Heuristic Search Based Planning by Minimizing Anticipated Search Efforts

Ishani Chatterjee

CMU-RI-TR-22-62

September 9, 2022

The Robotics Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA

Thesis Committee:
Maxim Likhachev, *Co-chair*
Manuela Veloso, *Co-chair*
Stephen Smith
Shlomo Zilberstein, *University of Massachusetts Amherst*

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

To my grandparents and parents.

Abstract

We focus on relatively low dimensional robot motion planning problems, such as planning for navigation of a self-driving vehicle, unmanned aerial vehicles (UAVs), and footstep planning for humanoids. In these problems, there is a need for fast planning, potentially compromising the solution quality. Often, we want to plan fast but are also interested in controlling the solution quality because we desire efficient planning. Bounded-suboptimal heuristic search algorithms are a popular alternative to optimal heuristic search algorithms that compromise solution quality for computation speed. Specifically, these searches aim to find a solution that can be computed faster than the optimal solution while guaranteeing that its cost is bounded within a specified factor of the optimal cost (bounded-suboptimality). Currently, most bounded-suboptimal heuristic search algorithms speed up planning by guiding the search using cost-to-goal estimates. Cost-to-goal estimates are not necessarily correlated with the search effort required to reach the goal. The key idea of this thesis is to *explicitly reason about anticipated search efforts required to reach the goal* instead of cost-to-goal estimates, and *achieve higher speedup by guiding the search along solutions that minimize these anticipated search efforts* while ensuring bounded-suboptimality of the computed solutions. Also, bounded-suboptimal heuristic search algorithms have been largely investigated in the context of deterministic planning. In this thesis, we use the key idea of this thesis to speed up planning while ensuring bounded suboptimality in the context of deterministic as well as probabilistic planning.

To this end, our first contribution is to speed up deterministic robot planning problems formulated as bounded-suboptimal heuristic search. Weighted A* (wA*) search is a popular bounded-suboptimal heuristic search algorithm. We investigate the problem of computing heuristics that explicitly aim to reduce the search efforts of wA*. For heuristic computation, it is common to solve a simpler planning problem in a relaxed space formed by relaxing some constraints in the original search space. We introduce conservative heuristics—novel heuristics that anticipate search efforts required to reach the goal. We first introduce the notion of a conservative path in the relaxed space, whose existence guarantees the existence of a feasible path in the original space. Conservative heuristics are computed such that if a conservative path exists in the relaxed space, then the search can follow the heuristic gradient to find a feasible path in the original space while expanding only those states that appear on this path. We propose an algorithm to compute conservative heuristics. We evaluate conservative heuristics theoretically as well as empirically using simulated experiments in humanoid footstep planning, planning for a UAV, and

a real-world experiment in footstep planning for a NAO robot.

Our second contribution is to speed up a particular class of planning under uncertainty problems. Many real-world planning problems involve robots having to plan in partially known environments. This frequently requires planning under uncertainty over missing information about the environment. Unfortunately, the computational expense of such planning often precludes its scalability to real-world problems. The Probabilistic Planning with Clear Preferences (PPCP) framework computes optimal policies in a scalable way for a subset of such planning problems wherein there exist clear preferences over the actual values of missing information. It runs a series of fast, deterministic, A*-like searches to construct and refine a policy, guaranteeing optimality under certain conditions. Aligning with the key idea of this thesis, we anticipate that while running a series of A*-like searches to compute a policy, search efforts are correlated with the amount of missing information each path relies upon to reach the goal. We observe that by exploiting this correlation and minimizing the amount of missing information each path relies upon, marginally suboptimal policies can be computed significantly faster. To that end, we introduce Fast-PPCP, a novel planning algorithm that computes a provably bounded-suboptimal policy using significantly lesser number of searches than that required to find an optimal policy, for the same subset of problems that can be solved by PPCP. We evaluate Fast-PPCP theoretically and experimentally, showing its benefit over popular baselines. Moreover, to evaluate Fast-PPCP in discounted-reward problems such as RockSample, we also formulate a transformation that converts discounted-reward problems into discounted-cost problems to which Fast-PPCP can be applied.

In the final part of the thesis, we are motivated by the application of Fast-PPCP to real-world robot navigation problems in partially-known environments. In such problems, Fast-PPCP can potentially waste search efforts in exploring multiple partial policies that use an unknown region to reach the goal before discovering that those partial policies are invalid. To reduce this wastage, we introduce an optimized version of Fast-PPCP for planning in environments with large unknown regions. We demonstrate its utility in off-road robot navigation in simulation and on a physical robot.

Acknowledgments

This thesis would not have been possible without the support, guidance, and encouragement of my advisors, Maxim Likhachev and Manuela Veloso. I am incredibly fortunate to be advised by them. I could not have asked for better advisors.

Specifically, I sincerely thank Maxim for his extremely insightful technical advice and commend his immense patience and understanding. I have always appreciated his willingness to take time out without fail to discuss research, even on weekends, beyond regular hours, and often at very short notice. His passion for planning inspires every student entering this field of research. Maxim has taught me how to take perceived failures in research in stride. I am forever grateful to Maxim for always believing in me, even during times when I did not believe in myself.

I am forever grateful to Manuela for her extremely helpful technical insights and terrific guidance in my Ph.D. research. Her pioneering work in robot soccer and many other creative topics in planning and learning has inspired me to stay in academia. I highly appreciate Manuela's extremely wise pieces of advice delivered with complete honesty. Her sincere critique of my approaches, whether about life in general or research, made me re-evaluate them and modify them for good. I thank her for her genuine care and empathy, which has always pushed me to do better. Manuela has been my sincere well-wisher and a source of tremendous inspiration from the day I met her during the RI Ph.D. open house in 2016.

Maxim and Manuela have taught me how to do scientific research and present it clearly. I am forever indebted to them for all the lessons they have imparted to me and for playing such an essential role in my journey of trying to become an independent researcher.

I thank my committee members—Shlomo Zilberstein and Stephen Smith—who, despite their busy schedules, provided valuable feedback on my research. Specifically, I thank Shlomo for correctly identifying some limitations of my work on planning under uncertainty which opened up exciting avenues for future research. I also thank him for pointing me to relevant literature produced by his group. I thank Steve for his encouraging comments about my body of work and for taking the time to give detailed feedback on my thesis document. I also thank him for being on my speaking qualifier committee and providing very useful insights into my work on developing heuristics for deterministic planning.

I thank my master's thesis advisor, Aaron Steinfeld, for imparting his valuable knowledge in the field of human-robot interaction and for being a very wise mentor throughout my Ph.D. journey. It is rare to find someone with

so much generosity as Aaron, and I take inspiration from him to be a better version of myself.

I am grateful to Michael Erdmann and Henny Admoni for providing me with an opportunity to TA. I thank Henny for giving me a chance to present some lectures in her course that introduced me to the challenges of teaching. Mike Erdmann has always amazed me with his excellent delivery of highly non-trivial technical material. His course has been a memorable experience during my time at CMU.

I am indebted to my undergraduate internship advisor, Francisco Rovira-Mas, who introduced me to AI and encouraged me to switch fields from agricultural engineering to robotics. I thank Ravi sir and Jasvinder sir, my first mentors in high school, whose passion for science and education was contagious.

I thank my collaborators, Yash Oza, Tushar Kusnur, and Ashwin Khadke, for taking time out for research discussions and helping with my experiments.

I sincerely thank Pratik Khandagale for his unwavering support and help during the final part of my thesis research. I thank my friends in Pittsburgh, Canada, and India, especially Prerna Chikersal, Christabel Wayllace, and Anindita Gandhi. They have encouraged me during my Ph.D. journey and lent a patient ear to my rants about paper rejections! I thank Maxim for letting me mentor some junior lab members. I thank my mentees–Jasmeet Kaur, Patrick Naughton, Alvin Shek, Lingyao Zhang, and Asritha Guduru, for introducing me to the challenges of mentorship.

This thesis would not have been possible without the unconditional love and support of my mother, Dr. Pamela Chatterjee, and my father, Dr. Subrata Chatterjee. My parents have been constant pillars throughout my life and have always encouraged me to fight against all odds. Through them, I learned the value of sincere efforts, hard work, perseverance, dedication, empathy, kindness, and forthrightness. I am incredibly fortunate to be their daughter.

I am also grateful to our golden retrievers back at my home in India, Ginger, and Custard, for being such entertainers and a breath of fresh air throughout my remote work from India. I am forever grateful to my auntie, Bhagya, who loved and supported me like her own child during my remote work from India.

Finally, my heartfelt gratitude towards my late grandparents, Dadu and Dida, two highly evolved eternal souls with an attitude towards education ahead of their times, whose dreams I could finally fulfill. My memories of their constant encouragement have urged me to go further in my academic journey. It is impossible to describe in words their role in my life, and I owe my life's journey to them. Dadu and Dida, your presence is felt every day. I dedicate this thesis to my grandparents and parents.

Contents

When this dissertation is viewed as a PDF, the page header is a link to this Table of Contents.

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

We focus on relatively low dimensional robot motion planning problems, where the robot has to search for a path from a start location to reach a goal location. Examples of low dimensional motion planning problems are:

- Planning in 4 dimensions (4D), i.e., x and y position, orientation and velocity for a self-driving vehicle.
- Planning in 5D, i.e., x and y position, orientation, velocity, and time for navigation of unmanned aerial vehicles (UAVs).
- Planning in 4D, i.e., x and y position, orientation and foot ID (left or right) for footstep planning for humanoids.

Heuristic search-based planning is a popular method for low dimensional robot motion planning ([78], [84], [74], [58], [83], [1]), where the planner searches for a path using an estimate of the cost to goal. This estimate is known as a heuristic. For example, the

Euclidean distance between the start and the goal is a heuristic for a robot trying to find the shortest path from start to goal.

There is a need for fast heuristic search-based planning when limited time is available for planning and execution of the computed plan. The nature of the task can introduce time sensitivity—for example, robots, while navigating in dynamic environments with moving pedestrians, need to react quickly to pedestrians.

For example, consider a scenario where the planner has computed a path from start to goal, and the robot executes this path. Suppose a new obstacle appears in the environment, which creates a narrow passage on the current path of the robot. A new obstacle previously not within the sensing range can appear in the environment if the robot can sense it now. A new obstacle can also get externally added to the environment; for example, a person brings in a piece of furniture. Suppose the passage is narrow enough for the robot to be unable to fit inside it. In that case, it becomes necessary for the planner to be able to quickly compute a new detour plan that goes around the obstacle to complete planning and execution within the allotted time. Also, aborting the execution of the current plan and computing a new plan has to be completed *before* the robot's inertia leads to a collision. These factors necessitate fast planning and re-planning.

Often, we want to plan fast but are also interested in controlling the solution quality because we desire *efficient* planning. In the footstep planning and in planning for self driving, if the goal has to be reached within a certain time then the planner cannot produce really long paths. Similarly, UAVs face energy usage constraints while flying. As a result, they need to preserve fuel while flying and thus can't take very long routes. In such examples, where controlling the solution quality is also essential in addition to fast planning, bounded-suboptimal heuristic search algorithms are a popular alternative to optimal heuristic search algorithms. These algorithms compromise solution quality

for computation speed. Specifically, these algorithms aim to find a solution that can be computed faster than the optimal solution, while guaranteeing that the solution cost is bounded within a specified factor of the optimal cost (bounded-suboptimality).

Bounded-suboptimal heuristic search algorithms have been largely investigated in deterministic planning—in planning where there is no uncertainty about the environment, or the outcomes of the actions executed by the robot, or the robot state. Currently, most bounded-suboptimal heuristic search algorithms speed up planning by guiding the search using cost-to-goal estimates. Cost-to-goal estimates are not necessarily correlated with the search effort required to reach the goal, which leads to wasted search efforts in environments with narrow passages. The first research question we investigate in this thesis is:

- How to increase the efficiency of bounded-suboptimal heuristic search in the context of deterministic planning in cluttered environments with narrow passages?

Further, we are also interested in problems where there is uncertainty regarding the environment, specifically, when the environment is partially known. In the real-world, robots often have to plan despite the environment being partially known. This frequently necessitates planning under uncertainty over missing information about the environment in order to generate solutions that are robust to uncertainty. For example, consider a robot navigating in a space with multiple rooms connected through doors—such as hospital spaces, university campuses, and home environments (Figure 1.1). The status of some of the doors is unknown, whether they are open or closed—there is uncertainty due to status of the doors being unknown. In general, planning under uncertainty is computationally expensive. The computational expense of planning under uncertainty often precludes its scalability to real-world problems. Thus, there is a need for developing efficient planning under uncertainty algorithms that can effectively reason about the uncertainty to generate a solution in real-time that is robust to uncertainty.

Figure 1.1: Room with doors whose status (open/closed) of green doors is unknown.

[80] focused on a specific subset of problems with partially known environments, wherein clear preferences exist over the possible values of unknown variables in the environment. For example, in Figure 1.1 the robot prefers that an unknown door be open rather than closed. For this subset, the method proposed by [80] computes an optimal solution in an efficient and scalable way that is significantly faster than existing methods. The second research question we investigate in this thesis is:

- How to further speed up planning under uncertainty when the environment is partially known and clear preferences exist, while aiming for bounded-suboptimality of the solution?

1.2 Key idea

As stated in section 1.1, bounded-suboptimal heuristic searches rely on cost-to-goal estimates. Cost-to-goal estimates are not necessarily correlated with search efforts. The key idea of this thesis is to *explicitly reason about anticipated search efforts required to reach the goal* instead of cost-to-goal estimates, and *achieve higher speedup by guiding the search along solutions that minimize these anticipated search efforts* while ensuring bounded-suboptimality of the computed solutions. In this thesis, we use this key idea to speed up planning while ensuring bounded-suboptimality in the context of *both deterministic planning as well as planning under uncertainty*. In the context of deterministic planning, we introduce a novel methodology for computing heuristics that try to anticipate the search efforts required to reach the goal. Specifically, the heuristic values are proportional to the *expected* number of search expansions needed by a bounded-suboptimal search algorithm to reach the goal. Under certain conditions, it can be shown that the heuristic value is, in fact, proportional to the *actual* number of search expansions required by a bounded-suboptimal search algorithm to reach the goal. In the context of planning under uncertainty, we anticipate that search efforts are correlated with the amount of uncertainty encountered on a path while trying to reach the goal. We introduce a novel algorithm that exploits this correlation and plans by minimizing the amount of uncertainty encountered on a path while trying to get to the goal.

1.3 Contributions

We now explicitly state the primary contributions of this thesis.

1.3.1 Introduction of Conservative Heuristics to reduce search efforts in search-based motion planning, its theoretical and experimental analysis

Our first contribution addresses our first research question. In particular, we aim to speed up deterministic robot planning problems formulated as bounded-suboptimal heuristic search. Weighted A^* (wA*) search is a popular bounded-suboptimal heuristic search algorithm. We investigate the problem of computing heuristics that explicitly aim to reduce the search efforts of wA*. For heuristic computation, it is common to solve a simpler planning problem in a relaxed space formed by relaxing some constraints in the original search space. We introduce conservative heuristics—novel heuristics that try to anticipate search efforts required to reach the goal.

We first introduce the notion of a conservative/non-conservative edge in the relaxed space, whose existence does/does not guarantee the existence of a corresponding feasible edge in the original space. This notion extends to a purely conservative path in the relaxed space, whose existence guarantees the existence of a feasible path in the original space. Conservative heuristics are computed such that if a purely conservative path exists in the relaxed space, then the search can follow the heuristic gradient to find a feasible path in the original space, while expanding only those states that appear on this path. Whenever this condition holds, conservative heuristic values can be shown to be proportional to the true search efforts required to reach the goal. In the case when a purely conservative path does not exist in the relaxed space, conservative heuristics values are proportional to the minimum number of non-conservative edges in the relaxed space required to reach the goal. Given no other information about non-conservative edges, all of them have the same number of expansions in expectation. Thus, the total number of non-conservative edges

needed to reach the goal is proportional to the number of expansions in expectation, which is one way to measure anticipated search efforts needed to reach the goal. Thus, minimizing the number of non-conservative edges in a path to a goal minimizes the anticipated search efforts needed to reach the goal.

We propose an algorithm to compute conservative heuristics. We theoretically derive the suboptimality bound, analyze the number of expansions made by wA* using conservative heuristics under certain conditions, and other properties of using conservative heuristics in wA*. Further, we evaluate conservative heuristics using simulated experiments in humanoid footstep planning and planning for a UAV. Results show significant benefits of conservative heuristics in terms of computation speed compared to baseline heuristics that estimate cost-to-goal.

1.3.2 Introduction of Fast-PPCP: a novel algorithm that minimizes anticipated search efforts in probabilistic planning, its theoretical and empirical evaluation

Our second contribution addresses our second research question. In particular, we aim to speed up a particular class of planning under uncertainty problems. We focus on the problem of planning under uncertainty over missing information about the environment. As noted in [79], real-world planning problems often possess the property of clear preferences (CP), wherein there exist clear preferences over the actual values of missing information. For example, consider a robot navigating in a partially known environment: it will clearly prefer any unknown region to be traversable rather than not. Likhachev and Stentz in [79] showed that the property of clear preferences, when combined with an assumption of perfect sensing (PS)—an assumption that there is no noise in sensing the value of missing information—can be utilized to compute optimal policies in an efficient and scalable way.

Specifically, they introduced the Probabilistic Planning with Clear Preferences (PPCP) framework that computes optimal policies in a scalable way for problems that exhibit clear preferences and assume perfect sensing (CP-PS problems). It runs a series of fast, deterministic, A*-like searches to construct and refine a policy, guaranteeing optimality under certain conditions.

Aligning with the key idea of this thesis, we anticipate that while running a series of A*-like searches to compute a policy, search efforts are correlated with the amount of missing information each path relies upon to reach the goal. We observe that by exploiting this correlation and minimizing the amount of missing information each path relies upon, marginally suboptimal policies can be computed significantly faster. To that end, we introduce FAST-PPCP, a novel planning algorithm that computes a provably bounded-suboptimal policy using significantly lesser number of searches than that required to find an optimal policy, for CP-PS problems. In these problems, FAST-PPCP runs much faster than PPCP, which outperforms other optimal solvers such as RTDP-BEL, LAO*, PAO* (Ferguson, Stentz, and Thrun 2004), and HSVI2 when applied to CP-PS problems. To the best of our knowledge, no heuristic search-based approach developed so far for planning under uncertainty – has attempted to speed up planning by guiding the search along solutions that explicitly minimize anticipated search efforts required to reach the goal while guaranteeing bounded suboptimality.

Differences with PPCP. We now highlight the algorithmic differences between PPCP and FAST-PPCP. FAST-PPCP uses a search that operates very differently compared to the search in PPCP search. It searches in an augmented state space to account for bounded-suboptimality, has different edge-costs to minimize the amount of missing information each path relies upon to reach the goal, has a different termination condition, and exploits pruning techniques to prune parts of the state space to increase search efficiency. Additionally,

FAST-PPCP also has novel strategies for scheduling the searches that ensure that the first time the search terminates, the policy is guaranteed to be bounded suboptimal, and ensure completeness.

We present the theoretical analysis of FAST-PPCP. We also present experimental analysis in the domain of robot navigation in partially unknown environments. Moreover, to evaluate FAST-PPCP in discounted-reward problems such as RockSample, we also formulate a transformation that converts discounted-reward problems into discounted-cost problems to which FAST-PPCP can be applied.

1.3.3 Adaptation and experimental evaluation of algorithms that achieve speedup by minimizing anticipated search-effort in real-world robotics problems

As part of experimental validation of our first contribution, as stated in subsection 1.3.1, we implement a footstep planner on a physical NAO robot based on [58]. The robot starts executing the plan computed by the footstep planner to reach from start to goal as shown in Figure 1.2. When a new obstacle is introduced into the environment, it creates a narrow passage between the white and brown boxes on the current path of the robot. The robot is unable to fit inside this narrow passage. We show the utility of using Conservative Heuristics to quickly compute a detour plan that does not go through the narrow passage, compared to a baseline heuristic.

As part of experimental validation of our second contribution, we aim to demonstrate the benefits of FAST-PPCP in real-world robot navigation problems in which the environment has large unknown regions. Partially known off-road terrains such as mines, military bases and disaster sites can have large unknown regions. Similarly, indoor environments can have large carpets or rugs. For some carpets that are thick, it is unknown at the time of planning

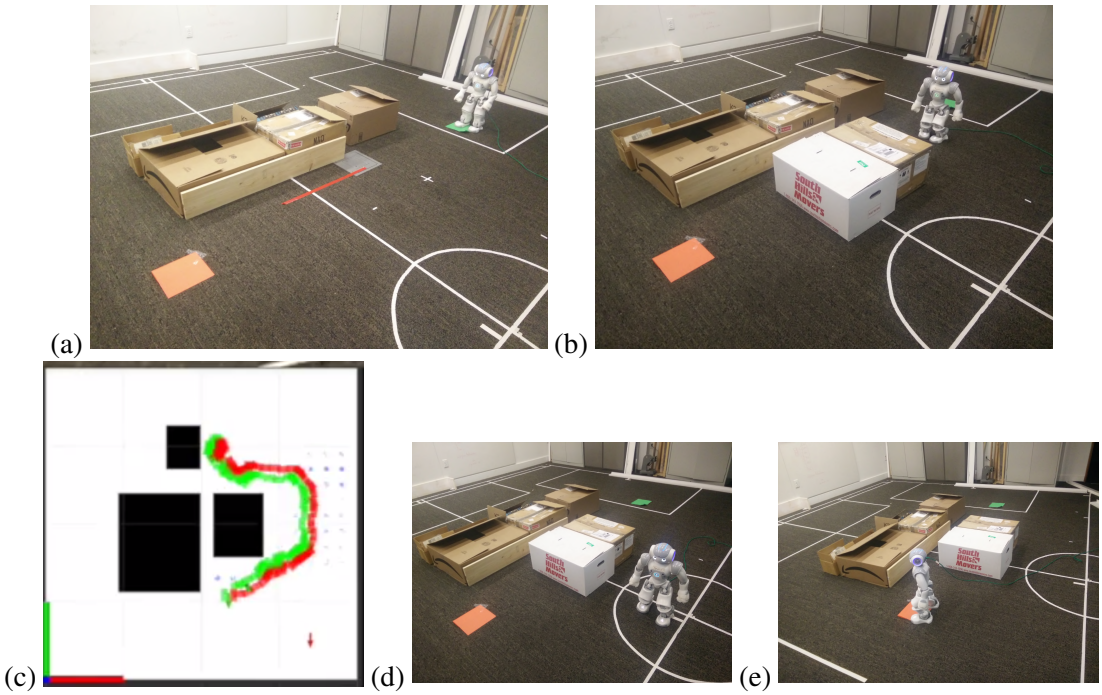


Figure 1.2: (a) Initial environment. (b) Narrow passage between white and brown box created by the sudden addition of a new obstacle, the humanoid robot NAO waiting for plan. (d), (e) NAO executing the recomputed footstep plan (c) that goes around the obstacle.

whether the robot can get on to the carpet and completely traverse it or not.

For an environment with large unknown regions, FAST-PPCP can potentially waste search efforts in exploring multiple partial policies that use an unknown region to reach the goal, before discovering that those partial policies are invalid. Our key idea is that FAST-PPCP can use the value of one invalid partial policy that uses a given unknown region, to update knowledge about the value of other partial policies that also use this unknown region. These updated values can then be used to preempt explicit exploration of invalid policies and thereby reduce wasted search efforts.

To this end, we introduce an *optimized version of* FAST-PPCP for planning in environments with large unknown regions. We have experimentally evaluated optimized FAST-PPCP in simulation over a real-world map of Fort Indiantown Gap, Pennsylvania,

constructed from satellite imagery (Figure 1.3). The traversability of most of the terrain

Figure 1.3: Satellite imagery of a part of Fort IndianaTown Gap, PA. There are large regions of unknown traversability due to dense forest canopies (dark green).

can be deduced from the satellite image, except for a few regions occluded by canopies in the aerial view. We have also demonstrated the working of optimized FAST-PPCP on a physical robot—Husarion ROSbot 2.0 PRO—in an indoor space with large carpets whose traversability is unknown at the time of planning.

1.4 Thesis outline

The following outline summarizes each chapter of the thesis.

Chapter 2 We review the relevant background on optimal and bounded suboptimal search-based planning. We also give an overview of heuristics computed in relaxed spaces using search-based planning which provides the background for chapter 4. Finally, we give an overview of the framework of probabilistic planning with clear preferences over missing

information. This provides a relevant background for chapters 5 and 6.

Chapter 3 We review the relevant literature.

Chapter 4 We present our first contribution. We introduce Conservative Heuristics– novel heuristics that aim to speed up deterministic robot planning problems formulated as bounded-suboptimal heuristic search. We present its theoretical and empirical analysis.

Chapter 5 We present our second contribution. We introduce FAST-PPCP– a novel planning algorithm that computes a provably bounded-suboptimal policy using significantly lesser number of searches than that required to find an optimal policy, for a special class of planning under uncertainty problems. We present its theoretical and empirical analysis.

Chapter 6 We present the application of FAST-PPCP in real-world robot navigation problems with large unknown regions. We introduce an optimization in FAST-PPCP, and demonstrate its utility in both simulation and real-world experiments.

Chapter 7 Additionally, we also present the application of PPCP to solve the problem of search-based planning with learned behaviors for navigation among pedestrians.

Chapter 8 We conclude the thesis with a summary of its contributions and present potential directions for future work.

Chapter 2

Background

In this chapter, we provide relevant background about optimal and bounded suboptimal search-based planning, abstraction heuristics in planning, and probabilistic planning with clear preferences over missing information.

2.1 Search-based planning

Search-based planning algorithms are one of the fundamental classes of algorithms used for robot motion planning in deterministic environments. Search-based planning algorithms are used in many other areas [70], [69], [61], apart from motion planning. These algorithms generate a graph representation of the planning problem and search the graph for a solution. The construction of the graph representation is often interleaved with the searching procedure. In the case of robot motion planning, a robot state that uniquely specifies the robot in terms of its base footprint pose, joint poses, etc., is represented as a node in the search graph. An edge in the graph represents the feasible transitions from one robot state to another in accordance with the kino-dynamic constraints of the robot. Each edge has

a weight that is proportional to the cost of transition that is represented by an edge. For a given start and goal states and a graph representing the problem, an optimal graph search algorithm can find a solution that minimizes the total cost from the start to the goal.

2.2 Search-based planning using A* search and its variants

Algorithm 1 A* Search

```
1:  procedure ExpandState(s)
2:    remove s from OPEN
3:    for all s' ∈ GetSuccessors(s) do
4:      if s' was not visited before then
5:        g(s') ← ∞
6:      if g(s') > g(s) + c(s, s') then
7:        g(s') ← g(s) + c(s, s')
8:        insert/update s' in OPEN with priority f(s') = g(s') + h(s')
9:  procedure A*(s_start, s_goal)
10:   OPEN ← ∅
11:   g(s_start) ← 0
12:   f(s_start) ← h(s_start)
13:   insert s_start in OPEN with priority f(s_start)
14:   while s_goal is not expanded do
15:     if OPEN is empty then return null
16:     remove s with the smallest f-value from OPEN
17:     ExpandState(s)
18:   return ReconstructPath()
```

Figure 2.1: A* search algorithm.

One of the most popular optimal graph search algorithms is A* search [45]. The A* algorithm shown in Figure 2.1 finds a minimum cost path from a start state $s_{start} \in S$ to a goal state $s_{goal} \in S$ in a directed graph $G(S,E)$. S is the set of nodes in the graph and E is the set of edges that connect the nodes. A* search only works for deterministic graphs where each edge $e \in E$ connects a single node s to another single node s'. An edge cost

function $c : S \times S \rightarrow R^{+}$ maps a pair of states, i.e., each edge $e(s,s')$ to a positive scalar-valued cost. If s and s' are not connected in the graph, the edge-cost $c(s,s')$ is assigned an infinite cost. For each state, A* search computes and updates three values associated with the state. The g-value $g(s)$ is the minimum path cost accumulated over the edges on the best path from s_{start} to s. The h-value $h(s)$ is a heuristically estimated minimum path cost accumulated over the edges on the best path from s to s_{goal}. The f-value $f(s) = g(s) + h(s)$ is an estimated minimum path cost from s_{start} to s_{goal} that passes through s and is also referred to as priority of state s. *OPEN* represents a priority queue called *OPEN* list that contains all the states that have been discovered but not yet expanded. Starting with *OPEN* containing s_{start} only, A* repeatedly expands the state in *OPEN* with the minimum f-value, i.e., with the highest priority. The state expansion process, ExpandState(), consists of two operations. One is to find or generate a successor of the expanded state for each action. Then the g-value of the successor is updated by the best path cost found so far. When s_{goal} is to be expanded, i.e., s_{goal} has the highest priority in *OPEN*, the search process terminates and return the best path found. Several theoretic properties of A* search depends on the heuristic function $h(s)$. If the heuristic is consistent, i.e., it satisfies the triangle inequality $h(s) \leq h(s') + c(s,s'), \forall s, s' \in S$ and $h(s_{goal}) = 0$, then A* is guaranteed not to expand a state more than once. If the heuristic is admissible, i.e., it never over-estimates the best path cost from s to s_{goal} for $\forall s \in S$, then A* is guaranteed to find the optimal (minimum cost) path. If a heuristic is consistent, it is also admissible, but not vice versa.

There are many variants of A* search. Weighted A* inflates the heuristic function which leads to suboptimality of the returned solution [94]. Anytime Repairing A* finds an initial solution quickly with a high inflation of the heuristic and then iteratively improves the solution quality over time by decreasing the inflation factor [81]. Multi-Heuristic A* leverages multiple and possibly inadmissible heuristics to guide the search efficiently [1].

2.3 Abstraction heuristics in planning

Admissible heuristics are often defined as optimal solutions to a *problem relaxation*, that is easier to solve than the original planning problem ([11], [16], [28], [27], [38], [30], [31], [32], [34], [35], [46], [47], [49], [51], [50], [52], [55], [57], [60], [63], [64], [69], [82], [97]). The relaxed problem is an abstraction of the original problem. Specifically, abstraction heuristics are computed by solving the relaxed problem through a search.

Definition of abstraction heuristics. We now give a formal definition of abstraction. We define a search problem as finding the minimum cost path through S from the start state s_{start} to a goal state s_{goal} from a set of goal states G, or prove that no path exists. Let $h^*_S(s)$ denote the cost of cheapest path through S from s to some $s_{goal} \in G$ (∞ if no path exists). An abstraction is a mapping, λ, from the states of S to some abstract space AS, which preserves labelled paths and goal states. In other words, if $s \xrightarrow{c} s' \in S$, then $\lambda(s) \xrightarrow{c'} \lambda(s') \in AS$, with $c' \leq c$. Here, c is the cost of a path from s to s' and c' is the cost of the corresponding path in the abstract space from $\lambda(s)$ to $\lambda(s')$. Also, if $s \in G$, then $\lambda(s) \in G_{AS}$, where G_{AS} is the set of projected goals in the abstract space. We now define abstraction heuristics. The corresponding abstraction heuristic is

$$h^{\lambda(s)} = h_{AS}(\lambda(s)). \tag{2.1}$$

An abstraction heuristic as defined in eq. 2.1 is admissible and consistent.

Properties of abstraction heuristics. Abstraction heuristics have attractive properties for domain-independent planning. They are general, i.e., abstractions exist for every planning domain/instance. Also, once an abstraction is chosen, the heuristic computation can be done by generic and automatic procedures. However, applying abstraction heuristics to planning presents some challenges. Typically, many abstractions are possible. It is a challenge

to automatically choose a good abstraction. Also, for domain specific planning, it is a challenge to find domains specific properties that help design better abstraction heuristics. We address this challenge in this thesis in chapter 4.

2.4 Probabilistic planning with clear preferences over missing information

[80] introduced the Probabilistic Planning with Clear Preferences (PPCP) framework that computes optimal policies in an efficient and scalable way for problems that exhibit clear preferences and assume perfect sensing (CP-PS problems), as stated in chapter 1. PPCP runs a series of fast, deterministic, A*-like searches to construct and refine a policy. The optimality of the final solution is guaranteed under certain conditions. In this section, we first explain the problem setup in which PPCP can be applied and its required assumptions. We then provide a brief overview of the working of PPCP.

2.4.1 Problem setup

We use the problem of robot navigation in partially known environments as a running example to explain the concepts throughout the rest of the paper. PPCP can be applied to any domain where the assumptions A1-5 listed in Table 2.1 hold true. Consider in Figure 2.2 a robot that has to navigate from start (cell $(11,48)$) to goal (cell $(51,57)$) in this environment represented by a 60×60 grid (Red cells indicate blocked space). We henceforth refer to this example environment as [**Ex.:**] . The robot can occupy a free grid-cell (white) and has 8 *move* actions that can move the robot in the cardinal and inter-cardinal directions (N, S, E, W, NE, NW, SE, SW) by one cell. There are some states in the environment whose status is unknown or hidden at the time of planning, which affects the

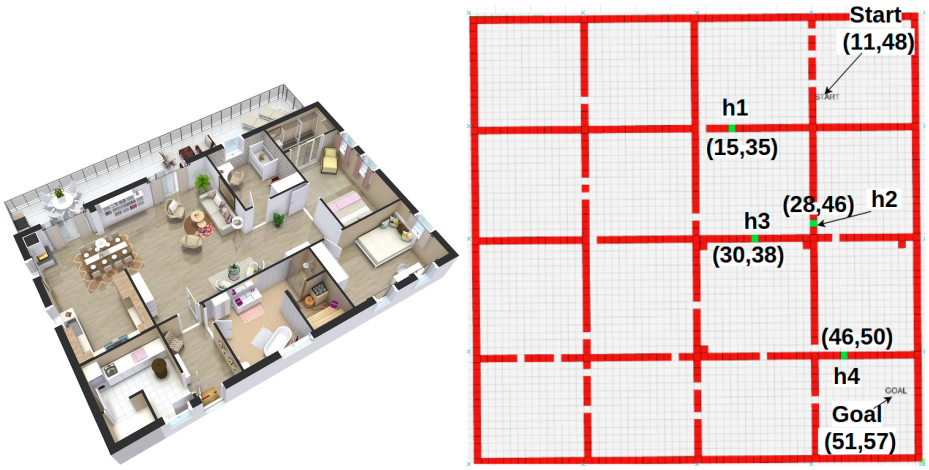


Figure 2.2: Robot navigation between rooms with uncertainty about doors being *OPEN* or closed.

outcomes of some actions. [**Ex.:** The status (free or blocked) of the four labelled green cells is unknown, since we do not know whether some of the room doors represented by the green cells are *OPEN* or closed.] .

PPCP formulates a belief state as a vector of discrete variables split into two components: $X = [S(X); H(X)]$. The *deterministic* component—$S(X)$—is a set of variables whose status is fully observable [**Ex.:** robot's 2D location] . The *hidden* component—$H(X)$—is a set of variables representing the statuses of all unknown/hidden states. We denote the i^{th} hidden variable in $H(X)$ by $h_i(X)$. $h_i(X) = u$ indicates that $h_i(X)$ is unknown. [**Ex.:** The four green cells are each represented by a hidden variable. $X_{st} = [(11,48); h_1 = u, h_2 = u, h_3 = u, h_4 = u]$ represents the start belief-state X_{st}, indicating that the values of all the four hidden variables are unknown before the robot starts planning] . The robot can *sense* an adjacent hidden cell to know if it is free ($h_i(X) = 0$) or blocked ($h_i(X) = 1$).

We denote the set of actions applicable at a belief state X by $A(S(X))$. $A(S(X))$ is applicable at any belief state Y that has $S(Y) = S(X)$ since the applicability of an action does not depend on the hidden variables in Y . However, the outcomes of an action depend

on a hidden variable; for an action a taken at $S(X)$, we denote the hidden variable in $H(X)$ affecting its outcome by $h^{S(X),a}$. An action is *deterministic* [**Ex.:** Move actions] if its outcome is unaffected by any hidden variable ($h^{S(X),a} = \texttt{NULL}$). A deterministic action has only one outcome because the underlying environment is deterministic. An action is *stochastic* if it can have multiple possible outcomes depending on the status of a hidden variable. [**Ex.:** Apart from the *move* actions, the robot can also take a stochastic action *sense and move*—that senses an adjacent cell and moves the robot to it only if it is sensed as free, else stays put.] .

We denote the set of possible outcomes (successors) of an action a taken at a belief-state X by $succ(X,a)$ in the belief-space and $succ(S(X),a)$ in the underlying deterministic space [**Ex.** 2D grid]. If $X \in succ(X',a)$, then $S(X) = succ(S(X'),a)$. We refer to $S(X')$ as the *predecessor* of $S(X)$ (and X' as the predecessor of X). $H(X)$ remains the same as $H(X')$ for a deterministic action, whereas, *for a stochastic action, $H(X)$ is the same as $H(X')$ except for $h^{S(X),a}$ which becomes known if it was unknown*. The probability distribution of the transitions $P(X|X',a)$ is the same as that of the hidden variable $h^{S(X'),a}$. Given the independence (A5), note that the belief state representation concisely represents a probability distribution over all possible states. [**Ex.:** h_1 in Figure. 2.2 represents the status of the unknown cell $(15,35)$ and affects the outcome of the action *sense-and-move* taken on the adjacent cell $(14,36)$. Let $X' = [(14,36); u,u,u,u]$ be a belief state. When taken on X', *sense-and-move* produces two belief-state outcomes: $X_1 = [(15,35); h_1 = 0,u,u,u]$ and $X_2 = [(14,36); h_1 = 1,u,u,u]$ both with a probability of 0.5] . The cost of an edge in the belief-space is the same as the cost of the corresponding edge in the deterministic environment: $c(X,a,X') = c(S(X),a,S(X'))$. [**Ex.:** the move actions have a cost proportional to the euclidean distance between $S(X)$ and $S(X')$ (1.4 for diagonal, 1 for others). The *sense-and-move* action follows the cost of *move* actions if the hidden cell is

A1	The environment is deterministic, i.e., if the environment were fully known at the time of planning, there would be no uncertainty in the outcome of an action.
A2	The agent has a probability distribution or *belief* over the status of these cells.
A3	The true status of a hidden variable becomes known immediately (perfect sensing assumption.
A4	Only one hidden variable can affect the outcome of an action a taken at $S(X)$. However, the same hidden variable is allowed to affect outcome of another action taken at another state.
A5	The variables in H are independent of each other and therefore $P(H) = \prod_{i=1}^{\lvert H \rvert} P(h_i)$.

Table 2.1: Assumptions used in PPCP

free, else it incurs a cost of 2] .

Clear preferences: We assume that every hidden variable's best (clearly preferred) value is known beforehand, meaning that we prefer a hidden cell to be free ($h^{S(X),a}$) rather than blocked ($h^{S(X),a}$). We define clear preferences [80] as: Let $V^*(X')$ denote the expected cost of executing an optimal policy (policy that minimizes expected cost to goal) from state X'. For any given state X' and stochastic action a such that $h^{S(X'),a}$ is unknown, there exists a successor state X^b such that $h^{S(X),a}(X^b) = b$ (we denote the best value using the variable b. In our example, b = 0) and $X^b = \arg\min_{X \in succ(S(X'),a)} \{c(S(X'),a,S(X)+V^*(X))\}$. The planning problem is to compute a policy (defined in Table 2.2) from X_{st}.

2.4.2 PPCP overview

The overall approach of PPCP is to compute an optimal policy in the belief-space by running a series of A*[45]-like searches in the underlying deterministic environment instead of the exponentially larger belief space. PPCP pseudocode and algorithmic details can be found in [79]. This approach turns out to be orders of magnitude faster than solving the full problem at once since the memory requirements are much lower.

PPCP iteratively constructs and refines a partial policy (defined in Table 2.2) from X_{st}, while updating v-values (expected cost-to-goal) of the states reachable by following the partial policy. PPCP in its first iteration computes the shortest path from $S(X_{st})$ in the

deterministic space, assuming all hidden variables (h_1 through h_4 in **[Ex.]**) are set to their best value (free in **[Ex.]**) and uses this path as an initial partial policy for the robot in the belief-space. This partial policy has either deterministic outcomes or only the best outcomes of sensing a hidden variable; the non-preferred outcomes have not been explored yet. In the first PPCP iteration, the shortest path from $S(X_{st})$ in **[Ex.]** passes through the hidden cell $(15,35)$ assuming it is free, and has a *sense-and-move* action at $(14,36)$. The partial policy defines an action for the best outcome $[(15,35);h_1 = 0,u,u,u]$; the non-preferred outcome $[(14,36);h_1 = 1,u,u,u]$ does not have an action defined from it yet. assuming its outcome is that the sensed hidden cell $(15,35)$ is free ($h_1 = 0$). However, the non-preferred outcome of the action has not been explored yet. The second PPCP iteration in **[Ex.]** computes a path from $(14,36)$, again in the deterministic space, remembering that $(14,36)$ was the outcome of *sense-and-move* when $(15,35)$ was blocked, but still assuming that h_2 through h_4 are free. PPCP then includes this path into the partial policy in the belief space from $[(14,36);h_1 = 1,u,u,u]$ and updates its v-value with the cost of this path (and an underestimate of V^* of the non-preferred outcomes encountered on this path). Now that PPCP has learnt the true expected cost to reach the goal through the hidden cell $(15,35)$ assuming other hidden cells are free, it starts the third iteration from $S(X_{st})$ again, to see if reaching the goal through a different hidden cell has a lower expected cost.

What has the most impact on PPCP's runtime? To explain from an algorithmic perspective, the updated v-value of $[(14,36);h_1 = 1,u,u,u]$ introduces a bellman error between the v-value of the predecessor $[(14,36);h_1 = u,u,u,u]$ and what it should be according to the expected cost over its successors in the current partial policy, which accumulates all the way upto X_{st} in this example. Hoping to correct this error, PPCP starts another iteration, searching from $S(X_{st})$. If it finds as optimal the same path as before, then the bellman error at $[(14,36);h_1 = u,u,u,u]$ gets corrected, else PPCP finds a new path from $S(X_{st})$ which

Term	Definition
Policy from bel state X	Tree rooted at X s.t every branch reaches a bel state X_g s.t $S(X_g) = S_g$ for a given goal S_g.
Unexplored pivot	Non-preferred outcome on partial policy which has not been a pivot yet
Partial Policy from a bel state X	Policy that has at least one bel state without a defined action.
Primary branch of a policy from X	Branch on the policy from X_p where every belief state on the branch is either an outcome of a deterministic action, or the best outcome of a stochastic action.

Table 2.2: Definitions used in the explanation of PPCP

might introduce more unexplored outcomes in the partial policy.

More generally, each non-preferred outcome on the partial policy leads to an additional PPCP iteration needed to define a policy from it. Also, whenever the v-value of a non-preferred outcome is updated, its predecessor on the policy gets a negative Bellman error, which gets accumulated up along the policy till the outcome of a stochastic action (or X_{st}, whichever comes first) is encountered, at which point PPCP starts another iteration from this outcome (or X_{st}). To conclude, the number of iterations increases with an increase in the number of stochastic actions in each branch of the partial policy.

Since PPCP continues to iterate until every outcome in the policy has an action defined and has no bellman error, the number of PPCP iterations can be really high for environments with a large number of hidden variables, especially if stochastic actions lie lower (closer to a leaf node) on the policy.

Chapter 3

Related Work

In this chapter, we review previous work in two primary areas: algorithms for efficient search-based deterministic planning, and efficient belief space planning algorithms.

3.1 Efficient search-based deterministic planning

This section reviews relevant work in efficient search-based planning when uncertainty is not considered during planning.

3.1.1 Bounded suboptimal heuristic search

We describe some popular bounded suboptimal variants of A^* [87]. Weighted A^* [94] inflates the heuristic function to place more emphasis on cost-to-goal. It computes bounded suboptimal solutions. Anytime Repairing A* (ARA*) [81] finds an initial path using a high weight on the heuristic, and then keeps decreasing the weight and improving solution quality. Multi-Heuristic A^* uses multiple inadmissible heuristics along with a one consistent heuristic function to compute bounded suboptimal solutions ([1]).

3.1.2 Using distance estimates in heuristic search

Explicit estimation search (EES) introduced by [110] use an estimate of node-distance to goal in bounded suboptimal search. However, the estimates used in the domains are all the number of nodes in the cost-wise shortest path to goal. Also, EES requires three priority queues, a heuristic function, and a distance function. With three queues to manage, EES has substantial overhead when compared to other bounded suboptimal search algorithms. Our work develops a heuristic function that anticipates search efforts, that can be used with many popular bounded suboptimal search algorithms in deterministic planning such as weighted A* and Multi-heuristic A*.

3.1.3 Abstract-space based heuristics in bounded suboptimal search

Abstract spaces have been investigated by [67] to reduce search efforts. AB-STRIPS, a modification of STRIPS, defines an abstraction space hierarchy from the STRIPS representation of a problem domain, and utilizes the hierarchy in solving problems ([101]). ([66]) identified a criterion for selecting useful abstractions, proposed a tractable algorithm for generating them, and empirically demonstrated that the abstractions reduce search.

Relaxed spaces and "safe" abstractions ([48]) have been used to compute solutions that could be refined/extended to the original space ([9]). Bacchus and Yang showed that plans in the relaxed space can be refined to a plan in original space with a high probability if the relaxed space satisfies the Downward Refinement Property (DRP). It is difficult to find non-trivial relaxed spaces with DRP. In our work, the relaxed spaces do not satisfy DRP, but we observe that we can identify conservative edges in the relaxed space that can direct the search in the original space towards goal. Also,we do not refine plans computed in the relaxed space but use them to compute a heuristic to be used by the original search. Many works use relaxed spaces to compute heuristics ([90]), ([54]), ([56]).

Pattern databases ([28]) store a table mapping states or sub-goals in a relaxed version of the original problem, to the cost-to-go of a pre-computed solution in the relaxed space to reach these sub-goals. ([49]) compute heuristics in abstract-spaces for automated-planning, where abstractions are computed using different sets of state variables.

The Fast Downward Planner attempts to combine refinement with heuristics ([49]). ([10]) define the weak refinement property (WRP) in studying the relationship between refinement and heuristics, and ([89]) define "strong matching", both of which come close to the conservative property, except that it is not a requirement in our relaxed spaces. Also, these works mainly deal with symbolic planning, whereas our focus is in motion planning.

([112]) use abstraction-based search in motion planning that divides the environment into overlapping regions and has the effect of heuristically guiding the search towards the next region, based on some computed bounds. However, they make assumptions about the convexity of the regions. In our case, there is no such assumption about the planning environment. To the best of our knowledge, no attempt has been made to identify and use conservative edges for heuristic computation in the context of motion planning.

3.2 Efficient planning under uncertainty

This section reviews relevant work in efficient planning under uncertainty.

3.2.1 Heuristic-search based methods

Heuristic-Search based methods utilize domain knowledge to compute heuristics that guide the search in belief spaces. Heuristic search methods do not need to evaluate the entire belief space while find the optimal policy, unlike dynamic programming methods. These methods typically minimize expected cost, as opposed to rewards typically used in other

classes of approaches, and are used for solving Goal-POMDPs—POMDPs with terminal states and no discounting of costs.

[44] proposed LAO* that is AO* for MDPs or belief MDPs with cycles. AO* algorithm is an extension of A* search algorithm that can solve problems formalized as an acyclic AND-OR graph (i.e., without loops). RTDP [12] performs a series of trials that consist of real-time lookahead searches leading to value backups on greedy paths to the goal in MDPs. The extension RTDP-BEL operates in the belief space [14]. It forward simulates a single branch from the start to the goal and updates the values only for the nodes on the branch. The initial values of the states are an admissible heuristic, and the values converge to the optimal value through repeated forward simulations of branches. Labeled RTDP is a variant of RTDP in which improves the converge of RTDP: values converge faster because the states with converged values are labelled as solved. [65] recently developed the Partially Observable Multi-Heuristic Dynamic Programming or POMHDP which is an anytime POMDP solver, that leverages multiple heuristics to efficiently compute high-quality solutions while guaranteeing asymptotic convergence to an optimal policy. Through iterative forward search, POMHDP utilizes domain knowledge to solve POMDPs with specific goals and an infinite horizon [65]. It utilizes a particle representation of a belief state instead of a factored belief space. Heuristic search-based methods for belief space planning do not explicitly reason about minimizing search efforts while ensuring bounded suboptimality.

3.2.2 Point-based methods

Point-based methods are a popular class of methods for planning in large POMDPs: they represent the value function by a vector set of sampled points and its piece-wise linear combination ([92]). This representation is used to update or backup the lower bound of the

value function (in the reward setting) by sampling a new belief state. Point-based methods compute the optimal policy by iteratively sampling a new belief state and backing up the lower bound. [92] introduced the point-based method called Point-Based Value Iteration (PBVI) with lower bound of the value function ([93]).

Point-based methods also the restrict value function updates to a subset of the belief space [103]. This subset is the search space that is reachable from the starting belief, as referred to as the reachable space, which grows as the search proceeds. [92] also introduced the idea of reachable belief spaces in their work on PBVI. They showed that the value function by a vector set of sampled points because the optimal value function is always piece-wise linear and convex in the belief space. However, point-based methods result in higher computational complexity and thus are not scalable to large search spaces typically encountered in real-world planning.

HSVI and HSVI2 utilize a heuristic strategy to focus the search towards regions of high uncertainty about the values [105, 106]. High uncertainty regions have a large gap between the upper and lower bounds of the value function. The initial values of the upper bounds and lower bounds can be computed as discussed in [103]. Apart from the domain of RockSample introduced in [105, 106], SARSOP (Successive Approximations of the Reachable Space under Optimal Policies) [72] is considered the state-of-the-art offline point-based POMDP solver in other benchmark domains. It also works by maintaining the upper and lower bounds of the value function. SARSOP uses heuristic exploration to converge to a subset of points optimally reachable from the start, often outperforming HSVI2 [72]. Anytime online variants of Point-based methods are possible ([100]).

Offline and online solvers: Typical POMDP solvers compute a policy prior to execution. They take significant time (sometimes hours) to terminate, and changes in environment dynamics require recomputing policies from scratch. In contrast, online solvers compute a

finite horizon (partial) policy for the current state of an agent, thereby interleaving planning and execution [100]. However, since these solvers have a finite, receding horizon, they are subject to getting stuck in local minima. Popular online POMDP solvers include DESPOT [107] and POMCP [104], which are Monte-Carlo based methods and discussed next.

3.2.3 Monte-Carlo based methods

The work on Partially Observable Monte Carlo Planning (POMCP) ([104]) introduced the idea of Monte Carlo-based planning in POMDPs. The idea of using Monte Carlo Planning in MDPs existed in Monte Carlo Tree Search (MCTS). These approaches maintain a a particle representation of a belief state and can sample the belief state and transitions, that converts the POMDP to an MDP. They use the Upper Confidence Bound applied to Trees (UCT) approach to guiding the tree search to balance between the exploration and exploitation. DESPOT (Deteminized Sparse Partially Observable Tree) ([107]) is a variant of POMCP. It gains efficiency by sparsification of the belief tree by sampling a small number of challenging scenarios.

However, when faced with large observation spaces, DESPOT becomes overly optimistic and computes sub-optimal policies, because of particle divergence [40]. [40] introduced a new online POMDP solver DESPOT-α, which builds upon DESPOT. DESPOT-α improves the practical performance of online planning for POMDPs with large observation as well as state spaces. Adaptive belief tree (ABT) was designed specifically to accommodate changes in the environment without having to replan from scratch ([71]). ABT is also based on POMCP. It reuses and improves the existing solution and update the solution as needed whenever the POMDP model changes ([108]). There are approaches to deal with continuous action and observation spaces ([102]), [108]).

3.2.4 Planning in special classes of belief-MDPs

[88] use a factored representation to represent separately the fully known and hidden components of a robot's state and derive a compact lower-dimensional representation of its belief space. With this representation, they use a point-based algorithm to compute approximate POMDP solutions. POMDP-lite [26] are a subclass of POMDPs in which the hidden state variables are constant or only change deterministically, and come up with a model-based Bayesian reinforcement learning algorithm to plan in this subclass. [18] propose a Bayesian reinforcement learning-based modelling and planning framework which uses Gaussian processes to model environmental uncertainty in a principled manner. They exploit the Bayesian RL formulation to plan more efficiently in these types of uncertain environments than previous methods are able to. [7] introduce Oracular Partially Observable Markov Decision Process (O-POMDP), a special type of POMDP in there is an "oracle" available—a human or a perfect sensor—in any state instead of observations, that tells the agent its exact state for a fixed cost. The agent is thus given access to information-gathering actions in addition to domain-level actions. They propose an algorithm to solve O-POMDP.

3.2.5 Planning under uncertainty in real-world robotics problems

In this section we review early and recent works that develop and evaluate planning under uncertainty algorithms in real-world robotics problems. Amato et. al. [5] have demonstrated for the first time that complex multi-robot domains can be solved with Dec-POMDP-based methods. [65] demonstrate the efficacy of their proposed POMHDP framework on a real-world, highly-complex, truck unloading application. [73] evaluate PPCP on a custom quad-rotor helicopter. Work in [6] enables a robot equipped with an arm to dynamically construct query-oriented MOMDPs for multi-modal predicate identification of objects. The robot's behavioral policy is learned from two datasets collected using real robots. Their

approach enables a robot to explore object properties in a way that is significantly faster while improving accuracies in comparison to existing methods.

There are many sources of uncertainty in real-world robot navigation problems. Wheel slippage and skidding can happen due to uneven terrain or slippery surfaces, which may introduce noise in the output of action execution. These noises may be hard to model precisely. There can be noise in sensing obstacles due to imperfect perception systems. A noisy localization system may make the navigation less robust. For robust navigation, it is important to effectively reason about uncertainties introduced by various sources.

An online version of the Despot algorithm introduced in [40] to effectively reasons about the uncertainties in a planning framework introduced by moving pedestrians. [42], [43] incorporate uncertainty in the state-space by modelling the belief state as a Gaussian distribution. They augment the state-space with the mean and covariance parameters of a Gaussian distribution. This state representation allows the formulation of the problem as a belief MDP, as well as the use of graph search algorithms on this state space to efficiently solve it. The results showed that this approach could find a path with the minimal cost that ensures the vehicle has a very little chance of collision with obstacles by approaching to feature-rich landmarks as necessary. ([17]) finds a path in the belief space by incrementally constructing a belief tree using RRT in Gaussian belief spaces . Other works also focus on using search algorithms to find a path in a Gaussian belief MDP ([76], [96]).

([86]) propose a motion planning framework in which the uncertainty is the articulated object models such as doors and drawers ([86]), and not in the robot state and object poses. They construct a belief MDP with a probability distribution of the articulated object model. They use LAO* to find a policy in this belief. They show that a robot can identify the articulation type of the given object by a sequence of actions from the planner. Some works in manipulation perform robust part feeding by utilizing contact between the parts and the

environment to align its pose ([2], [41]) . These approaches reduce uncertainty in object orientation on a plane by a sequence of actions. They require rigorous analysis of the configuration space that particularly depends on the shape of the object, which is usually not possible for high dimensional problems. Some recent works in robust grasping and in hand manipulation also utilize contact between the object parts and the environment to reduce uncertainty ([29], [33]). However, they do not use a motion planner.

Many works formulate object search as a POMDP. There is a very recent work that performs multi-resolution POMDP planning for multi-object search in 3D [118]. They design a novel octree-based belief representation to capture uncertainty of the target objects at different resolution levels, and derive abstract POMDPs at lower resolutions with dramatically smaller state and observation spaces. They demonstrate their approach on a mobile robot to find objects placed at different heights in two regions by moving its base and actuating its torso. Work by ([119]) uses a particle filter representation for a belief state and constructs a belief tree in offline by searching over a finite number of actions. They apply their work for 2D object pose identification. Notably, [8] first infers a room to search in then perform search by calculating candidate viewpoints in a 2D plane. [77] plans sensor movements online, yet assume objects are placed at the same surface level in a container with partial occlusion. [116] address object fetching on a cluttered tabletop where the robot's FOV fully covers the scene, and that occluding obstacles are removed permanently during search. [113] formulates the multi-object search (MOS) task on a 2D map using the proposed Object-Oriented POMDP (OO-POMDP).

3.2.6 Fully and partially observable non-deterministic planning

Fully Observable Non-Deterministic Planning, or more popularly known as FOND planning, is a body of work that relates to the probabilistic planning framework presented in this thesis.

Algorithms under FOND planning focus on planning in a partially observable environment, with sensing actions that implement perfect sensing. They use a series of calls to classical planners to iteratively construct and refine a policy.

Planning in a partially observable environment with perfect sensing (PPOS) can either be online, i.e., by interleaving planning, sensing, and acting; or offline, i.e., by generating a plan with decision points predicated on sensing outcomes. Online contingent plans ([4], [16], [14]) are generally easier to compute since integrating online sensing with planning eliminates the need to plan for a potentially exponential number of contingencies. In the absence of deadends, online contingent planning can be fast and effective.

On the other hand, offline contigent planning algorithms ([91], [98], [114], [13], [99]) constructs conditional plans with decision points for sensing outcomes and guarantees that the goal will be achieved if it is possible to do so. The plan is larger than an online plan but has the merit that it is generalized to deal with alternative sensing outcomes. PO-PRP [85] is a popular offline planner in PPOS environments that focuses on solving a compelling class of PPOS problems where the initial state specification includes a set of state constraints. PO-PRP demonstrates how PPOS problems can be solved offline by exploiting and extending a modern FOND planner. The main contribution of the PO-PRP work is that, in contrast to the commonly held belief that offline planning for PPOS is impractical, PO-PRP can produce conditional plans several orders of magnitude faster and smaller than the best planners. [75] shares similar execution structure with PO-PRP. Other offline contigent planners based on heuristic search include Contingent-FF ([54]), and the offline variant of CLG ([4]).

Partially-Observable NonDeterministic planner (POND) (Bryce, Kambhampati, and Smith 2006) relaxes the perfect sensing assumption in the FOND planning formulation. POND is a conditional progression planner that uses AO* search. It focuses on problems

where an agent starts in an uncertain state but has deterministic actions. ProbPRP [20] addresses the class of probabilistic planning problems where the objective is to maximize the probability of reaching a prescribed goal. ProbPRP leverages core similarities between probabilistic and fully observable non-deterministic (FOND) planning to construct a sound, offline probabilistic planner that exploits algorithmic advances from the state-of-the-art FOND planner, PRP [85], to compute compact policies that are guaranteed to bypass avoidable deadends.

RFF [109] is a work that closely relates to the framework of FAST-PPCP. It generate policies in MDPs by (1) determinizing the given MDP model into a classical planning problem; (2) building partial policies off-line by producing solution plans to the classical planning problem and incrementally aggregating them into a policy, and (3) using sequential Monte-Carlo (MC) simulations of the partial policies before execution, in order to assess the probability of replanning for a policy during execution. The objective of the RFF planner is to quickly generate policies whose probability of replanning is low and below a given threshold. However, the main differences between FAST-PPCP and RFF are as follows. RFF does not exploit the property of clear preferences used by Fast-PPCP. Neither does it aim to optimize or provide any guarantee on the value of the computed policy.

Chapter 4

Speeding Up Search-Based Motion Planning via Conservative Heuristics

4.1 Introduction

Weighted A* (wA*) [95], has been widely used for relatively low-dimensional motion planning problems such 2D navigation [37], path planning for UAVs [59], [83], and footstep planning for humanoids [58]. wA* with high weight shows better search efficiency in domains that show a strong correlation of the heuristic function with the node-distance-to-goal [115]. A weak correlation may create heuristic depression regions, or local minima, where the path suggested by the heuristic may not be feasible, severely degrading search efficiency [115]. We investigate the idea of computing heuristic functions that explicitly aim to reduce search expansions by wA*.

For heuristic computation, it is common to solve a simpler planning problem in a space formed by relaxing some constraints in the original space [19], [56]. We define an edge in the relaxed space as conservative if for each corresponding state in the original space

there is an edge to at least one corresponding successor state. Existence of a path composed only of conservative edges in the relaxed space, guarantees existence of a feasible path in the original space. If the heuristic computation finds such a path in the relaxed space, then simply following the heuristic gradient can guide the search to the goal, while expanding only the states that appear in the solution. Our first contribution is in observing that in motion planning problems formulated as heuristic search one can often identify conservative edges in the relaxed space. Secondly, we propose a heuristic computation algorithm that minimizes the use of non-conservative edges to reduce inefficient expansions in the original space.

Motivation behind the conservative property is similar to several ideas explored in classical hierarchical planning. Relaxed spaces and "safe" abstractions [48] have been used to compute solutions that could be refined/extended to the original space [9]. Bacchus and Yang showed that plans in the relaxed space can be refined to a plan in original space with a high probability if the relaxed space satisfies the Downward Refinement Property (DRP). It is difficult to find non-trivial relaxed spaces with DRP. In our work, the relaxed spaces do not satisfy DRP, but we observe that we can identify conservative edges in the relaxed space that can direct the search in the original space towards goal. Also, we do not refine plans computed in the relaxed space but use them to compute a heuristic to be used by the original search. Many works use relaxed spaces to compute heuristics [90], [54], [56]. Pattern databases [28] store a table mapping states or sub-goals in a relaxed version of the original problem, to the cost-to-go of a pre-computed solution in the relaxed space to reach these sub-goals. [51] compute heuristics in abstract-spaces for automated-planning, where abstractions are computed using different sets of state-variables. The Fast Downward Planner attempts to combine refinement with heuristics [49]. [10] define the weak refinement property (WRP) in studying the relationship between refinement and

heuristics, and [89] define "strong matching", both of which come close to the conservative property, except that it is not a requirement in our relaxed spaces. Also, these works mainly deal with symbolic planning, whereas our focus is in motion planning. [112] use abstraction-based search in motion-planning that divides the environment into overlapping regions and has the effect of heuristically guiding the search towards the next region, based on some computed bounds. However, they make assumptions about the convexity of the regions. In our case, there is no such assumption about the planning environment. To the best of our knowledge, no attempt has been made to identify and use conservative edges for heuristic computation in the context of motion planning.

4.2 Planning with conservative heuristics

In this section, we define notations and definitions and describe the problem. We then explain the algorithm to compute conservative heuristics.

4.2.1 Definitions, notations and problem description

Consider a graph $G = (S, E, c)$, where S is the set of states, $E = \{(s, s') | s, s' \in S\}$ denotes the set of feasible transitions/edges in the graph and, c is a cost-function such that $c(s_i, s_j)$ is the cost of an edge (s_i, s_j). A planning problem consists of finding a path $\pi(s_i, s_j)$ in G from s_i to s_j. $\pi^*(s_i, s_j)$ denotes the least-cost path between s_i and s_j. The cost of any path $\pi(s_i, s_j)$ is the cumulative cost of all edges along it and is denoted by $c(\pi(s_i, s_j))$. Let $h_c : S \to \mathbb{N}$ be our conservative heuristic function estimating cost-to-goal. We assume that $s_i \in S$ is a goal-state if and only if $h_c(s_i) = 0$. We use wA* to compute a path in G from a start state s_{st} to any state in the goal-set $S_g = \{s \in S \mid h_c(s) = 0\}$.

Heuristic space:

Consider another state-space $\tilde{S}$, an abstract space used to compute heuristics. We call it the heuristic-space. Let $\lambda : S \rightarrow \tilde{S}$ be a many-to-one mapping representing the projection of each state in S to the heuristic-space $\tilde{S}$, such that $|\tilde{S}| < |S|$. Moreover, $\lambda^{-1}(\tilde{s}) = \{s \in S \mid \lambda(s) = \tilde{s}\}\ \forall \tilde{s} \in \tilde{S}$. The heuristic-space has its own set of transitions $\tilde{E} = \{(\tilde{s}_i, \tilde{s}_j) | \tilde{s}_i, \tilde{s}_j \in \tilde{S}\}$. Let $\tilde{G}$ be the graph defined by $\tilde{S}$ and $\tilde{E}$. $\pi(\tilde{s}_i, \tilde{s}_j)$ denotes a path in $\tilde{G}$ from $\tilde{s}_i$ to $\tilde{s}_j$, and $c(\pi(\tilde{s}_i, \tilde{s}_j))$ denotes its cost in $\tilde{G}$. We assume that for every pair of states s_i and s_j in S,

$$c(\pi^*(s_i, s_j))) \geq c(\pi^*(\lambda(s_i), \lambda(s_j))) \tag{4.1}$$

$$\text{if } \exists\ \pi(s_i, s_j)\ \ s.t\ \ |c(\pi(s_i, s_j))| < \infty,$$
$$\text{then } \exists\ \pi(\lambda(s_i), \lambda(s_j))\ \ s.t\ \ |c(\pi(\lambda(s_i), \lambda(s_j)))| < \infty \tag{4.2}$$

We assume states in the goal-set S_g map to one goal-state $\tilde{s}_g$ in the heuristic space, ie, $\tilde{s}_g = \lambda(s_g) \forall s_g \in S_g$.

Conservative edges:

An edge $(\tilde{s}, \tilde{s}') \in \tilde{E}$ is conservative iff $\forall s \in \lambda^{-1}(\tilde{s})\ \exists\ (s, s') \in E$ s.t $s' \in \lambda^{-1}(\tilde{s}')$. A conservative edge $(\tilde{s}, \tilde{s}')$ guarantees that every state $s \in \lambda^{-1}(\tilde{s})$ is connected to at least one state in $\lambda^{-1}(\tilde{s}')$. Thus, existence of a path in the heuristic-space from $\lambda(s_i)$ to $\lambda(s_j)$ which consists of only conservative edges, guarantees existence of a path from s_i to s_j in the original space. Fig. 4.1 illustrates this reasoning. If the search in G is guided to always prefer successors connected via conservative edges in $\tilde{G}$, it would reach the goal by expanding only such successors, making the number of expansions equal to the solution size.

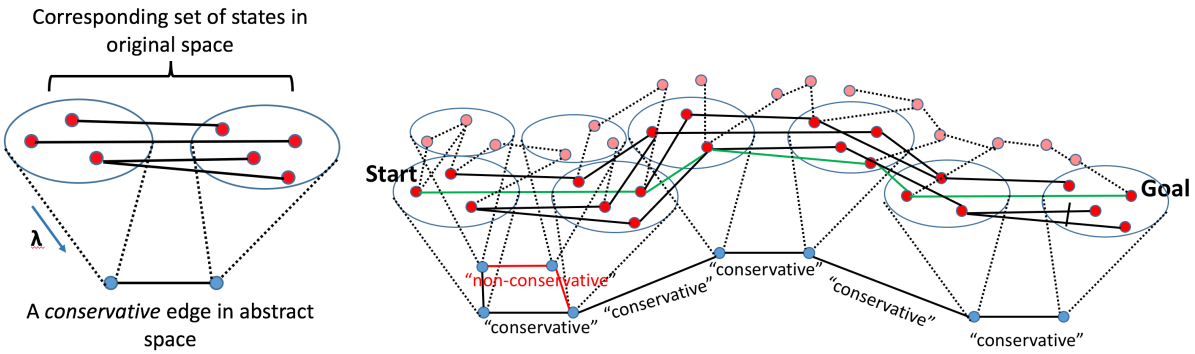


Figure 4.1: (a) A conservative edge in ($\tilde{S}$) and its corresponding set of states and edges in (G). (b) Non-conservative edges (red) with missing edges between some pairs of corresponding states in S. (c) Path (green) in S, comprising solely of conservative edges in $\tilde{S}$.

Identifying conservative edges in $\tilde{G}$ is a domain-dependent process. For example, when planning in $S = (x, y, orientation)$ for a non-circular robot, $\tilde{S}$ can be obtained by dropping the orientation, or, $\tilde{S} = (x, y)$. Here, the heuristic-space is obtained by abstracting away a specific geometric property of the robot (in this case the orientation), which is equivalent to treating the non-circular robot as a circular one in the heuristic-space. Inflating obstacles by the radius of the robot-footprint's circumcircle (circumradius) identifies the space that the robot can physically occupy for *all* possible orientations. States in $\tilde{S}$ lying outside of this inflated-obstacle region are surely not in collision with obstacles, therefore an omni-directional robot can surely move between neighbouring states. Thus, edges between these states are conservative. Fig. 4.2 shows the conservative and non-conservative edges in a 2D environment inflated by the robot circumradius.

4.2.2 Conservative heuristic computation

We want to compute $h_c(s)$ such that it (1) guides the search in the original space along paths that minimize the number of non-conservative edges in $\tilde{S}$, (2) prefers the shortest between all paths made purely of conservative edges and, (3) is α consistent ($\infty > \alpha > 1$). We define a heuristic to be α-consistent if for all $s, s' \in S$ such that s' is a successor of s,

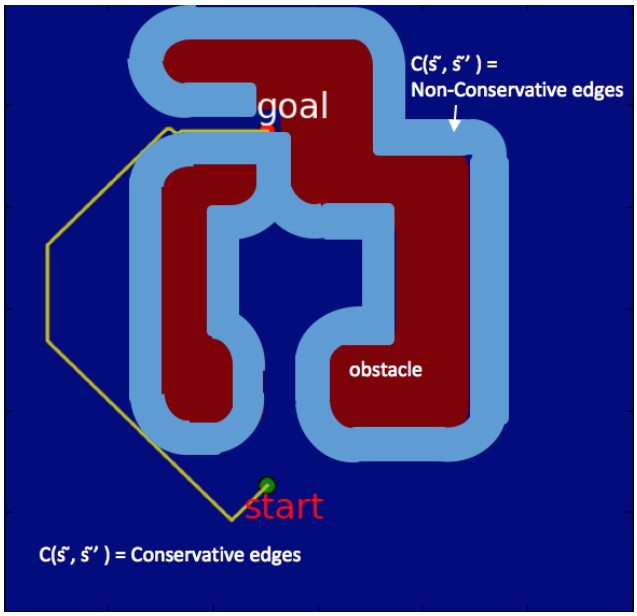


Figure 4.2: A planning environment with obstacles (red) inflated by the circumradius of a polygonal robot. Edges inside the inflated region (light blue) are non-conservative, while those outside (dark blue) are conservative.

$h_c(s) \leq \alpha c(s,s') + h_c(s')$, and $h_c(s) = 0$ for all $s \in S_g$. Consider a graph $\tilde{G}_m = (\tilde{S}, \tilde{E}, c_m)$, which is $\tilde{G}$ but with modified edge-cost. For any $s \in S$ and its image $\tilde{s} = \lambda(s)$, we define $h_c(s) = c_m(\pi^\star_m(\tilde{s}, \tilde{s}_g))$, where $\pi^\star_m$ is the optimal path in $\tilde{G}_m$ from $\tilde{s}$ to $\tilde{s}_g$, Let $\tilde{E}_{co}$ be the set of conservative edges in $\tilde{E}$, $\tilde{E}_{nco} = \tilde{E} \setminus \tilde{E}_{co}$. Let c_{min} be the minimum edge-cost in $\tilde{G}$. We define c_m in Eq. (4.3).

$$c_m(\tilde{s}_i, \tilde{s}_j) = \begin{cases} c_{min}/|\tilde{E}_{co}| & \text{if } (\tilde{s}_i, \tilde{s}_j) \in \tilde{E}_{co} \\ \alpha c_{min} & \text{if } (\tilde{s}_i, \tilde{s}_j) \in \tilde{E}_{nco} \end{cases} \tag{4.3}$$

From Eq. (4.3), we see that the optimal path $(\pi^\star_m(\tilde{s}, \tilde{s}_g))$ in $\tilde{G}_m$ would rather consist of all possible conservative edges than incorporating a single non-conservative one. Thus, the desired properties of h_c are achieved.

Pseudocode 1 shows the heuristic computation in detail. We need to first compute c_{min}

Pseudocode 1 Conservative Heuristic Computation

```
1:  c_min = min_{∀(s̃,s̃')∈Ẽ} c(s̃,s̃')
2:  n = 0
3:  for every (s̃,s̃') do                                   ▷ Compute |Ẽ_co|
4:      if is_conservative((s̃,s̃')) == True then
5:          n = n + 1
6:      end if
7:  end for
8:  |Ẽ_co| = n
9:  g(s̃_g) = 0
10: OPEN = s̃_g, CLOSED = ∅
11: while at least one s̃ ∈ S̃ hasn't been expanded and OPEN ≠ ∅ do
12:     remove s̃ from OPEN with minimum g(s̃)
13:     insert s̃ into CLOSED
14:     for every predecessor s̃' of s̃ s.t s̃ not in CLOSED do
15:         if is_conservative((s̃,s̃')) == True then
16:             c_m(s̃,s̃') = c_min/|Ẽ_co|
17:         else
18:             c_m(s̃,s̃') = α c_min
19:         end if
20:         if g(s̃) > g(s) + c_m(s̃,s̃') then
21:             g(s̃) = g(s) + c_m(s̃,s̃')
22:             Insert s̃ into OPEN with g(s̃) as key
23:         end if
24:     end for
25: end while
```

and total number of conservative edges $|\tilde{E}_{co}|$ (Lines 1:5). We then compute the shortest path in $\tilde{G}_m$ from $\tilde{s}_g$ to every state in $\tilde{G}_m$. This is done by running a backward Dijkstra's search on $\tilde{G}_m$ from $\tilde{s}_g$ to every state in $\tilde{G}_m$. $\tilde{G}_m$ is implicitly constructed: for each expanded $\tilde{s}$ and a predecessor $\tilde{s}'$, we check whether $(\tilde{s},\tilde{s}')$ is conservative and assign costs according to the scheme described in Eq. (4.3) (Lines 6:19).

For each state s expanded by wA* search in the original graph G, we first find its projection $\tilde{s} = \lambda(s)$ and query its g-value $g(\tilde{s})$, which was updated when $\tilde{s}$ was expanded in the heuristic computation search. $g(\tilde{s})$ is the cost of the shortest path in $\tilde{G}_m$. Therefore, $h_c(s) = g(\tilde{s})$.

We prove the following (full proofs can be found in [21]):

- h_c is α consistent.
- wA* using h_c is complete, with cost of the returned solution being no more than $w.\alpha$ times optimal solution cost in G (Theorem 1,3).
- The chosen cost-scheme guarantees that the longest purely conservative path has lower cost than any path with a non-conservative edge in $\tilde{G}_m$ (Theorem 4).
- If a purely conservative path exists in $\tilde{G}_m$ from $\tilde{s}_{st}$ to $\tilde{s}_g$, then number of expansions made by wA* with sufficiently large w equals the length of the shortest purely conservative path from $\tilde{s}_{st}$ to $\tilde{s}_g$ (Theorem 5). If not, h_c will still guide the search towards the use of edges in G that have conservative mappings in $\tilde{G}_m$. In doing so, wA* returns a solution with cost within the stated sub-optimality bound. However, no guarantees on the number of expansions can be provided.

It is to be noted that finding conservative edges (function `is_conservative`$((\tilde{s},\tilde{s}'))$ in Lines 3 and 13) is domain-dependent. However, in navigation-planning domains, domain-knowledge helps in computing conservative edges efficiently. In the aformentioned example of planning in $(x, y, \mathit{orientation})$, given the map of the environment and the circumradius r_c of the robot, an edge can be deemed conservative by checking if both the vertices comprising the edge are at a distance greater than r_c from the nearest obstacle [1]. This check is an $O(1)$ operation and can be performed during the implicit construction of $\tilde{G}_m$. The next section shows how conservative edges can be computed efficiently for two navigation domains: (1) path-planning for a UAV, and (2) humanoid foot-step planning.

[1] Distance-Transforms are typically used to compute and store the distance of each state in a 2D or 3D grid to its nearest obstacle.

4.3 Theoretical properties

Let $(\tilde{s},\tilde{s}')_{co}$ and $(\tilde{s},\tilde{s}')_{nco}$ denote a conservative and a non-conservative edge respectively. Let $c_m(\tilde{s},\tilde{s}')$ denote the cost of edge $(\tilde{s},\tilde{s}')$ in the modified abstract graph $\tilde{G}_m$. As mentioned, $c_m(\tilde{s},\tilde{s}')_{co} = c_{min}/|\tilde{E}_{co}|$ and $c_m(\tilde{s},\tilde{s}')_{nco} = \alpha c_{min}$. We call any path $\pi_{co}(\tilde{s},\tilde{s}_g)$ from $\tilde{s}$ to $\tilde{s_g}$ purely conservative if it consists of only conservative edges, and a path $\pi_{nco}(\tilde{s},\tilde{s}_g)$ non-conservative if it has at least one non-conservative edge. Let $\mathbb{P}_{co}$ and $\mathbb{P}_{nco}$ be the set of purely conservative and non-conservative paths respectively. We will denote conservative heuristics above by h_c. The open-list at expansion step i of wA* is denoted by $OPEN_i$

Theorem 1: *Completeness: For a finite graph G, wA* using h_c is complete, that is, if a finite-cost path exists in G wA* using h_c will find it.*

Proof. For a state s and goal s_g, $h_c(s)$ is the cost of the shortest path in the modified abstract graph $\tilde{G}_m$ from $\tilde{s}$ to $\tilde{s}_g$. If a finite-cost path exists in G from s to s_g, from the property of abstract graphs stated in (1) in the paper, a finite-cost path surely exists in $\tilde{G}$ and therefore in $\tilde{G}_m$, which has the same set of edges as $\tilde{G}$. Also, the edge-costs in $\tilde{G}_m$ are finite. Therefore, $h_c(s)$ is finite, and wA^* using h_c is guaranteed to find a path in G if it exists. □

Theorem 2: *$h_c(s)$ is α-consistent: $\forall s,s' \in S$, $s.t(s,s') \in E$, $h_c(s) \leq \alpha c(s,s') + h_c(s')$ and $h_c(s_g) = 0$.*

Proof. Let $\tilde{s} = \lambda(s)$, and $\tilde{s}' = \lambda(s')$. Since edge-costs in $\tilde{G}_m$ are positive, from triangle inequality: $c^*_m(\tilde{s},\tilde{s}_g) \leq c^*_m(\tilde{s},\tilde{s}') + c^*_m(\tilde{s}',\tilde{s}_g)$.
For $(\tilde{s},\tilde{s}')$, let π^*_c be the least cost path in $\tilde{G}$ with cost $c^*(\tilde{s},\tilde{s}')$ and number of edges n^{c^*}. Let $c_m(\pi^*_c)$ be its cost in $\tilde{G}_m$. Then, $c_m(\pi^*_c) \leq \max(c_m(\pi^*_c)) = \sum_{i=0}^{n^{c^*}} \alpha c_{min} = n^{c^*}\alpha c_{min}$.
Since $c^*_m(\tilde{s},\tilde{s}')$ is the cost of the shortest path in $\tilde{G}_m$,

$$c_m^*(\tilde{s},\tilde{s}') \leq c_m(\pi_c^*) \leq n^{c^*}\,\alpha c_{min} \quad .$$

$c^*(\tilde{s},\tilde{s}') = \sum_{i=0}^{n^{c^*}} c(\tilde{s}_i,\tilde{s}_{i+1}), \forall \tilde{s}_i \in \pi_c^*$.

$$\therefore c^*(\tilde{s},\tilde{s}') \geq n^{c^*} c_{min}$$

$\therefore c_m^*(\tilde{s},\tilde{s}') \leq c_m(\pi_c^*) \leq n^{c^*}\alpha c_{min} \leq \alpha c^*(\tilde{s},\tilde{s}') \leq \alpha c(s,s') \implies c_m^*(\tilde{s},\tilde{s}_g) \leq \alpha c(s,s') + c_m^*(\tilde{s}',\tilde{s}_g)$

Since $h_c(s) = c_m^*(\tilde{s},\tilde{s}_g)$ and $h_c(s') = c_m^*(\tilde{s}',\tilde{s}_g)$,

$\therefore h_c(s) \leq \alpha c(s,s') + h_c(s')$. $h_c(s_g) = c_m^*(\tilde{s}_g,\tilde{s}_g) = 0$ □

Theorem 3 *For a finite graph G, the solution returned is guaranteed to be no worse than $w \cdot \alpha$ times the optimal solution cost in G, where w is the heuristic inflation weight in wA*.*

Proof. Since h_c is α-consistent, wA* with or without re-expansions returns a solution that is $w \cdot \alpha$ times the optimal cost. □

Theorem 4 *In $\tilde{G}_m$ If $\exists\ \pi(\tilde{s},\tilde{s}_g) \in \mathbb{P}_{co}$ then the least-cost path from $\tilde{s}$ to $\tilde{s_g}$ is the shortest length purely conservative path, denoted by $\pi_{co}^*(\tilde{s},\tilde{s}_g)$.*

Proof. **Lemma 1**: The cost of any purely conservative path is strictly less than the cost of any non-conservative path in $\tilde{G}_m$.

Proof Let $\pi_{no} \in \mathbb{P}_{no}$ $\pi_{nco} \in \mathbb{P}_{nco}$ be arbitrary and fixed. **Lemma 1.a**: $c_m(\pi_{nco}) > c_{min}$

Let $E_{nco} \neq \{\}$ be the non-conservative edges of π_{nco}.

$$c_m(\pi_{nco}) \geq \sum_{e \in E_{nco}} c_m(e) = \sum_{e \in E_{nco}} \alpha c(e) > c_{min}$$

Lemma 1.b: $c_m(\pi_{co}) \leq c_{min}$

Let E_{co} be the set of edges in π_{co}

$$c_m(\pi_{co}) = \sum_{e \in E_{co}} c_m(e) = \sum_{e \in E_{co}} \frac{c_{min}}{\tilde{E}_{co}} = |E_{co}| \frac{c_{min}}{|\tilde{E}_{co}|} \leq c_{min}$$

Combining **1.a** and **1.b**, $c_m(\pi_{co}) \leq c_{min} < c_m(\pi_{nco})$

Thus if $\exists\ \pi(\tilde{s}, \tilde{s}_g) \in \mathbb{P}_{co}$ then the least-cost path $\pi^*_{co}(\tilde{s}, \tilde{s}_g)$ is purely conservative. As all conservative edges have the same cost in $\tilde{G}_m$, $\pi^*_{co}(\tilde{s}, \tilde{s}_g)$ must be the shortest length purely conservative path. □

Theorem 5 *For a state $s_{c0} \in S$, let us assume there exists a purely conservative path from $\tilde{s}_{c0}$ to $\tilde{s}_g$, and let $\pi^*_{co}(\tilde{s_{c0}}, \tilde{s}_g) = \tilde{s}_{c0}\tilde{s}_{c1}.....\tilde{s}_{n-1}\tilde{s}_g$ be the least-cost purely conservative path of size n from its abstract projection $\tilde{s}_{c0}$ to $\tilde{s}_g$. Let $\pi(s_0, s_g)$ be the solution found by wA*. Then, $\pi(s_{c0}, s_g) = s_{c0}s_{c1}....s_{n-2}s_g = \pi_{co}(s_{c0}, s_g)$, where $\pi_{co}(s_{c0}, s_g) = s_{c0}s_{c1}....s_{n-2}s_g$ is a path in G corresponding to a purely conservative path in $\tilde{G}_m$ s.t $\lambda(s_{ci}) = \tilde{s}_{ci}$, $\forall i \in [n]$. If s_{c0} is the start state, wA* with a sufficiently large weight w will expand only along $\pi_{co}(s_{c0}, s_g)$ in path order, thereby making the number of expansions equal to the solution size n.*

Proof. Sketch: Lemma 0: In each expansion step of wA* with $w \to \infty$, the state with $min(h_c(s))$ in OPEN will be expanded. To prove **Theorem 5** we need to show the statement $T(n):\ \forall i \in [n]$, *wA* with $w \to \infty$ expands s_{ci} on its expansion step i.* We prove this by induction on the expansion step number x. For x = 0, the open-list $OPEN_0$ has only $\{s_{c0}\}$ and is surely expanded. Assuming $T(j)$ is true for $j \leq i$ for some $i \in [n]$, we now show $T(i+1)$ is true, as follows:

Lemma 2 For any state s_{ci} on $\pi_c(s_{c0}, s_g)$ in original graph, let S'_i be the set of successors. Out of all successors in S'_i, $h_c(s'^*_{ci}) = h_c(s_{c(i+1)}) = \min_{s' \in S'_i} h_c(s')$. OR, the minimum h-value is of that successor $s_{c(i+1)}$ which lies on $\pi_{co}(s_{c0}, s_g)$, ie, whose abstract projection $\tilde{s}_{c(i+1)}$ lies on $\pi^*_{c0}(\tilde{s_{c0}}, \tilde{s}_g)$.

Lemma 3 For any state s_{ci} on $\pi_{co}(s_{c0}, s_g)$ and its successor $s_{c(i+1)}$ on $\pi_{co}(s_{c0}, s_g)$, $h_c(s_{c(i+1)}) < h_c(s_{ci})$.

If **Lemma 2** and **Lemma 3** are proved, then using them we prove $T(i+1)$ as follows: If s is expanded at x=i, $h_c(s) = \min_{s \in OPEN_i}(h_c(s))$ from Lemma 0, where $OPEN_i$ is the open list at expansion step i. $T(i)$ is true, $\implies h_c(s_{ci}) = \min_{s \in OPEN_i}(h_c(s))$. At x=i+1, $OPEN_{i+1} = \{OPEN_i - s_{ci}\} \cup S_i^{'}$. From **Lemma 2** and **Lemma 3** we have $h_c(s_{c(i+1)}) < h_c(s_{ci}) < h_c(s) \forall s \in \{OPEN_i - s_{ci}\}$. Thus, $h_c(s_{c(i+1)}) < h_c(s) \forall s \in S_i^{'}$ AND $h_c(s_{c(i+1)}) < h_c(s) \forall s \in \{OPEN_i - s_{ci}\} \implies h_c(s_{c(i+1)}) < h_c(s) \forall s \in S_i^{'} \cup \{OPEN_i - s_{ci}\}$ $\implies h_c(s_{c(i+1)}) < h_c(s) \forall s \in OPEN_{i+1}$.

This means that $h_c(s_{c(i+1)})$ is $\min_{s \in OPEN_{i+1}}(h_c(s))$, and thus $s_{c(i+1)}$ is expanded at x=i+1, proving $T(i+1)$.

Lemma 2: *Sketch of proof*: $h_c(s)$ of a successor s' in $S_i^{'}$ the cost in $\tilde{G}_m$ of either (a) a least-cost non-conservative path from $\tilde{s}'$, or (b) a least-cost purely conservative path from $\tilde{s}'$. From Theorem 4, $h_c(s'_{ci}) < h_c(s')$ for all s' of type (a), or, out of all successors, those that have a purely conservative path as the least-cost path always have h value strictly less than other successors that do not. Out of all abstract successors in (b), if $\tilde{s}_{ci}^{'*}$ is the abstract successor with the minimum valued least-cost conservative path among all successors, we show that $\tilde{s}_{ci}^{'*}$ lies on $\pi_{co}^*(\tilde{s_{ci}^{'*}}, \tilde{s}_g)\}$ which is a sub-path of $\pi_{co}^*(\tilde{s_{c0}}, \tilde{s}_g)$. $\pi_{co}^*(\tilde{s_{c0}}, \tilde{s}_g)$ maps to $\pi_{co}(s_{c0}, s_g)$ in G. Thus, $h_c(s_{ci}^{'*}) = h_c(s_{c(i+1)}) = \min_{s' \in S_i^{'}} h_c(s')$.

Lemma 3: *Proof*: $h_c(s_{ci}) = c_m(\pi_c^*(\tilde{s_{ci}}, \tilde{s}_g))$, and $h_c(s_{c(i+1)}) = c_m(\pi_c^*(s_{c(\tilde{i+1})}, \tilde{s}_g))$. $c_m(\pi_c^*(\tilde{s_{ci}}, \tilde{s}_g)) = c_{min}/|\tilde{E}_{co}| + c_m(\pi_c^*(\tilde{s_{ci}}, \tilde{s}_g))$ which follows from Lemma 2.

$c_{min}/|\tilde{E_{co}}| > 0 \therefore c_m(\pi_c^*(s_{c(\tilde{i+1})}, \tilde{s}_g)) < c_m(\pi_c^*(\tilde{s_{ci}}, \tilde{s}_g)). \therefore h_c(s_{c(i+1)}) < h_c(s_{ci})$. □

$|\tilde{E}_{co}|$ can be computed while computing $c *_m (\tilde{s}, \tilde{s}_g)$ for all states in $\tilde{G}$. However, if $c *_m (\tilde{s}, \tilde{s}_g)$ is computed lazily on-demand when the heuristic value of any state $\in \lambda^{-1}(\tilde{s})$ is queried, $|\tilde{E}_{co}|$ may not be known in advance. In such cases, $|\tilde{E}_{co}|$ can be replaced by

$|\tilde{E}|$ which is known in advance. Since $|\tilde{E}| \geq |\tilde{E}_{co}|$, $c_{min}/|\tilde{E}| \leq c_{min}/|\tilde{E}_{co}|$ and all the above theorems still hold.

4.4 Implementation and experimental analysis

In this section, we present the experimental setup and results of empirical analysis in the domains of path planning for a UAV and humanoid footstep planning.

4.4.1 Path planning for a UAV in (X,Y,Z)

We first evaluated the effectiveness of our heuristic computation for path planning in $S = (X,Y,Z)$ for a simulated omnidirectional UAV. We discretized the environment (Figure 4.3 bottom left), terrain (obstacles) in maroon) into $600 \times 600 \times 400$ cells. G is a 3D 26-connected grid with transition costs proportional to euclidean distance between states.

Heuristic space:

$\tilde{G}$ is defined by dropping the z, or, $\lambda([x,y,z]) = [x,y]$. Thus, $\tilde{G}$ is a 2D 8-connected grid. We assume a maximum flying range in z for the UAV given by z_{max}. We consider $\tilde{s} = [x,y] \in \tilde{S}$ as 'free' if there exists a $z < z_{max}$ for which $s = [x,y,z]$ is obstacle-free, thus ensuring Eq. (4.2) holds.

Conservative edge computation:

For every 2D $[x,y]$ state, obstacles in z are represented as the terrain elevation-map (Figure 4.3, example elevation values z_e shown in yellow). For a 2D edge $([x,y],[x',y'])$, let z_e and z'_e be the respective elevations for $[x,y]$ and $[x',y']$. We compute $z_d = |z_e - z'_e|$, the absolute difference in elevation of two adjacent 2D states forming the edge. Since the 3D grid is

26-connected, a z_d value of 0 or 1 implies that every $s = [x,y,z]$ projecting to $[x,y]$ has at least one collision-free edge to a $s' = [x',y',z']$ projecting to $[x',y']$, making $([x,y],[x',y'])$ conservative (Fig 4.3 (top left), red and blue). However, $z_d > 1$ indicates that there is one state $s = [x,y,z]$ projecting to $[x,y]$, which has no collision-free edge to any $s' = [x',y',z']$ projecting to $[x',y']$, making $([x,y],[x',y'])$ non-conservative (Fig 4.3 (top left), green. z_d = 2).

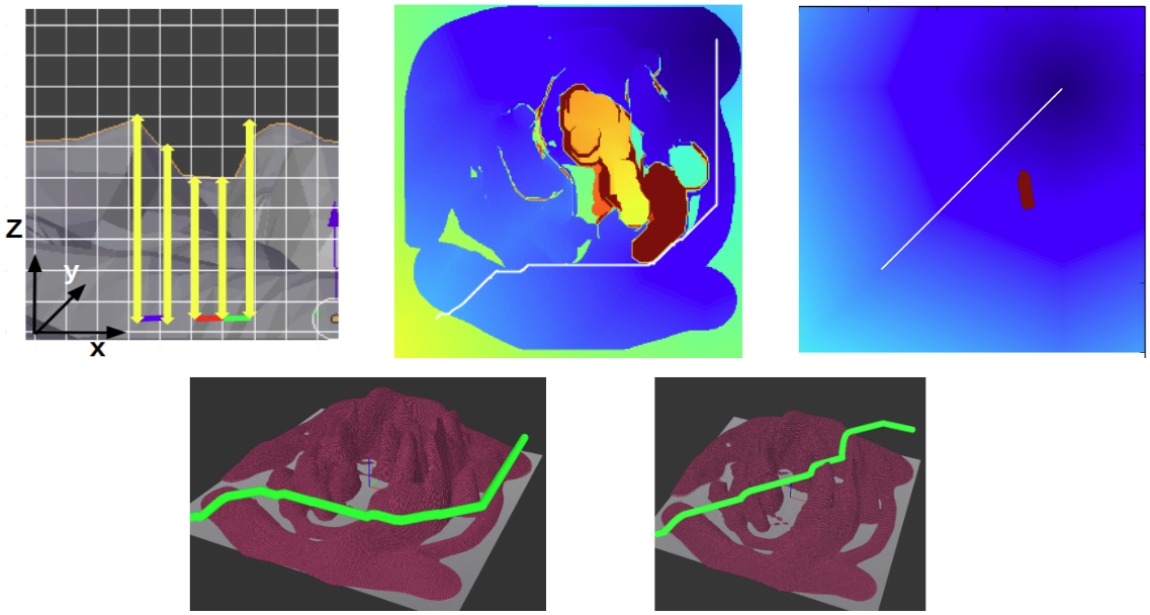


Figure 4.3: (top left) Cons.(red and blue) and a non-Cons. edge(green) for UAV domain, terrain elevation map (grey: terrain, yellow: elev values(z_e)). (top center, right) visualized h-values and optimal 2D paths by h_c and h_b respectively. (bottom left, right) 3D paths using h_c, h_b.

type of environment	heuristic	w in wA*	succ rate	# expansions	sol size	sol cost	planning time(s)	heur comp time(s)
difficult	h_c	100	100%	534±99	534±99	5333±994	0.002±0.001	0.21±0.003
difficult	h_b	100	80%	56687±30740	537±94	5357±943	0.028±0.03	0.21±0.01
easy	h_c	100	100%	894±1263	381±96	3798±969	0.001±0.001	0.21±0.009
easy	h_b	100	100%	1921±4300	346±112	3452±1160	0.002±0.002	0.20±0.05

Table 4.1: Comparison of h_c with baseline h_b for UAV domain in 'difficult' (more local minima) and 'easy' scenarios

heuristic	w in wA*	succ. rate	# expansions	sol size	sol cost	planning time(s)	heur comp time(s)
h_c	100	37/39	1720±5850	50±35	156250±110820	6±19.8	0.07 ±1.39
h_b	100	24/39	16000±19000	61.1±62.8	51630±52396	36±43	0.036±0.006

Table 4.2: Comparison of h_c with baseline h_b in Footstep Planning.

Results:

We compare h_c with a base-line heuristic h_b, which is cost of the optimal path in (x,y) space but with the regular euclidean 2D edge-costs. h_c is higher (yellow, red regions in Fig 4.3 (top center)) in the central region where elevation differences are steeper causing many infeasible 3D edges and more non-conservative edges. As a result, h_c guides the search away from this region and on purely conservative paths (Fig 4.3 (top center) white-line), unlike h_b (Fig 4.3 (top right)). We evaluate in 20 'Easy'(gradual terrain slopes, lesser depression regions) and 20 'Difficult' (steeper slopes). Table 4.1 shows results. Success rate indicates the number of instances when the planner finds a solution. Heuristic computation time includes time taken to identify conservative edges and run the Backward Dijkstra's search in $\tilde{G}_m$, Planning time indicates time taken by wA* to compute a solution, total time being the sum of both. Statistics are computed for cases in which both planners were able to find a solution. h_c has significantly less expansions, while producing similar solution costs and sizes. For the example in Fig 4.3, h_c (Fig 4.3 (bottom left)) generates a path around the steep regions, whereas h_b (Fig 4.3 (bottom right)) computes a path through the steep-sloped regions. The number of expansions using h_c is 829, which is exactly equal to the solution size (829), and significantly less than the number of expansions using h_b (20940). However, solution cost using h_c is 8280, which is slightly greater than that using h_b (6270). For the 'easy' scenarios where chances of encountering local minima are low, search using h_c and h_b performs similarly.

4.4.2 Humanoid footstep planning for bipedal walk

For this domain, state $s = [x_l, y_l, \theta_l, x_r, y_r, \theta_r, ID] \in S$ consists of global position and orientation of the two feet and ID of the foot being moved next (active foot). The environment was divided into 800x800 cells. θ has a resolution of 45°. The active foot moves relative to the pivot foot. For each foot as pivot, there are 15 feasible motions of the active foot in the form of $a = [\delta x, \delta y, \delta\theta]$ relative to pivot foot. Transition-costs are proportional to euclidean distance between the feet centers.

Heuristic Space:

$\tilde{G}$ is a 2D 8-connected grid. λ projects the 2 feet-positions in s to the 2D grid by computing their mean. Cost of an edge is proportional to the euclidean distance between 2D cell-centres.

Conservative edge computation:

For an $(\tilde{s}, \tilde{s}')$ to be conservative, *every* $s \in \lambda^{-1}(\tilde{s})$ should have *at least one* valid pose leading to a state in $\lambda^{-1}(\tilde{s}')$, such that the mean moves along $(\tilde{s}, \tilde{s}')$. Owing to the kinematic constraints of this humanoid, most edges in set $\tilde{E}$ are non-conservative. Consider the example in Fig 4.4, where the active foot (right) can have 2 feasible motions: $a_1 = [2, 0, 0]$ or $a_2 = [0, 0, 45^\circ]$. Let s_1, s_2 be states in $\lambda^{-1}(A)$, shown in Fig 4.4 (left) in red and black respectively. None of the actions a_1 or a_2 applied in s_2 result in a successor that projects to B. Similarly, no action applied in s_1 results in a successor that projects to C. Thus, (A, B) and (A, C) are both non-conservative edges.

However, consider the state D, 5 edges North of A in the heuristic-space (Fig 4.4 (center)). Applying a_1 thrice from s_1 (Fig 4.4 (center)) moves the mean to D. Applying a_2 and then a_1 four times from s_2 (Fig 4.4 (right)) also moves the mean to D. Thus, we

can choose a state $\tilde{s}'$, k edges away from A in heuristic space, with k being sufficiently large such that *every* $s \in \lambda^{-1}(A)$ has a sequence of valid steps in G to move the mean to $\tilde{s}'$. We call this sequence of moves as a 'macro-move'. Macro-moves exist in G as well as $\tilde{G}$. Fig 4.4 (center) and Fig 4.4 (right) depict macro-moves in G from s_1 and s_2 respectively, corresponding to the macro-move (A,D) in $\tilde{G}$. If we can find macro-moves from *every* $s \in \lambda^{-1}(A)$ to some $s' \in \lambda^{-1}(D)$, then (A,D) satisfies the conservative property and thus, becomes a conservative macro-move.

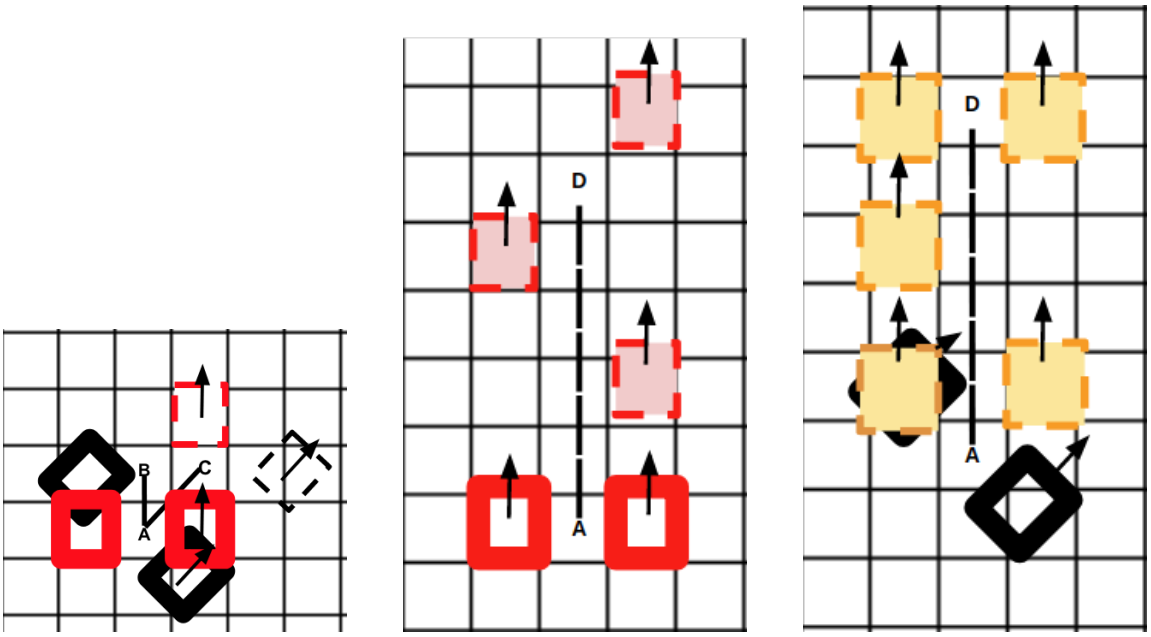


Figure 4.4: (left) Biped states s_1, s_2 (red, black) such that $\lambda(s_1) = \lambda(s_2) = A \in \tilde{S}$. States(pink) forming a "North" macro-move in E for s_1. (right) States(yellow) forming a "North" macro-move in E for s_2. (A,D) is the corresponding macro-move in $\tilde{E}$.

We can generate such macro-moves from A in other directions as well. Here, we select $k = 20$ and add macro-moves in 4 directions (North, East, West, South) in $\tilde{G}$. These macro-moves represent sequence of actions and are analogous to motion primitives and can be computed offline. Once computed for all $s \in \lambda^{-1}(A)$ for any $A \in \tilde{S}$, these can be applied to any state $s \in S$ to compute its relevant successors.

We compute conservative heuristic $h_c(s)$ of state s, by performing a Dijstra's search in $\tilde{G}$ with the macro-moves included. Note that, macro-moves are like any other edge and no edges have been removed from any of the graphs, hence all the guarantees described before still hold.

Results (Simulation):

We used the model of a full-size humanoid for our experiments and allowed a maximum planning time of 90 seconds. We chose a typical indoor environment with narrow passages, corridors and varying doorway sizes. Also, the environment has multiple pathways for traversing between different locations, therefore requiring the search to explore these options. Table 4.2 compares the performance of conservative heuristic (h_c) with the baseline 2D Dijkstra's shortest path heuristic (h_b). Results are averaged over 39 random starts and goals. h_c has a remarkably higher success rate (94.8%) compared to h_b (61.5%). Moreover, using h_c reduces planning times by an order of ≈ 6 and expansions in the final search by a factor of 10. The heuristic computation time is slightly higher for h_c (0.07 seconds) than h_b (0.036 seconds) owing to the addition of macro-moves in $\tilde{E}$. Two examples of the search using h_c and h_b shown in Fig 4.5 (left) and (right) respectively. In the first example (top), wA* with h_b takes a long time but finds a path through the narrow passage. In the second example (bottom), wA* with h_b isn't even able to find the solution in 90s.

Results (Robot experiment):

Fast re-planning is useful when the environment is changing. We implemented a footstep planner for the NAO robot based on [58] and show advantages of using h_c in quickly computing footstep plans in the following scenario:

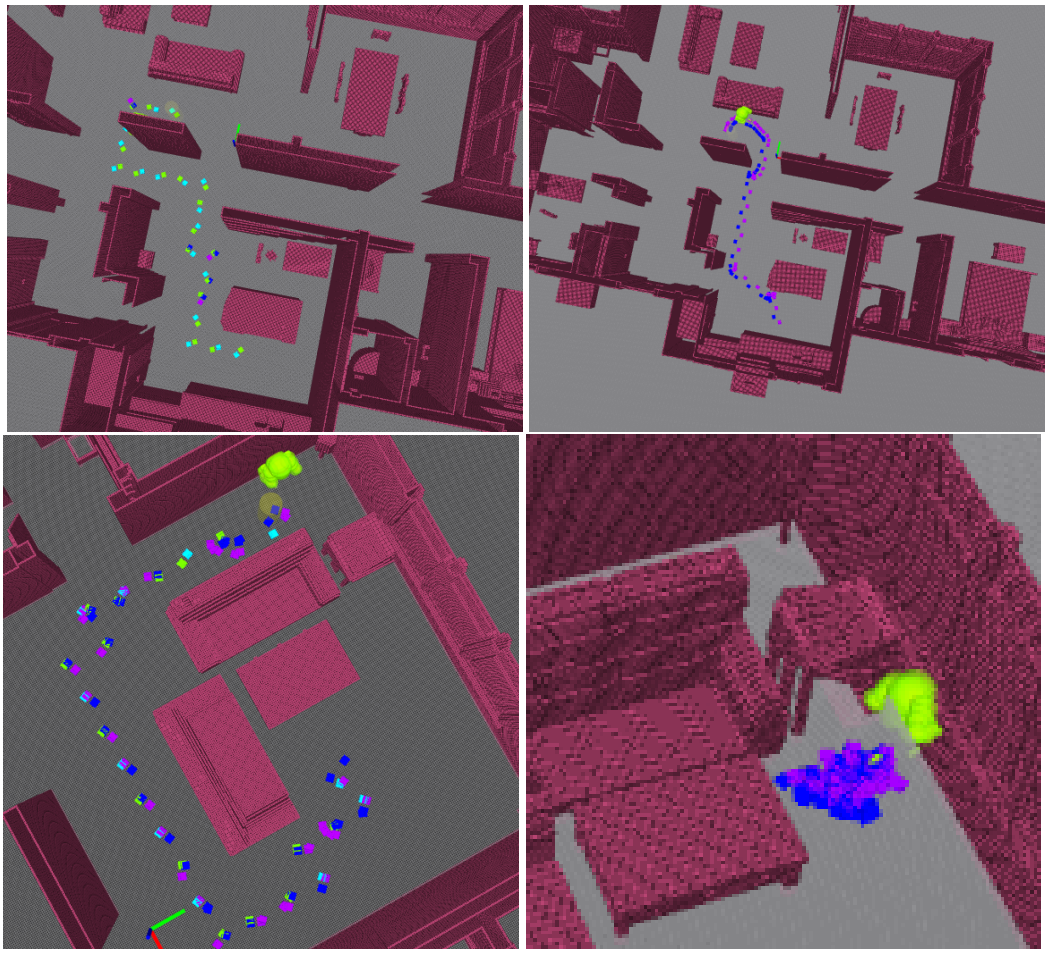


Figure 4.5: Results with h_c (left) and h_b (right) for footstep planning.

- We placed a set of static obstacles in the environment and computed initial footstep plans using h_c and h_b.
- During execution, we added an obstacle in the environment creating a narrow passage on the robot's path, triggering re-planning. wA*(w=8) using h_b keeps expanding states in the local minima created by the passage and fails to find a path in the given time. wA*(w=8) using h_c re-plans in 978 ms (heuristic computation time: 187 ms, planning time: 791 ms). A video demonstration of this experiment can be found here: https://youtu.be/zs4HX84jE1w

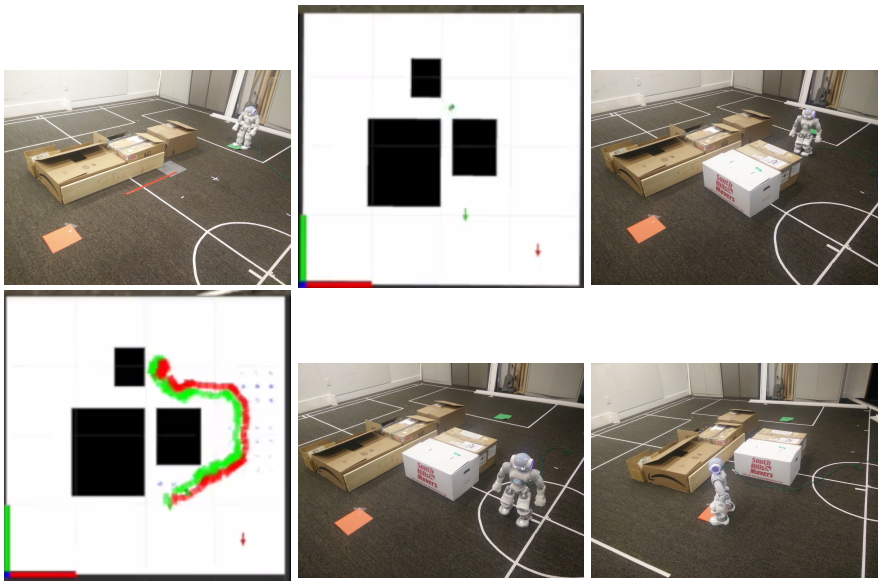


Figure 4.6: Initial environment (top left). Planner stuck in local minima (top center) created by the narrow passage between white and brown box using h_b, NAO waiting for plan (top right). NAO executing plan computed using h_c (bottom).

4.5 Conclusions, Discussion, and Lessons Learned

We proposed a heuristic computation algorithm using conservative edges in the heuristic-space for reducing state expansions by wA*, proved theoretical properties and applied our approach in several motion planning domains, observing significant reductions in expansions and planning times. Future work is to have a probabilistic definition of the conservative property.

Note that the computation of conservative edges is domain-dependent. However, it is this very domain knowledge that helps us to identify conservative edges efficiently. For example, suppose we find the maximum possible area the robot can occupy independent of its orientation, given by its circumcircle, and inflate obstacles by this radius. In that case, any edge lying outside this inflated region is a conservative edge. We realize that the notion of conservative edges is one way to utilize domain knowledge to compute

efficient heuristics. It is essential to devise other ways to see how domain knowledge can be compactly defined and used in an algorithm to develop heuristics and other ways to guide the search that lead to efficient planning.

We also observe that the speed-ups for a given start and a goal depend on the shortest purely conservative path length. Thus, where the conservative edges are present in a given environment plays a significant role in speed-ups. Suppose the conservative edges are positioned such that the shortest purely conservative path is extremely long. In that case, it may be possible to find a path with non-conservative edges with a much lesser number of search expansions than the length of the shortest purely conservative path.

It is possible to have domains where conservative edges do not exist in the original abstract space. In that case, one can potentially add conservative edges artificially by offline computation. Artificial conservative edges may be intuitive to compute for some planning problems, such as the humanoid footstep planning as presented in this chapter. However, it may not be easy to add conservative edges artificially for other planning problems, such as planning with dynamics.

For the number of expansions by weighted A* search to be equal to the solution size, the used weight on the h-value has to be sufficiently high. Given this fact, one might argue that we should use a purely greedy search since weighted A* with very high weight is practically a greedy search. However, we do not use a pure greedy search because weighted A* still allows the user to "control" the greediness by setting the weight. If we want the solution cost to be no more than w times optimal, using the f-value f = g + w·h instead of f = h (greedy) gives us the ability to preserve bounds. This way, we can use a weight w high enough to give significant practical speed-ups. Of course, if the weight w is really large, then the benefits of using weighted A* instead of a purely greedy search are diminished.

Chapter 5

Speeding Up Planning under Uncertainty with Fast-PPCP

5.1 Introduction

In many real-world planning problems, AI agents operate in partially known environments. One approach for the agent to plan in such environments is by taking a deterministic approach, i,e., planning by assuming some instantiation of the variables that represent missing information about the environment (unknown variables), executing few actions of the plan, and replanning in response to sensing. While computationally efficient, this approach may lead to highly suboptimal behavior. In contrast, planning under uncertainty allows an agent to be much more robust with respect to missing information but also becomes computationally dramatically more expensive as it is a special class of planning for Partially Observable Markov Decision Processes (POMDPs) [62, 68].

As noted in [80], real-world planning problems often possess the property of *clear preferences* (CP), wherein one can identify beforehand *clearly preferred* values for unknown

variables in the environment. For example, consider a robot navigating in a partially known environment: it will clearly prefer for any unknown region to be traversable rather than not. Another example is the air traffic management problem, with unknown weather conditions at certain locations: it is clearly preferred to have weather suitable for flying than not.

Likhachev and Stentz in [79] showed that the property of clear preferences, when combined with an assumption of *perfect sensing*—an assumption that there is no noise in sensing and any sensing operation returns the true state of the variable being sensed—can be utilized to compute optimal policies in an efficient and scalable way. Specifically, they introduced Probabilistic Planning with Clear Preferences (PPCP) that scales to large problems by running a series of fast, deterministic, A*-like searches to construct and refine a policy, guaranteeing optimality under certain conditions [80].

However, we often prefer feasible, marginally suboptimal policies over optimal ones if the former can be computed significantly faster. Our key insight is that marginally suboptimal policies can be found much faster if, when running a series of A*-like searches, a plan tries to minimize the number of unknown variables it makes assumptions about to reach its goal. To that end, we introduce FAST-PPCP, a novel planning algorithm that computes a provably bounded-suboptimal policy using much lesser number of searches than that required to compute an optimal policy, for problems that exhibit clear preferences and assume perfect sensing (CP-PS problems). The paper presents theoretical analysis of FAST-PPCP and shows its significant benefits in runtime on several domains. We present theoretical analysis of FAST-PPCP and shows its significant benefits in runtime on several domains (section 5.3.4, section 5.5).

PPCP is a solver that is specifically designed for CP-PS problems. As such it outperforms other optimal solvers such as RTDP-BEL, LAO*, PAO* [36], and HSVI2 when applied to CP-PS problems. FAST-PPCP also focuses on CP-PS problems but runs much

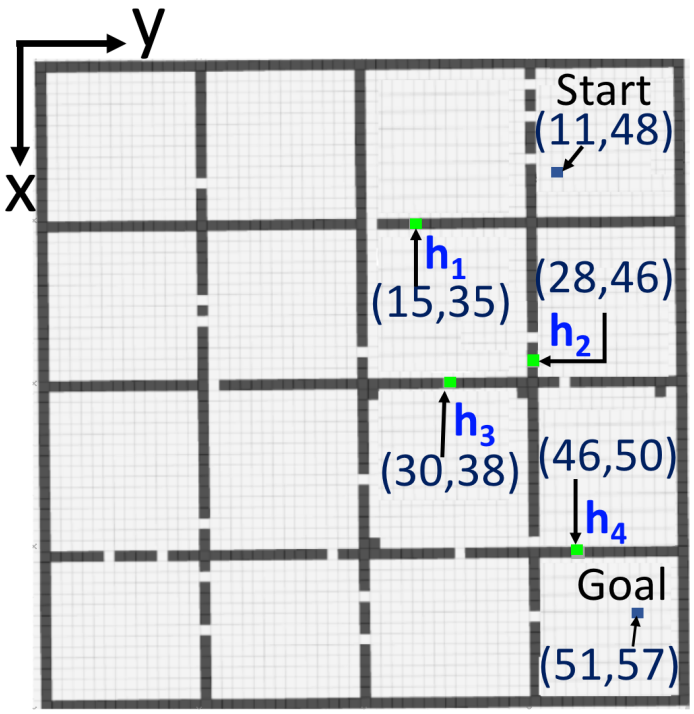


Figure 5.1: Robot navigation between rooms with doors open/closed.

faster than PPCP by trading off optimality of a solution for speed while still guaranteeing bounded suboptimality.

5.2 Problem definition, assumptions and background

Example domain. We use the problem of robot navigation in partially known environments as a running example. Consider in Figure 5.1 a robot that has to navigate from start cell $(11,48)$ to goal cell $(51,57)$ in the environment represented by the 60×60 grid (dark grey cells indicate blocked space). We refer to this example as [**Ex.:**] . The robot can occupy a free grid-cell (light grey) and has eight *move* actions to move in the cardinal and inter-cardinal directions by one cell. The status of some states in the environment is unknown or hidden at the time of planning, which affects the outcomes of some actions

[**Ex.:** The status (free/blocked) of the four cells h_1-h_4 (2D coordinates shown) is unknown] . The robot can also take a stochastic action *sense-and-move*, that senses an adjacent cell and moves the robot to it only if it is free, else stays put. The cost of a *move* action is equal to the Euclidean distance between two adjacent cells (1.4 for diagonal, 1 for others). The cost of a *sense-and-move* action is follows the cost of a *move* action if the hidden cell is free, else is 2.

Problem Definition and Assumptions. PPCP and FAST-PPCP formulate a belief state as a vector of discrete variables split into two components, $X = [S(X); H(X)]$. The *deterministic* component, $S(X)$, is a set of variables whose status is fully observable [**Ex.:** robot's 2D location] . The *hidden* component, $H(X)$, is a set of variables representing the statuses of all unknown/hidden states. We denote the i^{th} unknown variable in $H(X)$ by $h_i(X)$. $h_i(X) = u$ indicates that $h_i(X)$ is unknown. [**Ex.:** Cells h_1-h_4 are each represented by an unknown variable of the same name; $h_i = 0/1/u$ indicates the cell h_i is free/blocked/unknown. $X_{st} = [(11,48); h_1 = u, h_2 = u, h_3 = u, h_4 = u]$ represents the start belief-state X_{st}; the values of all the unknown variables are unknown before planning starts] .

We denote the set of actions applicable at a belief state X by $A(S(X))$. Action $a \in A(S(X))$ is applicable at any belief state Y where $S(Y) = S(X)$. However, the outcome of an action a depends on an unknown variable, and we denote the unknown variable in $H(X)$ affecting its outcome by $h^{S(X),a}$. An action is *deterministic* [**Ex.:** move actions] if its outcome is unaffected by any unknown variable ($h^{S(X),a} = \texttt{NULL}$), and so it has only one outcome because the underlying environment is deterministic. An action is *stochastic* [**Ex.:** sense-and-move actions] if it can have multiple possible outcomes depending on the status of an unknown variable. We assume that the assumptions A1-5 in Table 5.1 hold; FAST-PPCP can be applied to any domain where they hold.

A1	The environment is deterministic, i.e., if the environment were fully known at the time of planning, there would be no uncertainty in the outcome of an action.
A2	The agent has a probability distribution or *belief* over the status of these cells.
A3	The true status of a hidden variable becomes known immediately (perfect sensing assumption.
A4	Only one hidden variable can affect the outcome of an action a taken at $S(X)$. However, the same hidden variable is allowed to affect outcome of another action taken at another state.
A5	The variables in H are independent of each other and therefore $P(H) = \prod_{i=1}^{\lvert H \rvert} P(h_i)$.

Table 5.1: Assumptions used in FAST-PPCP.

We denote the set of possible outcomes of action a taken at a belief-state X' by $succ(X',a)$ in the belief-space and by $succ(S(X'),a)$ in the underlying deterministic space [**Ex.** 2D grid]. We refer to X' as the predecessor of X. Note that $H(X) = H(X')$ for a deterministic action, whereas for a stochastic action, $H(X)$ is the same as $H(X')$ except for $h^{S(X),a}$, which becomes known if it was unknown. The probability distribution of transitions $P(X|X',a)$ is the same as that of the unknown variable $h^{S(X'),a}$. Given assumption A5, X concisely represents a probability distribution over all possible states. [**Ex.** h_1 in Figure 5.1 represents the status of the unknown cell $(15,35)$ and affects the outcome of the action *sense-and-move* taken on the adjacent cell $(14,36)$. Let $X' = [(14,36);u,u,u,u]$ be a belief state. When taken on X', *sense-and-move* produces two belief-state outcomes: $X_1 = [(15,35);h_1 = 0,u,u,u]$ and $X_2 = [(14,36);h_1 = 1,u,u,u]$ both with a probability of 0.5]. If X^b, X_{np} are the preferred and non-preferred outcome of (X',a), then the cost $C(X',a,X^b)$ of the edge (X',a,X^b) in the belief-space $= C(S(X'),a,S(X^b))$, the cost of the corresponding edge in the deterministic space, and $C(X',a,X_{np}) = C(S(X'),a,S(X_{np}))$.

Clear preferences: We assume that every hidden variable's best (clearly preferred) value is known beforehand, meaning that we prefer a hidden cell to be free ($h^{S(X),a}$) rather than blocked ($h^{S(X),a}$). We define clear preferences [80] as: Let $V^*(X')$ denote the expected cost of executing an optimal policy (policy that minimizes expected cost to goal) from state X'. For any given state X' and stochastic action a such that $h^{S(X'),a}$ is unknown, there exists

Term	Definition
Full Policy from belief state X	Tree rooted at X s.t every branch reaches a belief state X_g s.t $S(X_g) = S_g$ for a given goal S_g.
Unexplored belief state	Non-preferred outcome on partial policy from which a path has not been searched yet
Policy-devoid belief state	Non-preferred outcome on partial policy without an action defined from it (Unexplored belief states and belief states from which policy growth failed)
Partial Policy from X	Policy rooted at belief state X that has at least one belief state without a defined action.

Table 5.2: Definitions used in the explanation of FAST-PPCP.

a successor state X^b such that $h^{S(X),a}(X^b) = b$ (we denote the best value using the variable b) [**Ex.:** b = 0] and $X^b = \arg\min_{X \in succ(X',a)} \{c(S(X'), a, S(X) + V^*(X))\}$.

The planning problem is to compute a policy (defined in Table 5.2) from X_{st}.

PPCP Overview and Motivation for FAST-PPCP. The overall approach of PPCP is to compute an optimal policy in the belief-space by running a series of A* [45]-like searches in the underlying deterministic environment instead of the exponentially larger belief space. This approach turns out to be orders of magnitude faster than solving the full problem at once since the memory requirements are much lower. PPCP iteratively constructs and refines a partial policy (defined in Table 5.2) from X_{st}, while updating V-values of the states reachable by following the partial policy.

We make an observation that each non-preferred outcome on the partial policy leads to an additional PPCP iteration needed to define a policy from it. Also, whenever the V-value of a non-preferred outcome is updated, its predecessor on the policy gets a negative Bellman error. This gets accumulated up along the policy till the outcome of a stochastic action (or X_{st}, whichever comes first) is encountered, at which point PPCP starts another iteration from this outcome (or X_{st}). To conclude, the number of iterations increases with

an increase in the number of stochastic actions in each branch of the partial policy.

Since PPCP continues to iterate until every outcome in the policy has an action defined and has no Bellman error, the number of PPCP iterations can be really high for environments with a large number of unknown variables, especially if stochastic actions lie lower (closer to a leaf node) on the policy. [**Ex.:** PPCP requires 159 iterations and 1.3 seconds to find the optimal policy in this environment that has only 4 unknown variables] . However, there exists a policy in [**Ex.:**] with expected cost of 93.8 that is only 1.04 times higher than that of the optimal policy (90.1). FAST-PPCP computes this policy with 3 search iterations in just 34 milliseconds (ms), which is 40 times faster than PPCP.

This is because a FAST-PPCP search operates very differently compared to PPCP search, in order to meet the following **search objective**: to compute a path that *explicitly minimizes the number of stochastic transitions* in it, while ensuring that including this path in the partial policy can result in a provably bounded suboptimal full policy π_f from X_{st}, when construction of the policy is completed. Also, it has a novel policy growth strategy such that the first time the search terminates, the policy is guaranteed to be bounded suboptimal which leads to significant speedup. It also has a novel strategy for scheduling the searches to ensure completeness.

5.3 Fast bounded suboptimal PPCP

We give an overview of the working of FAST-PPCP in the following subsection.

5.3.1 Operation and Intuition in a Nutshell

In **Figure** 5.2, we present a block diagram of the FAST-PPCP algorithm showing how its algorithmic components/blocks interact. The complete pseudocode is given in Appendix

B in chapter A. The notations introduced in this subsection appear in the pseudocode in chapter A.

FAST-PPCP iteratively develops partial policies into a full policy π_f rooted at X_{st} (definitions of partial and full policy are in Table 5.2). It aims to find a π_f that satisfies the following: $V^{\pi_f}(X_{st}) \leq \alpha V^*(X_{st})$, where $V^{\pi_f}(X_{st})$ is the V-value (expected cost-to-goal) of following the full policy π_f from X_{st}, $\alpha > 1$ is a user-defined constant, and $\alpha V^*(X_{st}) \triangleq \mathtt{B}^*$ is the desired suboptimality bound. We refer to such a full policy as a $\mathtt{B}^*$-bounded policy. However, $V^*(X_{st})$ is not known. Hence, before FAST-PPCP begins, it computes $V_L^*(X_{st})$, a lower bound on $V^*(X_{st})$ (**Block 1.**) which we describe at the end of this section. FAST-PPCP ensures that the full policy it computes satisfies $V^{\pi_f}(X_{st}) \leq \alpha V_L^*(X_{st}) \triangleq \mathtt{B}^{\mathtt{L}}$. We refer to such a policy as a $\mathtt{B}^{\mathtt{L}}$-bounded policy.

In an iteration i, FAST-PPCP either performs (1) **Policy Growth**: adds a new branch to the current partial policy $\hat{\pi}_i$, or (2) **Policy Correction**: replaces existing branches in $\hat{\pi}_i$. FAST-PPCP terminates as soon as the partial policy it maintains becomes a full policy. If the combination of Policy Growth and Policy Correction fails to find π_f, FAST-PPCP performs (3) **Improvement of** $\mathtt{B}^{\mathtt{L}}$. Throughout its operation, for every belief state X in $\hat{\pi}_i$, FAST-PPCP maintains and updates V-value estimates $\hat{V}(X)$. For every policy-devoid belief state X in $\hat{\pi}_i$ specifically, it maintains $V_u^*(X)$, an underestimate of $V^*(X)$, which is initially set to a known underestimate before FAST-PPCP begins. For any partial policy $\hat{\pi}$ from a belief state X, we define its *expected cost* $\hat{V}^{\hat{\pi}}(X)$, which is the same as the expected cost of a full policy except that the $\hat{V}$-value of each policy-devoid belief state X in $\hat{\pi}$ is set to its corresponding underestimate $V_u^*(X)$. For a partial policy $\hat{\pi}_i$ from X_{st}, FAST-PPCP maintains $\hat{V}^{\hat{\pi}_i}(X_{st})$.

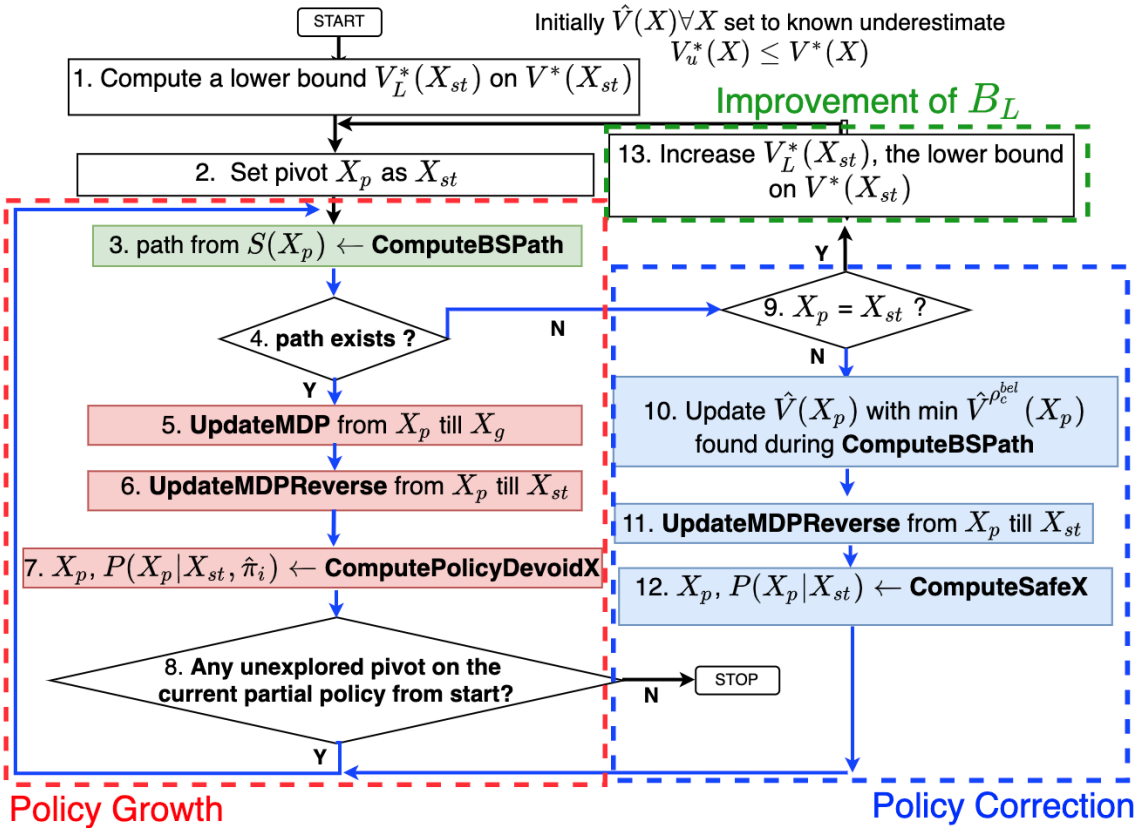


Figure 5.2: FAST-PPCP block diagram of Main function.

Policy Growth

In Policy Growth mode, FAST-PPCP first picks any belief state X_p on $\hat{\pi}_i$ without an action defined from it. Initially, $\hat{\pi}_i$ is empty and $X_p = X_{st}$ (**Block 2.**). FAST-PPCP attempts to *grow* $\hat{\pi}_i$ from X_p. Specifically, it searches for a path ρ_{bs} from $S(X_p)$ in the underlying deterministic space instead of the exponentially larger belief space. This deterministic space is constructed assuming every unknown variable in X_p is set to its preferred value, and every observed/known variable in X_p—that was initially unknown when planning began—is set to its observed value. [**Ex.:** It searches in the 2D grid assuming every unknown cell in X_p is set to free, and every hidden cell whose status is observed in $H(X_p)$ is set to the observed value (free/blocked)] . The path ρ_{bs} maps to a corresponding path in the belief

space, ρ_{bs}^{bel}, that consists of transitions that only correspond to either deterministic actions, or stochastic actions with clearly preferred outcomes. We refer to this as the *primary* branch corresponding to ρ_{bs} from X_p to goal. If ρ_{bs} is found, FAST-PPCP then updates $\hat{\pi}_i$ by adding ρ_{bs}^{bel} to $\hat{\pi}_i$. Next, we explain the objective and constraint used by COMPUTEBSPATH, the FAST-PPCP search to find ρ_{bs}.

COMPUTEBSPATH (**Block 3.**): In order to minimize the number of stochastic transitions as its search objective, FAST-PPCP ensures that the search computes the path with the *least number of stochastic transitions* first as a candidate, ρ_c, for ρ_{bs}. Adding the primary branch ρ_c^{bel} to $\hat{\pi}_i$ will result in the updated partial policy $\hat{\pi}_i^{\mathsf{c}}$. To meet the bounded suboptimality aim, FAST-PPCP additionally checks if $\hat{\pi}_i^{\mathsf{c}}$ is NOT invalid—one that cannot be developed into an $\mathtt{B}^{\mathsf{L}}$-bounded policy no matter which unexplored belief state on it is explored in the future. To do this, FAST-PPCP computes $\hat{V}^{\rho_c^{\mathsf{bel}}}(X_p)$, the expected cost of the partial policy that corresponds to the primary branch ρ_c^{bel}.

$\hat{V}^{\rho_c^{\mathsf{bel}}}(X_p)$ is a lower bound on the V-value of any policy containing ρ_c^{bel} that can be developed in future iterations. This is because the $\hat{V}$-value of any policy-devoid belief state X is maintained to be $V_u^*(X)$, and we show that any non-preferred outcome of a stochastic transition in ρ_c^{bel} can only be policy-devoid. FAST-PPCP then computes $\hat{V}^{\hat{\pi}_i^{\mathsf{c}}}(X_{st})$, the expected cost of $\hat{\pi}_i^{\mathsf{c}}$, which consequently is also a lower bound on the V-value of any policy containing $\hat{\pi}_i^{\mathsf{c}}$. If $\hat{V}^{\hat{\pi}_i^{\mathsf{c}}}(X_{st}) > \mathtt{B}^{\mathsf{L}}$ (rejection condition), no full $\mathtt{B}^{\mathsf{L}}$-bounded policy π_f containing $\hat{\pi}_i^{\mathsf{c}}$ can be developed thereafter. Candidate ρ_c is then rejected: ρ_c^{bel} is not added to $\hat{\pi}_i$. In this way, FAST-PPCP sequentially computes candidate paths from $S(X_p)$ *in increasing order of the number of stochastic transitions on them*, and rejects them until a candidate path ρ_c is found, such that $\hat{V}^{\hat{\pi}_i^{\mathsf{c}}}(X_{st}) \leq \mathtt{B}^{\mathsf{L}}$ (formally stated later in Theorem 1). We refer to ρ_c that meets this constraint as ρ_{bs} and the corresponding $\hat{V}^{\hat{\pi}_i^{\mathsf{c}}}(X_{st})$ as $\hat{V}^{\hat{\pi}_i^{\mathsf{bs}}}(X_{st})$.

UPDATEMDP **(Block 5.):** If ρ_{bs} is found (**Block 4. Outcome Y**), ρ_{bs}^{bel} is added to $\hat{\pi}_i$

from X_p to get the updated partial policy $\hat{\pi}_i^{\mathsf{bs}}$, and $\hat{V}(X_p)$ is updated to be the lower bound $\hat{V}^{\rho_{bs}^{\mathsf{bel}}}(X_p)$ computed during the search. FAST-PPCP also updates the $\hat{V}$-values of belief states on ρ_{bs}^{bel}.

UPDATEMDPREVERSE **(Block 6.):** Starting from X_p, it backs up $\hat{V}$-values of every belief state on the branch in $\hat{\pi}_i$ from X_p up till X_{st} to remove their respective Bellman errors. As a result, $\hat{V}$ of (X_{st}) gets updated to $\hat{V}^{\hat{\pi}_i^{\mathsf{bs}}}(X_{st})$.

COMPUTEPOLICYDEVOIDX **(Block 7.):** FAST-PPCP then finds a policy-devoid non-preferred outcome in $\hat{\pi}_i^{\mathsf{bs}}$. It starts the next iteration of Policy Growth from this outcome.

When the final primary branch gets added to the maintained partial policy and it has no more policy-devoid outcomes, a full policy π_f has been found and FAST-PPCP terminates (**Block 8. evaluation fails**). Since π_f is no longer a partial policy, $\hat{V}^{\pi_f}(X_{st})$ in this iteration is no longer a lower bound but is the actual V-value of π_f. Since $\hat{V}^{\pi_f}(X_{st}) \leq \mathtt{B}^{\mathsf{L}}$ has been ensured in Policy Growth, π_f is $\mathtt{B}^{\mathsf{L}}$-bounded, and also $\mathtt{B}^*$-bounded (formally stated later in Theorem 3).

Policy Correction

In Policy Growth, it is possible that *every* candidate path ρ_c from $S(X_p)$ is rejected by COMPUTEBSPATH (**Block 4. evaluation fails**) and X_p is not X_{st} (**Block 9. evaluation fails**) because adding any of them to $\hat{\pi}_i$ results in an invalid updated partial policy $\hat{\pi}_i^{\mathsf{c}}$ with $\hat{V}^{\hat{\pi}_i^{\mathsf{c}}}(X_{st}) > \mathtt{B}^{\mathsf{L}}$. While computing a ρ_c, FAST-PPCP maintains $\min \hat{V}^{\rho_c^{bel}}(X_p)$, the minimum value of $\hat{V}^{\rho_c^{bel}}(X_p)$ over all ρ_c computed so far. In this case, since every possible path has been visited, $\min \hat{V}^{\rho_c^{bel}}(X_p)$ is a lower bound on $V^*(X_p)$. FAST-PPCP updates $V_u^*(X_p)$ to $\min \hat{V}^{\rho_c^{bel}}(X_p)$ (**Block 10.**). Even if no branch is added to $\hat{\pi}_i$ from X_p, backups are still done as described from X_p up until X_{st} (**Block 11.**) to get the updated $\hat{V}^{\hat{\pi}_i}(X_{st})$ which now exceeds $\mathtt{B}^{\mathsf{L}}$. FAST-PPCP then *corrects* $\hat{\pi}_i$ by removing a *safe* primary branch in it and trying

to replace it with one that results in an updated partial policy $\hat{\pi}_i^{\mathsf{cs}}$ with $\hat{V}^{\hat{\pi}_i^{\mathsf{cs}}}(X_{st}) \leq \mathtt{B}^{\mathsf{L}}$. A primary branch is *safe* if removing it does not discard any $\mathtt{B}^{\mathsf{L}}$-bounded policy if one exists. This guarantees that FAST-PPCP finds a $\mathtt{B}^{\mathsf{L}}$-bounded policy if one exists, or, is B^{L}-*complete* (formally stated later in Theorem 2).

COMPUTESAFEX **(Block 12.):** To find a *safe* primary branch, FAST-PPCP looks for a non-preferred outcome X_{safe} in $\hat{\pi}_i$ with a primary branch from it such that $\hat{V}(X_{\mathsf{safe}})$ is a lower bound on V-value of any policy containing this primary branch. FAST-PPCP can guarantee this is the case only if: (1) this primary branch has deterministic transitions only, or (2) every non-preferred outcome X_{np} of a stochastic transition on this primary branch is policy-devoid, and therefore has $\hat{V}(X_{np})$ set to an underestimate of $V^*(X_{np})$. In this case, keeping this primary branch in $\hat{\pi}_i$ and performing Policy Growth via any such X_{np} to obtain an updated partial policy $\hat{\pi}^{+\times}$ will surely have $\hat{V}^{\hat{\pi}^{+\times}}(X_{st}) \geq \hat{V}^{\hat{\pi}_i}(X_{st}) > \mathtt{B}^{\mathsf{L}}$ (i.e., no $\mathtt{B}^{\mathsf{L}}$-bounded policy can develop from $\rho^{\mathsf{bel}}_{\mathsf{safe}}$ with the rest of $\hat{\pi}_i$ unchanged). Hence, this primary branch can be removed without missing a $\mathtt{B}^{\mathsf{L}}$ policy. FAST-PPCP resumes Policy Growth from X_{safe} in the next iteration. If Policy Growth fails again, it continues Policy Correction to further change $\hat{\pi}_i$ by removing another safe branch.

Computation/Improvement of $\mathtt{B}^{\mathsf{L}}$

Before it begins, FAST-PPCP computes $V_L^*(X_{st})$, a lower bound on $V^*(X_{st})$, by running one iteration of PPCP (**Block 1.**). For a very small α or $V_L^*(X_{st})$, it may be the case that *no* $\mathtt{B}^{\mathsf{L}}$-bounded policy from X_{st} exists. This is the case when every candidate path is rejected by COMPUTEBSPATH (**Block 4. evaluation fails**) when attempting to grow the policy from X_{st} (**Block 9. evaluation succeeds**). However, there may exist full policies that are not $\mathtt{B}^{\mathsf{L}}$-bounded but are $\mathtt{B}^*$-bounded, i.e., $\mathtt{B}^{\mathsf{L}} < V^{\pi}(X_{st}) \leq \mathtt{B}^*$ where $V^{\pi}(X_{st})$ is the V-value of the full policy π. To ensure completeness and find these policies, FAST-PPCP performs

Improvement of $\mathtt{B}^{\mathtt{L}}$: It increases $\mathtt{B}^{\mathtt{L}}$ by running PPCP iterations till $V_L^*(X_{st})$ increases (**Block 13.**), and restarts afresh from X_{st} (**Block 2.**) using this increased (looser) value for $\mathtt{B}^{\mathtt{L}}$. Increase in $V_L^*(X_{st})$ is guaranteed in [80].

5.3.2 Implementation details of some functions

We refer to X_p, the belief state from which FAST-PPCP starts an iteration, as a *pivot*. The complete pseudocode is given in Appendix B in chapter A.

COMPUTEBSPATH implementation

We now describe procedure COMPUTEBSPATH shown in Pseudocode 2. At iteration i starting from pivot X_p, FAST-PPCP aims to find a path from $S(X_p)$ in a deterministic graph $\mathcal{G}^i = \{S^i, E^i\}$ where S^i, E^i are the states and edges representing feasible transitions in the deterministic space. In order to ensure that the primary branch computed from X_p is feasible, the status of the hidden variables observed (known) according to $H(X_p)$ is used while determining S^i, E^i: state s is the clearly preferred/non-preferred outcome in $\mathcal{G}^i$ of action a executed at s' if $h^{s',a}$ is known to have clearly preferred/non-preferred value in $H(X_p)$. The unknown variables in $H(X_p)$ are assumed to have their clearly preferred value while determining S^i, E^i: state s is the clearly preferred outcome in $\mathcal{G}^i$ of action a executed at s' if $h^{s',a}$ is unknown in $H(X_p)$.

However, in order to meet the search objective and constraint explained in **Policy Growth**—which is to compute a path ρ_{bs} from $S(X_p)$ in $\mathcal{G}^i$ that minimizes the total number of stochastic transitions on it such that $\hat{V}^{\hat{\pi}_i^{bs}}(X_{st}) \leq \mathtt{B}^{\mathtt{L}}$—COMPUTEBSPATH is a backward A^*-like search run on an augmented deterministic graph $\mathcal{G}^i_{sto} = \{S^i_{sto}, E^i\}$. The search is backward in order to allow the computation of $\hat{V}^{\hat{\pi}_i^{bs}}(X_{st})$ by back-propagating values from the goal that has its $\hat{V}$-value $= 0$. A search state $n \in S^i_{sto}$ is defined as $n = [s, \hat{V}^{\rho}(s)]$, where

s is a state in S^i, ρ is a path from s to goal S_g in $\mathcal{G}^i$. E^i is the same set of edges as in $\mathcal{G}^i$. We first describe the edge-costs in $\mathcal{G}^i_{sto}$, then explain $\hat{V}^{\rho}(s)$.

Edge-costs in $\mathcal{G}^i_{sto}$: We derive the edge-costs with the following aims: (1) In $\mathcal{G}^i_{sto}$,

Pseudocode 2

```
1: procedure ComputeBSPath(X_p, α, V*_L(X_st), P(X_p|X_st, π̂_i))
2:     n_g = [S_g, 0]; besta(n_g) = NULL;
3:     g_sto(n_g) = 0;
4:     V*_u(X_p) = ∞; OPEN = ∅
5:     Insert n_g in Δ_dom(S_g) [1]
6:     Insert S_g in OPEN with priority g_sto(S_g) + h_c(S(X_p), S_g);
7:     while 1 do
8:         if OPEN is empty then                                   ▷ Policy Growth failed
9:             V̂(X_p) ← V*_u(X_p) updated during this search, π̂_i(X_p) = NULL
10:            and return FALSE
11:        end if
12:        Remove search-state n with the smallest g_sto(n) + h_c(S(X_p), s(n))
13:        in OPEN
14:        if s(n) = S(X_p) then
15:            V̂^{ρ^bel_c}(X_p) ← V̂^{ρ_c}(S(X_p)), V̂ component of n =
16:            [S(X_p), V̂^{ρ_c}(S(X_p))]
17:            Update V*_u(X_p) ← V̂^{ρ^bel_c}(X_p)) if V̂^{ρ^bel_c}(X_p)) < V*_u(X_p)
18:            Update V̂^{π̂^c_i}(X_st) by replacing old V̂(X_p) with V̂^{ρ^bel_c}(X_p), as
19:            in Eq. 5.8
20:            if V̂^{π̂^c_i}(X_st) ≤ αV*_L(X_st) then              ▷ Policy growth is successful
21:                V̂^{π̂^bs_i}(X_st) ← V̂^{π̂^c_i}(X_st)
22:                V̂(X_p) ← V̂^{ρ^bel_c}(X_p)
23:                return TRUE
24:            end if
25:        end if
26:        s ← s(n)
27:        for each action a∈A(s') and predecessor s' s.t s= S(succ(X', a)^b)   where X' = [s', H(X_p)]
28:            compute V̂^{ρ'}(s') using Eq. 5.1 with X' = [s', H(X_p)]
29:            search-predecessor n' = [s', V̂^{ρ'}(s')]
30:            if a is a stochastic action then
31:                g_sto(n') = g_sto(n) + |E|.c_max;
32:            else
33:                g_sto(n') = g_sto(n) + c_max;
34:            end if
35:            if n' not visited before or¬IsDominated(n') then
36:                Insert n' in Δ_dom(s')
37:                Insert n' in OPEN with priority g_sto(n') + h_c(S(X_p), s')
38:                and besta as a;
39:            end if
40:        end for
41:    end while
42: end procedure
43: procedure IsDominated(n')
44:    return (g_sto(n) ≥ g_sto(n') and V̂(n) ≥ V̂(n')) for any
45:    n ∈ Δ_dom(s(n'))                                            ▷ returns true if n' is dominated by an n ∈ list of
46:    Δ_dom(s(n')) undominated search-states of s(n')
47: end procedure
```

[1] $\hat{V}(X_p)$ initialized to $V^*_u(X_p) \leq V^*(X_p)$

ordering paths according to their modified cost is the same as ordering them according to their number of stochastic transitions. To ensure the path with minimum number of stochastic transitions is the least cost path, we need to make sure that the largest purely deterministic path (consisting of deterministic transitions only) is strictly less than the smallest path with a single stochastic transition (which is a one-edge path with only one stochastic edge). To satisfy this, we set the costs of deterministic and stochastic transitions as β and $|E|\beta$ respectively, where $|E|$ is the total number of edges in $\mathscr{G}^i$. (2) Assume that a non-zero consistent heuristic function $h_c(S(X_p),s)$ is known for the original cost function in $\mathscr{G}^i$ [**Ex.:** Euclidean distance is consistent heuristic given edge-costs are Euclidean distance] . We want to use the same heuristic in COMPUTEBSPATH. To do this, we need to ensure that $c_{sto}(n,a,n') \geq c(s,a,s')$ for all edges (s,a,s') in G^i. Setting $\min c_{sto}(n,a,n') = c_{max} = \max_{\forall(s,a,s')\in G^i}(c(s,a,s'))$ ensures this. Accordingly,

$$c_{sto}(n,a,n') = \begin{cases} |E|c_{max} & a \text{ is stochastic} \\ c_{max} & a \text{ is deterministic} \end{cases}$$

However, the cost c_{sto} of a path ρ_c from $S(X_p)$ to goal in $\mathscr{G}^i$ has no notion of the V-value of a policy from $S(X_p)$ containing ρ_c^{bel}. Maintaining the second component $\hat{V}^{\rho}(s)$ in state n serves this purpose.

$\hat{V}^{\rho}(s)$ **computation:** $\hat{V}^{\rho}(s)$ is computed as follows: Let ρ' be a path in $\mathscr{G}^i$ from s' to goal. Let s be the outcome of (s',a) for an action a in ρ'. Let ρ'^{bel} be the corresponding primary branch of ρ' from a belief state $X' = [s',H(X)]$. Let $X^b = succ(X',a)^b$ be the preferred outcome of belief state-action (X',a). Since ρ' maps to ρ'^{bel}, which is a primary branch, $s = S(X^b)$. Let ρ be the subpath from s to S_g and ρ^{bel} be its corresponding primary branch from X_b. FAST-PPCP recursively defines $\hat{V}^{\rho'}(s')$ which can be computed by backing

up from S_g with $\hat{V}(S_g) = 0$ as follows:

$$\hat{V}^{\rho\prime}(s') = \sum_{X_{np} \in succ(X',a) \setminus X^b} P(X_{np}|X',a)(c(s',a,S(X_{np})) + \hat{V}(X_{np})) + P(X^b|X',a)(c(s',a,s) + \hat{V}^{\rho}(s)) \quad (5.1)$$

$\hat{V}^{\rho}(s)$ is the same as $\hat{V}^{\rho^{\mathsf{bel}}}(X)$ which is lower bound on the V-value of any policy from X containing ρ^{bel}.

Predecessor generation: Note that COMPUTEBSPATH while searching does not keep track of $H(\cdot)$ part of the belief states: while computing the predecessor of s which corresponds to computing predecessor of X_b, FAST-PPCP has the predecessor X' *defined as* $X' = [s', H(X_p)]$ (line 29). This implements the assumption that the hidden variable $h^{s',a}$ sensed by executing a on s', if it was unknown in $H(X_p)$, remains unknown in the primary branch from X_p to X' and is sensed for the first time when X_b is generated as outcome. More generally, it can only find a primary branch from X_p that satisfies the condition **C1**: The primary branch does not have two belief states X_1 and X_2 with actions a_1 and a_2 respectively that sense the same hidden variable in $H(X_p)$, i.e, $h^{S(X_1),a_1} \neq h^{S(X_2),a_2}$ (Formally stated in Lemma 9). Since E^i is the same set of edges as in $\mathscr{G}^i$, while generating a predecessor n' from n, COMPUTEBSPATH assumes that each action a has only one outcome if , i.e., $s(n)$ is an outcome of action a executed at $s(n')$ (which is s') if and only if $s = S(succ(X',a)^b)$.

With search states and edge-costs as described, COMPUTEBSPATH searches backwards from $[S_g, 0]$, expanding states from the *OPEN* list in non-decreasing order of $f(n) = g_{sto}(n) + h_c(n, S(X_p))$ where h_c is a consistent heuristic (Line 13).

$\hat{V}^{\hat{\pi}_i^c}(X_{st})$ **computation:** The first time a search state $s = [S(X_p), \hat{V}^{\rho_c}(S(X_p))]$ is expanded, COMPUTEBSPATH has found a path ρ_c from $S(X_p)$ with the minimum number of stochastic transitions in $\mathscr{G}^i$, and the lower bound $\hat{V}^{\rho_c^{\mathsf{bel}}}(X_p) (= \hat{V}^{\rho_c}(S(X_p)))$ of its corresponding primary branch ρ_c^{bel} from X_p. COMPUTEBSPATH then computes $\hat{V}^{\hat{\pi}_i^c(X_{st})}$, a

lower bound on any policy that can develop from the updated partial policy $\hat{\pi}_i^c$, if ρ_c^{bel} is added to $\hat{\pi}_i$. To compute this, it replaces the old $\hat{V}(X_p)$ with the current estimate $\hat{V}^{\rho_c^{\mathsf{bel}}}(X_p)$ to get $\hat{V}^{\hat{\pi}_i^c}(X_{st})$ as (Line 19):

$$\hat{V}^{\hat{\pi}_i^c}(X_{st}) \leftarrow \hat{V}^{\hat{\pi}_i}(X_{st}) + P(X_p|X_{st},\hat{\pi}_i)(-\hat{V}(X_p) + \hat{V}^{\rho_c^{bel}}(X_p)) \tag{5.2}$$

where $P(X_p|X_{st},\hat{\pi}_i)$ is the probability of reaching the pivot X_p following the current partial policy $\hat{\pi}_i$ from X_{st}. It then checks if $\hat{V}^{\hat{\pi}_i^c}(X_{st}) \leq \alpha V_L^*(X_{st})$. If this is false, then ρ_c^{bel} gets rejected and COMPUTEBSPATH continues expansions from OPEN.

COMPUTEBSPATH termination: COMPUTEBSPATH terminates in either of the following cases: (1) when for the first time $n = [S(X_p), \hat{V}^{\rho_c^{\mathsf{bel}}}(X_p)]$ gets expanded such that $\hat{V}^{\hat{\pi}_i^c}(X_{st}) \leq \alpha V_L^*(X_{st})$ (Line 20). Upon termination it returns a path that meets the search objective given the constraint (Theorem 1). Or, (2) when OPEN list is empty (Line 8) indicating that policy growth failed from X_p. In this case, line 9. implements updating $\hat{V}(X_p)$ with the minimum value of $\hat{V}^{\rho_c^{bel}}(X_p)$ over all ρ_c in $\mathscr{G}^i$.

COMPUTEBSPATH performs dominance checks (Line 35) through procedure ISDOMINATED (Pseudocode 2 Line 43). If a search-state n' is dominated by any previously seen search state n with $s(n) = s(n')$ (Line 44-46), the sub-graph that can be generated from n' is pruned. This significantly increases search efficiency.

COMPUTESAFEX implementation

As mentioned in section 5.3.1, FAST-PPCP looks for a non-preferred outcome X_{safe} in $\hat{\pi}_i$ with a primary branch from it such that: (1) this primary branch has deterministic transitions only, or (2) every non-preferred outcome of a stochastic transition on this primary branch is policy-devoid. To find X_{safe}, COMPUTESAFEX first considers any arbitrary non-preferred outcome X_e on $\hat{\pi}_i$ that has an existing primary branch from it. Starting from X_e,

COMPUTESAFEX traverses down the primary branch from X_e. It sequentially visits each belief state X in the primary branch and checks if the action $\hat{\pi}_i(X)$ at X—given by the current partial policy $\hat{\pi}_i$—is deterministic, or every non-preferred outcome of taking $\hat{\pi}_i(X)$ at X is policy-devoid. *If this is the case*, then it moves down (towards the goal belief state X_g) on this primary branch to the preferred outcome of taking $\hat{\pi}_i(X)$ at X (there is only one outcome when $\hat{\pi}_i(X)$ is deterministic). *If not*, then it resets X_e as the non-preferred outcome of $\hat{\pi}_i(X)$ that is not policy-devoid and has an existing primary branch. It then restarts the traversal down this primary branch.

5.3.3 Working example

In Figure 5.3 we present a working example of FAST-PPCP in **[Ex.]** with $\alpha = 1.485$. Before it begins, FAST-PPCP computes $V_L^*(X_{st}) = 63.4$ by running one PPCP iteration. $\alpha V_L^*(X_{st}) \triangleq \mathtt{B}^{\mathtt{L}} = 94.1$.

In its first iteration (i = 0), it starts Policy Growth with $X_{st} = [(11,48), u, u, u, u]$ as the pivot. It attempts to find a path from start state $S(X_{st}) = (11,48)$ in the 2D grid $\mathscr{G}_0$ (5.3(a)) constructed assuming all the four hidden cells are set to their clearly preferred value, denoted by 0, since all of them are unknown in $H(X_{st})$. COMPUTEBSPATH first finds a deterministic path with zero stochastic transitions as the first candidate (when it expands from *OPEN* a search-state n with $s(n) = (11,48)$ for the first time). However, it gets rejected because its cost exceeds $\mathtt{B}^{\mathtt{L}}$. The second candidate path (5.3(a)) with one stochastic transition corresponding to a *sense-and-move* action at $(29,38)$—that senses the adjacent hidden cell h_3 $(30,38)$—is found when $n = [(11,48), \hat{V}^{\rho_{bs}}(S(X_{st})) = 71.6]$ gets expanded. $\hat{V}^{\rho_{bs}}(S(X_{st}))$ is the same as $\hat{V}^{\rho_{bs}^{bel}}(X_{st})$, the expected cost of the partial policy that corresponds to the primary branch ρ_{bs}^{bel} (dotted blue rectangles in 5.3(b)) that ρ_{bs} maps to.

The updated partial policy at the end of first iteration $\hat{\pi}_1 \triangleq \rho_{bs}^{bel}$, and $\hat{V}^{\hat{\pi}_1}(X_{st}) =$

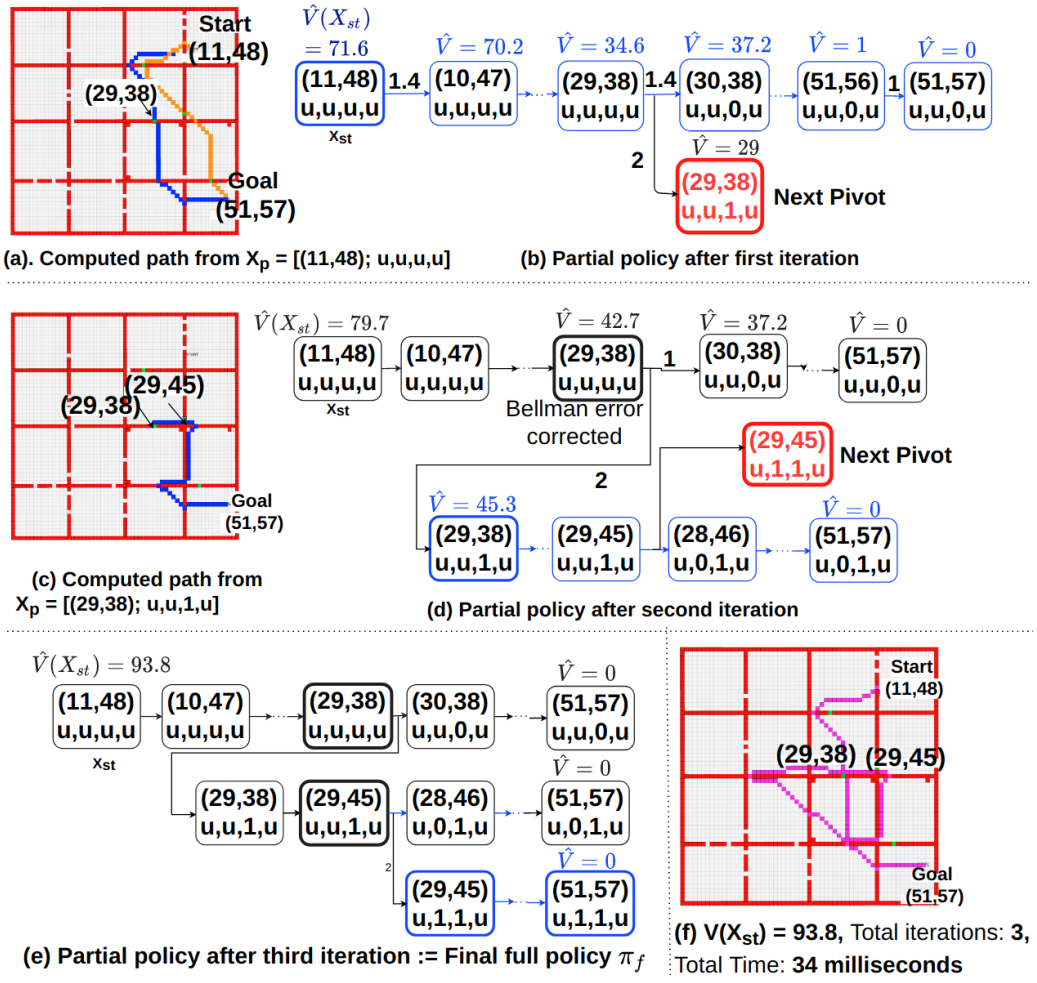


Figure 5.3: FAST-PPCP operation in **Ex.**. Number near an arrow (action) in a partial policy indicates transition costs (described in section 5.2). Some intermediate successors have been omitted (dotted).

$\hat{V}^{\rho_{bs}^{bel}}(X_{st}) = 71.6$. UPDATEMDP updates $\hat{V}$ values of belief states along ρ_{bs}^{bel} from X_{st}(5.3(b) blue bold $\hat{V}$-values). COMPUTEPOLICYDEVOIDX searches for a policy-devoid outcome in $\hat{\pi}_1$ and finds the non-preferred outcome $[(29,38);u,u,1,u]$ (5.3(b) black solid line rectangle) of the *sense-and-move* action at $(29,38)$ when $(30,38)$ is sensed as blocked.

The second iteration grows $\hat{\pi}_1$ from $X_p = [(29,38);u,u,1,u]$. This time it finds a path in $\mathscr{G}_1$ (5.3(c)) assuming the unknown hidden variables h_1, h_2 and h_4 are free but h_3 is blocked, because h_3 is known to be blocked (=1) in $H(X_p)$. COMPUTEBSPATH keeps rejecting other paths until it expands $n = [(29,38), \hat{V}^{\rho_{bs}^{bel}}(X_p) = 45.3]$ that corresponds to the path ρ_{bs} (5.3(c)) and the primary branch ρ_{bs}^{bel} (dotted blue rectangles in 5.3(d)). For this ρ_{bs}, $\hat{V}^{\hat{\pi}_1^{\mathsf{bs}}}(X_{st})$, the lower bound on the V-value of any policy that can develop from the updated partial policy $\hat{\pi}_1^{\mathsf{bs}}$ if ρ_{bs}^{bel} is added to $\hat{\pi}_1$, is 79.7 which is within B^{L}. The value 79.7 is computed as in eq. 5.8, using $P(X_p|X_{st},\hat{\pi}_i) = 0.5$, $\hat{V}(X_p) = 29$ which is the initialized Euclidean distance between $S(X_p) = (29,38)$ and goal, $\hat{V}^{\hat{\pi}_1}(X_{st}) = 71.6$, and $\hat{V}^{\rho_{bs}^{bel}}(X_p) = 45.3$ (5.3(d) blue bold $\hat{V}$). In addition to updating $\hat{V}$-values along ρ_{bs}^{bel}, UPDATEMDPREVERSE corrects the Bellman error of $\hat{V}$-value of $[(29,38);u,u,u,u]$ given its non-preferred outcome $[(29,38);u,u,1,u]$ got updated: its updated $\hat{V}$ according to the Bellman expectation equation becomes $42.7 = 0.5*(1+37.2)+0.5*(2+45.3)$ (5.3(d) pink italics $\hat{V}$). This Bellman backup is all the way up till (X_{st}). The third iteration begins from the policy-devoid belief state $[(29,45);u,1,1,u]$. A path with no stochastic transition qualifies as ρ_{bs} (ρ_{bs}^{bel} shown in 5.3(e) blue). Since no more policy-devoid outcomes exist in the updated partial policy (5.3(e)), FAST-PPCP terminates (full policy shown in 5.3(f) by superimposing paths found in the three iterations).

5.3.4 Theoretical analysis

We now present the theoretical analysis of FAST-PPCP.

We refer to an edge (s,a,s') in the graph $\mathcal{G}^i$ as deterministic (stochastic) if a is deterministic (stochastic). $|E|$ is the total number of edges in $\mathcal{G}^i$.

Lemma 1. ***Consistent heuristic for*** **COMPUTEBSPATH***: Let s and s' be two states in $S \in \mathbb{R}^n$. Let $\|s-s'\|_2$ be the euclidean distance between s and s' in $\mathbb{R}^n$. Let the modified edge-cost be $c_{sto}(s,a,s') = c_{max}$ if a is deterministic, and $c_{sto}(s,a,s') = |E|c_{max}$ if a is stochastic, where $c_{max} = \max_{\forall(s,a,s')\in\mathcal{G}^i}(c(s,a,s'))$ is the maximum given edge-cost in the deterministic graph $\mathcal{G}^i$ and $|E| = |E|$. With these modified edge-costs in $\mathcal{G}^i$, the euclidean distance heuristic $h_{cons}(s) = \left\|s - S(X_p)\right\|_2$ is a consistent heuristic for a backward search from the goal S_g to the pivot $S(X_p)$ to compute the shortest path in $\mathcal{G}^i$ if $c(s,a,s') > \|s-s'\|_2$. Or, $h_{cons}(s) = 0$ for $s = S(X_p)$, and*

$$\left\|s - S(X_p)\right\|_2 + c_{sto}(s,a,s') \geq \left\|s' - S(X_p)\right\|_2, \forall s \in S \; s.t \; s \neq S(X_p)$$

Proof. According to triangle inequality,

$$\left\|s - S(X_p)\right\|_2 + \left\|s - s'\right\|_2 \geq \left\|s' - S(X_p)\right\|_2. \tag{5.3}$$

If edge (s,a,s') is deterministic, then $c_{sto}(s,a,s') = c_{max}$, else $c_{sto}(s,a,s') = |E|.c_{max}$. Since $|E| \geq 1$, in both cases $c_{sto}(s,a,s') \geq c_{max} \geq c(S,a,s')$. Thus, if $c(s,a,s') \geq \|s-s'\|_2$,

$$\implies c_{sto}(s,a,s') \geq \left\|s - s'\right\|_2 \tag{5.4}$$

Using 5.4 in 5.3 we get

$$\left\|s - S(X_p)\right\|_2 + c_{sto}(s,a,s') \geq \left\|s' - S(X_p)\right\|_2. \tag{5.5}$$

□

We define a purely deterministic path as a path consisting of only those edges that are generated by deterministic actions.

Note that the set of paths in $\mathscr{G}^i$ is the same as the set of paths in $\mathscr{G}^i_{sto}$.

Lemma 2. *Let ρ^l be an arbitrary and fixed path in the set of all longest paths purely deterministic paths in $\mathscr{G}^i$, with cost $c_{sto}(\rho^l)$. Let ρ^{st} be any arbitrary and fixed path in $\mathscr{G}^i$ that has at least one stochastic edge, with cost c_{sto} (ρ^{st}). The cost c_{sto} of ρ^l_{det} is strictly less than the cost c_{sto} of any path having at least one stochastic action, or*

$$c_{sto}(\rho^l) < c_{sto}(\rho^{st})$$

Proof. For any arbitrary and fixed path ρ in $\mathscr{G}^i$, let N^{ρ}_{det} and N^{ρ}_{sto} be the total number of deterministic and stochastic edges in ρ. The cost $c_{sto}(\rho)$ of the path is given by:

$$c_{sto}(\rho) = N^{\rho}_{det} \cdot c_{max} + N^{\rho}_{sto} \cdot |E| c_{max}$$

For ρ^l_{det}, $N^{\rho}_{sto} = 0$ because it is purely deterministic, and $N^{\rho^l}_{det} < |E|$ as long as there exists at least one stochastic edge in $\mathscr{G}^i$. Therefore,

$$c_{sto}(\rho^l) = N^{\rho^l}_{det} c_{max} < |E| c_{max}$$

Let ρ^{st} be any arbitrary and fixed path in $\mathscr{G}^i$ that has at least one stochastic edge. For this path, $N^{\rho^{st}}_{sto} \geq 1$ and $N^{\rho^{st}}_{det} \geq 0$. Thus, its cost c_{sto} follows the following inequality:

$$c_{sto}(\rho^{st}) = N^{\rho^{st}}_{sto} |E| c_{max} + N^{\rho^{st}}_{det} c_{max} \geq |E| c_{max}$$

Thus, $c_{sto}(\rho^l) < c_{sto}(\rho^{st})$. □

Lemma 3. *Let ρ_1 and ρ_2 be two arbitrary and fixed paths in $\mathscr{G}^i$ from $S(X_p)$ to S_g. Let*

$N_{sto}^{\rho_1}$ and $N_{sto}^{\rho_2}$ be the total number of stochastic actions in ρ_1 and ρ_2 respectively. For a sufficiently large $|E|$, if $N_{sto}^{\rho_1} < N_{sto}^{\rho_2}$, then $c_{sto}(\rho_1) < c_{sto}(\rho_2)$.

Proof. For any arbitrary and fixed path ρ in $\mathscr{G}^i$, let N_{det}^{ρ} and N_{sto}^{ρ} be the total number of deterministic and stochastic edges in ρ. The cost $c_{sto}(\rho)$ of the path is given by:

$$c_{sto}(\rho) = N_{det}^{\rho} \cdot c_{max} + N_{sto}^{\rho}|E|c_{max}$$

For paths for which $\frac{N_{det}^{\rho}}{N_{sto}^{\rho}} << |E|$, $c_{sto}(\rho) \sim N_{sto}^{\rho}|E|c_{max} \implies c_{sto}(\rho) \propto N_{sto}^{\rho}$. Thus, if $N_{sto}^{\rho_1} < N_{sto}^{\rho_2}$, then $c_{sto}(\rho_1) < c_{sto}(\rho_2)$ □

Derivation of the cost-scheme in $\mathscr{G}_{sto}^i$. In $\mathscr{G}_i$, we want the search to consider paths as candidate for ρ_{bs} in increasing order of the number of stochastic transitions, starting from the one with minimum. This can be done by modifying edge-costs from given cost $c(s,a,s')$ in $\mathscr{G}_i$ such that the path with the minimum number of stochastic transitions is the shortest (minimum cost) path in $\mathscr{G}_i$, or more generally, ordering paths according to their cost using this modified cost-scheme is the same as ordering them according to the number of stochastic transitions. Then running a backward A^* from S_{goal} to $S(X_p)$ that expands in increasing order of $f = g_{sto} + h_c$ where h_c is consistent heuristic and expanding $S(X_p)$ for the first time would give the path with least number of stochastic transitions. The second time $S(X_p)$ is expanded, the second best path in terms of number of stochastic transitions trans is found, and so on. To get this desired search behavior, an intuitive cost scheme is 0/1 cost assignment (Note that to design a cost function that minimizes *number* of stochastic edges, all stochastic edges should have the same constant cost, and all deterministic edges should have another constant cost), i.e, assigning 1 to stochastic actions and 0 to deterministic ones. But assigning 0 cost to edges would mean that a non-zero heuristic function (for example euclidean distance) would not be consistent. It is essential

to have a non-zero heuristic function in order to speedup search. Therefore, we assign a positive cost β to the deterministic edges. To make sure the path with minimum number of stochastic transitions is the least cost path, we need to make sure that even the largest purely deterministic path is strictly less than the smallest path with a single stochastic transition (which is a one-edge path with only 1 stochastic edge). To satisfy this, if deterministic edges have cost β, stochastic edges are set to cost $|E|\beta$. Given this setting, the cost $c_{sto}(\rho)$ of a path with is:

$$c_{sto}(\rho) = N^{\rho}_{det} \cdot \beta + N^{\rho}_{sto} \cdot |E|\beta$$

, where N^{ρ}_{det} and N^{ρ}_{sto} are the total number of deterministic and stochastic edges in ρ. This cost scheme ensures that ordering of paths according to their cost c_{sto} is the same as their ordering according to the number of stochastic transitions for sufficiently large $|E|$. We assume that a non-zero consistent heuristic is known for the original (given) cost function (For example, in the robot navigation domain, euclidean distance to goal is a consistent heuristic function if the given edge-costs are euclidean distances). If we want to use the same heuristic function for the modified cost-scheme as well, we need to ensure that $c_{sto}(n,a,n') \geq c(s,a,s')$ for all edges (s,a,s') in G_i. Setting $minc_{sto}(n,a,n') = c_{max} = max_{\forall(s,a,s')\in G_i}(c(s,a,s'))$ ensures this. Thus, FAST-PPCP uses the following cost-scheme:

$$c_{sto}(n,a,n') = \begin{cases} |E|c_{max}, & \text{if a is stochastic} \\ c_{max}, & \text{otherwise (a is deterministic)} \end{cases}$$

Remark 1. COMPUTEBSPATH *is a backward search from S_g to $S(X_p)$ that expands search-nodes in increasing order of $f(n) = g_{sto}(n) + h(n, S(X_p))$, where a search-node $n = (s, \hat{V}^{\rho}(s))$ and corresponds to a path ρ from s to goal S_g with $\hat{V}$ being $\hat{V}^{\rho}(s)$.*

Lemma 4. COMPUTEBSPATH *expands nodes with state $s = S(X_p)$ in increasing order of*

their g_{sto} value.

Proof. Let $n_1 = (S(X_p), \hat{V}^{\rho_1}(s))$ and $n_2 = (S(X_p), \hat{V}^{\rho_2}(s))$ be two arbitrary nodes such that both n_1 and n_2 have state $s = S(X_p)$ and n_1 is expanded before n_2 by COMPUTEBSPATH. Using Remark 1, if during the search n_1 is expanded before n_2, then $f(n_1) \leq f(n_2)$.
Since both n_1 and n_2 have the state $s = S(X_p)$ and $h(n, S(X_p))$ is an estimate for cost of path from the state s in n to $S(X_p)$ (since backward search), therefore, $h_c(n_1, S(X_p)) = h_c(n_2, S(X_p)) = 0$.
Thus, $f(n_1) = g_{sto}(n_1)$ and $f(n_2) = g_{sto}(n_2)$, and using Remark 1 it follows that:

$$g_{sto}(n_1) \leq g_{sto}(n_2) \tag{5.6}$$

Since eq. 5.6 holds for any arbitrary nodes n_1 and n_2 having $s = S(X_p)$, hence lemma 4 holds. □

Theorem 1. ***Termination Condition of* COMPUTEBSPATH** *Let $\mathbb{P}$ be the set of all paths in $\mathscr{G}^i$ from $S(X_p)$ to S_g in a FPI i. For a path $\rho_c \in \mathbb{P}$, let $\hat{V}^{\hat{\pi}_i^{+c}}(X_{st})$ be the $\hat{V}$ of the updated partial policy $\hat{\pi}_i^{+c}$ from X_{st} assuming the primary branch ρ_c^{bel} gets added to $\hat{\pi}_i$.* COMPUTEBSPATH *upon termination computes a path $\rho_{bs} \in \mathbb{P}$ that satisfies:*

$$\rho_{bs} = \arg\min_{\rho_c \in \mathbb{P}} c_{sto}(\rho_c(S(X_p))) \; s.t \; \hat{V}^{\hat{\pi}_i^{+c}}(X_{st}) \leq \alpha V_L^*(X_{st}) \tag{5.7}$$

given ρ_{bs} exists.

Proof. In a FAST-PPCP iteration i, let π_i be the partial policy computed so far (from iteration 0 to i). If COMPUTEBSPATH expands a node $n = (S(X_p), \hat{V}^{\rho_c}(S(X_p)))$, it computes the updated v-value estimate of X_{st} assuming ρ is included in π_i, by replacing the old v-value

estimate $\hat{V}(X_p)$ of X_p with the new estimate $\hat{V}^{\rho_c^{bel}}(S(X_p))$ to get $\hat{V}^{\hat{\pi}_i^{+c}}(X_{st})$ as:

$$\hat{V}^{\hat{\pi}_i^{+c}}(X_{st}) \leftarrow \hat{V}^{\hat{\pi}_i}(X_{st}) + P(X_p|X_{st})(-\hat{V}(X_p) + \hat{V}^{\rho_c^{bel}}(X_p)) \tag{5.8}$$

where $P(X_p|X_{st})$ is the probability of reaching the pivot X_p following the current partial policy π_i from X_{st}.

COMPUTEBSPATH terminates when for the first time $n = [S(X_p), \hat{V}^{\rho_c^{\mathsf{bel}}}(X_p)]$ gets expanded such that $\hat{V}^{\hat{\pi}_i^{+c}}(X_{st}) \leq \alpha V_L^*(X_{st})$. Given this, and lemma 4, theorem 1 holds. □

Lemma 5. *At the iteration when* COMPUTEBSPATH *is computing a path* ρ *from* $S(X_p)$ *(and the corresponding* ρ^{bel} *from* X_p*), any non-preferred outcome of a stochastic transition in* ρ^{bel} *can only be policy-devoid.*

Proof. Case 1: When an unexplored policy-devoid outcome is explored for the first time, because sensing is perfect, the subgraphs that result from different outcomes of a single stochastic action are disjoint. Thus, a subgraph from X_p is disjoint with a subgraph from any other state pivot in any other branch in $\hat{\pi}_i$.

Case 2: When a safe branch is removed in policy correction from X_p, and another branch is added from X_p, any non-preferred outcome of a stochastic transition in ρ_c^{bel} can only be policy-devoid. (1) Only such an X_p gets its branch replaced which doesn't have a dependent branch from any non-preferred outcome X_{np} of a stochastic transition on this branch. (2) Because sensing is perfect, the subgraphs that result from different outcomes of a single stochastic action are disjoint. Thus, a subgraph from X_p is disjoint with a subgraph from any other previously explored pivot with an existing primary branch in $\hat{\pi}_i$. Thus, while searching for branch from X_p only way to encounter a branch that has a non-preferred outcome X_{np} shared with a previously existing branch is if we removed a branch from

X_p which had an existing branch ρ_{np} from X_{np} but did not remove ρ_{np} because of which $V(X_{np})$ was not set to underestimate of V^*. Since we never remove such a branch, this case never arises. □

Lemma 6. *Let ρ' be a path in $\mathcal{G}_i$ from s' to goal. Let s be the outcome of (s',a) for an action a in ρ'. Let ρ'^{bel} be the corresponding primary branch of ρ' from a belief state $X' = [s', H(X)]$. Let $X^b = succ(X',a)^b$ be the preferred outcome of belief state-action (X',a). Since ρ' maps to ρ'^{bel}, $s = S(X^b)$. Let ρ be the subpath from s and ρ^{bel} be its corresponding primary branch from X_b. For a path ρ' from s in $\mathcal{G}_i$,* FAST-PPCP *recursively defines $\hat{V}^{\rho'}(s')$ which can be computed by backing up from S_g with $\hat{V}(S_g) = 0$ as:*

$$\begin{aligned}\hat{V}^{\rho'}(s') = &\sum_{X_{np} \in succ(X',a) \setminus X^b} P(X_{np}|X',a)(c(s',a,S(X_{np})) + \hat{V}(X_{np})) \\ &+ P(X^b|X',a)(c(s',a,s) + \hat{V}^{\rho}(s))\end{aligned} \tag{5.9}$$

$\hat{V}^{\rho'}(s')$ is the same as $\hat{V}^{\rho'^{bel}}(X')$ which is a lower bound on the V-value of any policy containing ρ'^{bel}.

Proof. $\hat{V}^{\rho'}(s')$ is the same as $\hat{V}^{\rho'^{bel}}(X')$:

$$\begin{aligned}\hat{V}^{\rho'^{\mathsf{bel}}}(X') = &\sum_{X_{np} \in succ(X',a) \setminus X^b} P(X_{np}|X',a)(c(X',a,X_{np}) + \hat{V}(X_{np})) \\ &+ P(X^b|X',a)(c(X',a,X^b) + \hat{V}^{\rho^{\mathsf{bel}}}(X^b))\end{aligned} \tag{5.10}$$

$\hat{V}^{\rho'^{bel}}(X')$ is a lower bound on the V value of any policy that contains ρ'^{bel}. This is because: (1) $\hat{V}(X_{np})$ of policy-devoid outcomes are underestimates $V_u^*(X) \leq V^*(X)$. (2) We show in lemma 5 that at the iteration when ρ_c^{bel} is being computed, any non-preferred outcome of a stochastic transition in ρ_c^{bel} can only be policy-devoid. (3) Also, an edge in the belief-space has the same cost as its mapping in the deterministic space, i.e, $C(s',a,s) = C(X',a,X)$

where $s' = S(X')$ and $s = S(X)$. □

Lemma 7. *Between two subpaths ρ_2 and $\rho_1(s)$ from a state $s \in G_s$ to S_g, if $\rho_2(s)$ is dominated by $\rho_1(s)$, then for any path $\psi_2(S(X_p))$ from $S(X_{st})$ to S_g passing through $\rho_2(s)$, there exists a path $\psi_1(S(X_p))$ from $S(X_p)$ to S_g that dominates $\psi_2(S(X_p))$*

The proof of lemma 7 is deferred to Appendix A in chapter A.

Lemma 8. ***Pruning using dominance**: Let $\mathbb{P}$ be the set of all paths in G_s from $S(X_p)$ to S_g. Let $\mathscr{P}_\sigma \subset \mathbb{P}$ be a subset of paths from $S(X_p)$ to S_g such that no two paths have the same $\boldsymbol{g}$-value and every path $\sigma \in \mathscr{P}_\sigma$ satisfies*

$$\sigma = \arg\min_{p \in \mathbb{P}} g^p_{sto}(S(X_p)) \; s.t \text{ using } g^p_Q(S(X_p)) \; \hat{V}^p(X_{st}) \leq \alpha V^*_L(X_{st}) \tag{5.11}$$

For a state $s \in G_s$, let $\rho_1(s)$ and $\rho_2(s)$ be two sub-paths from s to S_g that the search visits with path costs $\boldsymbol{g}^{\rho_1}(s)$ and $\boldsymbol{g}^{\rho_2}(s)$. Let $\rho_2(s)$ be dominated by $\rho_1(s)$. For every state $s \in G_s$, we prune all paths from $S(X_p)$ to S_g that pass through $\rho_2(s)$. It is guaranteed that this pruning method does not prune any path in $\mathscr{P}_\sigma$.

The proof of lemma 8 is deferred to Appendix A in chapter A.

Remark 2. *For a non-preferred outcome X_{np} that has a primary branch ρ^{bel} from it in $\hat{\pi}_i$, it is redundant to consider any outcome on ρ^{bel} as pivot. This is because ρ_{bel} has either a success outcome or a deterministic outcome, and any eligible path (given the search objective) found from either type of outcomes is a subpath in an eligible path from X_{np}, which can be computed by considering X_{np} as pivot.*

Theorem 2. *(B^L-Completeness) Let Π_{bl} be a set of full policies from X_{st} in the belief-space, such that every $\pi_{bl} \in \Pi_{bl}$ satisfies two conditions: (C1) If there exists a pair of states $X_1 \in \pi_{bl}$ and $X_2 \in \pi_{bl}$, where X_2 can be reached with a non-zero probability from X_1 following π_{bl} and whose actions $\pi_{bl}(X_1)$ and $\pi_{bl}(X_2)$ are affected by the hidden variables $h^{S(X_1),\pi_{bl}(X_1)}$ and $h^{S(X_2),\pi_{bl}(X_2)}$, then it holds that $h^{S(X_1),\pi_{bl}(X_1)}$ is not the same as $h^{S(X_2),\pi_{bl}(X_2)}$.*

(C2) The V-value $V^{\pi_{bl}}(X_{st})$ of π_{bl} from X_{st} is $\leq \alpha V_L^*(X_{st}) \triangleq B^L$, where B^L is the lower bound of $V^*(X_{st})$ in the current iteration.
If there exists a non-empty set Π_{bl}, then FAST-PPCP *upon termination is guaranteed to find a full policy $\pi_f \in \Pi_{bl}$.*

Proof. ComputeBSPath while searching does not keep track of the hidden component $H(\cdot)$ of the belief states, therefore it can only find policies having primary branches that satisfy condition (C1): a primary branch does not have two belief states with actions that sense the same hidden variable in $H(X_p)$.

Among such policies from X_{st}, if there exists policies with V-value $<= \alpha V_L^*(X_{st})$, Theorem 2 states that Fast-PPCP will find one such policy. Or, the only time Fast-PPCP returns without solution from X_{st} is when no such policy exists.

This holds because of the following arguments.

1. When a candidate path (equivalently its corresponding candidate primary branch from X_p) is computed during policy growth, we show that any non-preferred outcome of a stochastic transition on this primary branch can only be policy-devoid. Fast-PPCP always maintains the $\hat{V}$-value of any policy-devoid outcome X to be an underestimate of $V^*(X)$. Given this, if a candidate primary branch is rejected, it is only when no full policy with V-value $<= \alpha V_L^*(X_{st})$ can develop from the updated partial policy—formed by adding this candidate primary branch to the current partial policy. Also, an existing primary branch is safely rejected during policy correction—without discarding any full policy with V-value $<= \alpha V_L^*(X_{st})$—only if every non-preferred outcome of stochastic transitions on this branch is policy-devoid. Thus, if a primary branch is rejected given a partial policy, the only policies Fast-PPCP discards without exploring fully are the ones that exceed $\alpha V_L^*(X_{st})$.
2. The previous statement applies to a primary branch from X_{st} as well—the only time

Fast-PPCP resumes policy growth from X_{st} and rejects a candidate or an existing primary branch from X_{st} is when every non-preferred outcome of stochastic transitions on this primary branch is policy-devoid. Since Fast-PPCP is never prevented from exploring any non-preferred outcome in a partial policy, in the worst case, this is when every non-preferred outcome X_{np} has been explored and for each X_{np}, all candidate primary branches from X_{np} have been rejected. Given 1., we conclude that the existing primary branch from X_{st} is rejected only when no full policy with V-value $<= \alpha V_L^*(X_{st})$ can be developed from this primary branch.

The only time Fast-PPCP returns no-solution from X_{st} is when it rejects every primary branch from X_{st}, indicating that no full policy with V-value $<= \alpha V_L^*(X_{st})$ can be developed from X_{st}. □

Lemma 9. *Completeness: Suppose there exists a policy π^* from X_{st} in the belief-space such that it satisfies the condition (C1): If there exists a pair of states $X_1 \in \pi_{bl}$ and $X_2 \in \pi_{bl}$, where X_2 can be reached with a non-zero probability from X_1 following π_{bl} and whose actions $\pi_{bl}(X_1)$ and $\pi_{bl}(X_2)$ are affected by the hidden variables $h^{S(X_1),\pi_{bl}(X_1)}$ and $h^{S(X_2),\pi_{bl}(X_2)}$, then it holds that $h^{S(X_1),\pi_{bl}(X_1)}$ is not the same as $h^{S(X_2),\pi_{bl}(X_2)}$.*
In the worst-case, FAST-PPCP *is guaranteed to find π^*.*

Proof. Since FAST-PPCP is $\mathtt{B}^{\mathtt{L}}$-complete, a particular $V_L^*(X_{st})$ is increased only when no solution is found from X_{st}, which indicates that every possible full policy has been considered (not necessarily fully constructed). Then $V_L^*(X_{st})$ is increased by running PPCP iterations. In the worst-case, in limit of the total of number PPCP iterations, it ends up finding π^*. However, for a particular $\mathtt{B}^{\mathtt{L}}$, if $\mathtt{B}^{\mathtt{L}}$-completeness is not guaranteed and FAST-PPCP misses finding a full policy π with $V^{\pi}(X_{st}) \leq \mathtt{B}^{\mathtt{L}}$, this π can also be missed in the future in spite of increasing $\mathtt{B}^{\mathtt{L}}$, because $V^{\pi}(X_{st}) \leq \mathtt{B}^{\mathtt{L}} \leq \mathtt{B}^{\mathtt{L}}_{inc}$ holds for $\mathtt{B}^{\mathtt{L}}_{inc}$, the increased

$\mathtt{B}^{\mathtt{L}}$. Thus, $\mathtt{B}^{\mathtt{L}}$-completeness guarantees that any other π that is not π^* but is $\mathtt{B}^*$-bounded is not missed. □

Theorem 3. *The full policy π_f starting from X_{st} computed by* FAST-PPCP *upon termination is bounded suboptimal.*

Proof. FAST-PPCP terminates when it can't find a policy-devoid belief-state on the current $\hat{\pi}_i$. Since π_f is no longer a partial policy, $\hat{V}^{\pi_f}(X_{st})$ in this iteration is no longer a lower bound but is the actual V-value of π_f. Since $\hat{V}^{\pi_f}(X_{st}) \leq \mathtt{B}^{\mathtt{L}}$ has been ensured in procedure COMPUTEBSPATH, π_f is $\hat{V}^{\pi_f}(X_{st}) \leq \mathtt{B}^{\mathtt{L}} \leq B^*$ □

5.4 Application to discounted reward-based belief-MDPs with clear preference and perfect sensing

In this section, we show how to apply FAST-PPCP and PPCP in discounted reward belief MDPs with clear preference and perfect sensing. We need this application such that we can experimentally evaluate FAST-PPCP and PPCP for discounted reward-based domains in addition to undiscounted cost-based domains stated so far in this thesis.

Consider a discounted reward-based belief-MDP M_R, with clear preference and having perfect sensing assumption (CP-PS belief MDP). We now show how to convert M_R into a discounted cost CP-PS belief-MDPs M_c in which FAST-PPCP and PPCP can converge to a bounded suboptimal and optimal policy respectively. FAST-PPCP computes the initial $V_L^*(X_{st}) \leq V^*(X_{st})$ by running a single PPCP search. This search is a backward A^* search that defines g-values in the underlying deterministic graph of M_c as $\hat{Q}$ values (Eq. 3 in PPCP (Likhachev 2006)). This definition makes the g-value of $S(X_{st})$ corresponding to the optimal path from $S(X_{st})$ a lower bound on $V^*(X_{st})$. For this to hold, $\hat{Q}$ values need to be monotonically increasing from S_g along an optimal path.

- Assume: minimum cost in M_c is c_{min}, and maximum cost in M_c is c_{max}
- Assume: For a belief state action pair (X,a) if best successor is X'_b with cost $c_{s'_b}$ and $S(X'_b) = s'_b$, non preferred outcome is X'_f with cost $c_{s'_f}$ and $S(X'_f) = s'_f$, then $c_{s'_b} \leq c_{s'_f}$ and $V^*(X'_b) \leq V^*(X'_f)$.
  Then, $c_{s'_b} + \gamma.V^*(X'_b) \leq c_{s'_f} + \gamma.V^*(X'_f)$.

- We want to see if we can derive conditions such that $\hat{Q}_{v,g^*}(S(X),a) > g^*(s'_b)$ for optimal trajectory.
  - If $Q_{v,g^*}(S(X),a) = \sum_{s' \in \{s'_b, s'_f\}} P_{s'} \max(c(s,a,s') + \gamma.V(s')), c_{s'_b} + \gamma.g^*(s'_b)) \geq c_{s'_b} + \gamma.g^*(s'_b) \geq c_{min} + \gamma.g^*(s'_b)$
  - Sufficient to show $c_{min} + \gamma.g^*(s'_b) > g^*(s'_b)$ or $c_{min} > (1-\gamma)g^*(s'_b)$
  - If assume optimal path from every state to goal has maximum steps T_{max}, then $(1-\gamma)g^*(s'_b) \leq (1-\gamma)(c_{max} + \gamma.c_{max} + \gamma^2.c_{max}..... + \gamma^{T_{max}}.c_{max} = (1-\gamma^{T_{max}}).c_{max}$

Also, we want M_c to be such that if a full policy π^* from X is optimal in M_R, it is also optimal in M_c. Specifically, we want the following relation `R1`:
`R1`: If $V^{\pi}_{M_R}(X) > V^{\pi^*}_{M_R}(X)$, then $V^{\pi}_{M_c}(X) > V^{\pi^*}_{M_c}(X)$, and if $V^{\pi}_{M_R}(s) = V^{\pi^*}_{M_R}(X)$, then $V^{\pi}_{M_c}(X) = V^{\pi^*}_{M_c}(X)$.

We assume:
(1) We know a constant T_{max} s.t. $T_{max} = \max_{X \in M_R}(|\pi^*(X)|)$ where $|\pi^*(X)|$ is the length of the longest branch in the optimal policy π^* from state X, and (2) We can compute a constant $\mathscr{C} > R_{max}$ = the maximum reward values in M_R We propose the following transformation from M_R to M_c such that if π^* from X is optimal in M_R, it is also optimal in M_c:
(1) $R \rightarrow -R + \mathscr{C}$ (map all rewards R in M_R to costs $-R + \mathscr{C}$ in M_C), and (2) For a belief

state $X \in M_R$ and a full policy π starting from X, let ρ_l^{bel} be the branch from X to goal in π having the largest number of actions (given by $|\rho_l^{bel}|$) among other branches in π. If $|\rho_l^{bel}| < T_{max}$, for each branch ρ^{bel} in π add $T_{max} - |\rho^{bel}|$ number of self-absorbing actions at goal with cost $\mathscr{C}$, else add $|\rho_l^{bel}|$ number of self-absorbing actions at goal with cost $\mathscr{C}$.
In practice, we implement this in PPCP's backwards A* search in the deterministic space by planning for T_{max} steps with the step number t included in the state space. This leads to following relationship between values :
For full policy π from X with $|\pi| = |\rho_l^{bel}|$ as defined,

$$V_{M_c}^{\pi}(X) = -V_R^{\pi}(X) + \mathscr{C}\frac{(1-\gamma^{|\pi|})}{(1-\gamma)}, /if|\pi| > T_{max} \tag{5.12}$$

$$V_{M_c}^{\pi}(X) = -V_R^{\pi}(X) + \mathscr{C}\frac{(1-\gamma^{T_{max}})}{(1-\gamma)}, /if|\pi| \leq T_{max} \tag{5.13}$$

In both cases, for $|\pi^*| \leq T_{max}$ R1 holds.
Further, if R_{min} and R_{max} are the minimum and maximum reward values in M_R respectively, we derive the following condition on $\mathscr{C}$ to ensure monotonicity: In M_c, let $c_{min} = \mathscr{C} - R_{max}$, and $c_{max} = \mathscr{C} - R_{min}$.
Recall that in order to get $Q_{v,g^*}(S(X),a) > g^*(s'_b)$ for optimal trajectory, we showed that it was sufficient to show:

$$c_{min} > (1-\gamma)g^*(s'_b) \tag{5.14}$$

In M_c, after the transformation, $g^*(s'_b) < \frac{(1-\gamma^{T_max})}{(1-\gamma)}\max(c_{max}, \mathscr{C})$.
Therefore, we need the following condition to be true:

$$c_{min} > (1-\gamma^{T_max})\max(c_{max}, \mathscr{C}) \tag{5.15}$$

For positive costs in M_c, we need $\mathscr{C} > R_{max}$

Combined condition on $\mathscr{C}$:

$$\mathscr{C} > \max\left(R_{max}, \frac{R_{max}}{\gamma^{T_{max}}}, \frac{R_{max} - (1-\gamma^{T_{max}})R_{min}}{\gamma^{T_{max}}}\right) \tag{5.16}$$

5.5 Experiments

All experiments were run on a machine with an Intel® Core™ i7-5600U CPU @ 2.60GHz × 4, and 15 GB RAM. All algorithms were implemented in C++11, compiled using the same optimization flag -03.

Domain 1: Robot Navigation in Partially Known Environment We compare FAST-PPCP with PPCP and weighted-RTDP-BEL (WRTDP-BEL)—with a weight on the admissible heuristic in RTDP-BEL [15]—which serves as a suboptimal baseline belief-MDP planner. We run experiments in the robot navigation domain on 2D grids of size 60×60 (small) with 7, 11 and 15 unknown variables and 300×300 (large) with 309 and 474 unknown variables (corresponding to 30%, and 50% of the doors being hidden).

Results: Results in Tables 5.3 (small environments) and 5.4 (large environments) are averaged over 40 planning instances.

Algorithm	Time(s)	Exp. cost	Iterations
unk. = 7; α = 1.5			
Fast-PPCP	**0.03** $\pm$ **0.05**	**83** $\pm$ **12**	**3.12** $\pm$ **2.51**
wRTDP-Bel	600 (timeout)	90 $\pm$ 10	237 $\pm$ 459
PPCP	1.10 $\pm$ 0.60	80 $\pm$ 10	138.87 $\pm$ 73.86
unk. = 11; α = 1.5			
Fast-PPCP	**0.03** $\pm$ **0.04**	**83** $\pm$ **11**	**2.90** $\pm$ **2.18**
wRTDP-Bel	600 (timeout)	90 $\pm$ 11	162 $\pm$ 305
PPCP	0.75 $\pm$ 0.46	79 $\pm$ 10	89.29 $\pm$ 56.20
unk. = 15; α = 1.5			
Fast-PPCP	**0.03** $\pm$ **0.06**	**82** $\pm$ **12**	**3.15** $\pm$ **2.53**
wRTDP-Bel	600 (timeout)	89 $\pm$ 11	171 $\pm$ 326
PPCP	1.17 $\pm$ 0.67	79 $\pm$ 10	140.57 $\pm$ 78.01
unk. = 7; α = 2.0			
Fast-PPCP	**0.01** $\pm$ **0.003**	**85** $\pm$ **12**	**2** $\pm$ **0**
wRTDP-Bel	600 (timeout)	94 $\pm$ 5	71 $\pm$ 65
PPCP	1.10 $\pm$ 0.60	80 $\pm$ 10	138.87 $\pm$ 73.86
unk. = 11; α = 2.0			
Fast-PPCP	**0.03** $\pm$ **0.04**	**84** $\pm$ **12**	**2** $\pm$ **0**
wRTDP-Bel	600 (timeout)	94 $\pm$ 5	67 $\pm$ 75
PPCP	0.75 $\pm$ 0.46	79 $\pm$ 10	89.29 $\pm$ 56.20
unk. = 15; α = 2.0			
Fast-PPCP	**0.01** $\pm$ **0.003**	**85** $\pm$ **13**	**2** $\pm$ **0**
wRTDP-Bel	600 (timeout)	95 $\pm$ 5	81 $\pm$ 100
PPCP	1.17 $\pm$ 0.67	79 $\pm$ 10	140.57 $\pm$ 78.01

Table 5.3: Navigation (small environments)

In small environments, FAST-PPCP with $\alpha = 1.5$ computes a policy in 3 iterations on average, with expected cost ~ 1.03 times higher than the optimal, while being $\sim$ 36 times faster in environments with 7 and 15 unknowns, and 25 times faster with 11 unknowns. However, with $\alpha > 1.6$ (we have reported for $\alpha = 2$), FAST-PPCP chooses purely deterministic paths over paths with stochastic actions. WRTDP-BEL with weight 2 is unable to converge within a timeout of 10 minutes (values reported at timeout). For large environments with 309 unknowns, FAST-PPCP with $\alpha = 1.5$ and 1.7 computes solutions ~ 1.03 higher than PPCP while being ~ 11 and ~ 25 times faster than PPCP. Similar results are seen for both weights in the 474 unknown variables case. WRTDP-BEL is unable to converge within 30 mins for large environments.

Algorithm	**Time(s)**	**Exp. cost**	**Iterations**
unk. = 309; α = 1.5			
Fast-PPCP	**13.16 ± 36.63**	**412 ± 60**	**4 ± 4.19**
PPCP	146 ± 23	380 ± 53	758 ± 525
unk. = 474; α = 1.5			
Fast-PPCP	**20.73 ± 43.7**	**514 ± 73**	**5.2 ± 5.6**
PPCP	146 ± 23	380 ± 53	758 ± 525
unk. = 309; α = 1.7			
Fast-PPCP	**5.97 ± 23.62**	**408 ± 62**	**2.51 ± 2.2**
PPCP	150 ± 0.1	384 ± 52	619 ± 379
unk. = 474; α = 1.7			
Fast-PPCP	**6.23 ± 24.58**	**539 ± 87**	**3.3 ± 3.7**
PPCP	150 ± 0.1	384 ± 52	619 ± 379

Table 5.4: Navigation (large environments)

Domain 2: RockSample. We chose the RockSample domain [105] that has a clear

preference of outcomes (it is preferred to sense a *good* rock over a *bad* rock) to evaluate PPCP and FAST-PPCP in a discounted belief-MDP setting with discount factor $\gamma = 0.95$. We maintain the perfect sensing assumption by removing "check" actions and update the belief states accordingly. We compare with HSVI2 [105] which, as shown in [72], performs better than SARSOP for the RockSample domain. Results are averaged over 3 environments each for Rocksample$(10, 10)$ and Rocksample$(15, 15)$ and planning instances generated by varying the start state from top to bottom at the left boundary for each environment (the goal is any state on the right boundary).

Results: Results are summarized in Table 5.5.

Algorithm	**Time(s)**	**Exp. reward**	**Iterations**
RockSample(10,10)			
Fast-PPCP ($\alpha = 1.1$)	**1.23** $\pm$ **0.20**	21 $\pm$ 25	**2** $\pm$ **0**
PPCP	4.21 $\pm$ 1.28	**51** $\pm$ **20**	79 $\pm$ 52
HSVI2	600 (timeout)	12 $\pm$ 1	2215 $\pm$ 184
RockSample(15,15)			
Fast-PPCP ($\alpha = 1.1$)	**5.12** $\pm$ **2.74**	51 $\pm$ 120	**2.11** $\pm$ **0.42**
PPCP	44.13 $\pm$ 30.37	**70** $\pm$ **150**	203 $\pm$ 179
HSVI2	-	-	-

Table 5.5: RockSample

Note that the values reported in column 2 are expected *rewards* V_R^π instead of expected costs V_C^π, the relation given in section 5.4. For $\alpha = 1.1$, in RockSample$(10, 10)$ problems, FAST-PPCP spends roughly a quarter of the time, in ~ 39 times fewer iterations than PPCP on average to terminate, and in RockSample$(15, 15)$ problems, FAST-PPCP spends roughly one-eighth of the time, in ~ 100 times fewer iterations than PPCP on average to terminate. For both RockSample domains, HSVI2 takes longer than 10 minutes to converge, and

computing an explicit belief space for some RockSample$(15, 15)$ instances even exhausts memory on a 15-GB RAM machine.

5.6 Conclusions, Discussion, and Lessons Learned

In this work, we present FAST-PPCP—a novel approach to probabilistic planning in domains with clear preferences over missing information. We achieve substantial decrease in run-time while incurring little loss in solution quality compared to PPCP, the state-of-the-art for planning on these problems, as well as other popular belief-MDP planners.

One limitation of FAST-PPCP is that *it may be possible* to construct worst-case scenarios in which FAST-PPCP performs computationally worse than PPCP. This is similar to the behavior of bounded-suboptimal deterministic searches such as weighted A*. Another limitation is the perfect sensing assumption which makes the scope of the problem narrower than POMDP planning.

We observe that dominance relationships between search nodes, that are exploited to prune parts of the search graph, significantly increase search efficiency. Without exploiting state dominance relationships, the FAST-PPCP search is around $\sim$5 times slower compared to the search using state dominance.

There are two main approaches to POMDP planning: offline policy computation and online search. In offline planning, the agent computes beforehand a policy contingent upon all possible future outcomes and executes the computed policy based on the observations received. One main advantage of offline planning is fast policy execution, as the policy is pre-computed. FAST-PPCP is an offline planner. In the RockSample domain, FAST-PPCP scales up dramatically for larger size RockSample problems than was possible previously by existing offline planners. Specifically, previous offline algorithms have achieved speed-up without exceeding the memory limit of standard CPUs for a maximum

size of RockSample(11,11) [117]. SARSOP [72], one of the fastest offline POMDP solvers, exceeds memory limits for RockSample problems larger than size (11,11). HSVI2 [106] and HSVI [105], two very popular offline POMDP solvers, exceed memory limits for RockSample problems larger than size (10,10) and (7,8) respectively. However, we have run FAST-PPCP on a RockSample problem of size (15,15) where FAST-PPCP terminates within $\sim$5 seconds.

Both FAST-PPCP and PPCP assume a clear preference over the outcomes of uncertain actions, that is, that we can identify a priori which outcomes of uncertain actions are known to be the best. While many problems exhibit clear preferences, there are many others for which it is difficult or impossible to predict clear preferences. However, in many of these cases, we can come up with *approximate* preferences, where we do not know a clear preference but have a bounded approximation on which outcomes we think may be the best. For example, while navigating in an environment with some surfaces that might be slippery, such as ice, we prefer that a surface is not slippery. But, the robot could "slip" along ice directly towards the goal with little effort. If the action costs are proportional to the efforts/energy required to reach the goal, it is not entirely true that it is best not to be slippery.

If approximate preference holds in a domain, the properties of completeness and bounded suboptimality would still hold for cases when FAST-PPCP finds a bounded suboptimal policy by iteratively executing Policy Growth only—by only adding branches to grow a partial policy in each iteration. If FAST-PPCP enters Policy Correction in an iteration, that is, it starts removing and replacing existing branches in the current partial policy, then the properties of completeness and bounded suboptimality would not hold. This is because of the following. FAST-PPCP updates the underestimate of the value of optimal policy from the state it decides to replace a branch. It performs the update by visiting all

paths from that state in the deterministic space and finding the minimum over the value of the partial policy corresponding to the primary branch that maps to a path. When clear preference holds, this minimum value is a lower bound on the value of any partial policy from the given state. However, this minimum value is not necessarily a lower bound when approximate preference holds.

Chapter 6

Optimizing Fast-PPCP for Planning in Environments with Large Unknown Regions

6.1 Motivation

In many real-world planning problems, AI agents operate in partially known environments. One approach for the agent to plan in such environments is by taking a deterministic approach, i,e., planning by assuming some instantiation of the variables that represent missing information about the environment (unknown variables), executing few actions of the plan, and replanning in response to sensing. While computationally efficient, this approach may lead to highly suboptimal behavior. In contrast, planning under uncertainty allows an agent to be much more robust with respect to missing information but also becomes computationally dramatically more expensive as it is a special class of planning

Figure 6.1: Satellite imagery of a part of Fort IndianaTown Gap, PA. There are large regions of unknown traversability due to dense forest canopies (dark green).

for Partially Observable Markov Decision Processes (POMDPs) [62, 68].

[80] showed that real-world planning problems often possess the property of *clear preferences* (CP), wherein one can identify beforehand *clearly preferred* values for unknown variables in the environment. For example, consider a robot navigating in a partially known environment: it will clearly prefer for any unknown region to be traversable rather than not. For problems that exhibit clear preferences and assume perfect sensing (CP-PS problems)—an assumption that there is no noise in sensing an unknown variable and any sensing operation returns the true state of the variable being sensed— [23] introduced FAST-PPCP, a novel planning algorithm that computes a provably bounded suboptimal policy. It provides a substantial gain in runtime over common alternative approaches to planning under uncertainty over missing information.

Starting from the start state, FAST-PPCP iteratively grows a partial policy into a full

policy. In each iteration it adds a path to the partial policy from a state in it that has no policy defined from it. For every policy-devoid state, FAST-PPCP maintains an underestimate of its optimal value.

In this work, we are motivated by the application of FAST-PPCP for goal-directed planning for navigation in partially known environments with large unknown regions. Partially known off-road terrains such as mines, military bases and disaster sites can have large unknown regions. Similarly, indoor environments can have large carpets or rugs. For some carpets that are thick, it is unknown at the time of planning whether the robot can get on to the carpet and completely traverse it or not.

In the domains of off-road navigation and indoor navigation, the agent typically has a static map of the environment. For example, the agent has access to a map of a terrain constructed from satellite imagery. The traversability of most of the terrain can be deduced from these aerial images, except for a few regions that are occluded by roofs or canopies in the aerial view (Figure 6.1). The agent can perfectly sense the traversability of each unknown region from any state adjacent to the boundary of this region (boundary-adjacent state). The preferred outcome of sensing is that the region is traversable.

When growing the partial policy constructed so far in this domain, consider the case when FAST-PPCP discovers that adding any path that senses an unknown region h from a boundary-adjacent state s leads to an *invalid* partial policy, i.e., one that would never grow into a bounded-suboptimal full policy. It discovers this invalidity only after exploring all paths from the non-preferred outcome of sensing h from s, which is a computationally expensive operation. Once discovered, it updates its knowledge of the value-underestimate—underestimate of the optimal value—of this non-preferred outcome with a more informed underestimate, i.e., the minimum over the cost of all the explored paths.

FAST-PPCP in its current form is agnostic to the correlation between value-underestimates

of sensing an unknown region from each boundary-adjacent state. As a result, it does not update value-underestimates of sensing h from other boundary-adjacent states when one of them gets its value-underestimate updated. Thus, it keeps adding paths that sense h from other boundary-adjacent states, and discovers their invalidity only after wasting search effort in explicitly exploring all paths from the non-preferred outcomes of sense actions in them. The search effort in these explorations can be quite high when the number of boundary-adjacent states is high due to large unknown regions. Large unknown regions in real world off-road terrains can be common due to forest canopies, mine-tunnels or structures covered with roofs. Similarly, even indoor environments can have large regions of unknown traversability such as mats or carpets on the floor.

In this work, we introduce an optimized version of FAST-PPCP with the aim of reducing wasted search effort incurred while planning in environments with large unknown regions. Our key insight is the following: if sensing h from a boundary-adjacent state s results in an invalid partial policy, then sensing h from other boundary-adjacent states near s are also likely to result in invalid partial policies. The optimized version of FAST-PPCP utilizes the correlation between value-underestimates of boundary-adjacent states; it uses the updated value-underestimate of sensing h from s to simultaneously compute updated value-underestimates of sensing h from other boundary-adjacent states. This update identifies invalid partial policies without having to waste search efforts to discover the invalidity via explicit exploration.

6.2 Problem Definition, Assumptions and Background

Example Domain. Consider an example environment in Figure 6.2, which is an approximate representation of the terrain in Figure 6.1. A robot has to navigate from *start* to *goal* in the terrain represented by the 2D grid. We refer to this example as [**Ex.:**]

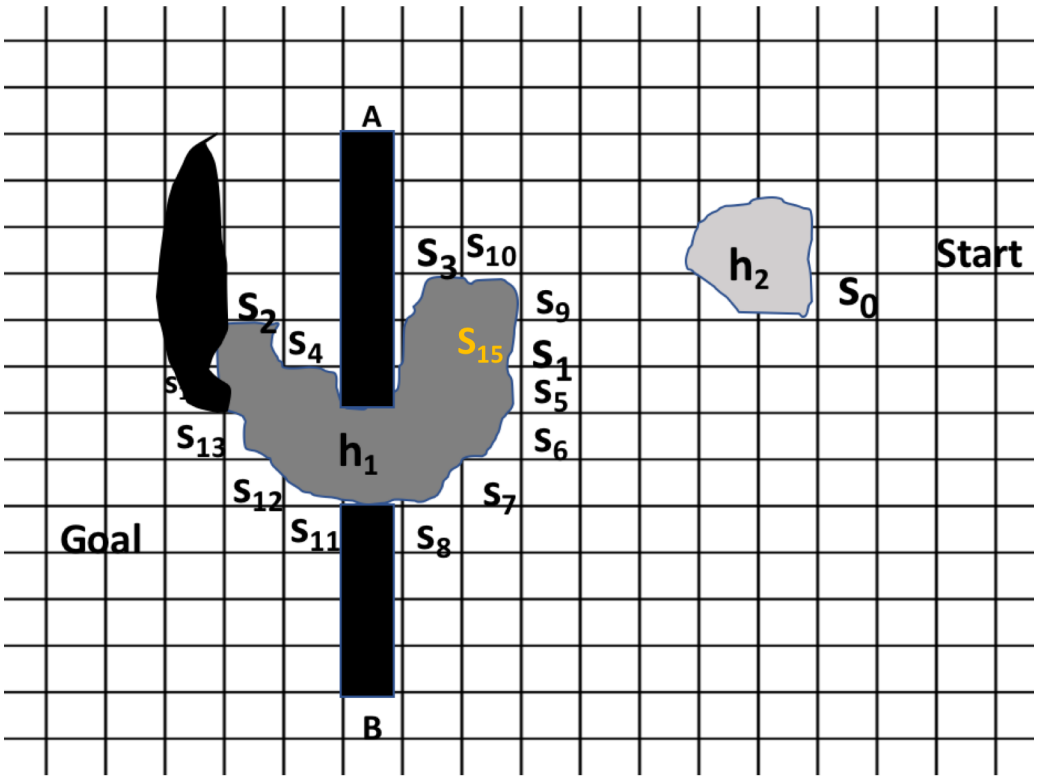


Figure 6.2: Approximate representation of the terrain in Figure 6.1 with two unknown regions (grey).

and use this throughout the paper. The terrain has grid-cells that may or may not be traversable. The robot can/cannot occupy the white/black grid-cells because they are known to be traversable/non-traversable. The robot can move between the traversable cells using eight *move* actions that move the robot in the cardinal and inter-cardinal directions by one cell. However, there are some cells ([**Ex.:** cells in light and dark grey regions]) whose traversability status is unknown at the time of planning, which affects the outcomes of some actions. The robot can execute a stochastic action *sense-and-move* at any boundary-adjacent state, that senses an unknown cell and moves the robot to it only if it is free, else stays put. The cost of a *move* action is equal to the Euclidean distance between two adjacent cells (1.4 for diagonal, 1 for others). The cost of a *sense-and-move* action follows the cost of a *move* action if the hidden cell is free, else is 2.

A1	The environment is deterministic, i.e., if the environment were fully known at the time of planning, there would be no uncertainty in the outcome of an action.
A2	The agent has a probability distribution or *belief* over the status of the unknown variables.
A3	**All states in an unknown region have the same true status.**
A4	**If an unknown state is sensed and its status is known, the status of the entire unknown region containing this state becomes known and is the same as that of the sensed state.**
A5	The true status of an unknown variable becomes known immediately upon being sensed (perfect sensing assumption).
A6	Only one unknown variable can affect the outcome of an action a taken at $s(X)$. However, the same unknown variable is allowed to affect outcome of another action taken at another state.
A7	The variables in H are independent of each other and therefore $P(H) = \prod_{i=1}^{\lvert H \rvert} P(h_i)$.

Table 6.1: Assumptions used in the optimized version of FAST-PPCP.

Problem definition and assumptions. We assume that the environment is deterministic and the uncertainty is only due to the traversability status of some cells being unknown (assumption A1 in Table 6.1). Given no uncertainty in localization or transition uncertainty, FAST-PPCP formulates a belief state as a vector of discrete variables split into two component, $X = [s(X), H(X)]$. $s(X)$ is the set of variables whose status is fully observable [**Ex.:** robot's 2D location] . The *hidden* component, $H(X)$, is a set of variables where each variable $H(X)$, known as an unknown variable, represents the traversability status of a given *unknown region*. An unknown region is a connected component such that all states in the component have the same status (assumption A3 in Table 6.1). [**Ex.:** All cells in an unknown *region* in our example domain have same traversability status due to either homogenous terrain type or even slope.]

We denote the i^{th} unknown variable in $H(X)$ by $h_i(X)$. $h_i(X) = u$ indicates that $h_i(X)$ is unknown. [**Ex.:** h_1 is an unknown variables representing the status of the dark grey region, h_2 $h_i = 0/1/u$ indicates the cell h_i is traversable/non-traversable/unknown. $X_{st} = [start; h_1 = u, h_2 = u]$ represents the start belief-state X_{st}; the values of all the unknown variables are unknown before planning starts] . We assume that the agent has a probability distribution or *belief* over the status of the unknown variables (A2 in Table 6.1).

We denote the set of actions applicable at a belief state X by $A(s(X))$. Action $a \in A(s(X))$ is applicable at any belief state Y where $S(Y) = s(X)$. However, the outcome of an action a depends on an unknown variable, and we denote the unknown variable in $H(X)$ affecting its outcome by $h^{s(X),a}$. There are either *deterministic* actions or *stochastic* sense actions. The outcome of a *deterministic* action [**Ex.:** move actions] is unaffected by any unknown variable, and so it has only one outcome because the underlying environment is deterministic. A *stochastic* sensing action [**Ex.:** sense-and-move actions] can have multiple possible outcomes depending on the true status of the unknown region being sensed. We assume that the assumptions A4 and A5 in Table 6.1 hold for a *stochastic* sense action. For a succcessor X' of X in the belief-space, note that the hidden component remains the same in X and X', i.e., $H(X) = H(X')$ for a deterministic action. For a stochastic action, $H(X)$ is the same as $H(X')$ except for $h^{s(X),a}$, which becomes known if it was unknown.

The probability distribution of transitions $P(X'|X,a)$ is the same as the probability distribution over statuses of the unknown variable $h^{s(X),a}$ that affects the outcome of executing a at X. Given that we represent the status of all cells in a given unknown region by a single unknown variable, we can make the assumption that the statuses of the unknown variables in $H(X)$ are independent of each other (A7 in Table 6.1) [**Ex.:** In our example domain, this assumption can be implemented when the unknown regions are far enough from each other such that knowing the traversability of one does not say anything about the traversability of other regions] . Given this independence assumption, X concisely represents a probability distribution over all possible states.

[**Ex.** h_1 in Figure 6.2 represents the status of the dark grey unknown region and affects the outcome of the action *sense-and-move* taken on any of the boundary-adjacent cells $s_1 - s_{14}$. Let $X = [s_1; u, u]$ be a belief state. When taken on X, *sense-and-move* produces two belief-state outcomes: $X_1 = [s_{15}; h_1 = traversable, u]$ and $X_2 = [s_1; h_1 =$

Term	Definition
Full Policy from belief state X	Tree rooted at X s.t every branch reaches a belief state X_g s.t $S(X_g) = S_g$ for a given goal S_g.
Partial Policy from X	Policy rooted at belief state X that has at least one belief state without a defined action.
Unexplored belief state	Non-preferred outcome on a partial policy from which a path has not been searched yet
Policy-devoid belief state	Non-preferred outcome on a partial policy without an action defined from it (Unexplored belief states and belief states from which policy growth failed)

Table 6.2: Definitions

$non-traversable, u]$ both with a probability of 0.5]. If X^b, X_{np} are the preferred and non-preferred outcome of (X,a), then the cost $C(X,a,X^b)$ of the edge (X,a,X^b) in the belief-space $= C(S(X),a,S(X^b))$, the cost of the corresponding edge in the deterministic space, and $C(X,a,X_{np}) = C(S(X),a,S(X_{np}))$.

Clear Preferences. We assume that every unknown variable's clearly preferred value is known beforehand. [**Ex.:** we prefer that an unknown region is traversable rather than blocked] . For any X, let $V^*(X)$ be the expected cost of executing an optimal policy (that minimizes expected cost to goal) from X. We define clear preferences [80] as follows: For any given state X and stochastic action a such that $h^{S(X),a}$ is unknown, there exists a successor state X^b such that $h^{s(X),a}(X^b) = b$ (we denote the clearly preferred value using the variable b) [**Ex.:** b = 0 representing traversable] and $X^b = \arg\min_{X \in succ(X,a)} \{c(S(X),a,s(X)) + V^*(X)\}$, where $succ(X,a)$ represents the set of successors of executing a at X.

Planning Problem. The planning problem is to compute a policy (defined in Table 6.2) from X_{st}.

6.3 Fast bounded suboptimal PPCP

Before describing an overview of FAST-PPCP's operation, we point the reader to Table 6.2 to familiarize with some definitions.

Key idea of FAST-PPCP. A FAST-PPCP search meets the following **search objective**: to compute a path that *explicitly minimizes the number of stochastic transitions* in it, while ensuring that including this path in the partial policy can result in a provably bounded suboptimal full policy π_f from X_{st}, when construction of the policy is completed. Also, it has a policy growth strategy such that the first time the search terminates, the policy is guaranteed to be bounded suboptimal which leads to significant speedup. It also has a strategy for scheduling the searches to ensure completeness.

6.3.1 Overview of FAST-PPCP operation

In order to understand how search efforts are wasted, we first briefly revisit some concepts related to FAST-PPCP using the environment in Figure 6.3 as an example.

Fast-PPCP: Search in the Deterministic Space. Starting from $X_{st} = [start, h_1 = u, h_2 = u]$ with an empty partial policy, FAST-PPCP attempts to iteratively grow a partial policy into a full bounded suboptimal policy. In order to grow a partial policy, FAST-PPCP first picks a policy-devoid belief state X_p in the current partial policy—a belief state without an action defined from it. FAST-PPCP then starts an iteration from X_p and attempts to add in the current partial policy a path from this belief state. To compute such a path, FAST-PPCP searches for a path from the deterministic component $s(X_p)$, in the underlying deterministic space instead of the exponentially larger belief space. This deterministic space is constructed assuming every unknown variable in X that is still unknown is set to its preferred value, and every known unknown variable is set to its known value. Because

of this construction, the found path p maps to a corresponding path p^{bel} in the belief space from X_p, that consists of transitions that only correspond to either deterministic actions, or stochastic actions with their clearly preferred outcomes.We refer to such a belief-space path corresponding to path p from $s(X_p)$ as the *primary* branch from X_p to goal.

Fast-PPCP: Rejection Condition for a Path. In a FAST-PPCP search from $S(X_p)$, FAST-PPCP sequentially computes candidate paths from $S(X_p)$ in increasing order of the number of stochastic transitions on them, and rejects a path if adding it to the current partial policy results in an *invalid* partial policy—one that cannot be developed into a bounded suboptimal policy no matter which unexplored belief state on it is explored in future iterations. While deciding whether to add or reject the path p during a FAST-PPCP search, the FAST-PPCP search computes $\hat{V}^p(X_p)$— the estimated expected cost of any full policy from X_p that contains the primary branch p^{bel} that maps to p. We define the estimated expected cost of a partial policy from a belief state in the same way as the conventional definition of the expected cost of a full policy, except that the V-value of a policy-devoid belief state X in this partial policy is set to an estimate $\hat{V}$-value $\hat{V}(X)$ maintained by FAST-PPCP. If the true minimum V-value of X, $V^*(X)$, was known for each policy-devoid belief state, then the estimated expected cost would be the true minimum expected cost. But $V^*(X)$ is not known to FAST-PPCP for all X in the belief space at the start of planning; however FAST-PPCP maintains an underestimate $\hat{V}^*(X)$ for all X. $\hat{V}^*(X)$ is initially set to a known underestimate before FAST-PPCP begins. Since $\hat{V}^*(X_2)$ is an underestimate of $V^*(X_2)$, the estimated expected cost $\hat{V}^{p_2}(X_p)$ is a lower bound on the minimum expected cost of any full policy from X_p containing p^{bel}. FAST-PPCP then computes the estimated expected cost of the updated partial policy if p^{bel} is added to the current partial policy from X_p, with $\hat{V}$-value of X_p as $\hat{V}^p(X_p)$. Since $\hat{V}^p(X_p)$ is a lower bound, the estimated expected cost of the updated partial policy is also a lower bound on the minimum expected cost of a

full policy containing this updated partial policy. FAST-PPCP uses this lower bound to reject a path: if this lower bound exceeds the suboptimality bound, then no full policy containing this updated partial policy can be developed thereafter whose V-value is within the suboptimality bound. Hence, p is rejected.

Fast-PPCP: Updating Underestimates of V^* in the case of `invalid-solution-found`. It is possible during the FAST-PPCP search that it explores *every* path from $S(X_p)$ as candidate and rejects all of them because adding any of them to the current partial policy results in an invalid updated partial policy. We refer to this case as `invalid-solution-found` from X_p given the current partial policy. In this case, since every possible path has been computed and its $\hat{V}^p(X_p)$-value is known, FAST-PPCP computes the minimum value of $\hat{V}^p(X_p)$ over all possible paths; this minimum is an underestimate of $V^*(X_p)$. FAST-PPCP then updates the initial underestimate of $V^*(X_p)$ to this minimum $\hat{V}^p(X_p)$. Note that FAST-PPCP in its current form updates underestimate of V^* for a policy-devoid belief state in the current partial policy only in the `invalid-solution-found` case.

6.4 Optimizing FAST-PPCP

In this section, we first present the motivation for optimizing FAST-PPCP using a running example and then present the optimization.

6.4.1 Motivation for optimization in FAST-PPCP:

We now show an example in Figure 6.3 where FAST-PPCP wastes computational efforts. At some FAST-PPCP iteration, let's say it explores the belief state $X_1 = [s_1, h_1 = non-traversable, h_2 = non-traversable]$ in the current partial policy. Consequently, FAST-PPCP concludes `invalid-solution-found` from X_1. Then, FAST-PPCP updates the

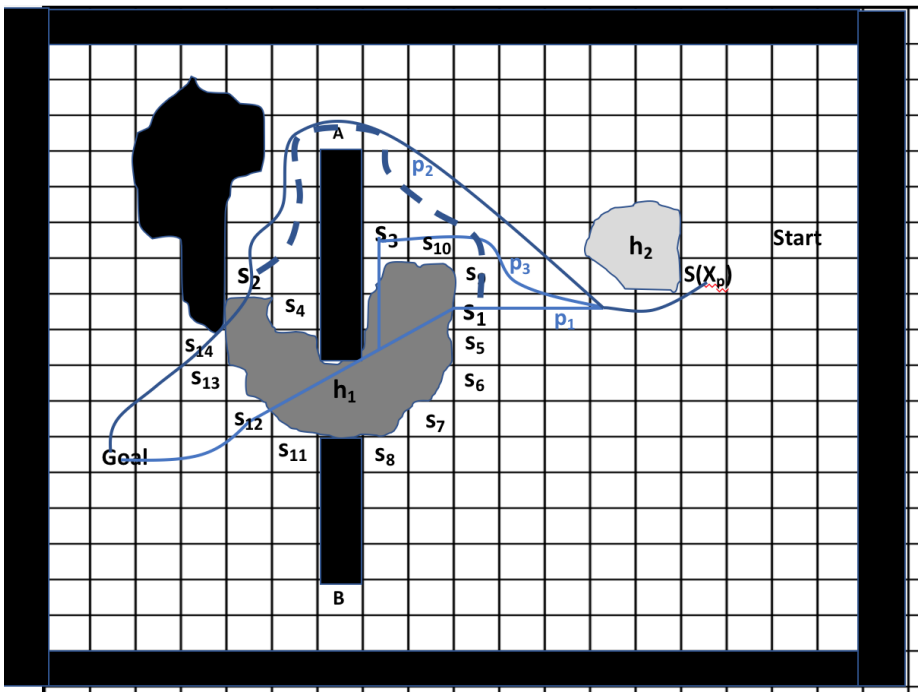


Figure 6.3: Running example to illustrate wasted computational efforts in FAST-PPCP. underestimate $\hat{V}^*(X_1)$ of $V^*(X_1)$ as explained in **Fast-PPCP: Updating Underestimates of V^* in the case of `invalid-solution-found`.**.

At a later iteration, let us assume $X_p = [S(X_p), h_1 = u, h_2 = non-traversable]$ is the belief state from which FAST-PPCP starts an iteration. Consider the path p_1 that has a `sense-and-move` action executed at s_1 that produces the non-preferred outcome X_1 in the belief space. The underestimate of $V^*(X_1)$ was updated when X_1 was previously explored as mentioned before. Let us say this updated underestimate of $V^*(X_1)$ is high enough such that adding p_1 makes the lower bound on the expected cost of partial policy exceed suboptimality bound, and as a result p_1 gets rejected. Consider any other path from $S(X_p)$ which also has a `sense-and-move` action to sense h_1 and has only this stochastic action—in Figure 6.3 this could be paths p_2 or p_3, or any other path from $S(X_p)$ that has only one `sense-and-move` action executed for sensing h_1 at the states $s_2....s_{14}$ that are neighbors of the unknown region represented by h_1. Note that FAST-PPCP sequentially computes

candidate paths from $S(Xp)$ in increasing order of the number of stochastic transitions on them. Given that these paths have the same number of stochastic actions as p_1, that is one, FAST-PPCP after rejecting p_1 would compute one of these paths as the next candidate. For the sake of example, let the candidate path be p_2.

Path p_2 has a `sense-and-move` action at state s_2 that senses the unknown variable h_1 which produces the non-preferred outcome $X_2 = [s_2, h_1 = non-traversable, h_2 = non-traversable]$ in the belief space. If X_2 is unexplored, the underestimate $\hat{V}^*(X_2)$ of $V^*(X_2)$ used while computing the lower bound of the expected cost of the partial polic if path p_2 is added, as mentioned in **Fast-PPCP: Rejection Condition for a Path**. This initial underestimate is likely to be much lower than the true minimum expected cost $V^*(X_2)$ of any policy from X_2. Because of this low underestimate, FAST-PPCP would estimate the lower bound of the expected cost of the partial policy if path p_2 is added as really low, which would not exceed the suboptimality bound. Because of this, FAST-PPCP would add p_2 in the current partial policy.

However, in reality the true $V^*(X_2)$ might be high, because of which no path from X_2 can be added to the current partial policy with its lower bound on expected cost within the suboptimality bound. This would mean that no path exists from X_2 adding which to the updated partial policy can give a full bounded suboptimal policy, and thus p_2 should have been rejected. Only at a future iteration will FAST-PPCP start from X_2 because it is unexplored, and then actually explore all paths from X_2 to conclude `invalid-solution-found`. Exploring all paths from X_2 itself can be quite computationally expensive especially for large environments.

After concluding `no-sol-found` from X_2, FAST-PPCP at a further later iteration might start again from X_p: this time, both p_1, p_2 would be rejected because their respective underestimates have been updated. However, any path p_i from $S(X_p)$ that only one

`sense-and-move` action executed at a neighboring state s_i of h_1 that is not s_1 or s_2—for example path p_3—with the unexplored non-preferred outcome X_i in the belief space—might get added and later removed, because of the same reason by which p_2 got added previously, further increasing computation.

More generally, in the worst case FAST-PPCP might end up adding all paths having a `sense-and-move` action executed at the neighboring states s_2.....s_{14} of h_1, and rejecting them in future iterations only after spending computational efforts to conclude `invalid-solution-found` from the non-preferred outcome of sensing h_1. If the unknown region represented by h_1 is large, then the number of such paths can be high. Thus, the overall computational effort spent to reject them could be quite high.

What if we could use the updated underestimate $\hat{V}^*(X_1)$ to update the underestimate $\hat{V}^*(X_i)$ from the initial really low underestimate to a higehr value, in such a way that FAST-PPCP preemptively rejects p_i in the same iteration when it rejects p_1, and thus never adds p_i in the current partial policy? This would save future iterations that would be spent in FAST-PPCP finding that X_i is policy devoid, and then actually exploring policies from X_i only to discover that no full bounded suboptimal policies exist from X_i given the current partial policy and then later rejecting p_i.

We propose an optimization to address the following two questions.

- Q1: With paths p_1, p_i and belief states X_1, X_i as defined earlier, and the true minimum expected cost $V^*(X_i)$ being high such that p_i should be rejected given the current partial policy, can we update the underestimate $\hat{V}^*(X_i)$ so that p_i gets rejected in the same iteration when p_1 is rejected?
- Q2: If this update requires any computation, is it significantly faster than that required in starting a FAST-PPCP iteration and exploring all paths from X_i to conclude `invalid-solution-found`.

6.4.2 Optimization

Note that X_1 and X_i have the same hidden component: $H(X_1)$ is the same as $H(X_i)$. This is because:

(1) p_1 and p_i are paths that map to their corresponding paths in the belief space from $X(p)$. Thus, unknown variables that are known in X_p are also known in X_1 and X_i. Also,

(2) both paths have a `sense-and-move` action that senses the same unknown variable h_1. Since X_1 and X_i are the outcomes of sensing h_1 when it is non-traversable, h_1 in both X_1 and X_i is non-traversable. Thus, the set of known (and unknown) unknown variables in X_1 is the same as that in X_i: $H(X_1) = H(X_i)$.

If $X_1 = [s_1, H(X_1)]$ and $X_i = [s_i, H(X_i)]$ have the same hidden component, then any feasible path from X_1 to X_i is purely deterministic, .i.e, consisting only of deterministic actions. This is because deterministic actions do not make any unknown variable known, and thus are the only actions that can be used to go from X_1 to X_i. Note that a deterministic path uniquely maps to a path from s_1 to s_i in the 2D grid constructed assuming whichever cells are known in $H(X_1)$ (equivalently $H(X_i)$) are set to their known value, as mentioned in **Fast-PPCP : Search in the Deterministic Space**. The dotted path from s_1 to s_2 in Figure 6.3 is an example of a feasible deterministic path in the 2D grid where cells in h_1 and h_2 are non-traversable because they are non-traversable in $H(X_1)$ (or $H(X_i)$). The cost of such a path from X_1 to X_i in the belief space is equal to the cost of the mapped path from s_1 to s_i. This is because (1) both paths have same actions, and (2) the cost of a deterministic transition (X, a, Y) in the belief space—where (X, a, Y) represents the transition that corresponds to executing deterministic action a at belief state X to produce outcome Y—is equal to the cost of the mapped transition $(s(X), a, S(Y))$ in the 2D grid. Given the cost of a deterministic path p^{bel} from X_1 to X_i is the same as the cost of its mapped path p from s_1 to s_i, our high-level intuition is that it is possible to derive a relationship

between $V^*(X_i)$ and $V^*(X_1)$—which are minimum expected costs in the belief space—using the cost of p, and subsequently between the underestimates $\hat{V}^*(X_1)$ and $\hat{V}^*(X_i)$. The cost of p can be computed by searching in the 2D grid space instead of the much larger belief space. Thus, if $\hat{V}^*(X_1)$ is updated and the cost of this path from s_1 to s_i is known, then $\hat{V}^*(X_i)$ can also be updated. This updated $\hat{V}^*(X_i)$ can make FAST-PPCP reject p_i without having to explore all paths from X_i to conclude `invalid-solution-found`. If the computational expense of computing the cost of the path from s_1 to s_i is significantly lower than that spent in starting a FAST-PPCP iteration and exploring all paths from X_i, then FAST-PPCP can achieve speedup.

We use this intuition to propose the following optimization: $c(s_1, s_2)$ can be computed offline using a Djikstra search from s_1 to s_2, thus this cost $c(s_1, s_2)$ should be used to update $V(b_2)$. Since costs in the 2D grid and belief space are same for deterministic actions, the following inequality holds:

$$\begin{aligned} & V(b_1) \leq V^*(b_1) \leq c(s_1, s_2) + V^*(b_2) \\ \textit{Or}, \quad & V(b_1) - c(s_1, s_2) \leq V^*(b_2) \end{aligned} \tag{6.1}$$

Thus, $V(b_1) - c(s_1, s_2)$ is an underestimate of $V^*(b_2)$. If this underestimate is higher than the previous underestimate of $V^*(b_2)$, then we update the previous underestimate with this new value. We perform this update for every X with $S(X)$ being a boundary-adjacent state whenever `invalid-solution-found` happens for a state X.

6.5 Simulation experiments

All experiments were run on a machine with an Intel® Core™ i7-5600U CPU @ 2.60GHz × 4, and 15 GB RAM. All algorithms were implemented in C++11, compiled using the

same optimization flag -03.

We evaluate optimized FAST-PPCP in simulation in the domain of robot navigation in partially known off-road terrain. We run experiments on a real-world map of the Fort IndianaTown Gap, PA (Figure 6.4) with a grid-size of 698×639 with 20 unknown regions of radius 100 cells. A `sense-and-move` action is applicable only at the boundary (blue in Figure 6.5) of an unknown region.

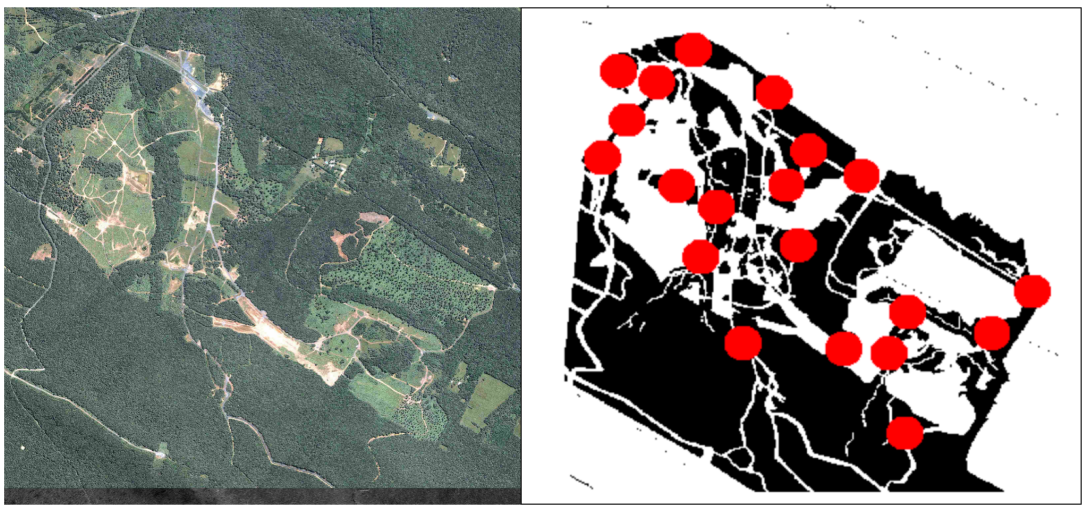


Figure 6.4: Real map of Fort IndianaTown Gap, PA (left). Unknown regions are shown in red (right).

We compare the optimized version of FAST-PPCP with an optimization introduced in the original PPCP paper. We compare the optimized version of FAST-PPCP with *optimized* PPCP which implements a similar optimization used in the original PPCP algorithm.

6.5.1 Results

Results in Tables 6.3 are averaged over 20 planning instances. In the map in Figure 6.4, optimized FAST-PPCP with $\alpha = 1.3$ computes a policy in ~ 5 iterations on average. The expected costs in the case of optimized FAST-PPCP are 1.07 times higher than the

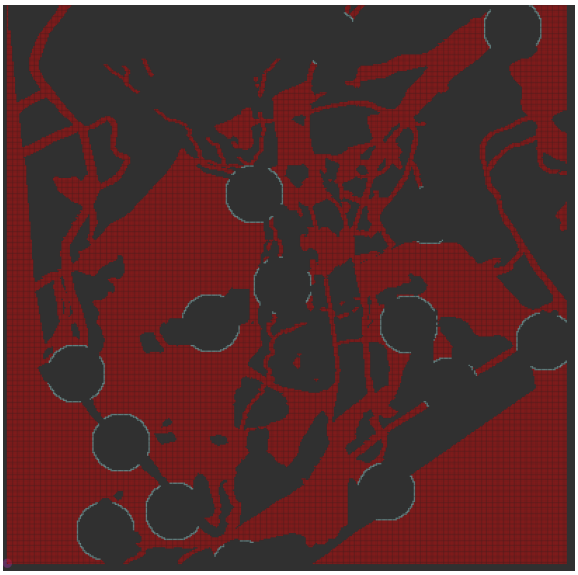


Figure 6.5: Visualization of a part of the map shown in Figure 6.4. Red regions are free, black cells are the regions whose traversability is unknown. Their boundaries are visualized in blue.

Algorithm	**Time (s)**	**Expected cost**	**Iterations**
unk. = 20; α = 1.3 Optimised FAST-PPCP	**194.77 ± 170.9**	**2763.7 ± 388.80**	**4.58 ± 2.81**
Optimised PPCP	417.75 ± 173.05	2576.6 ± 374.2	231 ± 101.49

Table 6.3: Performance comparison of optimized FAST-PPCP with optimized PPCP.

optimal expected cost computed by optimized PPCP. Optimized FAST-PPCP reduces the planning times by a factor of 2.07 as compared with optimized PPCP. The number of optimized FAST-PPCP iterations is much lower (51.3 times) than the number of iterations of optimized PPCP. However, planning time reduces only by a factor of 2.07 because the time taken per iteration in optimized FAST-PPCP is higher than that of optimized PPCP. An interesting future research direction would be to reduce the time taken per iteration in optimized FAST-PPCP, and consequently in FAST-PPCP).

We also compare with the original FAST-PPCP algorithm without the optimization introduced in this chapter. FAST-PPCP without optimization is unable to plan within a

timeout of 15 minutes for $\frac{4}{5}$ of the instances. For the remaining $\frac{1}{5}$ of the instances, as expected, it performs the same as optimized Fast-PPCP in all three metrics. This is because we do not encounter the Policy Correction mode in any of the instances which trigger the optimization. When the `invalid-solution-found` case does not arise, which occurs during Policy Correction, optimized FAST-PPCP is essentially the same as the original FAST-PPCP.

6.6 Robot experiments

In this section, we illustrate an example run of the optimized version of FAST-PPCP on a physical robot—Husarion ROSbot 2.0 PRO—in an indoor space with a large carpet whose traversability is unknown at the time of planning.

6.6.1 Husarion ROSbot 2.0 PRO

Overview. ROSbot (Figure 6.6) is a ROS powered 4x4 drive autonomous mobile robot platform equipped with LIDAR, RGB-D camera, IMU, encoders, and distance sensors. It is available in three version: "2" and "2 PRO" and "2R". In this thesis we use the 2 PRO version. ROSbot is an affordable robot platform for rapid development of autonomous robots. It can be a base for custom service robots, inspection robots and robots working in swarms. We chose ROSbot 2.0 PRO because of the following reasons:

- It runs an Ubuntu-based OS, customized to use ROS. This makes it easy to transfer implementations from simulation to real world experiments.
- It is affordable, lightweight and portable.
- There is an already available Gazebo simulation model for ROSbot which gives the flexibility to also test its performance in the Gazebo environment.

Figure 6.6: Husarion ROSbot 2.0 PRO.

Onboard sensors and components. The ROSbot 2.0 PRO is a 4-wheels mobile platform containing DC motors with encoders and an aluminum frame. It has the following components:

- an Orbbec Astra RGBD camera with RGB image of size 640×480 and depth image of size 640×480.
- an IMU sensor: a powerful 9-Axis MPU 9250 inertial sensor.
- a real-time CORE2 controller based on STM32F407 microcontroller.
- RPLIDAR A3 laser scanner, 360 degree and up to 25 meters range.

Software. Software for ROSbot can be divided into 2 parts:

- A low-level firmware that works on the real-time controller (CORE2).
- An OS based on Ubuntu 18.04 which runs on the SBC (UpBoard with 4 GB RAM, Quad-Core Intel Atom Z8350 1,92 GHz as CPU, a Intel® HD 400 Graphics as a GPU and 32GB eMMC.). The SBC runs on Ubuntu-based OS, customized to use ROS. The microSD card or MMC memory with OS is included with each ROSbot. The OS has been modified to make the file system insensitive to sudden power cuts.

6.6.2 Experimental setup

Consider an example environment in Figure 6.7. The rosbot has to navigate from the *start*

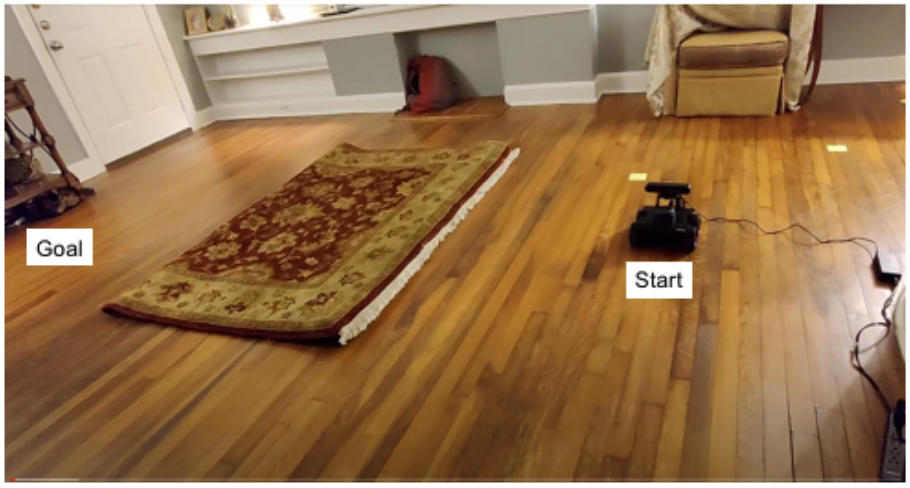


Figure 6.7: Experimental setup showing the carpet whose traversability is unknown at the time of planning, the start and the goal location.

location to the *goal* in the room environment represented by a 2D grid. Our experimental arena has dimensions 3 meters $\times$ 3 meters. With a map resolution of 0.1, the grid size for our experimental run is 30 $\times$ 30. The environment has regions whose traversability is known: the wooden floor is known to be traversable, and walls and furniture are known to be obstacles. The robot can move between the traversable cells using eight *move* actions that move the robot in the cardinal and inter-cardinal directions by one cell. However, there

is a region whose traversability status is unknown at the time of planning, which affects the outcomes of sensing actions. This region is the carpet, as shown in Figure 6.7 whose friction properties, thickness, and other characteristics are unknown to the robot. Because of the unknown carpet properties, it is unknown at the time of planning if the robot can successfully climb on top of the carpet and follow a path over the carpet. The robot can execute a stochastic *sense-and-move* action at any state adjacent to the boundary of the carpet. The cost of a *move* action is equal to the Euclidean distance between two adjacent cells (1.4 for diagonal, 1 for others). The cost of a *sense-and-move* action follows the cost of a *move* action if the unknown region is free, else the cost is 2. We assume that the robot has a static map of the environment where the free cells and obstacles cells are known. The location and 2D dimensions of the carpet are known in the map, and only the traversability status of the carpet is unknown.

6.6.3 Implementation details

In this section, we describe the implementation details of the localization, planning, and execution modules implemented to run FAST-PPCP on ROSbot. We use C++11 for all implementions and compile using the same optimization flag -03.

Localization. We use localization using sensor fusion via Extended Kalman Filter. The Extended Kalman Filter based odometry system fuses odometry sensor inputs from various sources into a locally accurate estimate of the robot's pose and velocity based on its motion. For the size of our experimental arena, localization using sensor fusion via Extended Kalman Filter produces reasonably accurate state estimates. The odometry sources could be IMU, LIDAR, RADAR, VIO, and wheel encoders. IMUs drift over time, while wheel encoders drift over distance traveled. Thus, they are often used together to counter each other's negative characteristics. In our experiments, the onboard IMU and wheel encoders

are used to localize the robot. The Robot Localization ROS package is a useful package to fuse information from arbitrary number of sensors using the Extended Kalman Filter (EKF) or the Unscented Kalman Filter (UKF).

Planner implementation. We implement FAST-PPCP as a global planner with the ROS navigation stack. The ROS navigation stack links together a global planner and a local controller to accomplish a global navigation task. It can support any global planner and any controller. Since ROSbot runs Ubuntu customized with ROS, we do not have to modify the implementation of FAST-PPCP used in the Gazebo simulation environment.

Implementing the perfect sensing assumption in this domain. We implement the perfect sensing assumption in the execution of a sensing action as follows. The sensing action is valid only at a periphery of 0.1 meters from the carpet boundary, following our assumption that a sensing action can be executed only at a state adjacent to the carpet boundary. If the robot position does not change beyond a very small threshold δ of 2 centimeters for K execution steps, and the robot yaw does not change by more than $d_\theta = 1$ degree, then the sensing action infers that the robot is stuck at the boundary of the carpet and is unable to climb on it. Therefore, the carpet is deemed to be non-traversable. Here, K is a tunable parameter. Note that the implementation can support more complex sensing actions.

Implementation of the Policy Executor module. We have introduced and implemented a new module in the ROS navigation stack: the *Policy Executor* module. The *Policy Executor* module provides a provision to execute a policy tree in the ROS navigation stack. This module connects the planner with the controller. The basic idea of our Policy Executor module is as follows. Once FAST-PPCP computes a full policy, the Policy Executor starts executing the primary branch from the start state in the full policy, using the local trajectory planners or controllers. Once the robot is in a cell adjacent to the unknown region r, the Policy Executor begins the execution of the sensing action, as described in the previous

paragraph. If the sensing action senses the region r as non-traversable, then the Policy Executor implements a *failure recovery behavior* of moving back for N steps and then latching onto the nearest point on the detour branch. This detour branch is the branch in the full policy from the belief state that corresponds to the non-preferred outcome of sensing the unknown region r. In our example scenario, a value of 8 for N produces smoooth controller trajectories. N is also a tunable parameter. The Policy Executor then continues executing the detour branch unless the robot either reaches the goal or is at a state adjacent to an unknown region. In the case when the sensing action senses the region r as traversable, then the Policy Executor continues the execution of the current branch.

Controller implementation We have made modifications in the implementation of the local trajectory planner (controller) provided in the ROS navigation stack. The controller package in the ROS navigation stack provides implementations of the Trajectory Rollout and Dynamic Window approaches to local robot navigation on a plane. Given a plan to follow, the controller produces velocity commands (d_x,d_y,d_θ) to send to a mobile base. The controller package supports both holonomic and non-holonomic robots, and any robot footprint that can be represented as a convex polygon or circle.

The basic idea of both the Trajectory Rollout and Dynamic Window Approach (DWA) algorithms is as follows:

1. Discretely sample in the robot's control space (d_x, d_y, d_θ)
2. For each sampled velocity, perform forward simulation from the robot's current state to predict what would happen if the sampled velocity were applied for some short period of time.
3. Score each trajectory resulting from the forward simulation, using a metric that incorporates characteristics such as: proximity to obstacles, proximity to the goal, proximity to the global path, and speed.

4. Pick the highest-scoring trajectory and send the associated velocity to the mobile base.
5. Rinse and repeat.

DWA differs from Trajectory Rollout in the way the control space of the robot is sampled. Trajectory Rollout samples from the set of achievable velocities over the entire forward simulation period given the acceleration limits of the robot, while DWA samples from the set of achievable velocities for just one simulation step given the acceleration limits of the robot. This means that DWA is a more efficient algorithm because it samples a smaller space, but may be outperformed by Trajectory Rollout for robots with low acceleration limits because DWA does not forward simulate constant accelerations. We conducted our example run in the Gazebo simulation environment as well as on the real robot. In our experiments, the Trajectory Rollout and the Dynamic Window controllers produced similar results. We finally used the Trajectory Rollout controller.

Tackling rotate-in-place behavior produced by the controller. In our experiments we observed that the controller in some cases causes the robot to start rotating-in-place after the controller latches on to the detour path. We identified that this issue is caused when the slope of the detour path is numeric zero at some places. We solve this issue by nominally perturbing the slope, i.e., setting the slope to a nominal numeric non-zero value of 0.03 instead of 0.

6.6.4 Results

Here is a video of an example run of the ROSbot 2.0 PRO navigating using the optimized version of FAST-PPCP as the planner. Figure 6.7 shows the snapshots of the video.

The full policy computed by FAST-PPCP is visualized in Figure 6.8. The ROSbot starts by following the path to goal assuming that the carpet is traversable (blue path in

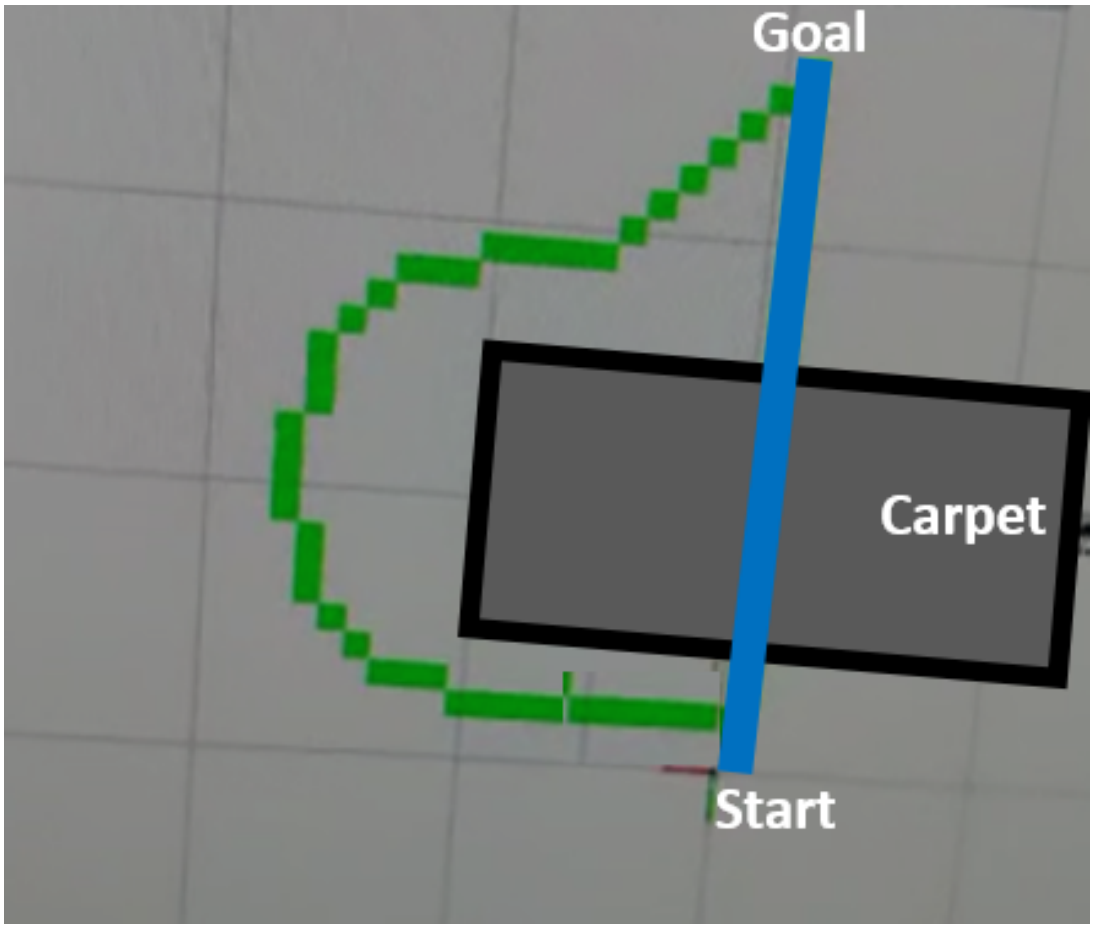
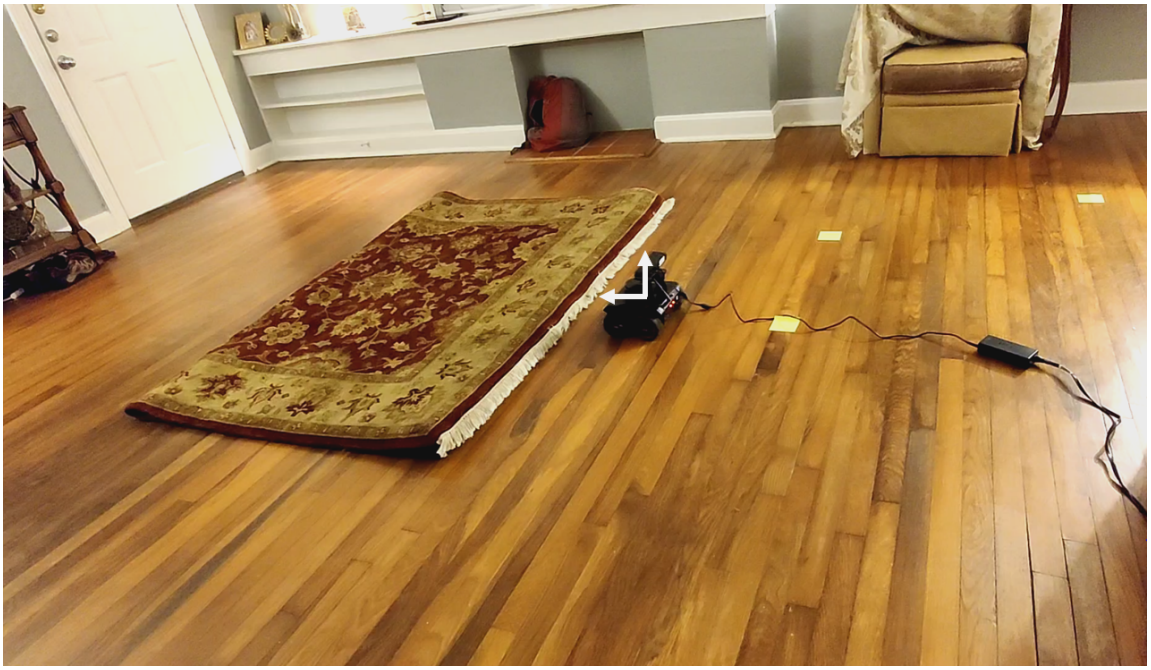


Figure 6.8: Full policy computed by FAST-PPCP for our experimental setup.

Figure 6.8). Upon reaching the boundary of the carpet, it executes the sensing action that now starts monitoring the robot position and orientation. When the robot tries to execute a forward motion to move onto the carpet, it is unable to do so and starts rotating about its pitch axis. When the pitch angle goes beyond $\sim$ 20 degrees while the robot position does not change beyond a very small threshold of 2 centimeters for 4 execution steps, and the robot yaw does not change by more than 1 degree, then the sensing action deems the region to be non-traversable. ROSbot then starts executing the failure recovery controller that moves back and then latches onto the nearest point on the detour branch (green path in Figure 6.8). It then sticks to this detour branch all the way until it reaches the goal. We

(a)

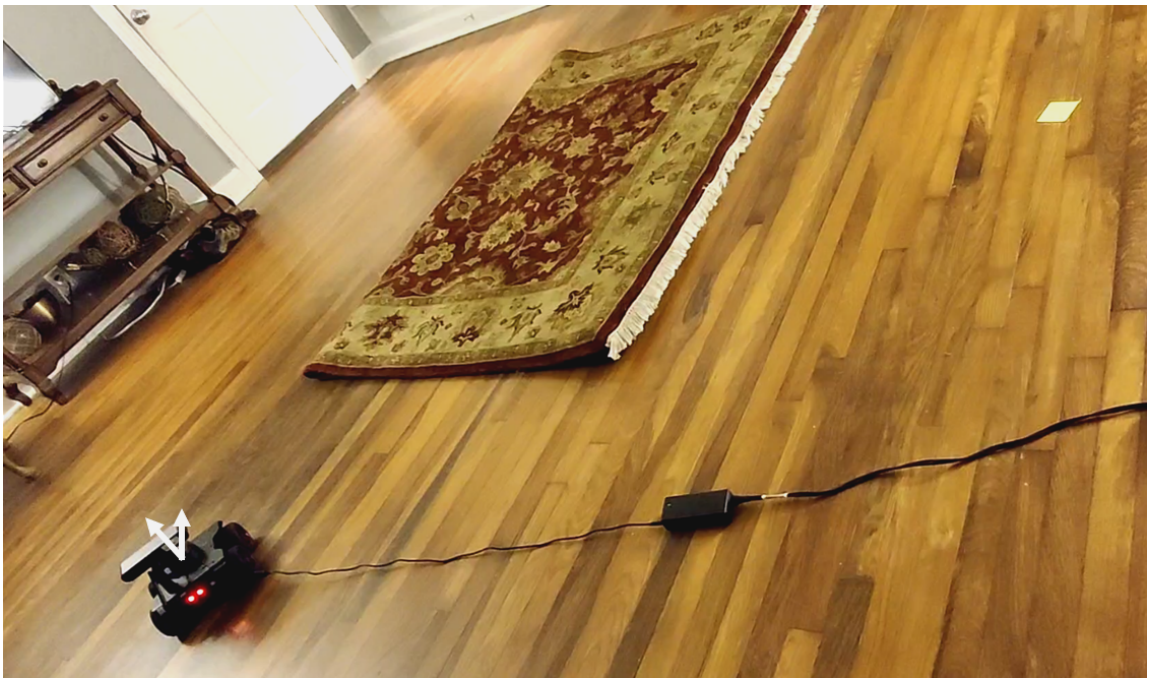
(b)



(c)

(d)

(e)



(f)

Figure 6.7: Snapshots of an example run of FAST-PPCP in our experimental setup. (a) to (f) are in the order of increasing timestamps during the policy execution.

have used a goal tolerance of 0.4 meters—execution stops when the robot position is within 0.4 meters of the goal location.

6.7 Conclusions, Discussion, and Lessons Learned

In this work, we present an optimized version of FAST-PPCP—a novel approach to probabilistic planning in domains with clear preferences over missing information. We achieve substantial decrease in run-time while incurring little loss in solution quality compared to the original version of FAST-PPCP as well as PPCP used in domains with clear preferences over missing information.

One limitation of optimized FAST-PPCP is that *it may be possible* to construct worst-case scenarios in which optimized FAST-PPCP performs computationally the same as the original version of FAST-PPCP. It is also possible to construct worst-case scenarios in which optimized FAST-PPCP performs computationally worse than PPCP. Future work could involve aiming to modify optimized FAST-PPCP. Another limitation of FAST-PPCP is the perfect sensing assumption, which narrows the scope of the problem more than typical POMDP planning without this assumption. A future research direction for relaxing this assumption is discussed in chapter 8.

The search time complexity of a FAST-PPCP search is a function of the number of sense locations per unknown region. The number of sense locations per unknown region can be pretty high for large unknown regions like the ones considered in this chapter. This increases the branching factor and the belief tree's size, resulting in a high search time.

We assume in FAST-PPCP that the traversability status of a large unknown region is represented by one unknown variable. Thus, if any part of the region is sensed, its traversability becomes known. This is typically true for homogeneous regions. However, in reality, traversability might differ within an unknown region. In that case, suppose the

agent is sensing the traversability of a region from one part of the region. In that case, the agent should decide whether or not to sense the region from more parts before inferring its traversability.

Chapter 7

Search-based Planning with Learned Behaviors for Navigation among Pedestrians

In addition to the primary contributions of this thesis as enumerated in chapter 1, we have also investigated the application of PPCP in the context of navigation among pedestrians.

Agent control among pedestrians is often approached in one of the three following ways: using predefined behaviors for agent navigation, learning navigation behaviors from data, or search-based planning on a graph where each edge is a feasible action chosen from a set of predefined actions. While the first approach often produces natural looking motions and the second learns and utilizes complex interactions with pedestrians, both lack global reasoning about how to sequence these behaviors to achieve the overall goal. The third approach, namely search-based planning, does incorporate global reasoning but relies on predefined actions that do not involve any interactions with pedestrians or assume predefined interactions that cannot model complex interactions. This is a significant

drawback since many situations such as going through a doorway blocked by other people require complex interactions in order to avoid highly suboptimal behaviors or not being able to get to the goal at all. To this end, we propose a search-based planning framework that constructs and searches a graph wherein each edge can be either a predefined action or a learned behavior. We further extend it to deal with the uncertainty arising from introducing learned behaviors. We present the algorithm, go over its theoretical analysis, and present experimental results.

7.1 Motivation

One goal of autonomous agents is to be able to navigate among pedestrians in places such as airports, shopping malls, etc. In such places, in addition to motion planning, the agent needs to seamlessly interact with pedestrians. For example, the agent needs to understand non-verbal cues presented by the nearby pedestrians—when to let people through, when to go through when someone clears the way, and how to move in a way that does not present a danger to the nearby people. In order to interact with pedestrians, the agent needs to model these interactions. While there exists hard-coded models of interaction—such as the Social Forces Model [Helbing and Molnar, 1995], [Ferrer et al., 2013], potential field based methods [Rimon and Koditschek, 1992] and fluid flow simulations [Helbing et al., 2005],—it is infeasible to have accurate hard-coded models of complex behaviors such as how to move in order to encourage pedestrians to clear a passage that they are blocking. In such cases, learning a model of interaction from pedestrian trajectories data can provide more generalization than hard-coded models. While there exists a body of literature that performs agent control in this domain by solely predicting trajectories using a learnt model, this approach typically requires a large amount of training data to generalize and also lacks global reasoning about how to sequence learned behaviors to achieve the overall goal.

For agent control, the advantage of search based planning (compared to control using hard-coded or learnt models) is that it incorporates global reasoning about sequencing actions to achieve the overall goal. Previous work has focused on search-based planning using predefined actions. We hypothesize that for agent control among pedestrians via search-based planning, solution quality can be improved if, in each step of planning, the agent is given access to learnt models of interaction as additional actions along with predefined actions.

We now summarize our contributions in this domain.

- We train an ensemble of LSTM-based models that produce the behavior of `barge-in`—a learnt behavior that aims to controls the robot to signal a group of pedestrians to move away from the exit of a passage they have been blocking.

We observe that the predicted outcome of learned models is typically a distribution over the agent's states. Therefore, we need to address the problem of incorporating this uncertainty into the search-based planning framework.

- We have addressed the problem of incorporating this uncertainty into the search-based planning framework: we have shown how to construct a graph with learned models as actions in addition to predefined actions. To construct a graph, we have (1) come up with a state representation to represent the uncertainty in outcomes, and (2) given this state representation, we have developed a successor generation method such that the final solution can be computed in real-time for this domain while adequately representing the uncertainty in successors.

We also observe that the learned model may fail to be executed when the agent encounters *novel* scenarios—those which are very different from the ones used for training this model.

- We have come up with an approach to estimate of the probability of failure in

execution of a learnt model.

- Note that a successful execution of a learned model is clearly preferred over a failed execution. Given this preferences, and our estimate of the probability of failure, we have shown that the planning problem in this domain can be formulated as finding a policy in a CP-PS belief MDP. We have investigated the use of PPCP to solve the CP-PS belief MDP in this context.

We now explain the details of this work.

7.2 Background: Planning in state-lattice with predefined behaviors (SLB)

The agent is in an environment that has moving pedestrians and static obstacles. We represent position of the agent, pedestrians, and obstacles by projecting their centers on a x-y plane. We assume the agent can perfectly sense its position and that of pedestrians and obstacles. The agent has a prediction model that for each pedestrian generates a time-parameterized path, an ordered set $\{X_p | X_p = (x_p, y_p, t)\}$ where (x_p, y_p, t) denotes position of a pedestrian p at time t. There is a goal position in the environment, reachable within T time-steps. The agent has to navigate from a given start position to the goal position within T time-steps.

7.2.1 SLB formulation

Our formulation builds upon the existing state-lattice based planning with predefined controllers/behaviors [Butzke et. al 2014], which we refer to as SLB. In motion planning, a state-lattice is a graph $G = \{S, E_a\}$ consisting of a set of states S and edges E_a which are feasible actions generated by taking the kino-dynamic constraints of the agent into

account, often called motion primitives. In our case, a state $s^t \in S$ is a tuple representing the (x,y) position of the agent at time-step t, i.e., $s^t = (x^t, y^t, t)$. We refer to actions generating E_a as predefined actions. [Butzke et. al 2014] introduced set of edges E_b in G formed by executing predefined behaviors to get the graph $G_{slb} = \{S, E_{slb}\}$ where $E_{slb} = E_a \cup E_b$. Unlike edges in E_a, an edge in E_b is computed by forward-simulating the execution of the behavior till a stopping condition is satisfied. Not all behaviors are available at all states. For a state $s \in S$, let $\mathscr{B}_p(s)$ be the set of available behaviors. For example, let *followWall(w)* and *followPedestrian(p)* be two applicable behaviors available to the agent at s. *followWall(w)* and *followPedestrian(p)* forward-simulate moving parallel to a wall w and tailgating pedestrian p respectively. $\mathscr{B}_p = \{$*followWall(w)*,*followPedestrian(p)*$\}$. For each $b_p \in \mathscr{B}_p$, there is a set $\mathscr{T}(b_p)$ of stopping conditions. In our example, for $b_p =$ *followWall(w)*, $\mathscr{T}(b_p) = \{$`endOfWall`,`nSteps`$\}$ is the set of two applicable stopping conditions to forward-simulate the execution of *followWall*(w) (1) till end of wall w is observed, or (2) after simulating `n` steps respectively. We define $E_b = \{(b_p, \tau)\} \forall b_p \in \mathscr{B}_p, \tau \in \mathscr{T}$.

7.3 Our approach: Introducing learned behaviors in state-lattice based planning

With `SLB`, a motion planning problem is converted to a graph search problem and a path can be found on G_{slb} with a heuristic search planner like A*. Our insight is that in addition to predefined behaviors, successful navigation among pedestrians requires complex behaviors that may be difficult to predefine and should be learned from data. We introduce State-Lattice with Predefined and learned Behaviors (`SLB-L`), a novel framework for planning under uncertainty with learned behaviors as edges. To plan using `SLB-L`, we first compute

an additional set of edges E_{lrn} formed by learned behaviors, to get a modified edge-set $E_{slb-l} = \{E_{slb} \cup E_{lrn}\}$. We then modify the deterministic SLB formulation to incorporate uncertainty due to execution of learned behavior edges. We extend SLB-L to introduce planning with SLB-CSL (State-Lattice with Predefined Behaviors and Confidence of Success of Learned Behaviors), a framework to incorporate uncertainty in the outcomes of a learned behavior due to chances of failure in its execution. This section gives details of our approach.

7.3.1 Learning behavior models

We define an agent behavior model as a function that takes a feature representation of the environment as an input from time $t-h$ through t for some $h > 0$, and predicts a distribution of the agent's position at $t' = t + \Delta t, \Delta t > 0$. We represent a learned behavior by an ensemble $EN = \{M_i | i \in \mathbb{N}, i \leq L\}$ of L models. Similar to a predefined behavior edge, we represent a learned behavior edge, e_{lrn} as a pair (EN, τ) of ensemble EN and stopping condition τ, where τ is defined in the same way as in section 7.2 for predefined behaviors. We now describe how the ensemble EN is learnt.

For a state s^t, we approximate the distribution of positions at t' (successor positions) with a Gaussian density, i.e., $s^{t'} = (x^{t'}, y^{t'}) \sim \mathcal{N}(\mu^{t'}, \Sigma^{t'})$. We want to learn a behavior model that predicts $\mu^{t'}, \Sigma^{t'}$. We combine the crowd-agent interaction network model of [25] that explicitly models pairwise interaction of each pedestrian with the agent, with social-LSTMs [3]. For each pedestrian [25] builds an $L \times L$ occupancy gridmap M having velocities for occupied cells and get the fixed size pairwise interaction vectors $e_p = \lambda(s^t, M, W_i)$, where λ is the interaction network which is an MLP with weights W_i. e_p for all pedestrians is combined to form a fixed size interaction vector c. We introduce c as an additional input to the social LSTM model. The outputs of this model are the mean and covariance $(\mu^{t'}, \Sigma^{t'})$

of the posterior position Gaussian distribution, computed through the following recurrence relations:

$$\begin{aligned}(\mu^{t'}, \Sigma^{t'}) &= \texttt{MLP}(h^{t'}; W_m) \\ h^{t'} &= \texttt{LSTM}(h^t, s^t, c^t; W_l)\end{aligned} \tag{7.1}$$

where W_l, W_m are model parameters for the LSTM and Multi-Layer Perceptron (MLP) networks respectively. Model parameters are learned by minimizing the negative log-likelihood loss
$-\log(\mathsf{P}(x^{t'}, y^{t'}|\mu^{t'}, \Sigma^{t'}, x^t, y^t))$. $\Sigma^{t'}$ represents uncertainty in successor positions in data (data uncertainty). [39] has shown that total prediction uncertainty of a model is the sum of data uncertainty and model uncertainty, which is uncertainty in model parameters due to lack of infinite training data. To account for model uncertainty, we initialize every model parameter with a random value and activate layer-wise dropouts while training, that sets each parameter in a layer to zero with probability p_{drop}. This generates $EN = \{M_i | i \in \mathbb{N}, i \leq L\}$ of L models, where models $M_i, M_j \in EN$ for $i \neq j$ differ in model parameters due to dropouts.

7.3.2 SLB-L formulation: Introducing prediction uncertainty of models into SLB

Since we have assumed that successor positions have a Gaussian distribution, the state-space of SLB-L should be a Gaussian belief-space. To keep a tractable graph-size for planning efficiency, we represent state-space of SLB-L by the parameters of a normal distribution instead of a non-parametric representation with particles. Specifically, $\Psi = (\mu, \Sigma, t) \forall \mu \in \mathbb{R}^n, \Sigma \in \mathbb{R}^{n \times n}$, where n is the dimension of position of the agent which in our formulation is two. The successor for a predefined behavior edge (b, τ) applied on a deterministic state

$s^t \in S_{slb}$ is generated as $s^{t'} = \phi_b(s^t, b, \tau)$, where ϕ_b simulates the execution of b starting at s^t until τ . Similarly $s^{t'} = \phi_a(s^t, e_a)$ generates successor for a predefined action e_a. To generate successor $\psi^{t'} \in \Psi$ for $\psi^t = (\mu^t, \Sigma^t, t)$ for (b, τ) and e_a, we propagate the mean of $\psi^{t'}$ as $\mu^{t'} = \phi_b(\mu^t, b, \tau)$ and $\phi_a(\mu^t, e_a)$ respectively. For propagating covariance for predefined behaviors and actions, we assume a linearly decreasing model with time: $\Sigma^{t'} = \Sigma^t - [\alpha_{ij}K], \forall i, j \in \mathbb{N}, i, j \leq n$ after K time-steps, α_{ij} is a tunable parameter. This mimics system-level implementation of the execution of predefined edges. For $s^t \sim \psi^t$, a predefined edge e_{slb} is typically a `PID` controllers that follows the trajectory generated by e_{slb} consisting of the set $\mathscr{P}_{e_{slb}} = \{\mu^t | t \in \mathbb{N}, t \leq K\}$, which is in global frame of reference and thus results in the shrinking of uncertainty per execution step.

Successor generation for ψ^t using $(EN, \tau) \in E_{slb-l}$ is described in pseudocode 3. We now describe pseudocode 3. We go for a sample-based approach: Starting from s^t, at each step we sample N i.i.d particles and propagate each of them through each model in the ensemble EN. If L be the total number of models in the ensemble, we get $N \times L$ output distributions after each step. We assume that the true population at the end of each step is a single bivariate gaussian density, therefore we sample M particles from each of $N \times L$ distributions and use them to find the Maximum Likelihood Estimate (MLE) of the output density parameters $\mu^{t'}$ and $\Sigma^{t'}$. This is repeated till a stopping trigger is reached. The number N is derived from how tight we want our sample convergence bounds to be, as will be shown in our theoretical analysis in Section 7.4. Since uncertainty propagation models for both predefined and learned behaviors deterministically produce a single successor belief state, i.e.,

$$p(\psi^{t'} | \psi^t, e) = 1 \forall e \in E_{slb-l} \tag{7.2}$$

we can formulate `SLB-L` as a deterministic graph $G_{SLB-L} = \{\Psi, E_{slb-l}\}$.

Pseudocode 3 Successor Generation in SLB-L for a learned behavior

Input state $\psi^t = \mathcal{N}(\mu^t, \Sigma^t, t)$
learned behavior ensemble $EN = \{M_1, M_2..M_L\}$
number of samples N = $\frac{1}{2\varepsilon^2} \ln \frac{4}{\delta}$ (refer sec. 4)
stopping condition τ
Output successor $\psi^{t'} = N(\mu^{t'}, \Sigma^{t'}, t'), t' = t + k$

v_t = vector of N i.i.d particles sampled from $\psi^t = N(\mu^t, \Sigma^t)$
$C_{sum} = 0$, $k = 0$, $t' = t$
$v_{t+(k-1)} = v_t$
while τ not satisfied **do**
    k = k + 1
    $v_{t+k} = \{\}$
    for each particle P_k^n in $v_{t+(k-1)}$ **do**
        while P_k^n not in collision **do**
            $\{N(\mu_1, \Sigma_1), ...N(\mu_L, \Sigma_L)\} = EN(P_k^n)$
            v' = $N \times L$ particles (N i.i.d samples each from $\{N(\mu_1, \Sigma_1), ...N(\mu_L, \Sigma_L)\}$
            v_{t+k} = Concatenate(v_{t+k}, v')
        end while
    end for
    Compute $\bar{P}_{t+k}$ = Sample mean of particles in v_{t+k}
    Compute S_{t+k} = Sample covariance of particles in v_{t+k}
    $\mu^{t'} = \hat{\mu} = \bar{P}_{t+k}$
    $\Sigma^{t'} = \hat{\Sigma} = \frac{|v_{t+k}|-1}{|v_{t+k}|} S_{t+k}$ ▷ $\hat{\mu}, \hat{\Sigma}$ are MLE for μ and Σ of a multivariate Gaussian
    Sample set v_{MLE} of N particles from $N(\mu^{t+k}, \Sigma^{t+k})$
    $v_{t+k} = v_{MLE}$, $|v_{MLE}| = N$
    $C_{sum} = C_{sum} + C(|v_k| \times |v_{k+t}|)$
    $v_{t+(k-1)} = v_{t+k}$, $t' = t' + k$
end while

7.3.3 SLB with Confidence of Success of Learned Behaviors (SLB-CSL)

In this subsection, we present how to incorporate the confidence of success of learned behaviors in the SLB formulation to get the SLB-CSL formulation and how to plan in SLB-CSL.

`SLB-CSL` formulation as a belief MDP

Consider a learned behavior e_{lrn} executed at agent position s^t to get a successor state $\psi^{t'}$. We assume that e_{lrn} has a set *SC* of *success* conditions that are evaluated on an agent position. We define the execution of e_{lrn} at s^t as a *success*, if every sampled position $s^{t'} \sim \psi^{t'}$ satisfies each success condition in *SC*. For example, consider the learned behavior `BargeIn` shown in Figure 7.1, that is executed at s^t to signal a group of pedestrians to move away from the exit of a passage they have been blocking. *SC* for this behavior has only one success condition: if the agent position lies within the defined success region marked by the blue oval near the exit. Executing `BargeIn` at s^t is considered a success, if every agent position $s^{t'}$ sampled from the successor state $\psi^{t'}$—for example the position labelled as success outcome in Figure 7.1(a)—lies within the blue oval. We use only those scenarios as training data for learning e_{lrn}, in which pedestrians responds to agent trajectories such that the agent position produced as outcome upon completing execution of a trajectory, satisfies all success conditions in *SC* corresponding to e_{lrn}. However, during the execution of e_{lrn}, when the scenario is *novel* because it was not seen in the training data for e_{lrn}, its execution may not necessarily produce agent positions that satisfy all success conditions, or may even produce potentially unsafe agent positions. For example, in Figure 7.1 (a) pedestrians move away from the exit as seen in training data when `BargeIn` is executed, but in Figure 7.1 (b) they do not, which is a *novel* scenario. Thus, in Figure 7.1 (b), executing `BargeIn` does not produce an outcome lying within the blue oval. We define this case as *failure* in execution of e_{lrn}. Upon failure we assume that the agent returns to ψ^t. This case necessitates estimation and reasoning about the probability of success in executing e_{lrn}, which we refer to as `Confidence of Success`.

Specifically, let $h_{e_{lrn}}^{\psi^t}$ be the variable that represents the outcome (*success*/*failure* denoted by `su`/`f`) of executing e_{lrn} at ψ^t. $h_{e_{lrn}}^{\psi^t}$ is unknown (denoted by `u`) until the execution of

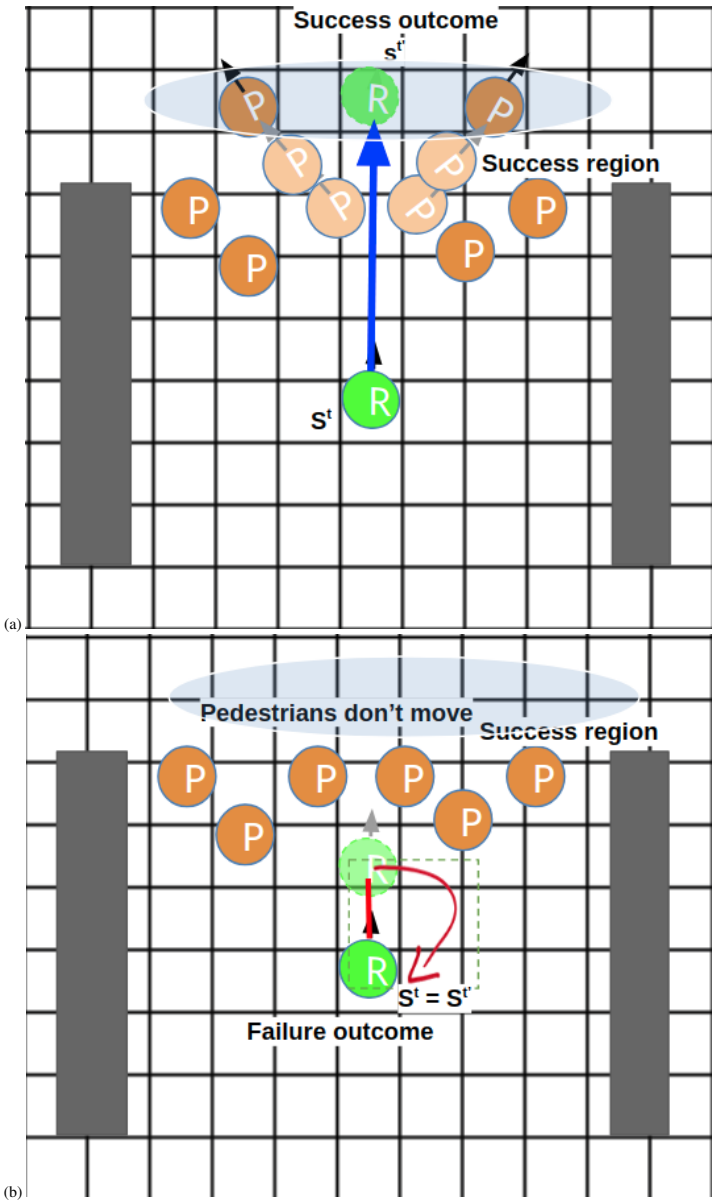


Figure 7.1: Example of a learned behavior `BargeIn` to generate an agent trajectory to make pedestrians clear a passage exit defined by walls (grey rectangles). `BargeIn` when applied to an s^t generates $s^{t'}$.(a) shows an $s^{t'}$ that lies in a success region (blue oval) defined near the passage exit because pedestrians have reacted to the agent and cleared the exit. (b) shows failure when pedestrians don't clear the exit and agent returns to s^t.

e_{lrn} is completed from ψ^t, and is equal to either `su` or `f` after execution. We define the probability $p(h^{\psi^t}_{e_{lrn}} = \texttt{su}|\psi^t, e_{lrn})$ as the `Confidence of Success` in executing e_{lrn} at ψ^t. We first describe how we extend the `SLB-L` framework to the `SLB-CSL` (SLB with Confidence of Success of Learned Behaviors) framework that incorporates this probability. We augment Ψ with a set H of variables representing the outcome of each $e_{lrn} \in E_{lrn}$ executed on each $\psi^t \in \Psi$ to get the belief-space $\mathscr{B} = [\Psi, H]$. To keep the dimension of H tractable, we use a coarse discretization function $f_C : \Psi \to \Psi_C$ that maps Ψ to a smaller space Ψ_C formed by coarsely discretizing μ, Σ and t. Then $H = \{h^{\psi_C}_{e_{lrn}} | \forall e_{lrn} \in E_{lrn}, \forall \psi_C \in \Psi_C\}$, where each $h^{\psi_C}_{e_{lrn}}$ represents the outcome of executing e_{lrn} on ψ^t, for all ψ^t that map to ψ_C. Predefined edges do not affect any variable in H. We state some assumptions:

- **A1**: All variables in H are independent of each other, and
- **A2**: We assume $h^{\psi_C}_{e_{lrn}}$ becomes known immediately after execution of e_{lrn} from ψ^t is completed.

Given **A1**, a belief-state $b^t = [\psi^t, H(b^t)]$ concisely represents the joint probability distribution or *belief* over all possible instantiations of b^t. This is because (1) ψ^t is always known, and (2) each $h^{\psi_C}_{e_{lrn}}$ in $H(b^t)$ concisely represents the probability $p(h^{\psi^t}_{e_{lrn}} = \texttt{su}|\psi^t, e_{lrn})$: the distribution over possible values of all $h^{\psi_C}_{e_{lrn}}$ can be represented as the product of the distribution of each $h^{\psi_C}_{e_{lrn}} \in H$. For a $b^t = [\psi^t, H(b^t)]$, if executing e_{lrn} on ψ^t leads to $\psi^{t'}$ and the outcome $h^{\psi_C}_{e_{lrn}}$ is a success, **A2** formally means that $p(h^{\psi_C}_{e_{lrn}} = \texttt{su}|\psi^{t'}, e_{lrn}) = 1$. Given **A2**, to get the updated belief-state $b^{t'}$ as a result of executing e_{lrn} on ψ^t and effectively on b^t, we set the variable $h^{\psi_C}_{e_{lrn}}$ which was `u` before the execution, to whatever outcome is observed: either `su` or `f`. This represents the belief-update step. As a small example, let ψ^0 be the Gaussian distribution at $t = 0$, $e_{lrn} = \{e_{lrn}\}$, and $\Psi_C = \{\psi_{Ci} | i \in \mathbb{N}, i \leq 4\}$. $\therefore$ $H = \{h^{\psi_{Ci}}_{e_{lrn}} | i \in \mathbb{N}, i \leq 4\}$. Every $\psi^t \in \Psi$ maps to a state in Ψ_C. Let $f_C(\psi^0) = \psi_{C1}$. The belief-state at $t = 0$ is $b^0 = [\psi^0, \{u, u, u, u\}]$. If e_{lrn} executed on ψ^0 leads to ψ^1 and a

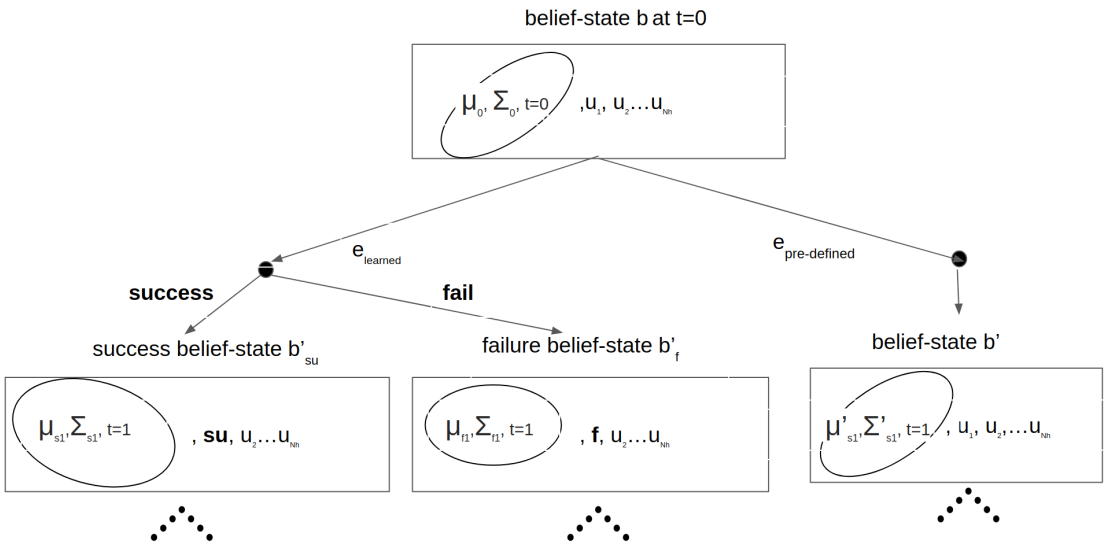


Figure 7.2: Construction of the belief tree. A learned behavior edge can lead to either a success (`su`) or a failure (`f`) distribution. Predefined edges do not make any hidden variable known.

success is observed, then $h^{\psi_{C1}}_{e_{lrn}} = \texttt{su}$, and the updated belief $b^1 = [\psi^1, \{s,u,u,u\}]$. Figure 7.2 shows a belief-tree rooted at b^0 with N_H number of variables in H having one predefined and one learned behavior edge.

The agent knows precisely the belief state it is in after every edge execution. Thus, SLB-CSL is formulated as a belief-MDP $M = \{\mathscr{B}, E_{slb-csl}, C, \mathbb{T}\}$, where $E_{slb-csl} = E_{slb-l}$, C: $\mathscr{B} \times E_{slb-csl} \times \mathscr{B} \rightarrow \mathbb{R}^+$ is the cost function, and $\mathbb{T}(b^t, a, b^{t'}) = p(b^{t'}|b^t, e)$. If $e \in E_{slb}$, then $p(b^{t'}|b^t, e) = 1$. Given **A2** and equation 7.2, for $e \in e_{lrn}$ we can write:

$$p(b^{t'}|b^t, e_{lrn}) = p(h^{\psi_C}_{e_{lrn}}|b^t, e_{lrn}) = p(h^{\psi_C}_{e_{lrn}}|\psi^t, e_{lrn}) \tag{7.3}$$

Let $b^{t'}_{su}$ and $b^{t'}_{f}$ be the corresponding successor belief-states for success, failure outcomes of executing e_{lrn} at b^t, and $c(b^{t'}_{su}, e_{lrn}, b^t)$ and $c(b^{t'}_{f}, e_{lrn}, b^t)$ are the costs. We use $c^t_{\texttt{su}}$ and $c^t_{\texttt{f}}$ to represent the corresponding costs concisely, and p^t_{su} and p^t_f for $p(h^{\psi_C}_{e_{lrn}} = \texttt{su}|b^t, e_{lrn})$ and $p(h^{\psi_C}_{e_{lrn}} = \texttt{f}|b^t, e_{lrn})$ respectively.

We are now ready to define the planning problem. We are given a start state s^0 and a goal

state s^g. We define ψ^0 and ψ^g as zero-variance Gaussians with means s^0 and s^g respectively. The start belief-state b^0 is $[\psi^0, H^0]$, with all variables in $H = \mathtt{u}$. We define a goal belief set as $\mathscr{B}_g = [\psi^g, H]$. For a b^t, we define a policy π as a tree rooted at b^t having all leaf nodes b^g in $\mathscr{B}_g$ The value of a policy π in M rooted at b^t is defined as:

$$V_M^{\pi}(b^t) = p_{su}^t(c_{su}^t + V_M^{\pi}(b_{\mathtt{su}}^t)) + p_f^t(c_f^t + V_M^{\pi}(b_{\mathtt{f}}^t)) \tag{7.4}$$

The planning problem is to find an optimal policy π^* in M rooted at a start belief-state b^0 that minimizes 7.4. $V^*(b^t)$ is the value of π^* rooted at b^t.

Efficient Belief-Space planning in `SLB-CSL`

We clearly prefer a success outcome compared to a failure.

$$c(b_{su}^{t'}, e_{lrn}, b^t) + V^*(b_{su}^{t'}) \leq c(b_f^{t'}, e_{lrn}, b^t) + V^*(b_f^{t'}) \tag{7.5}$$

Given equations 7.3, 7.5, **A1**, **A2**, we can use Probabilistic Planning with Clear Preferences (PPCP) [79], an existing belief-space planner that uses these assumptions to efficiently compute the optimal policy in the belief-space. PPCP performs a series of fast backward and forward searches in the the low-dimensional deterministic space ($\{\mu^t, \Sigma^t, t\}$ in our case) instead of searching in the belief-space. In case of learned behaviors, let EN_{fw} be the ensemble of models that generate the chronological `successor` $\psi^{t'}$ when applied on ψ^t. To run a backward search from goal, we need to produce the chronological `predecessor` ψ^t from $\psi^{t'}$. Thus, we also train an ensemble of models EN_{bw} for predecessor generation. However, we observe that EN_{fw} and EN_{bw} can be used to robustly estimate the `confidence of success`, which we describe next.

Estimating confidence of success in `SLB-CSL`

Since we are running a backward search, for a $\psi^{t'}$ we get a predecessor ψ^{bw} using Algorithm 1 with input $EN = E_{bw}$. We then re-apply Algorithm 1 on ψ^{bw} with $EN = E_{fw}$ to reconstruct $\psi^{t'}$. We denote this reconstructed state by ψ^{fw}. With the introduction of this reconstruction, we modify our definition of successful execution of e_{lrn} at ψ^t as two conditions being true:
C1: $s^{t'}$ sampled from the reconstructed state ψ^{fw} lies in the 95% confidence ellipse (error ellipse) of $\psi^{t'}$, and
C2: $s^{t'}$ from the reconstructed state ψ^{fw} satisfies all conditions in the set *SC* of success conditions of e_{lrn}.
Let $\hat{p}^t_{su}$ and $\hat{c}^t_{su}$ be an estimate of p^t_{su} and $\hat{c}^t_{su}$. To compute them, we sample N states from ψ^{fw} and check if it satisfies all success conditions of e_{lrn}. Let us denote the event where **C1** and **C2** are true by X_i, which is a random bernoulli variable with value =1 if state meets **C1** and **C2**, 0 otherwise. The observed (estimated) mean $\hat{p}^t_{\mathtt{su}}$ of this bernoulli distribution becomes $\hat{p}^t_{\mathtt{su}} = \frac{\sum_n X_i}{N}$.
We recognize that if a state $s^{t'} \sim \psi^{t'}$ is from the train distribution, then $s^{t'}$ will have a non-zero probability in the reconstructed distribution ψ^{fw}. Success condition **C1** approximates verification of this condition. Enforcing **C1** as a success condition makes $\hat{p}^t_{\mathtt{su}}$ high for familiar scenarios in which $\psi^{t'}$ and ψ^{fw} have similar sized error ellipses and similar means.
Let $\hat{C}$ and $\hat{\mathbb{T}}$ be the estimated C and $\mathbb{T}$ in the estimated belief-MDP $\hat{M} = \{\mathscr{B}, E_{slb-csl}, \hat{C}, \hat{\mathbb{T}}\}$. The planner (PPCP in our case) finds an optimal policy $\hat{\pi}^*$ in the estimated belief MDP $\hat{M}$.

7.4 Theoretical analysis

We are given $\varepsilon_p, \delta_p, \varepsilon_c$ and δ_c. The cost-function in M is bounded by $0 \leq c(b^{t'}, e, b^t) = c(\psi^{t'}, e, \psi^t) \leq c_{max}\ \forall b^{t'}, b^t, e$ in M.

Theorem 4. *For a state b^t, learned behavior edge e_{lrn} and a successor $b^{t'}$ that has a success outcome of $h^{\psi_C}_{e_{lrn}}$, let p^t_{su} and $\hat{p}^t$ be the true and the estimated probability of success. Then, the following bound holds true with probability at least $(1-\delta_p)$*

$$p^t_{su} \geq \hat{p}^t_{su} - \varepsilon_p, \text{ where } \varepsilon_p = \sqrt{\tfrac{1}{2N}\ln\tfrac{4}{\delta_p}}$$

Proof. This holds true using Hoeffding's inequality [53]. For practical implementation, this bound is used to compute N.

□

Theorem 5. *For ψ^t, that leads a successor state $\psi^{t'}$, that leads to a success outcome and learned behavior edge e^l, let c^t_{su} and $\hat{c}^t_{su}$ be the true and the estimated cost of the transition $(\psi^{t'}, e_{lrn}, \psi^t)$. Then, the following bound holds with probability at least $(1-\delta_c)$*

$$c^t_{su} \geq \hat{c}^t_{su} - \varepsilon_c, \text{ where } \varepsilon_c = \sqrt{\tfrac{1}{2N}\ln\tfrac{2}{\delta_c}}$$

Proof. This holds true using Hoeffding's inequality [53]. □

Theorem 6. *For any policy π of K time-steps rooted at b^0 having all leaf nodes $b^g \in \mathscr{B}_g$, let $V^\pi_M(b^0)$ and $V^\pi_{\hat{M}}(b^0)$ be its value in M and $\hat{M}$. Then, the following bound holds with probability at least 1-δ_p-δ_c:*

$$|V^\pi_M(b^t) - V^\pi_{\hat{M}}(b^t)| \leq \varepsilon_K \tag{7.6}$$

where $\varepsilon_K = 2\varepsilon_p K\left(1 + \frac{c_{max}(K-1)}{2}\right) + \varepsilon_c$

Proof. Starting from leaf nodes with $V^\pi_M(b^g) = 0$, for any t^{th} step node $b^t \in \pi$ we get $V^\pi_M(b^t) \leq (K-t)c_{max}$. Using bellman equations for $V^\pi_M(b^g)$ and $V^\pi_{\hat{M}}(b^g)$ and error-bounds from Theorems 4 and 5, we get the recurrence:

$$\begin{aligned} |V^\pi_M(b^t) - V^\pi_{\hat{M}}(b^t)| \leq 2\varepsilon_p |c_{max}(K-(t-1)| + \varepsilon_c \\ + |V^\pi_M(b^{t+1}) - V^\pi_{\hat{M}}(b^{t+1})| \end{aligned} \tag{7.7}$$

Using induction on t, we finally get Equation 7.6.

Let $|\hat{p}^t_{su} - p^t_{su}| \leq \varepsilon_p$ be event A, $|\hat{c}^t_{su} - c^t_{su}| \leq \varepsilon_c$ be event B. By union bounds, we get $p(A \cap B) \geq 1 - p(\neg A) - p(\neg B)$. Equation (6) holds when both A and B are true, hence:

$$p(|V^{\pi}_{M}(b^t) - V^{\pi}_{\hat{M}}(b^t)|) = p(A \cap B) \geq 1 - \delta_p - \delta_c \tag{7.8}$$

□

Using Theorem 6 we derive the error bound for $V^{\pi^*}_{M}$.

Theorem 7. *Let π^* and $\hat{\pi}^*$ be optimal policies in M and $\hat{M}$ rooted at b^0 having all leaf nodes $b^g \in \mathscr{B}_g$. π^* and $\hat{\pi}^*$ have T_{π^*} and $T_{\hat{\pi}^*}$ time-steps respectively, $T_{\pi^*}, T_{\hat{\pi}^*} \leq T$. Let $V^{\pi^*}_{M}$ and $V^{\hat{\pi}^*}_{M}$ be the value of π^* in M and $\hat{\pi}^*$ in $\hat{M}$ respectively. The following bound holds with probability at least 1-δ_p-δ_c:*

$$V^{\hat{\pi}^*}_{M} - V^{\pi^*}_{M} \leq 2\varepsilon_T \tag{7.9}$$

where $\varepsilon_T = 2\varepsilon_p T \left(1 + \frac{c_{max}(T-1)}{2}\right) + \varepsilon_c$

Proof. For π^*, using Theorem 5.3 we can write:

$$V^{\pi^*}_{\hat{M}} - \varepsilon_T \leq V^{\pi^*}_{M} \leq V^{\pi^*}_{\hat{M}} + \varepsilon_T,$$

where ε_T is defined in Equation (8). We can write a similar inequality for $V^{\hat{\pi}^*}_{M}$. Now,

$$V^{\hat{\pi}^*}_{M} - V^{\pi^*}_{M} \leq max(V^{\hat{\pi}^*}_{M}) - min(V^{\pi^*}_{M}).$$

Since $\hat{\pi}^*$ is the optimal policy in $\hat{M}$, $V^{\hat{\pi}^*}_{\hat{M}} - V^{\pi^*}_{\hat{M}} \leq 0 \implies V^{\hat{\pi}^*}_{M} - V^{\pi^*}_{M} \leq 2\varepsilon_T$. Using union bounds as shown for Theorem 6, we can state that

$$p(V^{\hat{\pi}^*}_{M} - V^{\pi^*}_{M} \leq 2\varepsilon_T) \geq 1 - \delta_p - \delta_c$$

□

7.5 Implementation and experimental analysis

In this section, we present our experimental setup and the experimental evaluation of the different formulations described in section 7.3.

Table 7.1: Performance comparison of `SL-CSL` with baselines `SL` and `SL-L` as described in the experimental setup in section 7.5

Scenario type	Planner type	Planning time(s)	Solution execution time(s)	# Emergency stops
Familiar (clear exit)	SL	5.81±0.32	62.90±0.87	0±0
	SL-L	47.4±23.5	6±2.3	0±0
	SL-CSL	37.16±18.70	8.4±2.83	0±0
Novel (Move Towards)	SL	1.8±1.4	22.00±5.62	1
	SL-L	40.28±6.11	12±10.03	7
	SL-CSL	44.52±3.33	19.6±6.56	3
Move Away	SL	5.74±0.19	63±0.84	0±0
	SL-L	40.3±0.20	6.7±2.86	0 ± 0
	SL-CSL	23.35±12.47	7.11±2.61	0 ± 0

7.5.1 Experimental setup

We perform evaluations in simulation. We represent the environment as an 8-connected 50×50 grid with 8 predefined actions for moving in directions: E, W, N, S, NE, NW, SE and SW. We have one learned behavior `Barge-In`, that produces successor positions to make pedestrians clear a passage exit as described in figure 7.1. For `Barge-In`, we use the termination condition: `terminate after K steps` with K = 5, and success condition: `successor position lies in exit region` (blue ellipse in figure 7.3).

Dataset collection and training: We use the RVO2 simulator [111] to generate training samples. A `Barge-In` scenario is created as follows: pedestrians are placed to block the exit of a passage, and an agent serving as an oracle is placed inside the passage with its goal in the exit region. Pedestrians clear the block for the oracle agent to reach the exit region: we record time-parameterized trajectories and velocities of the pedestrians and oracle agent to get one training instance. We introduce random perturbations in the start and goal positions to generate the training dataset. We follow the network parameters of [25] and [3] for the crowd-agent interaction network and LSTM network respectively. We train using dropout probability $p_{drop} = 0.4$ an ensemble EN of five social-attention based

learned `BargeIn` models.

Evaluation scenarios: We create an environment with narrow passages and long corridors to represent a typical indoor environment. Our evaluation scenarios can be classified into three types: (1) ***Familiar***: *similar* to training instances, i.e., people are *initially static* but move *away* from the exit of the passage once the agent moves towards them with a specific speed range, (2) ***Move-Towards***: where people are moving at a random speed in a direction *towards* the agent. Since the velocity direction of people is reversed compared to the training scenarios, we classify them as *novel* scenarios, and (3) ***Move-Away***: where people start moving inside the passage right from the start, *away* from the agent. The velocity direction is similar to train scenarios, but people are initially not static unlike train scenarios. This can be interpreted as a scenario somewhere in between a familiar and a novel scenario. We compare `SLB-CSL` with two baselines: (1) Planning with state-lattice with predefined actions (`SL`), and (2) Planning with state-lattice with learned behaviors (`SLB-L`), but no estimate of confidence of success.

7.5.2 Results and discussion

Qualitative results: Figure 7.3 shows the performance of `SL` and `SLB-CSL` on a familiar scenario. In a familiar scenario, initially, no feasible path exists to the goal. However, `SLB-CSL` has the `Barge-In` behavior that creates a feasible path-to-goal through the passage for the agent. Also, `SLB-CSL` correctly estimates a high confidence of succes. Thus, planning with `SLB-CSL` produces an optimal policy consisting of executing predefined actions for initial time-steps followed by `Barge-In` (Figure 7.3 center and right), resulting in a shorter path (execution time = 5 seconds). Whereas, planning with `SL` lacks the learned behavior models and incorrectly assumes that pedestrians will keep blocking the exit. Thus, it is finds a longer path around it (execution time = 62 seconds).

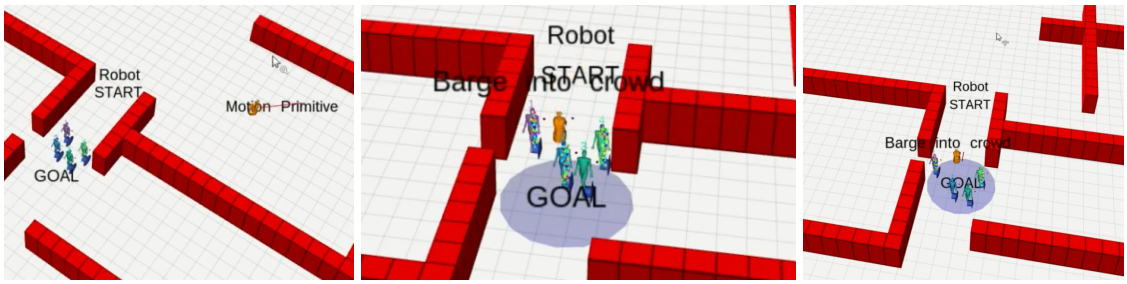


Figure 7.3: center and right: Our approach SLB-CSL in a *familiar* scene (people moving away from exit), is able to come up with a much shorter path compared to SL (left). left: since SL does not have an interaction model, the agent goes around the passage.

Figure 7.4 shows the performance of SLB-L and SLB-CSL on a novel scenario. Pedestrians are initially blocking the passage exit but start moving towards the agent. SLB-CSL estimates low confidence of success and and finds a path around the passage to generate a longer but less riskier path (Figure 7.4 (a)). Optimal policy generated by SLB-L has BargeIn as an action because it does not estimate confidence of success. Upon the execution of this policy, the agent enters into the passage and approaches pedestrians head-on and comes dangerously close, triggering an emergency stop.

Quantitative results: Quantitative results are summarized in Table 7.1 for the three scenario-types. For all scenarios, the agent is assigned random start and goal locations near the entry and exit of the passage respectively. We generate 10 (start,goal) pairs for scenarios of each type for averaging the results, resulting in a total of 30 different scenarios. Table 7.1 reports the mean and standard deviations of the following performance metrics: (1) Planning time (secs): time required to compute a solution. Note that planning with SL and SLB-L run (A^*) search while SLB-CSL runs PPCP to compute an executable policy, (2) Solution Execution Time (secs): time taken to execute the solution to reach from start to goal, (3) # Emergency Stops: number of times emergency stop was triggered during execution (when the agent comes closer than 0.5m to a pedestrian). Learned behavior edges are expensive to generate: hence planning times are higher and similar for both SLB-CSL and SLB-L,

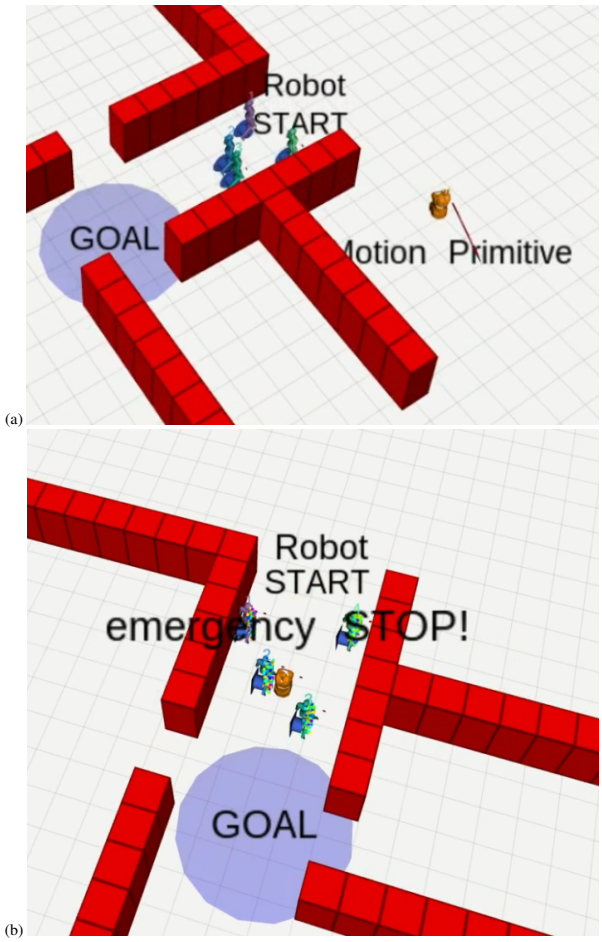


Figure 7.4: (a) `SLB-CSL` in a novel scene (people moving towards agent) estimates low probability of success and goes around the passage. (b) `SLB-L` executes `BargeIn` and comes dangerously close to pedestrians, triggering an emergency stop.

and lower for `SL`. For *familiar* scenarios, `SLB-CSL` on average correctly identifies them by producing high confidence estimates and executes the shorter path that consists of executing one `Barge-In` behavior. `SLB-L` being analogous to `SLB-CSL` with probability of success = 1 always, also behaves similarly. However, `SL` assumes that the passage is blocked for all time-steps, and plans a path with execution time $10X$ that of `SLB-CSL`. We observe that our approach is very useful in scenarios with long corridors and detours, as it provides a 30% improvement over `SL` in terms of total time from the start of planning till the completion of execution. For *novel* scenarios, both `SLB-CSL` and `SL` find longer paths with similar

execution times, as compared to SLB-L. However, SLB-L exhibits risky behavior leading to emergency stops in more than double the number of such behaviors for SLB-CSL. Results for the *Move-Away* scenario are similar to that of *familiar*.

7.6 Conclusions, Discussion, and Lessons Learned

In this work we have demonstrated how to construct a graph using learned behaviors as edges in addition to predefined behaviors and actions, and estimate confidence of success of executing learned behaviors, for the problem of agent navigation among pedestrians. Experimental results indicate that using learned behaviors, the agent can interact with pedestrians to produce better paths.

One limitation of this approach is that every time we want to add a new learned behavior, we must train a new ensemble model for that learned behavior. This can render our approach potentially hard to scale up to complex navigation scenarios that need many learned behaviors.

Our approach divides pedestrian navigation behavior into easily learnable simple behaviors that can be trained from very little data. More training data will be required if we want complicated behaviors that combine two or more simple behaviors. On the other hand, combining multiple simple behaviors into one complex behavior would decrease the branching factor of the graph, which can ultimately reduce planning times. Thus, reducing the branching factor to increase planning speed might require increasing training data to learn complex behaviors.

Chapter 8

Conclusions and Future work

In this chapter, we summarize the contributions of this thesis. We then discuss future directions of research.

8.1 Contributions

Introduction of Conservative Heuristics to reduce search efforts in Search-Based Motion Planning, its theoretical and experimental analysis. We define the notion of conservative edges in the abstract space [22]. We propose a heuristic computation algorithm in the abstract space that minimizes the number of non-conservative edges. We perform the theoretical analysis of our algorithm and show that conservative heuristics reduce state expansions by weighted A* under certain conditions. We evaluate conservative heuristics in several motion planning domains and observe significant reductions in expansions and planning times compared to a popular baseline heuristic.

Introduction of Fast-PPCP: a novel algorithm that minimizes anticipated search efforts in probabilistic planning, its theoretical and empirical evaluation. We focus on

the problem of planning under uncertainty over missing information about the environment. We develop FAST-PPCP—a novel approach to planning under uncertainty for problems wherein a clear preference exists over the actual values of missing information and the assumption of perfect sensing holds [23]. FAST-PPCP plans by a series of searches, where each search explicitly minimizes the amount of missing information it relies upon to reach the goal. FAST-PPCP uses novel search scheduling strategies and guarantees completeness and bounded-suboptimality of the final solution. We achieve a substantial decrease in runtime while incurring little loss in solution quality compared to popular baseline methods for planning under uncertainty.

Adaptation and experimental evaluation of algorithms that achieve speedup by minimizing anticipated search-effort in real-world robotics problems. We implement a footstep planner that uses weighted A* search combined with Conservative heuristics on a physical NAO robot. We show the utility of using Conservative Heuristics for quickly computing footstep plans for the NAO robot in a real-world setting. We demonstrate the benefits of FAST-PPCP in real-world robot navigation problems in which the environment has large unknown regions. We develop an optimized version of Fast-PPCP for application in environments with large unknown regions. We evaluate it in both simulation on a real-world map and a physical robotic system—Husarion ROSbot 2.0 PRO.

Application of PPCP in the context of navigation among pedestrians. Additionally, we have explored the application of the PPCP planner in the context of navigation among pedestrians [24]. We start with the key idea of incorporating learned behaviors into traditional graph search-based planning for agent navigation among pedestrians. We then demonstrate how to construct a graph with learned behaviors as edges, in addition to predefined actions typically used in search-based planning. We estimate the probability of successful execution of a learned behavior and incorporate this probability while planning

using the PPCP planner. We perform theoretical analysis and experimental evaluation in simulation.

8.2 Future Work

In this section, we present exciting directions for future research.

8.2.1 Application of FAST-PPCP in the domain of navigation among pedestrians

We have shown in chapter 5 that FAST-PPCP can efficiently compute a policy in CP-PS belief MDP faster than the following baselines: PPCP, weighted RTDP-bel, and HSVI2 in the domain of robot navigation in partially known environments. We have also applied an optimized version of Fast-PPCP in environments with large unknown regions. An interesting future research direction is to adapt the optimized version of FAST-PPCP in the domain of navigation among pedestrians and experimentally evaluate the utility of FAST-PPCP over the stated baselines.

Benefit. By using optimized FAST-PPCP instead of PPCP to plan for navigation among pedestrians, we aim to further reduce planning times which would be quite useful in this challenging dynamic domain. We can perform the evaluations across multiple environments created in the PedSim simulator used in our experiments in chapter 7. This research direction would consequentially also involve applying weighted RTDP-bel and HSVI2 in the domain of navigation among pedestrians, which has not been investigated before.

Challenges. Since we want to achieve real-time planning speeds, the main challenge is to decrease the time per iteration of FAST-PPCP. Also, there must be optimizations at the implementation level to achieve planning times in milliseconds. Training more complex

learned behaviors is also a non-trivial task.

8.2.2 Accounting for a change in the status of an unknown variable after it is sensed

We assume that once the true state of a hidden variable is sensed, it remains unchanged during a planning and execution episode. This assumption is generally true for the domains discussed in chapter 6, i.e., off-road environments and indoor environments with carpets or mats on the floor. In the case of off-road environments, terrain characteristics determine the traversability of a region. Natural terrain characteristics do not change frequently: for example, a large sandy region remains so for an extended period. Because of this, once an unknown region is sensed as traversable or non-traversable during execution, its status typically remains the same throughout planning and execution. Similarly, this assumption holds for carpets because their material and thickness will not change. However, this assumption may not hold for the indoor navigation domain of chapter 5, where the status of a door is unknown. Once a door is sensed as open, someone might close the door, and as a result, its status may change during the execution.

Benefit. Future work could involve modifying the FAST-PPCP framework to account for a change in the status of an unknown variable after it is sensed. This modification would broaden the range of domains where FAST-PPCP can be applied.

Challenge. Since FAST-PPCP is currently equipped to handle only such cases where the status of a hidden variable does not change, we may need to make non-trivial changes at the algorithmic level.

8.2.3 Introducing actions in the FAST-PPCP framework that do not exhibit perfect sensing

Another significant future research direction is to introduce `sense-from-far` actions for long-range sensing. By definition, a `sense-from-far` action $a_{h_i}(s)$ would sense the unknown region represented by the hidden variable h_i from the cell s that is not necessarily adjacent to the hidden variable h_i. `Sense-from-far` would typically use visual observations from a visual sensor. It is unrealistic to assume that there is no sensing noise in a visual sensor. In other words, long-range sensing using a `sense-from-far` action is *imperfect*: a `sense-from-far` action $a_{h_i}(s)$ senses the unknown region represented by h_i from state s but instead of returning its true status, it returns a noisy observation from the set $\{traversable, non-traversable\}$. We can assume that an observation noise model $P(O|h_i = O, s)$ is available that gives the probability of getting the observation O after executing the `sense-from-far` action at s when the true status of h_i is O.

We will now explain the benefit and challenges of this research direction.

Benefit. A `Sense-from-far` action would let the robot sense the status of an unknown region without having to move all the way to that region in order to find out its status. Such an action is beneficial when the cost of moving to the hidden region, sensing it as non-traversable and taking the detour path is higher than the cost of sensing it from far and preemptively avoiding the hidden region.

Challenges. We now explain how the size of the belief tree is affected by the introduction of sense-from-far actions. Firstly, multiple hidden variables can be within the sensing range of a `sense-from-far` action, as opposed to a single hidden variable being sensed by the sense-and-move action considered in this thesis. For k hidden variables within the sensing range, the branching factor in the belief state space increases by $2k$ because there are k sensing actions and each action can produce two possible observations. Additionally, with

the perfect sensing assumption, for N states in the graph representing the environment and $|H|$ total number of hidden variables, the size of the belief state space is N times the number of possible beliefs over a hidden variable. In a CP-PS belief-MDP, because of the perfect sensing assumption, there are only 3 possible beliefs–unknown, 1, or 0, and the size of the belief state space is $N \cdot 3^{|H|}$. With the introduction of `sense-from-far` actions, the belief over the status of an unknown hidden variable can be any value between 0 to 1, as opposed to only 3 possible values as is in the case of a CP-PS belief-MDP. Thus, the size of the belief state-space is infinite: a CP-PS belief-MDP is much narrower than a belief-MDP with sense actions that have imperfect sensing. To deal with the infinite belief space, point-based methods ([92]) use a vector set of sampled points to represent the value function and restrict value function updates to a subset of the belief space. There are other approaches that discretize the belief space. With a reasonably fine discretization the size of belief states now becomes significantly larger than $N \cdot 3^{|H|}$, the size of belief space in CP-PS belief MDPs. In our experiments we have shown that FAST-PPCP gives ~ 5 seconds planning times in the CP-PS belief-MDP with ~ 100 hidden variables and 30 m $\times$ 30 m environment. With the increase in the size of the belief state-space and branching factor, Fast-PPCP as it is may not be able to provide the few seconds planning time in environments as large as our experimental ones. Future work is to either make algorithmic modifications to FAST-PPCP or devise a novel algorithm that: (1) computes a policy in this larger belief MDP by reducing the amount of missing information the policy relies upon to reach goal, and (2) ensures bounded-suboptimality of this policy.

8.2.4 Interleaving planning and execution in the FAST-PPCP framework

If we want to use FAST-PPCP for more challenging real-world robotics planning problems compared to the problems considered in this thesis—such as navigation among pedestrians– then we must interleave planning with execution in the FAST-PPCP framework.

Benefit. The agent can start executing its current policy and the FAST-PPCP planner can simulatenously work on improving this policy. This way the agent does not have to wait for the planner to converge to a full policy before it can start execution.

One potential idea to interleave planning with execution is as follows. FAST-PPCP computes whichever partial policy it can find within a given time-bound, and the robot starts executing this partial policy. After a few seconds of execution, the current robot state becomes the start state for the planner. FAST-PPCP can potentially try to find a `better` partial policy from this updated start state, where the definition of `better` can be a metric such as the one that has a higher probability of reaching the goal location. FAST-PPCP then abandons the current partial policy and replaces it with the newly computed partial policy only if the latter is `better` than the current partial policy. Also, if FAST-PPCP while re-planning can re-use the already computed values of the states instead of planning from scratch, this could be a much more efficient way of re-planning.

Challenges. We discuss a research challenge that might arise while interleaving planning and execution in FAST-PPCP. FAST-PPCP starts with a lower bound optimal value from the start belief state and aims to compute a full policy from the start belief-state whose value is within α times the lower bound for a user-defined constant alpha. We term $alpha$ times the lower bound as the suboptimality bound. Starting from the start belief-state, it iteratively develops a partial policy into a full policy by performing either (1) policy growth by adding branches from policy-devoid outcomes in the current partial policy or

(2) policy correction by replacing existing branches. In its current version, if the value of the current partial policy exceeds the suboptimality bound, then FAST-PPCP removes and replaces an existing branch such that the partial policy with the replaced branch is bounded-suboptimal. However, suppose we aim for interleaving planning and execution. In that case, a situation could arise when FAST-PPCP chooses a branch to replace, but the robot has already committed to this branch and started executing this branch. In this case, FAST-PPCP cannot remove the entire branch. The research challenge for future work in this exciting direction is to determine the state on the current execution branch from which FAST-PPCP starts the replacement. Also, while investigating this question, it is worthwhile to consider if we need to use a different definition of bounded-suboptimality of a policy compared to the one used in this thesis in chapter 5.

Appendix A

Fast Bounded-Suboptimal Probabilistic Planning with Clear Preferences on Missing Information: Supplementary Material

A.1 Appendix A: Proofs

In this section we present proofs of lemmas 7 and 8.

Lemma 7. *Between two subpaths ρ_2 and $\rho_1(s)$ from a state $s \in G_s$ to S_g, if $\rho_2(s)$ is dominated by $\rho_1(s)$, then for any path $\psi_2(S(X_p))$ from $S(X_{st})$ to S_g passing through $\rho_2(s)$, there exists a path $\psi_1(S(X_p))$ from $S(X_p)$ to S_g that dominates $\psi_2(S(X_p))$*

Proof. There are finite number of paths from $S(X_p)$ to s. Consider any such path $\tau(S(X_p), s)$, with $\{c^{\tau}_{sto}(S(X_p), s), \hat{V}^{\tau}(S(X_p), s)\}$. Let $\psi_2(S(X_p))$ be the path from $S(X_p)$ to S_g composed

of τ from $S(X_p)$ to s, and $\rho_2(s)$ from s to S_g, with path cost $\mathbf{g}^{\psi_2}(S(X_p))$. Similarly, let $\psi_1(S(X_p))$ be the path from $S(X_p)$ to S_g composed of τ from $S(X_p)$ to s, and $\rho_1(s)$ from s to S_g, with path cost $\mathbf{g}^{\psi_1}(S(X_p))$. The following holds:

$$\begin{aligned} c_{sto}^{\psi_1}(S(X_p)) &= c_{sto}(\rho_1)(s) + c_{sto}^{\tau}(S(X_p), s) \\ c_{sto}(\psi_2)(S(X_p)) &= c_{sto}(\rho_2)(s) + c_{sto}^{\tau}(S(X_p), s) \end{aligned} \tag{A.1}$$

Since $\rho_2(s)$ is dominated by $\rho_1(s)$,

$$c_{sto}(\rho_1)(s) < c_{sto}(\rho_2)(s) \; and \; g_Q^{\rho_1}(s) < g_Q^{\rho_2}(s)$$

Since the same quantity $c_{sto}^{\tau}(S(X_p), s)$ is added to $c_{sto}(\rho_1)(s)$ and $c_{sto}(\rho_2)(s)$ in eq.A.1, therefore the inequality relationship between $c_{sto}(\psi_1)(S(X_p))$ and $c_{sto}(\psi_2)(S(X_p))$ follows the relationship between $c_{sto}(\rho_1)(s)$ and $c_{sto}(\rho_2)(s)$, i.e.,

$$c_{sto}(\psi_1)(S(X_p)) < c_{sto}(\psi_2)(S(X_p))$$

Similarly, if τ had N stochastic transitions with probabilities of preferred outcomes being $P_1, P_2 ...., P_n$, g_Q values are of the form:

$$\begin{aligned} g_Q^{\psi_1}(S(X_p)) &= \prod_{i=1}^{N} P_i . g_Q^{\rho_1}(s) + K \\ g_Q^{\psi_2}(S(X_p)) &= \prod_{i=1}^{N} P_i . g_Q^{\rho_2}(s) + K \end{aligned} \tag{A.2}$$

Where K is a positive quantity. Since $g_Q^{\rho_1}(s)$ and $g_Q^{\rho_2}(s)$ are multiplied by the same positive quantity $\prod_{i=1}^{N} P_i$ and added by the same quantity K, using similar argument as before the inequality relationship between $g_Q^{\psi_1}(S(X_p))$ and $g_Q^{\psi_2}(S(X_p))$ follows the relationship

between $g_Q^{\rho_1}(s)$ and $g_Q^{\rho_2}(s)$, i.e.,

$$g_Q^{\psi_1}(S(X_p)) < g_Q^{\psi_2}(S(X_p))$$

Since this relationship holds for any τ, if subpath $\rho_2(s)$ is dominated by $\rho_1(s)$, path $\psi_2(S(X_p))$ is also dominated by $\psi_1(S(X_p))$ and Lemma 7 holds. □

Let $g_Q(X_{st})$ computed using $g_Q^{\psi_1}(S(X_p))$ and $g_Q^{\psi_2}(S(X_p))$ be denoted by $\hat{V}^{\psi_1}(X_{st})$ and $\hat{V}^{\psi_2}(X_{st})$ respectively. Since $g_Q(X_{st})$ is computed using current partial policy which does not change in a given FAST-PPCP iteration, $\hat{V}^{\psi_1}(X_{st})$ and $\hat{V}^{\psi_2}(X_{st})$ are of the form:

$$\begin{aligned} \hat{V}^{\psi_1}(X_{st}) &= \beta_i g_Q^{\psi_1}(S(X_p)) + \gamma_i \\ \hat{V}^{\psi_2}(X_{st}) &= \beta_i g_Q^{\psi_2}(S(X_p)) + \gamma_i \end{aligned} \tag{A.3}$$

where β_i and γ_i are the remaining terms that appear in the expression of $\hat{V}(X_{st})$ which is the v-value of X_{st} following the current partial policy in the i^{th} FAST-PPCP iteration. Since β_i is positive and β_i and γ_i are added to both $g_Q^{\psi_1}(S(X_p))$ and $g_Q^{\psi_2}(S(X_p))$ in eq. A.3, the following conclusion can be drawn:

$$g_Q^{\psi_1}(S(X_p)) < g_Q^{\psi_2}(S(X_p)) \implies \hat{V}^{\psi_1}(X_{st}) < \hat{V}^{\psi_2}(X_{st})$$

.

Lemma 8. ***Pruning using dominance****: Let $\mathbb{P}$ be the set of all paths in G_s from $S(X_p)$ to S_g. Let $\mathscr{P}_\sigma \subset \mathbb{P}$ be a subset of paths from $S(X_p)$ to S_g such that no two paths have the same **g**-value and every path $\sigma \in \mathscr{P}_\sigma$ satisfies*

$$\sigma = \arg\min_{p \in \mathbb{P}} g_{sto}^p(S(X_p)) \; s.t \text{ using } g_Q^p(S(X_p)) \; \hat{V}^p(X_{st}) \leq \alpha V_L^*(X_{st}) \tag{A.4}$$

For a state $s \in G_s$, let $\rho_1(s)$ and $\rho_2(s)$ be two sub-paths from s to S_g that the search visits with path costs $\mathbf{g}^{\rho_1}(s)$ and $\mathbf{g}^{\rho_2}(s)$. Let $\rho_2(s)$ be dominated by $\rho_1(s)$. For every state $s \in G_s$, we prune all paths from $S(X_p)$ to S_g that pass through $\rho_2(s)$. It is guaranteed that this pruning method does not prune any path in $\mathscr{P}_\sigma$.

Proof. Estimated value of X_{st} under the current fixed policy in i^{th} FAST-PPCP iteration computed using the g_Q value of a path p from $S(X_p)$ to S_g is given by:

$$\hat{V}^p(X_{st}) = \beta_i g_Q^p(S(X_p)) + \gamma_i \tag{A.5}$$

Using A.5, eq. A.4 can also be written as:

$$\sigma = \arg\min_{p \in \mathbb{P}} g_{sto}^p(S(X_p)) \;\; s.t \;\; g_Q^p(S(X_p)) \;\leq\; \frac{\alpha V_L^*(X_{st}) - \gamma_i}{\beta_i}$$

For brevity we will use κ in place of $\frac{\alpha V_L^*(X_{st}) - \gamma_i}{\beta_i}$ for the rest of this proof.
All paths in $\mathscr{P}_\sigma$ have the same cost c_{sto}s but may have different g_Q-values. Let us consider one path $\sigma \in \mathscr{P}_\sigma$, denote its g_{sto} and g_Q values by $g_{sto}^\sigma(S(X_p))$ and $g_Q^\sigma(S(X_p))$.
Let $\psi_1(S(X_p))$, $\rho_1(s)$, $\psi_2(S(X_p))$ and $\rho_2(s)$ be defined the same way as in lemma 7. As shown in lemma 7, if $\rho_2(s)$ is dominated by $\rho_1(s)$, then $\psi_2(S(X_p))$ is dominated by $\psi_1(S(X_p))$.
Now, consider the following cases that cover all possible inequality relationships between $\mathbf{g}^\sigma(S(X_p))$ and $\mathbf{g}^{\psi_1}(S(X_p))$. For each case we will show that if $\psi_2(S(X_p))$ is dominated by $\psi_1(S(X_p))$, $\mathbf{g}^{\psi_2}(S(X_p))$ can never be equal to $\mathbf{g}^\sigma(S(X_p))$, implying that $\psi_2(S(X_p))$ can never be in $\mathscr{P}$.

- **Case 1**: $g_{sto}^\sigma(S(X_p)) = g_{sto}^{\psi_1}(S(X_p))$ and $g_Q^\sigma(S(X_p)) = g_Q^{\psi_1}(S(X_p))$: Since $g_{sto}^{\psi_1}(S(X_p)) < g_{sto}^{\psi_2}(S(X_p))$ and $g_Q^{\psi_1}(S(X_p)) < g_Q^{\psi_2}(S(X_p))$, therefore $g_{sto}^\sigma(S(X_p)) < g_{sto}^{\psi_2}(S(X_p))$ and $g_Q^\sigma(S(X_p)) < g_Q^{\psi_2}(S(X_p))$. Thus, $\mathbf{g}^{\psi_2}(S(X_p)) \neq \mathbf{g}^\sigma(S(X_p))$.

- **Case 2**: $g^{\sigma}_{sto}(S(X_p)) < g^{\psi_1}_{sto}(S(X_p))$ and $g^{\sigma}_{Q}(S(X_p)) < g^{\psi_1}_{Q}(S(X_p))$: Using the same argument as **Case 1**, $\mathbf{g}^{\psi_2}(S(X_p)) \neq \mathbf{g}^{\sigma}(S(X_p))$.
- **Case 3**: $g^{\sigma}_{sto}(S(X_p)) > g^{\psi_1}_{sto}(S(X_p))$ and $g^{\sigma}_{Q}(S(X_p)) > g^{\psi_1}_{Q}(S(X_p))$: $g^{\sigma}_{Q}(S(X_p))$ satisfies the condition: $g^{\sigma}_{Q}(S(X_p)) \leq \kappa$. Since σ is a solution of eq **??** and both $g^{\sigma}_{Q}(S(X_p))$ and $g^{\psi_1}_{Q}(S(X_p))$ are $\leq \kappa$, this implies that $g^{\sigma}_{sto}(S(X_p)) \leq g^{\psi_1}_{sto}(S(X_p))$. Therefore, $g^{\sigma}_{sto}(S(X_p)) > g^{\psi_1}_{sto}(S(X_p))$ is a contradiction and **Case 3** can never happen.
- **Case 4**: $g^{\sigma}_{sto}(S(X_p)) > g^{\psi_1}_{sto}(S(X_p))$ and $g^{\sigma}_{Q}(S(X_p)) = g^{\psi_1}_{Q}(S(X_p))$: Using the same argument as **Case 4**, $g^{\sigma}_{sto}(S(X_p)) > g^{\psi_1}_{sto}(S(X_p))$ is a contradiction and **Case 4** can never happen.
- **Case 5**: $g^{\sigma}_{sto}(S(X_p)) < g^{\psi_1}_{sto}(S(X_p))$ and $g^{\sigma}_{Q}(S(X_p)) = g^{\psi_1}_{Q}(S(X_p))$: Using the same reasoning as in **Case 1**, $\mathbf{g}^{\psi_2}(S(X_p)) \neq \mathbf{g}^{\sigma}(S(X_p))$.
- **Case 6**: $g^{\sigma}_{sto}(S(X_p)) < g^{\psi_1}_{sto}(S(X_p))$ and $g^{\sigma}_{Q}(S(X_p)) > g^{\psi_1}_{Q}(S(X_p))$: Since $g^{\sigma}_{sto}(S(X_p)) < g^{\psi_2}_{sto}(S(X_p))$, $\mathbf{g}^{\psi_2}(S(X_p)) \neq \mathbf{g}^{\sigma}(S(X_p))$.
- **Case 7**: $g^{\sigma}_{sto}(S(X_p)) > g^{\psi_1}_{sto}(S(X_p))$ and $g^{\sigma}_{Q}(S(X_p)) < g^{\psi_1}_{Q}(S(X_p))$: This is possible only if $g^{\psi_1}_{Q}(S(X_p)) > \kappa$. Since $g^{\psi_2}_{Q}(S(X_p)) > g^{\psi_1}_{Q}(S(X_p))$, $g^{\psi_2}_{Q}(S(X_p))$ is clearly $> \kappa$. Thus, $\mathbf{g}^{\psi_2}(S(X_p)) \neq \mathbf{g}^{\sigma}(S(X_p))$.
- **Case 8**: $g^{\sigma}_{sto}(S(X_p)) = g^{\psi_1}_{sto}(S(X_p))$ and $g^{\sigma}_{Q}(S(X_p)) < g^{\psi_1}_{Q}(S(X_p))$: Using the same reason as **Case 1**, $\mathbf{g}^{\psi_2}(S(X_p)) \neq \mathbf{g}^{\sigma}(S(X_p))$.
- **Case 9**: $g^{\sigma}_{sto}(S(X_p)) = g^{\psi_1}_{sto}(S(X_p))$ and $g^{\sigma}_{Q}(S(X_p)) > g^{\psi_1}_{Q}(S(X_p))$: In this case, since $g^{\sigma}_{Q}(S(X_p)) \leq \kappa$, $g^{\psi_1}_{Q}(S(X_p))$ is also $\leq \kappa$ which means that $\psi_1 \in \mathscr{P}_{\sigma}$. However, even if $g^{\psi_2}_{Q}(S(X_p)) \leq \kappa$, $\mathbf{g}^{\psi_2}(S(X_p)) \neq \mathbf{g}^{\sigma}(S(X_p))$ because $g^{\psi_2}_{sto}(S(X_p)) > g^{\psi_1}_{sto}(S(X_p))$.

Thus we show that $\psi_2(S(X_p))$ can't be in $\mathscr{P}_{\sigma}$. Since $\psi_2(S(X_p))$ can be any path from $S(X_p)$ to S_g passing through $\rho_2(s)$, pruning $\rho_2(s)$ does not prune any path from $\mathscr{P}_{\sigma}$. Also, since all of the above statements hold true for any $s \in G_s$, pruning all paths from $S(X_p)$ to S_g passing through $\rho_2(s)$ $\forall s \in G_s$ also does not prune any path from $\mathscr{P}_{\sigma}$. □

A.2 Appendix B: Complete Pseudocode of FAST-PPCP

In this section of the appendix, we present the complete pseudocode of FAST-PPCP. We first present pseudocode of the FAST-PPCP main function. We then present the pseudocode of the each individual procedure that is a component of the main function.

Pseudocode 4 FAST-PPCP Main function

```
 1: procedure MAIN()
 2:     N = 1; π̂_i = ∅
 3:     V*_L(X_st) ← run PPCP iterations till X_st is a PPCP pivot for the first N times;
 4:     X_p ← X_st; V̂^{π̂_i}(X_st) ← V*_ut(X_st);
 5:     while X_p ≠ NULL do
 6:         sol ←COMPUTEBSPATH(X_p, α, V*_L(X_st), P(X_p|X_st, π̂_i));
 7:         UPDATEMDPREVERSE(X_p)
 8:         if ¬sol then
 9:             if X_p = X_st then
10:                 N ← N + 1 and Goto Line 3 to increase V*_L(X_st);
11:             else
12:                 X_p ← COMPUTESAFEX(X_st)
13:             end if
14:         else
15:             π̂_i^bs ←UPDATEMDP(X_p)
16:             X_p, P(X_p|X_st, π̂_i^bs) ←COMPUTEPOLICYDEVOIDX()
17:             π̂_i ← π̂_i^bs
18:         end if
19:     end while
20: end procedure
```

Pseudocode 5

procedure COMPUTEBSPATH($X_p, \alpha, V_L^*(X_{st}), P(X_p|X_{st}, \hat{\pi}_i)$)
  $n_g = [S_g, 0]$; besta(n_g) = NULL;
  $g_{sto}(n_g) = 0$;
  $V_u^*(X_p) = \infty$; $OPEN = \emptyset$
  Insert n_g in $\Delta_{dom}(S_g)$ [1]
  Insert S_g in $OPEN$ with priority $g_{sto}(S_g) + h_c(S(X_p), S_g)$;
  while 1 **do**
    if $OPEN$ is empty **then** ▷ Policy Growth failed
      $\hat{V}(X_p) \leftarrow V_u^*(X_p)$ updated during this search, $\hat{\pi}_i(X_p) =$ NULL
      and **return** FALSE
    end if
    Remove search-state n with the smallest $g_{sto}(n) + h_c(S(X_p), s(n))$
    in $OPEN$
    if $s(n) = S(X_p)$ **then**
      $\hat{V}^{\rho_c^{bel}}(X_p) \leftarrow \hat{V}^{\rho_c}(S(X_p))$, $\hat{V}$ component of $n =$
      $[S(X_p), \hat{V}^{\rho_c}(S(X_p))]$
      Update $V_u^*(X_p) \leftarrow \hat{V}^{\rho_c^{bel}}(X_p))$ if $\hat{V}^{\rho_c^{bel}}(X_p)) < V_u^*(X_p)$
      Update $\hat{V}^{\hat{\pi}_i^c}(X_{st})$ by replacing old $\hat{V}(X_p)$ with $\hat{V}^{\rho_c^{bel}}(X_p)$, as
      in Eq. 5.8
      if $\hat{V}^{\hat{\pi}_i^c}(X_{st}) \leq \alpha V_L^*(X_{st})$ **then** ▷ Policy growth is successful
        $\hat{V}^{\hat{\pi}_i^{bs}}(X_{st}) \leftarrow \hat{V}^{\hat{\pi}_i^c}(X_{st})$
        $\hat{V}(X_p) \leftarrow \hat{V}^{\rho_c^{bel}}(X_p)$
        return TRUE
      end if
    end if
    $s \leftarrow s(n)$
    for each action $a \in A(s')$ and predecessor s' s.t $s = S(succ(X', a)^b)$ where $X' = [s', H(X_p)]$
      compute $\hat{V}^{\rho\prime}(s')$ using Eq. 5.1 with $X' = [s', H(X_p)]$
      search-predecessor $n' = [s', \hat{V}^{\rho\prime}(s')]$
      if a is a stochastic action **then**
        $g_{sto}(n') = g_{sto}(n) + |E|.c_{max}$;
      else
        $g_{sto}(n') = g_{sto}(n) + c_{max}$;
      end if
      if n' not visited before **or**¬ISDOMINATED(n') **then**
        Insert n' in $\Delta_{dom}(s')$
        Insert n' in $OPEN$ with priority $g_{sto}(n') + h_c(S(X_p), s')$
        and *besta* as a;
      end if
    end for
  end while
end procedure
procedure ISDOMINATED(n')
  return $(g_{sto}(n) \geq g_{sto}(n')$ **and** $\hat{V}(n) \geq \hat{V}(n'))$ for any
  $n \in \Delta_{dom}(s(n'))$ ▷ returns true if n' is dominated by an $n \in$ list of
  $\Delta_{dom}(s(n'))$ undominated search-states of $s(n')$
end procedure

Pseudocode 6

procedure UPDATEMDP(X_p)
  $X \leftarrow X_p$
  while $S(X) \neq S_g$ **do**
    $\hat{V}(X) \leftarrow \hat{V}^{\rho^{bel}_{bs}}(S(X))$, $\hat{\pi}^{\mathsf{bs}}_i(X) \leftarrow besta(S(X))$
    $X \leftarrow succ(X, \hat{\pi}^{\mathsf{bs}}_i(X))^b$ ▷ $\hat{\pi}^{\mathsf{bs}}_i(X)$ gives policy at X
  end whilereturn $\hat{\pi}^{\mathsf{bs}}_i$
end procedure
procedure UPDATEMDPREVERSE(X_p)
  $X \leftarrow X_p$
  while X has a predecessor **do**
    $X' \leftarrow predecessor(X)$ on current policy
    $\hat{V}(X') \leftarrow \mathbb{E}_{X \in succ(X', \hat{\pi}^{\mathsf{bs}}_i(X'))}(c(S(X'), a, S(X)) + \hat{V}(X)$
    $X \leftarrow X'$
  end while
end procedure
procedure COMPUTEPOLICYDEVOIDX
  Find X on current $\hat{\pi}^{\mathsf{bs}}_i$ s.t $S(X) \neq S_g$ **and**$besta(X) =$ `NULL`
  If X not found then **return** `NULL`else compute $P(X|X_{st}, \hat{\pi}^{bs}_i)$ and **return** X
end procedure

Pseudocode 7

procedure COMPUTESAFEX
  $X_e \leftarrow$ any non-preferred outcome on $\hat{\pi}_i$ that has an existing primary
  branch from it; $X \leftarrow X_e$
  while (**do**)
    if $\hat{\pi}_i(X)$ is deterministic or every non-preferred outcome of
      $succ(X, \hat{\pi}_i(X))$ is policy-devoid **then**
      $X \leftarrow succ(X, \hat{\pi}_i(X))^b$ ▷ move down on the primary branch
      from X
      if $S(X) = S_g$ **then**
        $X_{safe} \leftarrow X_e$ and **return** X_{safe}
      end if
    else$X_e \leftarrow$ another non-preferred outcome on $\hat{\pi}_i$ having an existing
      primary branch from it; $X \leftarrow X_e$
    end if
  end while
end procedure

Bibliography

[1] Sandip Aine, Siddharth Swaminathan, Venkatraman Narayanan, Victor Hwang, and Maxim Likhachev. Multi-heuristic a. *The International Journal of Robotics Research*, 35(1-3):224–243, 2016. 1.1, 2.2, 3.1.1

[2] Srinivas Akella and Matthew T Mason. Parts orienting with partial sensor information. In *Proceedings. 1998 IEEE International Conference on Robotics and Automation (Cat. No. 98CH36146)*, volume 1, pages 557–564. IEEE, 1998. 3.2.5

[3] Alexandre Alahi, Kratarth Goel, Vignesh Ramanathan, Alexandre Robicquet, Li Fei-Fei, and Silvio Savarese. Social lstm: Human trajectory prediction in crowded spaces. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 961–971, 2016. 7.3.1, 7.5.1

[4] Alexandre Albore, Héctor Palacios, and Hector Geffner. A translation-based approach to contingent planning. In *Twenty-First International Joint Conference on Artificial Intelligence*, 2009. 3.2.6

[5] Christopher Amato, George Konidaris, Gabriel Cruz, Christopher A Maynor, Jonathan P How, and Leslie P Kaelbling. Planning for decentralized control of multiple robots under uncertainty. In *2015 IEEE international conference on robotics and automation (ICRA)*, pages 1241–1248. IEEE, 2015. 3.2.5

[6] Saeid Amiri, Suhua Wei, Shiqi Zhang, Jivko Sinapov, Jesse Thomason, and Peter Stone. Multi-modal predicate identification using dynamically learned robot controllers. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI-18)*, 2018. 3.2.5

[7] Nicholas Armstrong-Crews and Manuela Veloso. An approximate algorithm for solving oracular pomdps. In *2008 IEEE International Conference on Robotics and Automation*, pages 3346–3352. IEEE, 2008. 3.2.4

[8] Alper Aydemir, Andrzej Pronobis, Moritz Göbelbecker, and Patric Jensfelt. Active visual object search in unknown environments using uncertain semantics. *IEEE Transactions on Robotics*, 29(4):986–1002, 2013. 3.2.5

[9] Fahiem Bacchus and Qiang Yang. The downward refinement property. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence - Volume*

1, IJCAI'91, pages 286–292, San Francisco, CA, USA, 1991. Morgan Kaufmann Publishers Inc. ISBN 1-55860-160-0. URL http://dl.acm.org/citation.cfm?id=1631171.1631214. 3.1.3, 4.1

[10] Christer Bäckström and Peter Jonsson. Bridging the gap between refinement and heuristics in abstraction. In *IJCAI*, pages 2261–2267, 2013. 3.1.3, 4.1

[11] Marcel Ball and Robert C Holte. The compression power of symbolic pattern databases. In *ICAPS*, pages 2–11, 2008. 2.3

[12] Andrew G Barto, Steven J Bradtke, and Satinder P Singh. Learning to act using real-time dynamic programming. *Artificial intelligence*, 72(1-2):81–138, 1995. 3.2.1

[13] Piergiorgio Bertoli, Alessandro Cimatti, Marco Roveri, and Paolo Traverso. Planning in nondeterministic domains under partial observability via symbolic model checking. In *IJCAI*, volume 2001, pages 473–478, 2001. 3.2.6

[14] Blai Bonet. Solving large POMDPs using real time dynamic programming. In *In Proc. AAAI Fall Symp. on POMDPs*. Citeseer, 1998. 3.2.1, 3.2.6

[15] Blai Bonet and Hector Geffner. Solving pomdps: Rtdp-bel vs. point-based algorithms. In *IJCAI*, pages 1641–1646. Pasadena CA, 2009. 5.5

[16] Ronen I Brafman and Carmel Domshlak. Structure and complexity in planning with unary operators. *Journal of Artificial Intelligence Research*, 18:315–349, 2003. 2.3, 3.2.6

[17] Adam Bry and Nicholas Roy. Rapidly-exploring random belief trees for motion planning under uncertainty. In *2011 IEEE international conference on robotics and automation*, pages 723–730. IEEE, 2011. 3.2.5

[18] Matthew Budd, Paul Duckworth, Nick Hawes, and Bruno Lacerda. Mission planning in unknown environments as bayesian reinforcement learning. 3.2.4

[19] Vadim Bulitko, Nathan R Sturtevant, Jieshan Lu, and Timothy Yau. Graph abstraction in real-time heuristic search. *J. Artif. Intell. Res.(JAIR)*, 30:51–100, 2007. 4.1

[20] Alberto Camacho, Christian Muise, and Sheila McIlraith. From fond to robust probabilistic planning: Computing compact policies that bypass avoidable deadends. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 26, 2016. 3.2.6

[21] Ishani Chatterjee, Maxim Likhachev, Ashwin Khadke, and Manuela Veloso. Speeding up search-based motion planning via conservative heuristics. Tech. Report, The Robotics Institute, Carnegie Mellon University., 2019. 4.2.2

[22] Ishani Chatterjee, Maxim Likhachev, Ashwin Khadke, and Manuela Veloso. Speeding up search-based motion planning via conservative heuristics. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 29, pages 674–679, 2019. 8.1

[23] Ishani Chatterjee, Tushar Kusnur, and Maxim Likhachev. Fast bounded suboptimal probabilistic planning with clear preferences on missing information. In *Proceedings of the International Symposium on Combinatorial Search*, volume 12, pages 37–45, 2021. 6.1, 8.1

[24] Ishani Chatterjee, Yash Oza, Maxim Likhachev, and Manuela Veloso. Search-based planning with learned behaviors for navigation among pedestrians. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 7953–7960. IEEE, 2021. 8.1

[25] Changan Chen, Yuejiang Liu, Sven Kreiss, and Alexandre Alahi. Crowd-robot interaction: Crowd-aware robot navigation with attention-based deep reinforcement learning. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 6015–6022. IEEE, 2019. 7.3.1, 7.5.1

[26] Min Chen, Emilio Frazzoli, David Hsu, and Wee Sun Lee. Pomdp-lite for robust robot planning under uncertainty. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5427–5433. IEEE, 2016. 3.2.4

[27] Joseph C Culberson and Jonathan Schaeffer. Searching with pattern databases. In *Conference of the Canadian Society for Computational Studies of Intelligence*, pages 402–416. Springer, 1996. 2.3

[28] Joseph C Culberson and Jonathan Schaeffer. Pattern databases. *Computational Intelligence*, 14(3):318–334, 1998. 2.3, 3.1.3, 4.1

[29] Nikhil Chavan Dafle, Alberto Rodriguez, Robert Paolini, Bowei Tang, Siddhartha S Srinivasa, Michael Erdmann, Matthew T Mason, Ivan Lundberg, Harald Staab, and Thomas Fuhlbrigge. Extrinsic dexterity: In-hand manipulation with external forces. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1578–1585. IEEE, 2014. 3.2.5

[30] Stefan Edelkamp. Automated creation of pattern database search heuristics. In *International Workshop on Model Checking and Artificial Intelligence*, pages 35–50. Springer, 2006. 2.3

[31] Stefan Edelkamp. *Symbolic shortest path planning*. Dekanat Informatik, Univ., 2007. 2.3

[32] Stefan Edelkamp. Planning with pattern databases. In *Sixth European Conference on Planning*, 2014. 2.3

[33] Clemens Eppner, Raphael Deimel, José Alvarez-Ruiz, Marianne Maertens, and Oliver Brock. Exploitation of environmental constraints in human and robotic grasping. *The International Journal of Robotics Research*, 34(7):1021–1038, 2015. 3.2.5

[34] Ariel Felner, Uzi Zahavi, Jonathan Schaeffer, and Robert C Holte. Dual lookups in

pattern databases. In *IJCAI*, pages 103–108, 2005. 2.3

[35] Ariel Felner, Richard E Korf, Ram Meshulam, and Robert C Holte. Compressed pattern databases. *Journal of Artificial Intelligence Research*, 30:213–247, 2007. 2.3

[36] Dave Ferguson, Anthony Stentz, and Sebastian Thrun. Pao for planning with hidden state. In *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04. 2004*, volume 3, pages 2840–2847. IEEE, 2004. 5.1

[37] Dave Ferguson, Maxim Likhachev, and Anthony Stentz. A guide to heuristic-based path planning. In *Proceedings of the international workshop on planning under uncertainty for autonomous systems, international conference on automated planning and scheduling (ICAPS)*, pages 9–18, 2005. 4.1

[38] Bernd Finkbeiner and A Podelski. Directed model checking with distance-preserving abstractions. *Proceedings of the 13th International SPIN*, 2006. 2.3

[39] Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pages 1050–1059, 2016. 7.3.1

[40] Neha Priyadarshini Garg, David Hsu, and Wee Sun Lee. Despot-alpha: Online pomdp planning with large state and observation spaces. In *Robotics: Science and Systems*, 2019. 3.2.3, 3.2.5

[41] Kenneth Y Goldberg. Orienting polygonal parts without sensors. *Algorithmica*, 10 (2):201–225, 1993. 3.2.5

[42] Juan Pablo Gonzalez and Anthony Stentz. Planning with uncertainty in position an optimal and efficient planner. In *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2435–2442. IEEE, 2005. 3.2.5

[43] Juan Pablo Gonzalez and Anthony Stentz. Planning with uncertainty in position using high-resolution maps. In *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pages 1015–1022. IEEE, 2007. 3.2.5

[44] Eric A Hansen and Shlomo Zilberstein. LAO*: A heuristic search algorithm that finds solutions with loops. *Artificial Intelligence*, 129(1-2):35–62, 2001. 3.2.1

[45] Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968. 2.2, 2.4.2, 5.2

[46] Patrik Haslum, Blai Bonet, Héctor Geffner, et al. New admissible heuristics for domain-independent planning. In *AAAI*, volume 5, pages 9–13, 2005. 2.3

[47] Patrik Haslum, Adi Botea, Malte Helmert, Blai Bonet, Sven Koenig, et al. Domain-independent construction of pattern database heuristics for cost-optimal planning. In *AAAI*, volume 7, pages 1007–1012, 2007. 2.3

[48] Patrik Haslum et al. Reducing accidental complexity in planning problems. In *IJCAI*,

pages 1898–1903, 2007. 3.1.3, 4.1

[49] Malte Helmert. The fast downward planning system. *Journal of Artificial Intelligence Research*, 26:191–246, 2006. 2.3, 3.1.3, 4.1

[50] Malte Helmert and Robert Mattmüller. Accuracy of admissible heuristic functions in selected planning domains. In *AAAI*, pages 938–943, 2008. 2.3

[51] Malte Helmert, Patrik Haslum, Jörg Hoffmann, et al. Flexible abstraction heuristics for optimal sequential planning. In *ICAPS*, pages 176–183, 2007. 2.3, 4.1

[52] István T Hernádvölgyi and Robert C Holte. Experiments with automatically created memory-based heuristics. In *International Symposium on Abstraction, Reformulation, and Approximation*, pages 281–290. Springer, 2000. 2.3

[53] Wassily Hoeffding. Probability inequalities for sums of bounded random variables. In *The collected works of Wassily Hoeffding*, pages 409–426. Springer, 1994. 7.4, 5

[54] Jörg Hoffmann and Bernhard Nebel. The FF planning system: fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14:253–302, 2001. 3.1.3, 3.2.6, 4.1

[55] Robert C Holte and István T Hernádvölgyi. A space-time tradeoff for memory-based heuristics. In *AAAI/IAAI*, pages 704–709. Citeseer, 1999. 2.3

[56] Robert C Holte, Taieb Mkadmi, Robert M Zimmer, and Alan J MacDonald. Speeding up problem solving by abstraction: A graph oriented approach. *Artificial Intelligence*, 85(1-2):321–361, 1996. 3.1.3, 4.1

[57] Robert C Holte, Maria B Perez, Robert M Zimmer, and Alan J MacDonald. Hierarchical a*: Searching abstraction hierarchies efficiently. In *AAAI/IAAI, Vol. 1*, pages 530–535. Citeseer, 1996. 2.3

[58] Armin Hornung, Daniel Maier, and Maren Bennewitz. M.: Search-based footstep planning. In *In: ICRA Workshop Progress Open Problems in Motion Planning Navigation for Humanoids*, 2013. 1.1, 1.3.3, 4.1, 4.4.2

[59] Myung Hwangbo, James J. Kuffner, and Takeo Kanade. Efficient two-phase 3d motion planning for small fixed-wing uavs. *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pages 1035–1041, 2007. 4.1

[60] Peter Jonsson and Christer Bäckström. State-variable planning under structural restrictions: Algorithms and complexity. *Artificial Intelligence*, 100(1-2):125–176, 1998. 2.3

[61] Andreas Junghanns and Jonathan Schaeffer. Domain-dependent single-agent search enhancements. In *IJCAI*, pages 570–577, 1999. 2.1

[62] Leslie Pack Kaelbling, Michael L Littman, and Anthony R Cassandra. Planning and acting in partially observable stochastic domains. *Artificial intelligence*, 101(1-2): 99–134, 1998. 5.1, 6.1

[63] Michael Katz and Carmel Domshlak. Optimal additive composition of abstraction-based admissible heuristics. In *ICAPS*, pages 174–181, 2008. 2.3

[64] Michael Katz and Carmel Domshlak. Implicit abstraction heuristics. *Journal of Artificial Intelligence Research*, 39:51–126, 2010. 2.3

[65] Sung-Kyun Kim, Oren Salzman, and Maxim Likhachev. Pomhdp: Search-based belief space planning using multiple heuristics. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 29, pages 734–744, 2019. 3.2.1, 3.2.5

[66] Craig A Knoblock. Learning abstraction hierarchies for problem solving. In *AAAI*, pages 923–928, 1990. 3.1.3

[67] Craig A Knoblock. Search reduction in hierarchical problem solving. In *AAAI*, volume 91, pages 686–691, 1991. 3.1.3

[68] Mykel J Kochenderfer. *Decision making under uncertainty: theory and application*. MIT press, 2015. 5.1, 6.1

[69] Richard E Korf. Finding optimal solutions to rubik's cube using pattern databases. In *AAAI/IAAI*, pages 700–705, 1997. 2.1, 2.3

[70] Richard E Korf and Larry A Taylor. Finding optimal solutions to the twenty-four puzzle. In *Proceedings of the national conference on artificial intelligence*, pages 1202–1207. Citeseer, 1996. 2.1

[71] Hanna Kurniawati and Vinay Yadav. An online pomdp solver for uncertainty planning in dynamic environment. In *Robotics Research*, pages 611–629. Springer, 2016. 3.2.3

[72] Hanna Kurniawati, David Hsu, and Wee Sun Lee. SARSOP: Efficient point-based pomdp planning by approximating optimally reachable belief spaces. In *Robotics: Science and systems*, volume 2008. Zurich, Switzerland., 2008. 3.2.2, 5.5, 5.6

[73] Aleksandr Kushleyev and Maxim Likhachev. Time-bounded lattice for efficient planning in dynamic environments. In *2009 IEEE International Conference on Robotics and Automation*, pages 1662–1668. IEEE, 2009. 3.2.5

[74] Tushar Kusnur, Shohin Mukherjee, Dhruv Mauria Saxena, Tomoya Fukami, Takayuki Koyama, Oren Salzman, and Maxim Likhachev. A planning framework for persistent, multi-uav coverage with global deconfliction. In *Field and Service Robotics*, pages 459–474. Springer, 2021. 1.1

[75] Ugur Kuter, Dana Nau, Elnatan Reisner, and Robert Goldman. Conditionalization: Adapting forward-chaining planners to partially observable environments. In *ICAPS 2007—workshop on planning and execution for real-world systems*, 2007. 3.2.6

[76] David Lenz, Markus Rickert, and Alois Knoll. Heuristic search in belief space for motion planning under uncertainties. In *2015 IEEE/RSJ International Conference*

on Intelligent Robots and Systems (IROS), pages 2659–2665. IEEE, 2015. 3.2.5

[77] Jue Kun Li, David Hsu, and Wee Sun Lee. Act to see and see to act: Pomdp planning for objects search in clutter. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5701–5707. IEEE, 2016. 3.2.5

[78] Maxim Likhachev and Dave Ferguson. Planning long dynamically feasible maneuvers for autonomous vehicles. *The International Journal of Robotics Research*, 28 (8):933–945, 2009. 1.1

[79] Maxim Likhachev and Anthony Stentz. Ppcp: Efficient probabilistic planning with clear preferences in partially-known environments. In *Proceedings of the National Conference on Artificial Intelligence*, volume 21, page 860. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2006. 1.3.2, 2.4.2, 5.1, 7.3.3

[80] Maxim Likhachev and Anthony Stentz. Probabilistic planning with clear preferences on missing information. *Artificial Intelligence*, 173(5-6):696–721, 2009. 1.1, 2.4, 2.4.1, 5.1, 5.2, 5.3.1, 6.1, 6.2

[81] Maxim Likhachev, Geoff Gordon, and Sebastian Thrun. Ara*: Anytime a* with provable bounds on sub-optimality. sl: Advances in neural information processing systems, 2004. 2.2, 3.1.1

[82] Carlos Linares López. Multi-valued pattern databases. In *ECAI 2008*, pages 540–544. IOS Press, 2008. 2.3

[83] Sikang Liu, Kartik Mohta, Nikolay Atanasov, and Vijay Kumar. Search-based motion planning for aggressive flight in se (3). *IEEE Robotics and Automation Letters*, 3(3):2439–2446, 2018. 1.1, 4.1

[84] Brian MacAllister, Jonathan Butzke, Alex Kushleyev, Harsh Pandey, and Maxim Likhachev. Path planning for non-circular micro aerial vehicles in constrained environments. In *2013 ieee international conference on robotics and automation*, pages 3933–3940. IEEE, 2013. 1.1

[85] Christian Muise, Vaishak Belle, and Sheila McIlraith. Computing contingent plans via fully observable non-deterministic planning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 28, 2014. 3.2.6

[86] Venkatraman Narayanan and Maxim Likhachev. Task-oriented planning for manipulating articulated mechanisms under model uncertainty. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3095–3101. IEEE, 2015. 3.2.5

[87] Nils J Nilsson. Problem-solving methods in. *Artificial Intelligence*, 1971. 3.1.1

[88] Sylvie CW Ong, Shao Wei Png, David Hsu, and Wee Sun Lee. Pomdps for robotic tasks with mixed observability. In *Robotics: Science and Systems*, volume 5, page 4,

2009. 3.2.4

[89] Bo Pang and Robert C Holte. State-set search. In *Fourth Annual Symposium on Combinatorial Search*, 2011. 3.1.3, 4.1

[90] Judea Pearl. Heuristics: intelligent search strategies for computer problem solving. 1984. 3.1.3, 4.1

[91] Mark A Peot and David E Smith. Conditional nonlinear planning. In *Artificial Intelligence Planning Systems*, pages 189–197. Elsevier, 1992. 3.2.6

[92] Joelle Pineau, Geoff Gordon, Sebastian Thrun, et al. Point-based value iteration: An anytime algorithm for POMDPs. In *IJCAI*, volume 3, pages 1025–1032, 2003. 3.2.2, 8.2.3

[93] Joelle Pineau, Geoffrey Gordon, and Sebastian Thrun. Anytime point-based approximations for large pomdps. *Journal of Artificial Intelligence Research*, 27:335–380, 2006. 3.2.2

[94] Ira Pohl. First results on the effect of error in heuristic search. *Machine Intelligence*, 5:219–236, 1970. 2.2, 3.1.1

[95] Ira Pohl. Heuristic search viewed as path finding in a graph. *Artificial intelligence*, 1 (3-4):193–204, 1970. 4.1

[96] Samuel Prentice and Nicholas Roy. The belief roadmap: Efficient planning in belief space by factoring the covariance. *The International Journal of Robotics Research*, 28(11-12):1448–1465, 2009. 3.2.5

[97] Armand E Prieditis. Machine discovery of effective admissible heuristics. *Machine learning*, 12(1):117–141, 1993. 2.3

[98] Louise Pryor and Gregg Collins. Planning for contingencies: A decision-based approach. *Journal of Artificial Intelligence Research*, 4:287–339, 1996. 3.2.6

[99] Jussi Rintanen. Complexity of planning with partial observability. In *ICAPS*, volume 4, pages 345–354, 2004. 3.2.6

[100] Stéphane Ross, Joelle Pineau, Sébastien Paquet, and Brahim Chaib-Draa. Online planning algorithms for pomdps. *Journal of Artificial Intelligence Research*, 32: 663–704, 2008. 3.2.2

[101] Earl D Sacerdoti. Planning in a hierarchy of abstraction spaces. *Artificial intelligence*, 5(2):115–135, 1974. 3.1.3

[102] Konstantin M Seiler, Hanna Kurniawati, and Surya PN Singh. An online and approximate solver for pomdps with continuous action space. In *2015 IEEE international conference on robotics and automation (ICRA)*, pages 2290–2297. IEEE, 2015. 3.2.3

[103] Guy Shani, Joelle Pineau, and Robert Kaplow. A survey of point-based POMDP solvers. *Autonomous Agents and Multi-Agent Systems*, 27(1):1–51, 2013. 3.2.2

[104] David Silver and Joel Veness. Monte-carlo planning in large pomdps. In *Advances in neural information processing systems*, pages 2164–2172, 2010. 3.2.2, 3.2.3

[105] Trey Smith and Reid Simmons. Heuristic search value iteration for POMDPs. *arXiv preprint arXiv:1207.4166*, 2012. 3.2.2, 5.5, 5.6

[106] Trey Smith and Reid Simmons. Point-based POMDP algorithms: Improved analysis and implementation. *arXiv preprint arXiv:1207.1412*, 2012. 3.2.2, 5.6

[107] Adhiraj Somani, Nan Ye, David Hsu, and Wee Sun Lee. Despot: Online pomdp planning with regularization. In *Advances in neural information processing systems*, pages 1772–1780, 2013. 3.2.2, 3.2.3

[108] Zachary Sunberg and Mykel Kochenderfer. Online algorithms for pomdps with continuous state, action, and observation spaces. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 28, 2018. 3.2.3

[109] Florent Teichteil-Königsbuch, Ugur Kuter, and Guillaume Infantes. Incremental plan aggregation for generating policies in mdps. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1*, pages 1231–1238, 2010. 3.2.6

[110] Jordan Tyler Thayer and Wheeler Ruml. Bounded suboptimal search: A direct approach using inadmissible estimates. In *IJCAI*, pages 674–679, 2011. URL https://doi.org/10.5591/978-1-57735-516-8/IJCAI11-119. 3.1.2

[111] Jur van den Berg, Stephen J Guy, Jamie Snape, Ming C Lin, and Dinesh Manocha. Rvo2 library: Reciprocal collision avoidance for real-time multi-agent simulation, 2011. 7.5.1

[112] William Vega-Brown and Nicholas Roy. Admissible abstractions for near-optimal task and motion planning. *arXiv preprint arXiv:1806.00805*, 2018. 3.1.3, 4.1

[113] Arthur Wandzel, Yoonseon Oh, Michael Fishman, Nishanth Kumar, Lawson LS Wong, and Stefanie Tellex. Multi-object search using object-oriented pomdps. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 7194–7200. IEEE, 2019. 3.2.5

[114] Daniel S Weld, Corin R Anderson, and David E Smith. Extending graphplan to handle uncertainty & sensing actions. In *Aaai/iaai*, pages 897–904, 1998. 3.2.6

[115] Christopher Makoto Wilt and Wheeler Ruml. When does weighted a* fail? In *SOCS*, pages 137–144, 2012. 4.1

[116] Yuchen Xiao, Sammie Katt, Andreas ten Pas, Shengjian Chen, and Christopher Amato. Online planning for target object search in clutter under partial observability. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8241–8247. IEEE, 2019. 3.2.5

[117] Nan Ye, Adhiraj Somani, David Hsu, and Wee Sun Lee. Despot: Online pomdp

planning with regularization. *Journal of Artificial Intelligence Research*, 58:231–266, 2017. 5.6

[118] Kaiyu Zheng, Yoonchang Sung, George Konidaris, and Stefanie Tellex. Multi-resolution pomdp planning for multi-object search in 3d. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2022–2029. IEEE, 2021. 3.2.5

[119] Jiaji Zhou, Robert Paolini, Aaron M Johnson, J Andrew Bagnell, and Matthew T Mason. A probabilistic planning framework for planar grasping under uncertainty. *IEEE Robotics and Automation Letters*, 2(4):2111–2118, 2017. 3.2.5