Fast Bounded Suboptimal Probabilistic Planning with Clear Preferences on Missing Information

Ishani Chatterjee, Tushar Kusnur, Maxim Likhachev
The Robotics Institute, Carnegie Mellon University
{ichatter, tkusnur, mlikhach}@andrew.cmu.edu

Abstract

In the real-world, robots must often plan despite the environment being partially known. This frequently necessitates 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 focuses on a specific subset of such planning problems wherein there exist clear preferences over the actual values of missing information (Likhachev and Stenz 2009). PPCP exploits the existence and knowledge of these preferences to perform provably optimal planning via a series of deterministic A*-like searches over particular instantiations of the environment. Such decomposition leads to much better scalability with respect to both the size of a problem and the amount of missing information in it. The run-time of PPCP however is a function of the number of searches it has to run until convergence. In this paper, we make a key observation that the number of searches PPCP has to run can be dramatically decreased if each search computes a plan that minimizes the amount of missing information it relies upon. 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. We present FAST-PPCP with its theoretical analysis, compare with common alternative approaches to planning under uncertainty over missing information, and experimentally show that FAST-PPCP provides substantial gain in runtime over other approaches while incurring little loss in solution quality.

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) (Kaelbling, Littman, and Cassandra 1998; Kochenderfer 2015).

As noted in (Likhachev and Stentz 2009), 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 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 (Likhachev and Stentz 2009).

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.

Related Work

Planning for CP-PS problems is a special class of POMDP planning. As such a multitude of POMDP solvers are applicable. We summarize three main groups.

Point-based Methods. Point-based methods use a vector set of sampled points to represent the value function and restrict value function updates to a subset of the belief space (Shani, Pineau, and Kaplow 2013). PBVI operates only on reachable beliefs (Pineau et al. 2003); HSVI and HSVI2 utilize a heuristic strategy to focus on regions with high uncertainty (Smith and Simmons 2012); SARSOP uses heuristic exploration to converge the to a subset of points optimally reachable from the start, often outperforming HSVI2 (Kurniawati, Hsu, and Lee 2008).

Heuristic-search Based Methods. RTDP (Barto, Bradtke, and Singh 1995) 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 (Bonet 1998). LAO* generalizes heuristic search to solving belief MDPs (Hansen and Zilberstein 2001). PO-PRP (Muise, Belle, and McIlraith 2014) and ProbPRP (Camacho, Muise, and McIlraith 2016) use a series of calls to classical planners to iteratively construct and refine a policy for planning in a partially observable environment. However, these two methods neither exploit clear preferences nor aim to provide any guarantee on the value of the computed policy.

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 (Ross et al. 2008). However, since these solvers have a finite, receding horizon, they are subject to getting stuck in local minima. Popular Monte Carlo simulation based online solvers include POMCP (Silver and Veness 2010) and DESPOT (Somani et al. 2013).

PPCP is a solver that is specifically designed for CP-PS problems. It outperforms optimal solvers such as RTDP-BEL, LAO*, PAO* (Ferguson, Stentz, and Thrun 2004), and HSVI2 when applied to CP-PS problems. FAST-PPCP also focuses on CP-PS problems but runs much faster than PPCP—as we show experimentally— while guaranteeing the value of the policy to be within a user-specified factor of the optimal value (bounded suboptimality).

Problem Definition, Assumptions and Background

Example Domain. We use the problem of robot navigation in partially known environments as a running example. Consider in Fig. 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.

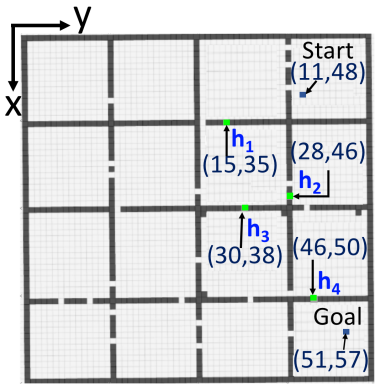

Figure 1: Robot navigation between rooms with doors whose status (open/closed) is unknown.

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 ($h^{S(X),a} =$ `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 Tab. 1 hold; FAST-PPCP can be applied to any domain where they hold.

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

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	The true status of an unknown variable becomes known immediately (perfect sensing assumption).
A4	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.
A5	The variables in H are independent of each other and therefore $P(H) = \prod_{i=1}^{\|H\|} P(h_i)$.

Table 1: Assumptions

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 2: Definitions

assumption A5, X concisely represents a probability distribution over all possible states. [**Ex.** h_1 in Fig. 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 unknown variable's clearly preferred value is known beforehand. [**Ex.** we prefer that a hidden cell is free 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 (Likhachev and Stentz 2009) 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 free] 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 Tab. 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* (Hart, Nilsson, and Raphael 1968)-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 tab. 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.

Fast Bounded Suboptimal PPCP

Operation and Intuition in a Nutshell

In **Fig. 2**, we present a block diagram of the FAST-PPCP algorithm showing how its algorithmic components/blocks interact. FAST-PPCP iteratively develops partial policies into a full policy π_f rooted at X_{st} (definitions of partial and full policy are in Tab. 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 $V^*(X_{st})$

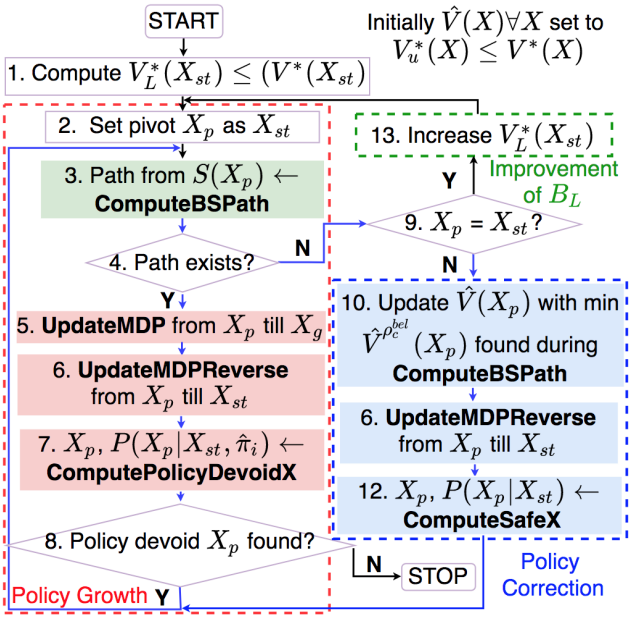

Figure 2: FAST-PPCP block diagram of Main function (Complete pseudocode in Appendix A in (Chatterjee 2021))

(**Block 1.**) which we describe at the end of this section. FAST-PPCP ensures that the full policy it computes satisfies $V^{\pi_f}(X_{st}) \leq \alpha V_L^*(X_{st}) \triangleq \mathsf{B}^{\mathsf{L}}$. We refer to such a policy as a B^{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** B^{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 $\hat{V}^{\hat{\pi}_i}(X_{st})$.

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 B^{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 $\hat{V}^{\hat{\pi}_i^{\mathsf{c}}}(X_{st}) > \mathsf{B}^{\mathsf{L}}$ (rejection condition), no full B^{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 $\hat{V}^{\hat{\pi}_i^{\mathsf{c}}}(X_{st}) \leq \mathsf{B}^{\mathsf{L}}$ (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 $\mathtt{B}^{\mathsf{L}}$-*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 $\hat{\pi}^{+\mathsf{x}}$ will surely have $\hat{V}^{\hat{\pi}^{+\mathsf{x}}}(X_{st}) \geq \hat{V}^{\hat{\pi}_i}(X_{st}) > \mathtt{B}^{\mathsf{L}}$ (i.e., no $\mathtt{B}^{\mathsf{L}}$-bounded policy can develop from $\rho_{\mathsf{safe}}^{\mathsf{bel}}$ 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 $\mathtt{B}^{\mathsf{L}}$: It increases $\mathtt{B}^{\mathsf{L}}$ 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 $\mathtt{B}^{\mathsf{L}}$. Increase in $V_L^*(X_{st})$ is guaranteed in (Likhachev and Stentz 2009).

Implementation Details of Some Functions

We refer to X_p, the belief state from which FAST-PPCP starts an iteration, as a *pivot*. [1] We assume familiarity with A^* search, g-, f- and h-values.

COMPUTEBSPATH Implementation. Pseudocode of Procedure COMPUTEBSPATH is shown in Pseudocode 1. 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 $\hat{V}^{\hat{\pi}_i^{bs}}(X_{st}) \leq \mathtt{B}^{\mathsf{L}}$—COMPUTEBSPATH is a backward A^*-like search run on an augmented deterministic graph $\mathcal{G}_{sto}^i = \{S_{sto}^i, E^i\}$. 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 $n \in S_{sto}^i$ 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 $\mathcal{G}_{sto}^i$, then explain $\hat{V}^{\rho}(s)$.

Edge-costs in $\mathcal{G}_{sto}^i$: We derive the edge-costs (Lemma 1-3, Appendix B in (Chatterjee 2021)) with the following aims: (1) In $\mathcal{G}_{sto}^i$, 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 $\mathcal{G}^i$. (2) Assume that a non-zero consistent heuristic function $h_c(S(X_p), s)$ is known for the original cost function in $\mathcal{G}^i$ [**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,

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

[1] Complete pseudocode in Appendix A in (Chatterjee 2021)

Pseudocode 1

1: **procedure** COMPUTEBSPATH($X_p, \alpha, V_L^*(X_{st}), P(X_p|X_{st}, \hat{\pi}_i)$)
2: $n_g = [S_g, 0]$; besta(n_g) = NULL;
3: $g_{sto}(n_g) = 0$;
4: $V_u^*(X_p) = \infty$; $OPEN = \emptyset$
5: Insert n_g in $\Delta_{dom}(S_g)$
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: $\hat{V}(X_p) \leftarrow V_u^*(X_p)$ updated during this search, $\hat{\pi}_i(X_p)$ = NULL
10: and **return** FALSE
11: Remove search-state n with the smallest $g_{sto}(n) + h_c(S(X_p), s(n))$
12: in $OPEN$
13: **if** $s(n) = S(X_p)$ **then**
14: $\hat{V}^{\rho_c^{bel}}(X_p) \leftarrow \hat{V}^{\rho_c}(S(X_p))$, $\hat{V}$ component of $n =$
15: $[S(X_p), \hat{V}^{\rho_c}(S(X_p))]$
16: Update $V_u^*(X_p) \leftarrow \hat{V}^{\rho_c^{bel}}(X_p))$ if $\hat{V}^{\rho_c^{bel}}(X_p)) < V_u^*(X_p)$
17: 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
18: in Eq. 2
19: **if** $\hat{V}^{\hat{\pi}_i^c}(X_{st}) \leq \alpha V_L^*(X_{st})$ **then** ▷ Policy growth is successful
20: $\hat{V}^{\hat{\pi}_i^{bs}}(X_{st}) \leftarrow \hat{V}^{\hat{\pi}_i^c}(X_{st})$
21: $\hat{V}(X_p) \leftarrow \hat{V}^{\rho_c^{bel}}(X_p)$
22: **return** TRUE
23: $s \leftarrow s(n)$
24: **for** each action $a \in A(s')$ and predecessor s' s.t $s = S(succ(X', a)^b)$
where $X' = [s', H(X_p)]$
25: compute $\hat{V}^{\rho'}(s')$ using Eq. 1 with $X' = [s', H(X_p)]$
26: search-predecessor $n' = [s', \hat{V}^{\rho'}(s')]$
27: **if** a is a stochastic action **then**
28: $g_{sto}(n') = g_{sto}(n) + |E|.c_{max}$;
29: **else**
30: $g_{sto}(n') = g_{sto}(n) + c_{max}$;
31: **if** n' not visited before **or** $\neg$ISDOMINATED(n') **then**
32: Insert n' in $\Delta_{dom}(s')$
33: Insert n' in $OPEN$ with priority $g_{sto}(n') + h_c(S(X_p), s')$
34: and $besta$ as a;
35: **procedure** ISDOMINATED(n')
36: **return** $(g_{sto}(n) \geq g_{sto}(n')$ **and** $\hat{V}(n) \geq \hat{V}(n'))$ for any
37: $n \in \Delta_{dom}(s(n'))$ ▷ returns true if n' is dominated by an $n \in$ list of
38: $\Delta_{dom}(s(n'))$ undominated search-states of $s(n')$

However, the cost c_{sto} of a path ρ_c from $S(X_p)$ to goal in $\mathcal{G}^i$ 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 $\mathcal{G}^i$ 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 action a taken on belief X'. Since ρ' maps to ρ'^{bel}, which is a primary branch, $S(X^b) = s$. 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:

$$\hat{V}^{\rho'}(s') = \sum_{Y \in succ(X',a) \setminus X^b} P(Y|X', a)(c(s', a, S(Y)) + \hat{V}(Y)) + P(X^b|X', a)(c(s', a, s) + \hat{V}^{\rho}(s)) \quad (1)$$

$\hat{V}^{\rho}(s)$ is the same as $\hat{V}^{\rho^{\mathsf{bel}}}(X^b)$ which is lower bound on the V-value of any policy from X^b containing ρ^{bel} (Lemma 6, Appendix B in (Chatterjee 2021)).

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 26). 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 Theorem 2). Since E^i is the same set of edges as in $\mathcal{G}^i$, 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 $g_{sto}(n) + h_c(n, S(X_p))$ (Line 12), where h_c is a consistent heuristic and $g_{sto}(n)$ is the cost of the path ρ from s to goal that corresponds to the state $n = [s, \hat{V}^{\rho}(s)]$ (Lines 3, 27-30).

$\hat{V}^{\hat{\pi}_i^c}(X_{st})$ *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* $\mathcal{G}^i$, 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 18):

$$\hat{V}^{\hat{\pi}_i^c}(X_{st}) \leftarrow \hat{V}^{\hat{\pi}_i}(X_{st}) + P(X_p|X_{st}, \hat{\pi}_i)(-\hat{V}(X_p) + \hat{V}^{\rho_c^{bel}}(X_p)) \quad (2)$$

where $P(X_p|X_{st}, \hat{\pi}_i)$ is the probability of reaching X_p following the current partial policy $\hat{\pi}_i$ from X_{st}. It then checks if $\hat{V}^{\hat{\pi}_i^c}(X_{st}) \leq \mathsf{B}^{\mathsf{L}}$. 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 $\hat{V}^{\hat{\pi}_i^c}(X_{st}) \leq \mathsf{B}^{\mathsf{L}}$ (Line 19). 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 $\mathcal{G}^i$.

COMPUTEBSPATH performs dominance checks (Line 31) through procedure ISDOMINATED (Pseudocode 1 Line 35). If a search-state n' is dominated by any previously seen search state n with $s(n) = s(n')$ (Line 36-38), the sub-graph

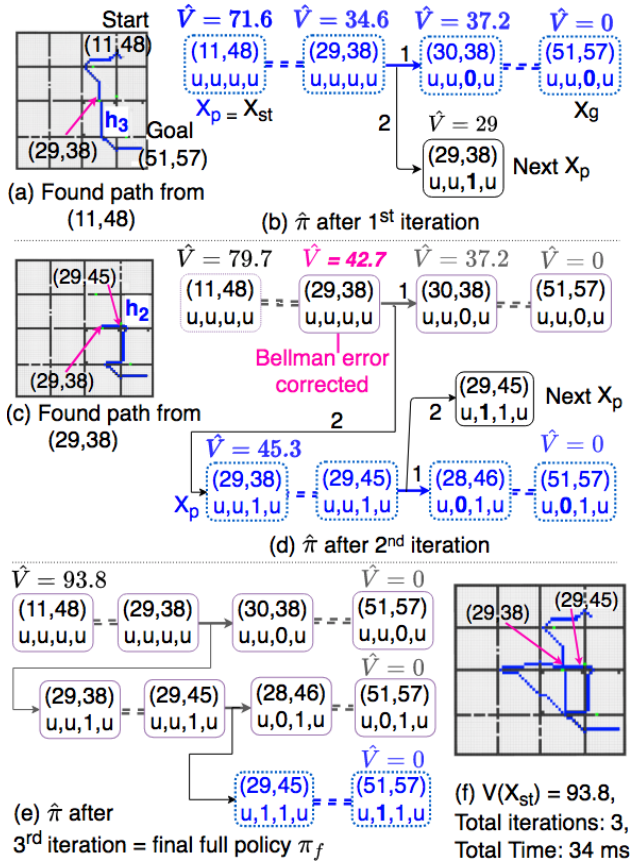

Figure 3: FAST-PPCP operation in [**Ex.**]. Numbers 1 and 2 near a stochastic transition (solid arrows in a partial policy) indicate transition costs. Outcomes of deterministic *move* actions have been omitted (visualized as dotted lines).

that can be generated from n' is pruned. This significantly increases search efficiency.

COMPUTESAFEX Implementation. As mentioned in **Policy Correction**, 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.

Working Example. In Fig. 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 $\mathcal{G}_0$ (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 (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 $\hat{V}^{\rho^{bel}_{bs}}(X_{st})$, the expected cost of the partial policy that corresponds to the primary branch ρ^{bel}_{bs} (dotted blue rectangles in 3(b)) that ρ_{bs} maps to.

The updated partial policy at the end of first iteration $\hat{\pi}_1 \triangleq \rho^{bel}_{bs}$, and $\hat{V}^{\hat{\pi}_1}(X_{st}) = \hat{V}^{\rho^{bel}_{bs}}(X_{st}) = 71.6$. UPDATEMDP updates $\hat{V}$ values of belief states along ρ^{bel}_{bs} from X_{st}(3(b) blue bold $\hat{V}$-values). COMPUTEPOLICYDEVOIDX searches for a policy-devoid outcome in $\hat{\pi}_1$ and finds the non-preferred outcome $[(29, 38); u, u, 1, u]$ (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 $\mathcal{G}_1$ (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 $n = [(29, 38), \hat{V}^{\rho^{bel}_{bs}}(X_p) = 45.3]$ that corresponds to the path ρ_{bs} (3(c)) and the primary branch ρ^{bel}_{bs} (dotted blue rectangles in 3(d)). For this ρ_{bs}, $\hat{V}^{\hat{\pi}^{\text{bs}}_1}(X_{st})$, the lower bound on the V-value of any policy that can develop from the updated partial policy $\hat{\pi}^{\text{bs}}_1$ if ρ^{bel}_{bs} is added to $\hat{\pi}_1$, is 79.7 which is within $\mathtt{B}^{\mathtt{L}}$. The value 79.7 is computed as in eq.2, 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 $\hat{V}^{\rho^{bel}_{bs}}(X_p) = 45.3$ (3(d) blue bold $\hat{V}$). In addition to updating $\hat{V}$-values along ρ^{bel}_{bs}, 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 $42.7 = 0.5 * (1 + 37.2) + 0.5 * (2 + 45.3)$ (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^{bel}_{bs} shown in 3(e) blue). Since no more policy-devoid outcomes exist in the updated partial policy (3(e)), FAST-PPCP terminates (full policy shown in 3(f) by superimposing paths found in the three iterations).

Theoretical Analysis[2]

Theorem 1. *Let $\mathbb{P}$ be the set of all paths in $\mathcal{G}^i$ (which is same as the set of paths in $\mathcal{G}^i_{sto}$) from $S(X_p)$ to S_g in a Fast-PPCP iteration i. For a path $\rho_c \in \mathbb{P}$, let $\hat{V}^{\hat{\pi}^c_i}(X_{st})$ be the $\hat{V}$ of the updated partial policy $\hat{\pi}^c_i$ from X_{st} assuming the primary branch ρ^{bel}_c gets added to $\hat{\pi}_i$. Let $c_{sto}(\rho_c(S(X_p)))$ be the cost of ρ_c in $\mathcal{G}^i_{sto}$.* COMPUTEBSPATH *upon termination computes a path $\rho_{bs} \in \mathbb{P}$ that satisfies:*

$$\rho_{bs} = \arg\min_{\rho_c \in \mathbb{P}} c_{sto}(\rho_c(S(X_p))) \; s.t \; \hat{V}^{\hat{\pi}^c_i}(X_{st}) \leq \mathcal{B}^{\mathsf{L}} \quad (3)$$

given ρ_{bs} exists.

Theorem 2. *($\mathcal{B}^{\mathsf{L}}$-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 $\leq \mathcal{B}^{\mathsf{L}}$, where $\mathcal{B}^{\mathsf{L}}$ is the lower bound on $\alpha V^*(X_{st})$ 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 $\pi_f \in \Pi_{bl}$.*

Theorem 3. *The full policy π_f starting from X_{st} computed by* FAST-PPCP *upon termination satisfies $V^{\pi_f}(X_{st}) \leq \mathcal{B}^{\mathsf{L}} \leq \alpha V^*(X_{st})$ and thus is bounded suboptimal.*

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 (Bonet and Geffner 2009)—which serves as a suboptimal baseline belief-MDP planner. We run experiments 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 3 (small environments) and 4 (large environments) are averaged over 40 planning instances. In small environments, FAST-PPCP with $\alpha = 1.5$ computes a policy in 3 iterations on average, with expected cost $\sim$ 1.03 times higher than the optimal, while being ~ 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.

[2]Full proofs/proof sketches in Appendix B in (Chatterjee 2021)

Algorithm	Time(s)	Exp. cost	Iterations
un. = 7; α = 1.5			
Fast-PPCP	**0.03 ± 0.05**	83 ± 12	**3 ± 3**
wRTDP-Bel	600 (timeout)	90 ± 10	237 ± 459
PPCP	1.10 ± 0.60	**80 ± 10**	139 ± 74
un. = 11; α = 1.5			
Fast-PPCP	**0.03 ± 0.04**	83 ± 11	**3 ± 2**
wRTDP-Bel	600 (timeout)	90 ± 11	162 ± 305
PPCP	0.75 ± 0.46	**79 ± 10**	90 ± 56
un. = 15; α = 1.5			
Fast-PPCP	**0.03 ± 0.06**	82 ± 12	**3 ± 6**
wRTDP-Bel	600 (timeout)	89 ± 11	171 ± 326
PPCP	1.17 ± 0.67	**79 ± 10**	141 ± 78
un. = 7; α = 2.0			
Fast-PPCP	**0.01 ± 0.003**	85 ± 12	**2 ± 0**
wRTDP-Bel	600 (timeout)	94 ± 5	71 ± 65
PPCP	1.10 ± 0.60	**80 ± 10**	139 ± 74
un. = 11; α = 2.0			
Fast-PPCP	**0.03 ± 0.04**	84 ± 12	**2 ± 0**
wRTDP-Bel	600 (timeout)	94 ± 5	67 ± 75
PPCP	0.75 ± 0.46	**79 ± 10**	90 ± 56
un. = 15; α = 2.0			
Fast-PPCP	**0.01 ± 0.003**	85 ± 13	**2 ± 0**
wRTDP-Bel	600 (timeout)	95 ± 5	81 ± 100
PPCP	1.17 ± 0.67	**79 ± 10**	141 ± 78

Table 3: Navigation (small environments)

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

Domain 2: RockSample. We chose the RockSample domain (Smith and Simmons 2012) 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. Appendix C in (Chatterjee 2021) details the transformation we formulate to convert discounted-reward CP-PS belief-MDPs into discounted-cost belief-MDPs to which PPCP and FAST-PPCP can be applied. We compare with HSVI2 (Smith and Simmons 2012) which, as shown in (Kurniawati, Hsu, and Lee 2008), performs better than SARSOP for the RockSample domain. Results are averaged over 3 environments each

Algorithm	Time(s)	Exp. reward	Iterations
RS(10,10)			
Fast-PPCP	$\mathbf{1.23 \pm 0.20}$	21 ± 25	$\mathbf{2 \pm 0}$
PPCP	4.21 ± 1.28	$\mathbf{51 \pm 20}$	79 ± 52
HSVI2	600 (timeout)	12 ± 1	2215 ± 184
RS(15,15)			
Fast-PPCP	$\mathbf{5.12 \pm 2.74}$	51 ± 120	$\mathbf{2.11 \pm 0.5}$
PPCP	44.13 ± 30.37	$\mathbf{70 \pm 150}$	203 ± 179
HSVI2	-	-	-

Table 5: RockSample (RS), with $\alpha = 1.1$ for Fast-PPCP

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 Tab. 5. 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 Appendix C in (Chatterjee 2021). 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 exhausts memory.

Conclusion and Limitations of FAST-PPCP

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 as well as some popular baselines. 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. Another limitation is the perfect sensing assumption which makes the scope of the problem narrower than POMDP planning.

Acknowledgements

This work was supported in part by ONR grant N00014-18-1-2775.

References

Barto, A. G.; Bradtke, S. J.; and Singh, S. P. 1995. Learning to act using real-time dynamic programming. *Artificial intelligence* 72(1-2): 81–138.

Bonet, B. 1998. Solving large POMDPs using real time dynamic programming. In *In Proc. AAAI Fall Symp. on POMDPs*. Citeseer.

Bonet, B.; and Geffner, H. 2009. Solving POMDPs: RTDP-Bel vs. Point-based Algorithms. In *IJCAI*, 1641–1646. Pasadena CA.

Camacho, A.; Muise, C.; and McIlraith, S. 2016. 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.

Chatterjee, I. 2021. Fast Bounded Suboptimal Probabilistic Planning with Clear Preferences on Missing Information: Proofs and Complete Pseudocode. Technical Report CMU-RI-TR-21-20, Carnegie Mellon University, Pittsburgh, PA.

Ferguson, D.; Stentz, A.; and Thrun, S. 2004. PAO for planning with hidden state. In *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04. 2004*, volume 3, 2840–2847. IEEE.

Hansen, E. A.; and Zilberstein, S. 2001. LAO*: A heuristic search algorithm that finds solutions with loops. *Artificial Intelligence* 129(1-2): 35–62.

Hart, P. E.; Nilsson, N. J.; and Raphael, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics* 4(2): 100–107.

Kaelbling, L. P.; Littman, M. L.; and Cassandra, A. R. 1998. Planning and acting in partially observable stochastic domains. *Artificial intelligence* 101(1-2): 99–134.

Kochenderfer, M. J. 2015. *Decision making under uncertainty: theory and application*. MIT press.

Kurniawati, H.; Hsu, D.; and Lee, W. S. 2008. SARSOP: Efficient point-based pomdp planning by approximating optimally reachable belief spaces. In *Robotics: Science and systems*, volume 2008. Zurich, Switzerland.

Likhachev, M.; and Stentz, A. 2009. Probabilistic planning with clear preferences on missing information. *Artificial Intelligence* 173(5-6): 696–721.

Muise, C.; Belle, V.; and McIlraith, S. 2014. Computing contingent plans via fully observable non-deterministic planning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 28.

Pineau, J.; Gordon, G.; Thrun, S.; et al. 2003. Point-based value iteration: An anytime algorithm for POMDPs. In *IJCAI*, volume 3, 1025–1032.

Ross, S.; Pineau, J.; Paquet, S.; and Chaib-Draa, B. 2008. Online planning algorithms for POMDPs. *Journal of Artificial Intelligence Research* 32: 663–704.

Shani, G.; Pineau, J.; and Kaplow, R. 2013. A survey of point-based POMDP solvers. *Autonomous Agents and Multi-Agent Systems* 27(1): 1–51.

Silver, D.; and Veness, J. 2010. Monte-Carlo planning in large POMDPs. In *Advances in neural information processing systems*, 2164–2172.

Smith, T.; and Simmons, R. 2012. Point-based POMDP algorithms: Improved analysis and implementation. *arXiv preprint arXiv:1207.1412* .

Somani, A.; Ye, N.; Hsu, D.; and Lee, W. S. 2013. DESPOT: Online POMDP planning with regularization. In *Advances in neural information processing systems*, 1772–1780.