $(B_{s,t}^k V)(s) = (B^k V)(s)$ by Lemma 3.1. We prove the other case, i.e., $m > k$, by induction on $i = k - m$:

- If $i = 1$, then $(B_{s,t}^k V)(s) = (B_{s,t}(B_{s,t}^{k-1}V))(s) = (B_{s,t}(B_{s,t}^m V))(s)$ thus, by Lemma 3.1,

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

  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, $(B_{s,t}^k V)(s) = (B_{s,t}(B^m V))(s) \leq (B^k V)(s)$.

- 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

$$\begin{aligned}
(B_{s,t}^k V)(s) &= (B_{s,t}(B_{s,t}^{m+c}(V))(s) \\
&= \min_a \sum_{s' \in \mathsf{S}_{s,t} \setminus \mathsf{G}_a} P(s'|\hat{s},a) \left[C(\hat{s},a,s') + (B_{s,t}^{m+c} V)(s')\right] \\
&\qquad + \sum_{s' \in \mathsf{G}_a} P(s'|\hat{s},a) \left[C(\hat{s},a,s') + V(s')\right].
\end{aligned}$$

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, $(B_{s,t}^{m+c}V)(s') \leq (B^{m+c}V)(s')$. Thus,

$$\begin{aligned}(B_{s,t}^{k}V)(s) &\leq \min_{a} \sum_{s' \in \mathsf{S}_{s,t}} P(s'|\hat{s},a)\left[C(\hat{s},a,s') + (B^{m+c}V)(s')\right] \\ &= \min_{a} \sum_{s' \in \mathsf{S}} P(s'|\hat{s},a)\left[C(\hat{s},a,s') + (B^{m+c}V)(s')\right],\end{aligned}$$

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

□

Bibliography

[Archibald et al., 1993] Archibald, T., McKinnon, K., and Thomas, L. (1993). Serial and Parallel Value Iteration Algorithms for Discounted Markov Decision Processes. *European Journal of Operational Research*, 67(2):188–203.

[Archibald et al., 1995] Archibald, T., McKinnon, K., and Thomas, L. (1995). Performance Issues for the Iterative Solution of Markov Decision Processes on Parallel Computers. *INFORMS Journal on Computing*, 7(3):349–357.

[Barto et al., 1995] Barto, A., Bradtke, S., and Singh, S. (1995). Learning to Act Using Real-Time Dynamic Programming. *Artificial Intelligence*, 72(1-2):81–138.

[Bernstein et al., 2002] Bernstein, D., Givan, R., Immerman, N., and Zilberstein, S. (2002). The Complexity of Decentralized Control of Markov Decision Processes. *Mathematics of Operations Research*, pages 819–840.

[Bertsekas, 1995] Bertsekas, D. (1995). *Dynamic Programming and Optimal Control*. Athena Scientific.

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

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

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

[Bonet and Geffner, 2009] Bonet, B. and Geffner, H. (2009). Solving POMDPs: RTDP-Bel Vs. Point-Based Algorithms. In *Proc. of the 21st Int. Joint Conf. on Artificial Intelligence (IJCAI)*.

[Bonet and Givan, 2007] Bonet, B. and Givan, R. (2007). 2th International Probabilistic Planning Competition (IPPC-ICAPS). `http://www.ldc.usb.ve/~bonet/ipc5/` (accessed on May 12, 2013).

[Boutilier et al., 1999] Boutilier, C., Dean, T., and Hanks, S. (1999). Decision-Theoretic Planning: Structural Assumptions and Computational Leverage. *Journal of Artificial Intelligence Research*, 11:1–94.

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

[Burns et al., 2009] Burns, E., Lemons, S., Ruml, W., and Zhou, R. (2009). Suboptimal and

Anytime Heuristic Search on Multi-Core Machines. In *Proc. of the 19th Int. Conf. on Automated Planning and Scheduling (ICAPS)*.

[Burns et al., 2010] Burns, E., Lemons, S., Ruml, W., and Zhou, R. (2010). Best-First Heuristic Search for Multicore Machines. *Journal of Artificial Intelligence Research*, 39(1):689–743.

[Coles et al., 2012] Coles, A. J., Coles, A., García Olaya, A., Jiménez, S., Linares López, C., Sanner, S., and Yoon, S. (2012). A Survey of the Seventh International Planning Competition. *AI Magazine*, 33(1):83–88.

[Dai and Goldsmith, 2007] Dai, P. and Goldsmith, J. (2007). Topological Value Iteration Algorithm for Markov Decision Processes. In *Proc. of the 20th Int. Joint Conf. on Artificial Intelligence (IJCAI)*.

[Dai et al., 2009] Dai, P., Weld, D. S., et al. (2009). Focused Topological Value Iteration. In *Proc. of the 19th Int. Conf. on Automated Planning and Scheduling (ICAPS)*.

[Dean et al., 1995] Dean, T., Kaelbling, L., Kirman, J., and Nicholson, A. (1995). Planning Under Time Constraints in Stochastic Domains. *Artificial Intelligence*, 76(1-2):35–74.

[Delgado et al., 2011] Delgado, K., Sanner, S., and De Barros, L. (2011). Efficient Solutions to Factored MDPs with Imprecise Transition Probabilities. *Artificial Intelligence*.

[Fernández and Veloso, 2006] Fernández, F. and Veloso, M. (2006). Probabilistic Policy Reuse in a Reinforcement Learning Agent. In *Proc. of the 5th Int. Joint Conf. on Autonomous Agents and Multiagent Systems (AAMAS)*.

[Gerevini and Schubert, 1998] Gerevini, A. and Schubert, L. (1998). Inferring State Constraints for Domain-Independent Planning. In *Proc. of the 15th AAAI Conf. on Artificial Intelligence (AAAI)*.

[Gerevini and Schubert, 2000] Gerevini, A. and Schubert, L. K. (2000). Discovering State Constraints in DISCOPLAN: Some New Results. In *Proc. of the 17th AAAI Conf. on Artificial Intelligence (AAAI)*.

[Givan et al., 2000] Givan, R., Leach, S. M., and Dean, T. (2000). Bounded-Parameter Markov Decision Processes. *Artificial Intelligence*, 122(1-2):71–109.

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

[Haslum, 2007] Haslum, P. (2007). Reducing Accidental Complexity in Planning Problems. In *Proc. of the 20th Int. Joint Conf. on Artificial Intelligence (IJCAI)*, pages 1898–1903.

[Haslum et al., 2007] Haslum, P., Botea, A., Helmert, M., Bonet, B., and Koenig, S. (2007). Domain-Independent Construction of Pattern Database Heuristics for Cost-Optimal Planning. In *Proc. of the 22th AAAI Conf. on Artificial Intelligence (AAAI)*.

[Haslum and Jonsson, 2000] Haslum, P. and Jonsson, P. (2000). Planning with Reduced Operator Sets. In *Proc. of the 5th Conf. on Artificial Intelligence Planning Systems (AIPS)*.

[Helmert, 2006] Helmert, M. (2006). The Fast Downward Planning System. *Journal of Artificial Intelligence Research*, 26:191–246.

[Hoffman et al., 2004] Hoffman, J., Porteous, J., and Sebastia, L. (2004). Ordered Landmarks

in Planning. *Journal of Artificial Intelligence Research*, 22:215–278.

[Hoffmann, 2011] Hoffmann, J. (2011). Analyzing Search Topology Without Running Any Search: On the Connection Between Causal Graphs and H+. *Journal of Artificial Intelligence Research*, 41(2):155–229.

[Hoffmann and Nebel, 2001] Hoffmann, J. and Nebel, B. (2001). The FF Planning System: Fast Plan Generation Through Heuristic Search. *Journal of Artificial Intelligence Research*, 14(1):253–302.

[Howard, 1960] Howard, R. (1960). *Dynamic Programming and Markov Processes*. MIT Press.

[Keller and Eyerich, 2012] Keller, T. and Eyerich, P. (2012). Probabilistic Planning Based on UCT. In *Proc. of 22nd Int. Joint Conf. on Automated Planning and Scheduling (ICAPS)*.

[Keyder and Geffner, 2008] Keyder, E. and Geffner, H. (2008). The HMDP Planner for Planning with Probabilities. In *3rd Int. Probabilistic Planning Competition (IPPC-ICAPS)*.

[Kishimoto et al., 2009] Kishimoto, A., Fukunaga, A., and Botea, A. (2009). Scalable, Parallel Best-First Search for Optimal Sequential Planning. In *Proc. of the 19th Int. Conf. on Automated Planning and Scheduling (ICAPS)*.

[Kishimoto et al., 2010] Kishimoto, A., Fukunaga, A., and Botea, A. (2010). On the Scaling Behavior of HDA*. In *Proc. of the 3rd Symposium on Combinatorial Search (SoCS)*.

[Kocsis and Szepesvri, 2006] Kocsis, L. and Szepesvri, C. (2006). Bandit Based Monte-Carlo Planning. In *Proc. of the 17th European Conf. on Machine Learning (ECML)*.

[Kolobov et al., 2009] Kolobov, A., Mausam, and Weld, D. S. (2009). ReTrASE: Integrating Paradigms for Approximate Probabilistic Planning. In *Proc. of the 21st Int. Joint Conf. on Artificial Intelligence (IJCAI)*.

[Korf, 1990] Korf, R. E. (1990). Real-Time Heuristic Search. *Artificial intelligence*, 42(2):189–211.

[Little and Thiébaux, 2007] Little, I. and Thiébaux, S. (2007). Probabilistic Planning vs Replanning. In *Proc. of ICAPS Workshop on IPC: Past, Present and Future*.

[McMahan et al., 2005] McMahan, H., Likhachev, M., and Gordon, G. (2005). Bounded Real-Time Dynamic Programming: RTDP with Monotone Upper Bounds and Performance Guarantees. In *Proc. of the 22nd Int. Conf. on Machine Learning (ICML)*.

[Melo and Veloso, 2009] Melo, F. and Veloso, M. (2009). Learning of Coordination: Exploiting Sparse Interactions in Multiagent Systems. In *Proc. of the 8th Int. Joint Conf. on Autonomous Agents and Multiagent Systems (AAMAS)*.

[Melo and Veloso, 2011] Melo, F. and Veloso, M. (2011). Decentralized MDPs with Sparse Interactions. *Artificial Intelligence*, 175(11):1757–1789.

[Nebel et al., 1997] Nebel, B., Dimopoulos, Y., and Koehler, J. (1997). Ignoring Irrelevant Facts and Operators in Plan Generation. In *Proc. of the 4th European Conf. on Planning (ECP)*.

[Newell and Simon, 1963] Newell, A. and Simon, H. (1963). GPS: A Program that Simulates Human Thought. In Feigenbaum, E. and Feldman, J., editors, *Computers and Thought*, pages 279–298. McGraw-Hill Book Company.

[Nourbakhsh and Genesereth, 1996] Nourbakhsh, I. R. and Genesereth, M. R. (1996). Assumptive Planning and Execution: A Simple, Working Robot Architecture. *Autonomous Robots*, 3(1):49–67.

[Papadimitriou and Tsitsiklis, 1987] Papadimitriou, C. H. and Tsitsiklis, J. N. (1987). The Complexity of Markov Decision Processes. *Mathematics of Operations Research*, 12(3):441–450.

[Pearl, 1985] Pearl, J. (1985). *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley, Menlo Park, California.

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

[Rosenthal et al., 2010] Rosenthal, S., Biswas, J., and Veloso, M. (2010). An Effective Personal Mobile Robot Agent Through Symbiotic Human-Robot Interaction. In *Proc. of the 9th Int. Joint Conf. on Autonomous Agents and Multiagent Systems (AAMAS)*.

[Russel and Norvig, 2003] Russel, S. J. and Norvig, P. (2003). *Artificial Intelligence - A Modern Approach*. Prentice Hall, 2nd edition.

[Samadi et al., 2012] Samadi, M., Kollar, T., and Veloso, M. (2012). Using the Web to Interactively Learn to Find Objects. In *Proc. of the 26th AAAI Conf. on Artificial Intelligence (AAAI)*.

[Sanner et al., 2009] Sanner, S., Goetschalckx, R., Driessens, K., and Shani, G. (2009). Bayesian Real-Time Dynamic Programming. In *Proc. of the 21st Int. Joint Conf. on Artificial Intelligence (IJCAI)*.

[Satia and Lave Jr, 1973] Satia, J. K. and Lave Jr, R. E. (1973). Markovian Decision Processes with Uncertain Transition Probabilities. *Operations Research*, 21(3):728–740.

[Smith and Simmons, 2006] Smith, T. and Simmons, R. G. (2006). Focused Real-Time Dynamic Programming for MDPs: Squeezing More Out of a Heuristic. In *Proc. of the 21st AAAI Conf. on Artificial Intelligence (AAAI)*.

[Sondik, 1971] Sondik, E. (1971). *The Optimal Control of Partially Observable Markov Decision Processes*. PhD thesis, Stanford University.

[Sulewski et al., 2011] Sulewski, D., Edelkamp, S., and Kissmann, P. (2011). Exploiting the Computational Power of the Graphics Card: Optimal State Space Planning on the GPU. In *Proc. of the 21st Int. Conf. on Automated Planning and Scheduling (ICAPS)*.

[Teichteil-Koenigsbuch et al., 2008] Teichteil-Koenigsbuch, F., Infantes, G., and Kuter, U. (2008). RFF: A Robust, FF-Based MDP Planning Algorithm for Generating Policies with Low Probability of Failure. *3rd Int. Planning Competition (IPPC-ICAPS)*.

[Teichteil-Königsbuch et al., 2011] Teichteil-Königsbuch, F., Vidal, V., and Infantes, G. (2011). Extending Classical Planning Heuristics to Probabilistic Planning with Dead-Ends. In *Proc. of the 26th AAAI Conf. on Artificial Intelligence (AAAI)*.

[Trevizan et al., 2007] Trevizan, F., Cozman, F., and de Barros, L. (2007). Planning Under Risk and Knightian Uncertainty. In *Proc. of the 20th Int. Joint Conf. on Artificial Intelligence (IJCAI)*.

[Trevizan and Veloso, 2012a] Trevizan, F. and Veloso, M. (2012a). Short-Sighted Stochastic Shortest Path Problems. In *In Proc. of the 22nd Int. Conf. on Automated Planning and Scheduling (ICAPS)*.

[Trevizan and Veloso, 2012b] Trevizan, F. and Veloso, M. (2012b). Trajectory-Based Short-Sighted Probabilistic Planning. In *Advances in Neural Information Processing Systems (NIPS)*.

[Trevizan and Veloso, 2013] Trevizan, F. and Veloso, M. (2013). Depth-Based Short-Sighted Probabilistic Planning. *Artificial Intelligence (to appear)*.

[Veloso and Blythe, 1994] Veloso, M. and Blythe, J. (1994). Linkability: Examining Causal Link Commitments in Partial-Order Planning. In *Proc. of the 2nd Conf. on Artificial Intelligence Planning Systems (AIPS)*.

[Vidal et al., 2010] Vidal, V., Bordeaux, L., and Hamadi, Y. (2010). Adaptive K-Parallel Best-First Search: A Simple but Efficient Algorithm for Multi-Core Domain-Independent Planning. In *Proc. of the 3rd Symposium on Combinatorial Search (SoCS)*.

[Yoon et al., 2007] Yoon, S., Fern, A., and Givan, R. (2007). FF-Replan: A Baseline for Probabilistic Planning. In *Proc. of the 17th Int. Conf. on Automated Planning and Scheduling (ICAPS)*.

[Yoon et al., 2008] Yoon, S., Fern, A., Givan, R., and Kambhampati, S. (2008). Probabilistic Planning Via Determinization in Hindsight. In *Proc. of the 23rd AAAI Conf. on Artificial Intelligence (AAAI)*.

[Yoon et al., 2010] Yoon, S., Ruml, W., Benton, J., and Do, M. B. (2010). Improving Determinization in Hindsight for Online Probabilistic Planning. In *Proc. of the 20th Int. Conf. on Automated Planning and Scheduling (ICAPS)*.

[Younes and Littman, 2004] Younes, H. and Littman, M. (2004). PPDDL 1.0: An Extension to PDDL for Expressing Planning Domains with Probabilistic Effects. Technical Report CMU-CS-04-167, Carnegie Mellon University.

[Younes et al., 2005] Younes, H., Littman, M., Weissman, D., and Asmuth, J. (2005). The 1st Probabilistic Track of the International Planning Competition. *Journal of Artificial Intelligence Research*, 24(1):851–887.

[Zhou and Hansen, 2007] Zhou, R. and Hansen, E. (2007). Parallel Structured Duplicate Detection. In *Proc. of the 22nd AAAI Conf. on Artificial Intelligence (AAAI)*.

[Zhou et al., 2010] Zhou, R., Schmidt, T., Hansen, E., Do, M., and Uckun, S. (2010). Edge Partitioning in Parallel Structured Duplicate Detection. In *Proc. of the 3rd Symposium on Combinatorial Search (SoCS)*.

[Zickler and Veloso, 2010] Zickler, S. and Veloso, M. (2010). Variable Level-Of-Detail Motion Planning in Environments with Poorly Predictable Bodies. In *Proc. of the 19th European Conf. on Artificial Intelligence (ECAI)*.

MACHINE LEARNING
DEPARTMENT

Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA15213

Carnegie Mellon