Online Partial Conditional Plan Synthesis for POMDPs With Safe-Reachability Objectives: Methods and Experiments

Yue Wang, Abdullah Al Redwan Newaz, *Member, IEEE*, Juan David Hernández, *Senior Member, IEEE*, Swarat Chaudhuri, and Lydia E. Kavraki, *Fellow, IEEE*

***Abstract*—The framework of partially observable Markov decision processes (POMDPs) offers a standard approach to model uncertainty in many robot tasks. Traditionally, POMDPs are formulated with optimality objectives. In this article, we study a different formulation of POMDPs with *Boolean objectives*. For robotic domains that require a correctness guarantee of accomplishing tasks, Boolean objectives are natural formulations. We investigate the problem of POMDPs with a common Boolean objective: *safe reachability*, requiring that the robot eventually reaches a goal state with a probability above a threshold while keeping the probability of visiting unsafe states below a different threshold. Our approach builds upon the previous work that represents POMDPs with Boolean objectives using symbolic constraints. We employ a satisfiability modulo theories (SMTs) solver to efficiently search for solutions, i.e., policies or conditional plans that specify the action to take contingent on every possible event. A full policy or conditional plan is generally expensive to compute. To improve computational efficiency, we introduce the notion of *partial conditional plans* that cover sampled events to approximate a full conditional plan. Our approach constructs a partial conditional plan parameterized by a *replanning probability*. We prove that the failure rate of the constructed partial conditional plan is bounded by the replanning probability. Our approach allows users to specify an appropriate bound on the replanning probability to balance efficiency and correctness. Moreover, we update this bound properly to quickly detect whether the current partial conditional plan meets the bound and avoid unnecessary computation. In addition, to further improve the efficiency, we cache partial conditional plans for sampled belief states and reuse these cached plans if possible. We validate our approach in several robotic domains. The results show that our approach outperforms a previous policy synthesis approach for POMDPs with safe-reachability objectives in these domains.**

***Note to Practitioners*—This article was motivated by two observations. On the one hand, in robotics applications where uncertainty in sensing and actions is present, the solution to the classical partially observable Markov decision process (POMDP) formulation is expensive to compute in general. On the other hand, in certain practical scenarios, formulations other than the classical POMDP make a lot of sense and can provide flexibility in balancing efficiency and correctness. This article considers a modified POMDP formulation that includes a Boolean objective, namely safe reachability. This article uses the notion of a partial conditional plan. Rather than explicitly enumerating all possible observations to construct a full conditional plan, this work samples a subset of all observations to ensure bounded replanning probability. Our theoretical and empirical results show that the failure rate of the constructed partial conditional plan is bounded by the replanning probability. Moreover, these partial conditional plans can be cached to further improve the performance. Our results suggest that for domains where replanning is easy, increasing the replanning probability bound usually leads to better scalability, and for domains where replanning is difficult or impossible in some states, we can decrease the bound and allocate more computation time to achieve a higher success rate. Hence, in certain cases, the practitioner can take advantage of their knowledge of the problem domain to scale to larger problems. Preliminary physical experiments suggest that this approach is applicable to real-world robotic domains, but it requires a discrete representation of the workspace. How to deal with continuous workspace directly is an interesting future direction.**

***Index Terms*—Partially observable Markov decision processes (POMDPs) with Boolean objectives, planning, robots, safe reachability, uncertainty.**

Manuscript received August 8, 2019; revised December 17, 2020; accepted January 13, 2021. Date of publication February 23, 2021; date of current version July 2, 2021. This article was recommended for publication by Editor L. Tapia upon evaluation of the reviewers' comments. This work has been supported in part by NSF grants 1139011, 1162076, 1317849, and 1514372. *(Yue Wang and Abdullah Al Redwan Newaz contributed equally to this work.) (Corresponding author: Lydia E. Kavraki.)*

Yue Wang was with the Department of Computer Science, Rice University, Houston, TX 77005 USA. He is now with Facebook, Bellevue, WA 98004 USA (e-mail: yw27@rice.edu).

Abdullah Al Redwan Newaz was with the Department of Computer Science, Rice University, Houston, TX 77005 USA. He is now with North Carolina Agricultural and Technical State University, Greensboro, NC 27411 USA (e-mail: redwan.newaz@rice.edu).

Juan David Hernández was with the Department of Computer Science, Rice University, Houston, TX 77005 USA. He is now with the Centre in AI, Robotics and Human-Machine Systems (IROHMS), School of Engineering, Cardiff University, Cardiff CF24 3AA, U.K. (e-mail: juandhv@rice.edu).

Swarat Chaudhuri was with the Department of Computer Science, Rice University, Houston, TX 77005 USA. He is now with the Computer Science Department, The University of Texas at Austin, Austin, TX 78712 USA (e-mail: swarat@rice.edu).

Lydia E. Kavraki is with the Department of Computer Science, Rice University, Houston, TX 77005 USA (e-mail: kavraki@rice.edu).

This article has supplementary material provided by the authors and color versions of one or more figures available at https://doi.org/10.1109/TASE.2021.3057111.

Digital Object Identifier 10.1109/TASE.2021.3057111

I. Introduction

PLANNING robust executions under uncertainty, e.g., uncertain effects from imperfect controllers and sensors, is a fundamental concern in robotics. Partially observable Markov decision processes (POMDPs) [1] provide a standard framework for modeling many robot tasks under uncertainty (see [2]–[7]). The solutions to POMDPs are *policies* [1] or *conditional plans* [8] that specify the actions to take under *all possible* events during execution.

Traditionally, the goal of solving POMDPs is to find optimal solutions with respect to a quantitative objective such as that maximize (discounted) rewards [2], [3], [5], [8]–[13]. While this purely quantitative formulation is suitable for

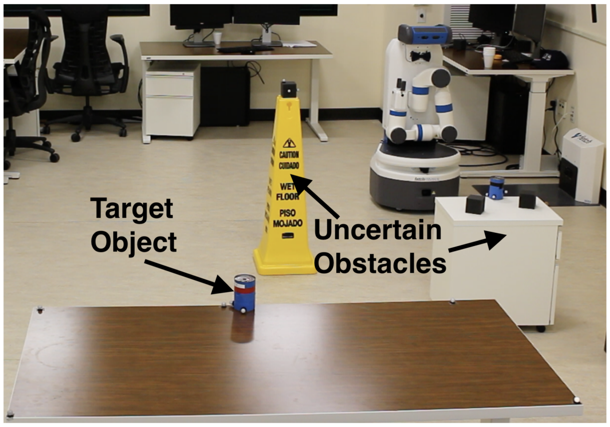


Fig. 1. Robot with imperfect actuation and perception navigates through an office to pick up the blue can on the table while avoiding collisions with uncertain obstacles such as floor signs and file cabinets.

many applications, some robotic settings demand synthesis concerning Boolean requirements. For example, consider a robot with imperfect actuation and perception working in an office environment with uncertain obstacles, such as floor signs and furniture (see Fig. 1). Due to uncertainty, the locations of the robot and the obstacles are partially observable, and the robot's action effects and observations are both probabilistic. In this probabilistic setting, a reasonable task requirement for the robot is to eventually pick up the target object with a probability above a threshold while keeping the probability of collision below a different threshold. This task requirement is naturally formulated as a Boolean objective written in temporal logic. Moreover, formulating Boolean requirements implicitly as quantitative objectives by assigning proper rewards for goal states and unsafe states does not always yield good solutions for certain domains [14]. Therefore, POMDPs with explicit Boolean objectives are better formulations than quantitative POMDPs in these domains.

Policy synthesis for POMDPs with Boolean objectives has been studied in previous works [4], [15]–[17], where the goal is to satisfy a temporal property with *probability 1* (almost-sure satisfaction). A more general policy synthesis for POMDPs with Boolean objectives is to synthesize policies that satisfy a temporal property with *a probability above a threshold*. In this work, we study this problem for the special case of *safe-reachability* objectives, which require that with a probability above a threshold, a goal state is eventually reached while keeping the probability of visiting unsafe states below a different threshold. Many robot tasks such as the one in Fig. 1 can be formulated as a safe-reachability objective.

Our previous work [6] has presented a method called bounded policy synthesis (BPS) for POMDPs with safe-reachability objectives. BPS computes a valid policy over the *goal-constrained belief space* rather than the entire belief space to improve efficiency. The goal-constrained belief space only contains beliefs visited by desired executions achieving the safe-reachability objective and is generally much smaller than the original belief space. BPS is an *offline* synthesis method that computes a full policy before execution. Another category of approaches to planning under uncertainty is *online planning* that interleaves planning and execution [5], [12], [13], [18]–[21]. Offline synthesis offers a strong correctness guarantee, but it is difficult to scale. Online planning is much more scalable and works well when replanning is likely to succeed, but it often fails when replanning is difficult or infeasible in some states, making it hard to ensure correctness.

In this work, our goal is to scale up our previous BPS method further through online planning. Specifically, we present a method called *online partial conditional plan synthesis* (OPCPS) for POMDPs with safe-reachability objectives. OPCPS is based on the new notion of *partial conditional plans*, which only contains a sampled subset of all possible events and approximates a full policy. OPCPS computes a partial conditional plan parameterized by a *replanning probability*, i.e., the probability of encountering an event not covered by the partial conditional plan, thus requiring replanning. We offer a theoretical analysis of this framework, showing that the failure rate of the constructed partial conditional plan is bounded by the replanning probability. OPCPS allows users to specify an appropriate bound on the replanning probability to balance efficiency and correctness: for domains where replanning is likely to succeed, increasing the bound usually leads to better scalability, and for domains where replanning is difficult or impossible in some states, users can decrease the bound and allocate more time to achieve a higher success rate.

To further improve the performance, OPCPS updates the replanning probability bound properly during the partial conditional plan construction. This bound update enables quicker detection of the current partial conditional plan meeting the bound and avoids unnecessary computation. For a better safety guarantee, OPCPS checks whether the successor belief of every uncovered observation of the constructed partial conditional plan satisfies the safety requirement. Thus, OPCPS guarantees that the robot still satisfies the safety requirement when replanning fails. Section IV-B has more details on the bound update and the safety guarantee of OPCPS. What is more, we cache partial conditional plans for sampled belief states and reuse these plans if possible to avoid repetitive computation. In certain cases, as we explain in Section IV-C, caching partial conditional plans leads to increased computational efficiency.

We evaluate OPCPS in the kitchen domain presented in [6] and the tag domain [3]. We also validate OPCPS on a Fetch robot for the domain shown in Fig. 1. The results demonstrate that OPCPS scales better than BPS and can solve problems that are beyond the capabilities of BPS within the time limit.

This article is a significant extension of the preliminary findings presented in [22]. First, we extend the OPCPS algorithm presented in [22] with partial conditional plan caching. Second, we show that OPCPS with caching greatly improves running times in the experiments. Third, we conducted a physical experiment to validate OPCPS with the new caching option on a Fetch robot. Hence, the algorithms presented in this article can be regarded as improved versions of the algorithms in [22].

II. Related Work

The analysis of POMDPs can be divided into three categories. In the first category, the objective is to find optimal solutions concerning quantitative rewards. Many previous POMDP algorithms [3], [5], [8], [10]–[13] focus on maximizing (discounted) rewards. In the second category, the objective combines the quantitative rewards of the traditional POMDPs with notions of risk and cost. Recently, there has been a growing interest in constrained POMDPs [14], [23]–[25], chance-constrained POMDP [26], and risk-sensitive POMDPs [27], [28] that handle cost/risk constraints explicitly. The third category consists of POMDPs with high-level Boolean requirements written in temporal logic. Some works [4], [15] have investigated *almost-sure satisfaction* of POMDPs with temporal properties, where the goal is to check whether a given temporal property can be satisfied with probability 1. A more general policy synthesis problem of POMDPs with safe-reachability objectives has been introduced in our previous work [6]. It has been shown that for robotic domains that require a correctness guarantee of accomplishing tasks, POMDPs with safe reachability provide a better guarantee of safety and reachability than the quantitative POMDP formulations [6].

While several works [29], [30] propose different types of reinforcement learning algorithms to address policy synthesis problems for MDP, to the best of our knowledge, there are a few works on policy synthesis based on reinforcement learning for POMDPs [31]. In recent work, Flaspohler*et al.* [32] presented macroaction discovery from a low-level POMDP model by chaining sequences of open-loop actions together with the task-specific value of information. Bhattacharya *et al.* [33] proposed a reinforcement learning-based POMDP solver for autonomous sequential repair problems. They use a neural network classifier for approximating successive policies. Garg *et al.* [34] modeled adaptive grasping using tactile and visual sensors as a POMDP problem and proposed a combination of the model-based POMDP planning and imitation learning to learn a robust strategy for grasping previously unseen objects.

In this work, we focus on POMDPs with safe-reachability objectives and evaluate our previous BPS approach [6]. While BPS synthesizes a full policy (conditional plan) offline that covers all possible events, our approach is an online method that interleaves the computation of a partial conditional plan and execution. Since a partial conditional plan only contains a sampled subset of all possible events, our method achieves better scalability than BPS and can solve problems that are beyond the capabilities of BPS within the time limit.

The idea of partial conditional plans resembles the state-of-the-art online POMDP algorithm based on determinized sparse partially observable tree (DESPOT) [5], [12]. Both DESPOT and our partial conditional plans contain a subset of all possible observations to improve efficiency. There are two major differences between our method and DESPOT. First, DESPOT handles POMDPs with (discounted) rewards, whereas our approach solves POMDPs with safe-reachability objectives. Second, DESPOT contains all action branches, whereas our approach constructs partial conditional plans (see Fig. 2) that only contains one action per step, which is part of the desired execution satisfying the safe-reachability objective.

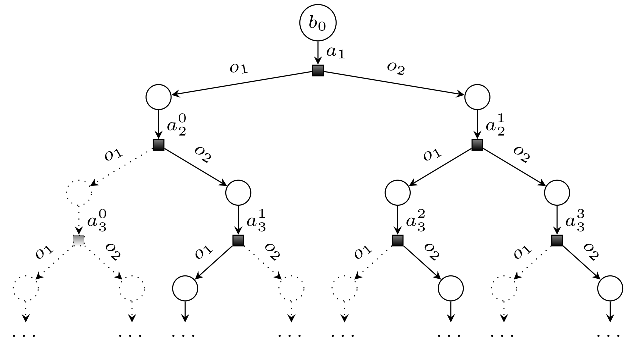


Fig. 2. Full conditional plan γ_k contains both solid and dotted branches. $a_1, a_2^0, \ldots$ are actions. o_1 and o_2 are observations. A partial conditional plan γ_k^p contains only solid branches.

III. Problem Formulation

We follow the notation in [6] for POMDPs with safe-reachability objectives.

A. Partially Observable Markov Decision Process

Definition 1 (POMDP [1]): A POMDP is a tuple $P = (\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{O}, \mathcal{Z})$, where $\mathcal{S}$ is a *finite* set of states, $\mathcal{A}$ is a *finite* set of actions, $\mathcal{T}$ is a probabilistic transition function, $\mathcal{O}$ is a *finite* set of observations, and $\mathcal{Z}$ is a probabilistic observation function. $\mathcal{T}(s, a, s') = \Pr(s'|s, a)$ specifies the probability of moving to state $s' \in \mathcal{S}$ after taking action $a \in \mathcal{A}$ in state $s \in \mathcal{S}$. $\mathcal{Z}(s', a, o) = \Pr(o|s', a)$ specifies the probability of observing $o \in \mathcal{O}$ after taking action $a \in \mathcal{A}$ and reaching $s' \in \mathcal{S}$.

Due to uncertainty, states are partially observable, and typically, we maintain a probability distribution (*belief*) over all states $b : \mathcal{S} \mapsto [0, 1]$ with $\sum_{s\in\mathcal{S}} b(s) = 1$. The set of all beliefs $\mathcal{B} = \{b : \mathcal{S} \mapsto [0, 1] \mid \sum_{s\in\mathcal{S}} b(s) = 1\}$ is the *belief space*.

The belief space transition function $\mathcal{T}_\mathcal{B} : \mathcal{B} \times \mathcal{A} \times \mathcal{O} \to \mathcal{B}$ is *deterministic*. $b_a^o = \mathcal{T}_\mathcal{B}(b, a, o)$ is the successor belief for a belief $b \in \mathcal{B}$ after taking an action $a \in \mathcal{A}$ and receiving an observation $o \in \mathcal{O}$, defined according to Bayes rule: $\forall\, s' \in \mathcal{S}$, $b_a^o(s') = ((\mathcal{Z}(s', a, o) \sum_{s\in\mathcal{S}} \mathcal{T}(s, a, s')b(s))/(\Pr(o|b, a)))$, where $\Pr(o|b, a) = \sum_{s'\in\mathcal{S}} \mathcal{Z}(s', a, o) \sum_{s\in\mathcal{S}} \mathcal{T}(s, a, s')b(s)$ is the probability of receiving the observation o after taking the action a in the belief b.

Definition 2 (Plan): A k-step *plan* is a sequence $\sigma = (b_0, a_1, o_1, \ldots, a_k, o_k, b_k)$ such that for all $i \in (0, k]$, the belief updates satisfy the transition function $\mathcal{T}_\mathcal{B}$, i.e., $b_i = \mathcal{T}_\mathcal{B}(b_{i-1}, a_i, o_i)$, where $a_i \in \mathcal{A}$ is an action and $o_i \in \mathcal{O}$ is an observation.

B. Safe-Reachability Objective

In this work, we consider POMDPs with *safe-reachability* objectives.

Definition 3 (Safe-Reachability Objective): A *safe-reachability objective* is a tuple $\mathcal{G} = (\text{Dest}, \text{Safe})$, where $\text{Safe} = \{b \in \mathcal{B} \mid \sum_{s \text{ violates safety}} b(s) < \delta_2\}$ is a set of safe beliefs and $\text{Dest} = \{b \in \text{Safe} \mid \sum_{s \text{ is a goal state}} b(s) > 1-\delta_1\} \subseteq \text{Safe}$ is a set of goal beliefs. δ_1 and δ_2 are small values that represent tolerance.

A safe-reachability objective $\mathcal{G}$ compactly represents the set $\Omega_{\mathcal{G}}$ of valid plans:

Definition 4 (Valid Plan): A k-step plan $\sigma = (b_0, a_1, o_1, \ldots, a_k, o_k, b_k)$ is *valid* with respect to a safe-reachability objective $\mathcal{G} = (\text{Dest}, \text{Safe})$ if b_k is a goal belief ($b_k \in \text{Dest}$) and all beliefs visited before step k are safe beliefs ($\forall\ i \in [0, k), b_i \in \text{Safe}$).

Note that the safety requirement in the safe-reachability objective only states that for every step of the plan, the probability of being in an unsafe state is within the threshold. This safety requirement does not necessarily extend to the safety of the whole plan, i.e., the probability of visiting an unsafe state is within the same threshold when executing the plan starting from the initial belief. To achieve the safety of the whole plan, we should consider the chance constraints presented in [26], which is beyond the scope of this article and a possible future extension of this work.

C. Solution to POMDPs With Safe-Reachability Objective

The solution to POMDPs with safe-reachability objective is a *valid* policy that specifies the action to take contingent on all possible events:

Definition 5 (Valid Policy): A *valid* policy $\pi : \mathcal{B} \to \mathcal{A}$ is a function that maps a belief $b \in \mathcal{B}$ to an action $a \in \mathcal{A}$. A policy π defines a set of plans in belief space: $\Omega_\pi = \{\sigma = (b_0, a_1, o_1, \ldots) \mid \forall i > 0, a_i = \pi(b_{i-1}), \text{and } o_i \in \mathcal{O}\}$. For each plan $\sigma \in \Omega_\pi$, the action a_i at each step i is chosen by the policy π. For a valid policy, the set Ω_π of plans defined by the policy π is all *valid plans*.

D. Partial Conditional Plan

Computing an exact policy over the entire belief space $\mathcal{B}$ is intractable, due to the curse of dimensionality [35]: $\mathcal{B}$ is a high-dimensional space with an infinite number of beliefs. To make the problem tractable, we can exploit the *reachable belief space* $\mathcal{B}_{b_0}$ [3], [10]. $\mathcal{B}_{b_0}$ only contains beliefs reachable from the initial belief b_0 and is generally much smaller than $\mathcal{B}$. Therefore, instead of computing a policy $\pi : \mathcal{B} \mapsto \mathcal{A}$ over the entire belief space, we only compute a policy $\pi_{\mathcal{B}_{b_0}} : \mathcal{B}_{b_0} \mapsto \mathcal{A}$ over the reachable belief space.

Our previous BPS work [6] has shown that the performance of policy synthesis for POMDPs with safe-reachability objectives can be further improved based on the notion of a *goal-constrained belief space* $\mathcal{B}_{\mathcal{G}}$. $\mathcal{B}_{\mathcal{G}}$ combines the reachable belief space $\mathcal{B}_{b_0}$ and the set $\Omega_{\mathcal{G}}$ of valid plans defined by the safe-reachability objective $\mathcal{G}$. $\mathcal{B}_{\mathcal{G}}$ only contains beliefs reachable from the initial belief b_0 under a valid plan $\sigma \in \Omega_{\mathcal{G}}$ and is generally much smaller than the reachable belief space $\mathcal{B}_{b_0}$.

Previous results [36]–[38] have shown that the problem of policy synthesis for POMDPs is generally undecidable. However, when restricted to a bounded horizon, this problem becomes PSPACE-complete [35], [39]. Therefore, BPS computes a bounded policy π over the goal-constrained belief space $\mathcal{B}_{\mathcal{G}}$ within a bounded horizon h, where the horizon (number of steps) of the policy is less than a given bound h. This bounded policy π is essentially a set of conditional plans [8].

Definition 6 (Conditional Plan): A k-step conditional plan $\gamma_k \in \Gamma_k$ is a tuple $\gamma_k = (b, a, \nu_k)$, where $b \in \mathcal{B}$ is a belief, $a \in \mathcal{A}$ is an action, and $\nu_k : \mathcal{O} \mapsto \Gamma_{k-1}$ is an *observation strategy* that maps an observation $o \in \mathcal{O}$ to a $(k-1)$-step conditional plan $\gamma_{k-1} = (b', a', \nu_{k-1}) \in \Gamma_{k-1}$, where $b' = \mathcal{T}_{\mathcal{B}}(b, a, o)$ is the successor belief.

Fig. 2 shows an example k-step conditional plan $\gamma_k = (b_0, a_1, \nu_k)$. γ_k defines a set Ω_{γ_k} of k-step plans $\sigma_k = (b_0, a_1, o_1, \ldots, a_k, o_k, b_k)$. For each plan $\sigma_k \in \Omega_{\gamma_k}$, the action a_1 at step 1 is chosen by the k-step conditional plan γ_k, the action a_2 at step 2 is chosen by the $(k-1)$-step conditional plan $\gamma_{k-1} = \nu_k(o_1), \ldots,$ and the action a_k at step k is chosen by the one-step conditional plan $\gamma_1 = \nu_2(o_{k-1})$.

Definition 7 (Valid Conditional Plan): A k-step conditional plan γ_k is *valid* with respect to a safe-reachability objective $\mathcal{G}$ if every plan in Ω_{γ_k} is valid ($\Omega_{\gamma_k} \subseteq \Omega_{\mathcal{G}}$).

It is clear that the number of valid plans in a valid k-step conditional plan γ_k grows exponentially as the horizon k increases. To address this challenge, our method computes *partial* conditional plans that only contain a small number of valid plans to approximate full conditional plans:

Definition 8 (Partial Conditional Plan): A k-step *partial* conditional plan is a tuple $\gamma_k^p = (b, a, \mathcal{O}_k^p, \nu_k^p)$, where $b \in \mathcal{B}$ is a belief, $a \in \mathcal{A}$ is an action, $\mathcal{O}_k^p \subseteq \mathcal{O}$ is a subset of the observation set $\mathcal{O}$, and $\nu_k^p : \mathcal{O}_k^p \mapsto \Gamma_{k-1}^p$ is a *partial observation strategy* that maps an observation $o \in \mathcal{O}_k^p$ to a $(k-1)$-step partial conditional plan $\gamma_{k-1}^p = (b', a', \mathcal{O}_{k-1}^p, \nu_{k-1}^p)$, where $b' = \mathcal{T}_{\mathcal{B}}(b, a, o)$ is the successor belief. When $\mathcal{O}_k^p = \mathcal{O}$, the partial conditional plan γ_k^p is a full conditional plan $\gamma_k \in \Gamma_k$. For $k = 1$, the observation strategy of γ_1^p is $\nu_1 = \emptyset$.

Similarly, a k-step partial conditional plan γ_k^p defines a set $\Omega_{\gamma_k^p}$ of k-step plans σ_k in belief space, and we can define a *valid* partial conditional plan:

Definition 9 (Valid Partial Conditional Plan): A k-step partial conditional plan γ_k^p is *valid* with respect to a safe-reachability objective $\mathcal{G}$ if every plan in $\Omega_{\gamma_k^p}$ is valid.

E. Replanning Probability

Since a partial conditional plan $\gamma_k^p = (b, a, \mathcal{O}_k^p, \nu_k^p)$ only contains a subset of all observation branches at each step (see Fig. 2), during online execution, it is possible that an observation branch $o \in \mathcal{O} - \mathcal{O}_k^p$ that is not part of the partial conditional plan is visited. In this case, we need to recursively compute a new partial conditional plan for this new branch o. However, since γ_k^p does not consider all possible observation branches, it is possible that the action chosen by γ_k^p is *invalid* for the new observation branch o, even for a valid partial conditional plan. As a result, there are no partial conditional plans for the new observation branch o and execution fails.

To preserve correctness, we would like to bound the *failure rate* $p_{\text{fail}}(\gamma_k^p) = \Pr(\text{failure}|\gamma_k^p)$ measured under a valid partial conditional $\gamma_k^p = (b, a, \mathcal{O}_k^p, \nu_k^p)$. However, computing p_{fail} is costly because it requires checking whether the action a chosen by γ_k^p is valid for every uncovered observation branch $o \in \mathcal{O} - \mathcal{O}_k^p$, which essentially computes a *full* conditional

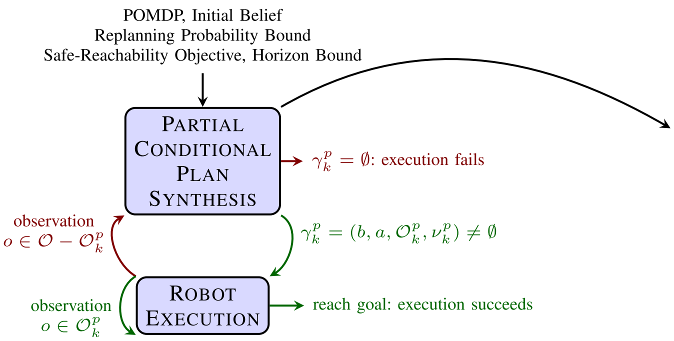


Fig. 3. Overall structure of the OPCPS algorithm.

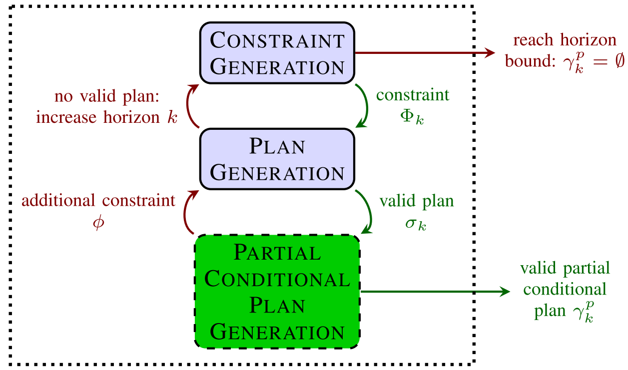


Fig. 4. Component of partial conditional plan synthesis in Fig. 3.

plan. Alternatively, we can easily compute the *replanning probability* $p_{\text{replan}}(\gamma_k^p) = \Pr(\text{replanning}|\gamma_k^p)$ of reaching an uncovered observation branch $o \in \mathcal{O} - \mathcal{O}_k^p$ and requiring *replanning*

$$\begin{aligned} p_{\text{replan}}\left(\gamma_k^p\right) = &\sum_{o \in \mathcal{O}_k^p} \Pr(o|b, a)\, p_{\text{replan}}\left(\nu_k^p(o)\right) \\ &+ \sum_{o \in \mathcal{O} - \mathcal{O}_k^p} \Pr(o|b, a). \end{aligned} \quad (1)$$

For the base case $k = 1$, $p_{\text{replan}}(\gamma_1^p) = \sum_{o \in \mathcal{O} - \mathcal{O}_1^p} \Pr(o|b, a)$.

The following theorem states that for a *valid* partial conditional plan γ_k^p, the failure rate $p_{\text{fail}}(\gamma_k^p)$ is bounded by the replanning probability $p_{\text{replan}}(\gamma_k^p)$:

Theorem 1: For any valid partial conditional plan γ_k^p, $p_{\text{fail}}(\gamma_k^p) \leq p_{\text{replan}}(\gamma_k^p)$.

Proof: We prove Theorem 1 by induction. First, we define $\delta_{\text{fail}}(b) : \mathcal{B} \mapsto \{0, 1\}$ as an indicator, and when $\delta_{\text{fail}}(b) = 1$, there are no valid partial conditional plans for belief b and execution fails.

1) *Base Case ($k = 1$):* Since $\gamma_1^p = (b, a, \mathcal{O}_1^p, \emptyset)$ is valid, for every covered observation $o \in \mathcal{O}_1^p$, $b' = \mathcal{T}_{\mathcal{B}}(b, a, o) \in \text{Dest}$ is the terminal goal belief and thus $\delta_{\text{fail}}(b') = 0$. Therefore,

$$\begin{aligned} p_{\text{fail}}\left(\gamma_1^p\right) &= \sum_{o \in \mathcal{O} - \mathcal{O}_1^p} \Pr(o|b, a)\delta_{\text{fail}}(b') \\ &\leq \sum_{o \in \mathcal{O} - \mathcal{O}_1^p} \Pr(o|b, a) = p_{\text{replan}}\left(\gamma_1^p\right) \end{aligned}$$

since $\delta_{\text{fail}}(b') \leq 1$ where $b' = \mathcal{T}_{\mathcal{B}}(b, a, o)$ is the successor belief for the uncovered observation $o \in \mathcal{O} - \mathcal{O}_1^p$.

2) *Inductive Case ($k > 1$):* Since $\gamma_k^p = (b, a, \mathcal{O}_k^p, \nu_k^p)$ is valid, for every covered observation $o \in \mathcal{O}_k^p$, the corresponding $(k-1)$-step partial conditional plan $\nu_k^p(o)$ is also valid. Assume that $p_{\text{fail}}(\nu_k^p(o)) \leq p_{\text{replan}}(\nu_k^p(o))$, and then

$$\begin{aligned} p_{\text{fail}}\left(\gamma_k^p\right) = &\sum_{o \in \mathcal{O}_k^p} \Pr(o|b, a)\, p_{\text{fail}}\left(\nu_k^p(o)\right) \\ &+ \sum_{o \in \mathcal{O} - \mathcal{O}_k^p} \Pr(o|b, a)\delta_{\text{fail}}(b') \\ \leq &\sum_{o \in \mathcal{O}_k^p} \Pr(o|b, a)\, p_{\text{replan}}\left(\nu_k^p(o)\right) \\ &+ \sum_{o \in \mathcal{O} - \mathcal{O}_k^p} \Pr(o|b, a) = p_{\text{replan}}\left(\gamma_k^p\right) \end{aligned}$$

since $\delta_{\text{fail}}(b') \leq 1$, where $b' = \mathcal{T}_{\mathcal{B}}(b, a, o)$ is the successor belief for the uncovered observation $o \in \mathcal{O} - \mathcal{O}_k^p$.

Therefore, for any k-step valid partial conditional plan $\gamma_k^p = (b, a, \mathcal{O}_k^p, \nu_k^p)$, $p_{\text{fail}}(\gamma_k^p) \leq p_{\text{replan}}(\gamma_k^p)$. □

F. Problem Statement

Given a POMDP P, an initial belief b_0, a replanning probability bound $\delta_{p_{\text{replan}}}$, a safe-reachability objective $\mathcal{G}$, and a horizon bound h, our goal is to synthesize a *valid* k-step ($k \leq h$) partial conditional plan $\gamma_k^p = (b_0, a, \mathcal{O}_k^p, \nu_k^p)$ with a replanning probability $p_{\text{replan}}(\gamma_k^p) \leq \delta_{p_{\text{replan}}}$.

Since the replanning probability $p_{\text{replan}}(\gamma_k^p)$ is bounded by $\delta_{p_{\text{replan}}}$, by Theorem 1, γ_k^p guarantees achieving the given safe-reachability objective with a probability at least $1 - \delta_{p_{\text{replan}}}$. Note that when $p_{\text{replan}}(\gamma_k^p) = 0$, γ_k^p is a full conditional plan.

IV. Online Partial Conditional Plan Synthesis

Fig. 3 shows the overall structure of OPCPS (Algorithm 1). OPCPS follows the typical online planning paradigm [40] that interleaves synthesis of valid partial conditional plans (line 1) and execution (lines 6–8). If there are no valid partial conditional plans within the horizon bound (line 2), execution fails. Otherwise, OPCPS follows the generated partial conditional plan until a goal belief is reached (line 9: execution succeeds) or a new observation $o \in \mathcal{O} - \mathcal{O}_k^p$ is received (line 13). In the latter case, OPCPS recursively replans for the observation o. Next, we describe the partial conditional plan synthesis algorithm (Fig. 4) used in OPCPS.

A. Partial Conditional Plan Synthesis

In partial conditional plan synthesis (Fig. 4 and Algorithm 2), we replace the policy generation component in BPS [6] with a new partial conditional plan generation (the green dashed component). For completeness, we offer a brief summary of the constraint generation and plan generation components in BPS (see [6] for more details).

In constraint generation (Fig. 4), given a POMDP P, an initial belief b_0, and a safe-reachability objective $\mathcal{G} = (\text{Dest}, \text{Safe})$, we first construct a constraint Φ_k to symbolically encode the goal-constrained belief space over a

Algorithm 1 OPCPS

Input: POMDP $P = (\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{O}, \mathcal{Z})$, Initial Belief b_{init}, Replanning Probability Bound $\delta_{p_{\text{replan}}}$, Safe-Reachability Objective $\mathcal{G} = (\text{Dest}, \text{Safe})$, Horizon Bound h
Output: A boolean: true - success, false - failure

```
/* Generate the partial conditional plan */
1  γ_k^p ← PartialContionalPlanSynthesis(P, b_init, G, δ_{p_replan}, h)
2  if γ_k^p = ∅ then
       /* No partial conditional plans: failure */
3      return false
4  repeat
5      (a, O_k^p, ν_k^p) ← γ_k^p
6      Execute action a
7      Receive observation o
8      b_init ← T_B(b_init, a, o)          /* Update belief */
9      if b_init ∈ Dest then
           /* reach a goal belief: success */
10         return true
       /* Get the next partial conditional plan */
11     γ_k^p ← ν_k^p(o)
12     h ← h − 1                 /* Reduce the horizon bound */
13 until γ_k^p = ∅
/* recursively perform OPCPS on new branch */
14 return OPCPS(P, b_init, G, h)
```

Algorithm 2 PartialConditionalPlanSynthesis

Input: POMDP P, Initial Belief b_{init}, Replanning Probability Bound $\delta_{p_{\text{replan}}}$, Safe-Reachability Objective $\mathcal{G} = (\text{Dest}, \text{Safe})$, Horizon Bound h
Output: *Valid* partial conditional plan γ_k^p with $p_{\text{replan}}(b_{init}, \gamma_k^p) \leq \delta_{p_{\text{replan}}}$

```
/* Φ_k is the constraint to symbolically encode
   the goal-constrained belief space */
15 Φ_k ← (b_0 = b_init)    /* Start from initial belief */
16 k ← 0                   /* k is the number of steps */
17 while k ≤ h do
       /* Add transition at step k if k > 0 */
18     if k > 0 then
19         Φ_k ← Φ_k ∧ (b_k = T_B(b_{k−1}, a_k, o_k))
20     push(Φ_k)                           /* Push scope */
       /* Add goal constraints at step k */
21     Φ_k ← Φ_k ∧ G(σ_k, G, k)
22     repeat
           /* Plan generation: check satisfiability of
              Φ_k via an satisfiability modulo
              theory (SMT) solver [41] */
23         σ_k ← IncrementalSMT(Φ_k)
24         if σ_k ≠ ∅ then /* Find valid plan */
               /* Generate partial conditional plan */
25             γ_k^p, φ = PartialConditionalPlanGeneration(P,
               δ_{p_replan}, G, σ_k, 1, k)
26             if ∅ = γ_k^p then /* Generation failed */
                   /* Blocking invalid plans */
27                 Φ_k ← Φ_k ∧ φ
28             else
29                 return γ_k^p
30     until σ_k = ∅
31     pop(Φ_k)             /* Pop goal and φ at step k */
32     k ← k + 1                /* Increase the horizon */
33 return ∅
```

bounded horizon k based on the encoding from bounded model checking [42] (lines 15, 19, and 21). Φ_k compactly represents the requirement of reaching a goal belief $b \in \text{Dest}$ safely in k steps. In constraint generation (Fig. 4), we use the bounded model checking [42] encoding to construct Φ_k, which contains three parts.

1) Start from the initial belief (line 15): $b_0 = b_{\text{init}}$.
2) Unfold the transition up to horizon k (line 19): $\bigwedge_{i=1}^{k}(b_i = \mathcal{T}_{\mathcal{B}}(b_{i-1}, a_i, o_i))$.
3) Satisfy the objective $\mathcal{G}$ (line 21): $G(\sigma_k, \mathcal{G}, k) = \bigvee_{i=0}^{k}(b_i \in \text{Dest} \wedge (\bigwedge_{j=0}^{i-1}(b_j \in \text{Safe})))$.

Then, in plan generation (see Fig. 4), we compute a valid *plan* σ_k by checking the satisfiability of Φ_k (line 23) through an SMT solver [41]. Note that the horizon k restricts the plan length, and thus, the robot can only execute k actions before reaching a goal belief $b \in \text{Dest}$.

If Φ_k is satisfiable, the SMT solver returns a valid plan $\sigma_k = (b_0^{\sigma_k}, a_1^{\sigma_k}, o_1^{\sigma_k}, \ldots, b_k^{\sigma_k})$. This valid plan σ_k only covers a particular observation $o_i^{\sigma_k}$ at step i. In partial conditional plan generation (see Fig. 4), we generate a valid partial conditional plan γ_k^p with a replanning probability $p_{\text{replan}}(\gamma_k^p) \leq \delta_{p_{\text{replan}}}$ (line 25) from this valid plan σ_k by sampling a subset $\mathcal{O}_k^p \subseteq \mathcal{O}$ of observations (solid branches in Fig. 2) at each step, where $\delta_{p_{\text{replan}}}$ is the given replanning probability bound. If this partial conditional plan generation fails, we construct an additional constraint ϕ to block invalid plans (line 27) and force the SMT solver to generate another better plan. Note that ϕ is only valid for current horizon k, and when we increase the horizon, we should *pop* the scope related to the additional constraints ϕ from the stack of the SMT solver (line 31) so that we can *revisit* σ_k with the increased horizon. The incremental SMT solver can efficiently generate alternate valid plans by maintaining a *stack* of *scopes* for the "knowledge" learned from previous satisfiability checks [6], [41], [43].

If Φ_k is unsatisfiable and there is no valid plan for the current horizon, we increase the horizon (line 32) and repeat the above steps until a *valid* partial conditional plan is found (line 29) or a given horizon bound is reached (line 17). Next, we describe the new partial conditional plan generation component.

B. Partial Conditional Plan Generation

In partial conditional plan generation (Algorithm 3), we construct a valid partial conditional plan γ_k^p that satisfies the given bound $\delta_{p_{\text{replan}}}$ from a valid plan σ_k. For each step i, we first recursively construct a next-step conditional plan γ_{next}^p for $o_i^{\sigma_k}$ (line 38). If the replanning probability $p_{\text{replan}}(\gamma_k^p)$ is greater

Algorithm 3 PartialConditionalPlanGeneration

Input: POMDP $P = (\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{O}, \mathcal{Z})$, Replanning Probability Bound $\delta_{p_{\text{replan}}}$, Safe-Reachability Objective $\mathcal{G} = (\text{Dest}, \text{Safe})$, Valid k-Step Plan $\sigma_k = (b_0^{\sigma_k}, a_1^{\sigma_k}, o_1^{\sigma_k}, \ldots, b_k^{\sigma_k})$, Step i, Horizon Bound h

Output: *Valid* partial conditional plan γ_k^p with replanning probability $p_{\text{replan}}(\gamma_k^p) \le \delta_{p_{\text{replan}}}$, Constraint ϕ for blocking invalid plans

34 **if** $i > k$ **then** /* Reach the last step k */
  /* Terminal belief: γ_k^p specifies nothing */
35 $\gamma_k^p \leftarrow (b_k^{\sigma_k}, \emptyset, \emptyset, \emptyset)$
36 **return** $\gamma_k^p, \emptyset$
/* Initialize */
37 $\mathcal{O}_k^p \leftarrow \emptyset$, $\delta'_{p_{\text{replan}}} \leftarrow \delta_{p_{\text{replan}}}$, $b \leftarrow b_{i-1}^{\sigma_k}$, $a \leftarrow a_i^{\sigma_k}$, $o' \leftarrow o_i^{\sigma_k}$
/* Recursively process next step */
38 $\gamma_{\text{next}}^p \leftarrow$ PartialConditionalPlanGeneration$(P, \delta'_{p_{\text{replan}}}, \mathcal{G}, \sigma_k, i+1, h)$
39 **if** $\gamma_{next}^p = \emptyset$ **then** /* Construction failed */
40 Construct ϕ using Formula 2, **return** $\emptyset$, ϕ
/* Add o' to γ_k^p */
41 $\mathcal{O}_k^p \leftarrow \mathcal{O}_k^p \cup \{o'\}$, $\nu_k^p(o') \leftarrow \gamma_{\text{next}}^p$, $\gamma_k^p \leftarrow (b, a, \mathcal{O}_k^p, \nu_k^p)$
42 **while** $p_{\text{replan}}(\gamma_k^p) > \delta_{p_{\text{replan}}}$ **do**
  /* Bound update */
43 $\delta'_{p_{\text{replan}}} \leftarrow \delta'_{p_{\text{replan}}} + \frac{\Pr(o'|b,a)(\delta'_{p_{\text{replan}}} - p_{\text{replan}}(\nu_k^p(o')))}{\sum_{o \in \mathcal{O} - \mathcal{O}_k^p - \{o'\}} \Pr(o|b,a)}$
44 $o' \leftarrow$ sampled observation in $\mathcal{O} - \mathcal{O}_k^p$ based on the probability of occurrence
45 $b' \leftarrow \mathcal{T}_{\mathcal{B}}(b, a, o')$ /* Get new initial belief */
46 **if** $b' \in \mathbb{M}$ **then** /* Check the cache $\mathbb{M}$ */
47   $\gamma_{\text{next}}^p \leftarrow \mathbb{M}(b')$
48 **else**
   /* Recursively construct a next-step partial conditional plan */
49   $\gamma_{\text{next}}^p \leftarrow$ PartialConditionalPlanSynthesis$(P, b', \delta'_{p_{\text{replan}}}, \mathcal{G}, i, h)$
   /* Update the cache */
50   $\mathbb{M}(b') \leftarrow \gamma_{\text{next}}^p$
51 **if** $\gamma_{next}^p = \emptyset$ **then** /* Construction failed */
52   Construct ϕ using Formula 2, **return** $\emptyset$, ϕ
53 $\mathcal{O}_k^p \leftarrow \mathcal{O}_k^p \cup \{o'\}$, $\nu_k^p(o') \leftarrow \gamma_{\text{next}}^p$ /* Add o' to γ_k^p */
/* Final safety check */
54 **foreach** *observation* $o \in \mathcal{O} - \mathcal{O}_k^p$ **do**
55 $b' \leftarrow \mathcal{T}_{\mathcal{B}}(b, a, o)$ /* Try observation o */
56 **if** $b' \notin$ *Safe* **then** /* Violates safety */
57   Construct ϕ using Formula 2, **return** $\emptyset$, ϕ
58 **return** $\gamma_k^p, \emptyset$

than the bound $\delta_{p_{\text{replan}}}$ (line 42), we add more observation branches to γ_k^p by sampling a new observation o' according to the probability of occurrence (line 44) and recursively constructing a next-step partial conditional plan γ_{next}^p for o' (line 49). This is another partial conditional plan synthesis problem with a new initial belief b' (line 45) and can be solved recursively using Algorithm 2 shown in Fig. 4.

If we successfully construct a valid γ_{next}^p for o', we add o' to γ_k^p (line 41 or 53). Otherwise, this input plan σ_k cannot be an element of a *valid* partial conditional plan γ_k^p ($\sigma_k \notin \Omega_{\gamma_k^p}$). Therefore, the prefix $(b_0^{\sigma_k}, a_1^{\sigma_k}, o_1^{\sigma_k}, \ldots, b_{i-1}^{\sigma_k}, a_i^{\sigma_k})$ of the input plan σ_k is invalid for the current horizon k and we construct the following additional constraint ϕ to block invalid plans:

$$\neg\left(\left(b_0 = b_0^{\sigma_k}\right) \wedge \left(a_i = a_i^{\sigma_k}\right) \wedge \left(\bigwedge_{m=1}^{i-1} \left(a_m = a_m^{\sigma_k}\right) \wedge \left(o_m = o_m^{\sigma_k}\right) \wedge \left(b_m = b_m^{\sigma_k}\right)\right)\right). \quad (2)$$

ϕ blocks the invalid plans that have this prefix and avoids unnecessary checks of these plans (checking σ_k has already shown that these plans are invalid).

1) Updating Replanning Probability Bound: As we add more observation branches to the current partial conditional plan $\gamma_k^p = (b, a, \mathcal{O}_k^p, \nu_k^p)$, we update the replanning probability bound $\delta'_{p_{\text{replan}}}$ (line 43) for the remaining uncovered observation branches $\mathcal{O} - \mathcal{O}_k^p$ to avoid unnecessary computation.

Initially, $\mathcal{O}_k^p$ is empty and $\delta'_{p_{\text{replan}}}$ is the input bound $\delta_{p_{\text{replan}}}$ (line 37). $\delta'_{p_{\text{replan}}}$ bounds the replanning probability $p_{\text{replan}}(\nu_k^p(o))$ of the next-step partial conditional plan $\nu_k^p(o)$ for every remaining uncovered observation $o \in \mathcal{O} - \mathcal{O}_k^p$. $\delta'_{p_{\text{replan}}}$ guarantees that the replanning probability $p_{\text{replan}}(\gamma_k^p)$ satisfies the original bound $\delta_{p_{\text{replan}}}$, i.e., $p_{\text{replan}}(\gamma_k^p) = \sum_{o \in \mathcal{O}} \Pr(o|b,a) p_{\text{replan}}(\nu_k^p(o)) \le \sum_{o \in \mathcal{O}} \Pr(o|b,a) \delta'_{p_{\text{replan}}} \le \delta'_{p_{\text{replan}}} = \delta_{p_{\text{replan}}}$ since $p_{\text{replan}}(\nu_k^p(o)) \le \delta'_{p_{\text{replan}}}$ based on the definition of $\delta'_{p_{\text{replan}}}$.

During partial conditional plan generation, after adding a new observation $o' \in \mathcal{O} - \mathcal{O}_k^p$ to the partial conditional plan γ_k^p (line 41 or 53), we update $\delta'_{p_{\text{replan}}}$ to avoid unnecessary computation. Suppose that we construct a new next-step partial conditional plan γ_{next}^p with the same replanning probability α for every remaining uncovered observation $o \in \mathcal{O} - \mathcal{O}_k^p - \{o'\}$. Then, the replanning probability of the observation branches $\mathcal{O} - \mathcal{O}_k^p$ is $\Pr(o'|b,a)$ $p_{\text{replan}}(\nu_k^p(o')) + \alpha \sum_{o \in \mathcal{O} - \mathcal{O}_k^p - \{o'\}} \Pr(o|b,a) \le \sum_{o \in \mathcal{O} - \mathcal{O}_k^p} \Pr(o|b,a) \delta'_{p_{\text{replan}}}$. Therefore, $\alpha \le \delta'_{p_{\text{replan}}} + ((\Pr(o'|b,a)$ $(\delta'_{p_{\text{replan}}} - p_{\text{replan}}(\nu_k^p(o')))/(\sum_{o \in \mathcal{O} - \mathcal{O}_k^p - \{o'\}} \Pr(o|b,a)))$. Then, the new bound for the remaining uncovered observation $o \in \mathcal{O} - \mathcal{O}_k^p - \{o'\}$ should be $\delta'_{p_{\text{replan}}} + ((\Pr(o'|b,a)$ $(\delta'_{p_{\text{replan}}} - p_{\text{replan}}(\nu_k^p(o')))/(\sum_{o \in \mathcal{O} - \mathcal{O}_k^p - \{o'\}} \Pr(o|b,a)))$, and this new bound is at least $\delta'_{p_{\text{replan}}}$ since $p_{\text{replan}}(\nu_k^p(o')) \le \delta'_{p_{\text{replan}}}$ according to the definition of $\delta'_{p_{\text{replan}}}$. When the replanning probability bound becomes larger, computing a partial conditional plan is usually less expensive. Therefore, updating the replanning probability bound (line 43) usually improves efficiency and still makes the current partial conditional plan γ_k^p satisfy the original bound $\delta_{p_{\text{replan}}}$.

2) Safety Guarantee: After we construct a valid partial conditional plan $\gamma_k^p = (b, a, \mathcal{O}_k^p, \nu_k^p)$, if the uncovered observation set is not empty ($\mathcal{O} - \mathcal{O}_k^p \ne \emptyset$), then the replanning probability $p_{\text{replan}}(\gamma_k^p) > 0$. Though this replanning probability is bounded by the given bound $\delta_{p_{\text{replan}}}$ and by Theorem 1, we know that the execution failure rate $p_{\text{fail}}(\gamma_k^p)$ is also bounded by $\delta_{p_{\text{replan}}}$.

However, if $p_{\text{replan}}(\gamma_k^p) > 0$, during execution, the robot might receive an uncovered observation $o \in \mathcal{O} - \mathcal{O}_k^p$ and there are no valid partial conditional plans for this observation o. Then, execution fails due to unsuccessful replanning. In this case, though we cannot achieve the safe-reachability objective, a guarantee of the robot still satisfying the safety requirement is preferable to the situation where the robot violates the safety requirement. Our approach OPCPS can provide this safety guarantee by checking whether the successor belief of every uncovered observation $o \in \mathcal{O} - \mathcal{O}_k^p$ of the constructed partial conditional plan γ_k^p is a *safe* belief (lines 54–57).

C. Caching

The algorithm we have discussed so far recursively constructs a partial conditional plan for every sampled belief state. In some cases, those sampled beliefs are revisited under similar k-step plans starting from the initial belief. For instance, different invalid k-step plans can lead to the same belief state that violates our safety requirement. The original OPCPS presented in [22] does not cache partial conditional plans for sampled belief states, resulting in repetitive computation of partial conditional plans for revisited belief states. Computing partial conditional plans requires invoking the incremental SMT solver, which is typically quite expensive. Therefore, it is more efficient to reuse previous computed partial conditional plans rather than constructing a new one from scratch. Moreover, for revisited belief states that violate the safety constraints and thus correspond to the empty partial conditional plan ϕ, caching also helps quickly invalidate the plans since we cached the empty partial conditional plan ϕ for these invalid beliefs.

Algorithm 3 augments the corresponding procedure from [22] with caching. For every sampled belief state, we first check whether this belief state is in the cache (line 46). In this work, we are focusing on *discrete* POMDPs and the belief state specifies the probability for each discrete state, which can be represented as a finite vector. When checking whether a belief state is in the cache, we are checking whether the belief state matches any belief state in the cache, i.e., we compare finite vectors. If we find this belief state in the cache, we can reuse the previous computed partial conditional plan (line 47). Otherwise, we compute a partial conditional plan for this belief state as in the previous OPCPS (line 49). Then, we cache the new partial conditional plans for this sampled belief state (line 50). One can argue that in a large belief space caching, each sampled belief might not be a feasible approach. However, we are dealing with the goal constrained belief space $\mathcal{B}_{\mathcal{G}}$, which is generally much smaller than the reachable belief space $\mathcal{B}_{b_0}$. In our case, the lack of caching previously computed partial conditional plan leads to a slower convergence rate. To provide some intuitions, consider an invalid plan where there is a constraint violation near the goal belief but far from the initial belief. The incremental SMT solver dodges this violation by slightly modifying the k-step plan. In this case, the new k-step plan does not change drastically compared to the previous invalid plan. When not caching the previous solution conditional plans, OPCPS will spend a lot of time to recursively compute a new partial conditional plan at each planning step. It is reasonable to recursively compute a partial conditional plan in very dynamic or adversarial environments where one can observe constraint violation in each planning step. However, in many applications, the environment is mostly static and it is more efficient to reuse a previous solution rather than constructing a new one from scratch.

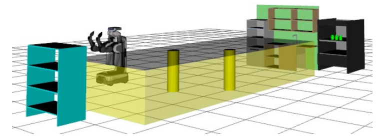


Fig. 5. Kitchen domain [6]: a robot navigates through the kitchen to pick up a green cup from the black storage area (reachability) while avoiding collisions with uncertain obstacles (e.g., chairs) modeled as cylinders placed in the yellow "shadow" region (safety).

D. Algorithm Complexity

In the worst case, OPCPS will generate a full conditional plan (policy) and requires $O(I|\mathcal{O}|^h)$ calls to the SMT solver similar to BPS [6], where I is the number of interactions between plan generation and partial conditional plan generation, $|\mathcal{O}|$ is the size of observation set $\mathcal{O}$, and h is the horizon bound. In general cases, OPCPS can achieve a much better practical performance compared to BPS, due to the carefully designed *partial* conditional plan generation with replanning probability bound update and caching.

V. Experiments

We test OPCPS on the kitchen domain (horizon bound $h = 30$) presented in [6] and the classic tag domain [3] ($h = 100$). We use Z3 [41] as our backend SMT solver. All experiments were conducted on a 3.0 GHz Intel processor with 32 GB of memory. We set the time-out to be 1800 s. For all the tests of the kitchen and tag domains, the results are averaged over 50 independent runs.

In a kitchen domain [6] (see Fig. 5), a robot needs to eventually pick up a cup from the storage while avoiding collisions with M uncertain obstacles. This kitchen domain is an example scenario that requires a correctness guarantee of accomplishing tasks, and POMDPs with safe-reachability objectives provide a better correctness guarantee than the traditional quantitative POMDP formulations [6].

The kitchen environment is discretized into $N = 36$ regions. The actuation and perception of the robot are imperfect, modeled as ten uncertain robot actions: *move* and *look* in four directions, pick-up using the left or right hand. We assume that the robot starts at a known initial location. However, due to the robot's imperfect perception, the location of the robot and the locations of obstacles are all partially observable during execution. This kitchen domain has a large state space $|S| = C(N, M) \cdot N$, where $C(N, M)$ is the number of M-combinations from the set of N regions. In the largest test ($M = 7$), there are more than 10^8 states (see [6] for more details regarding the kitchen domain POMDP setup). We also validate the presented approach on a Fetch robot [44].

A. Performance

We evaluate our previous BPS method [6] and OPCPS (with the replanning probability bound $\delta_{p_{\text{replan}}}$ ranging

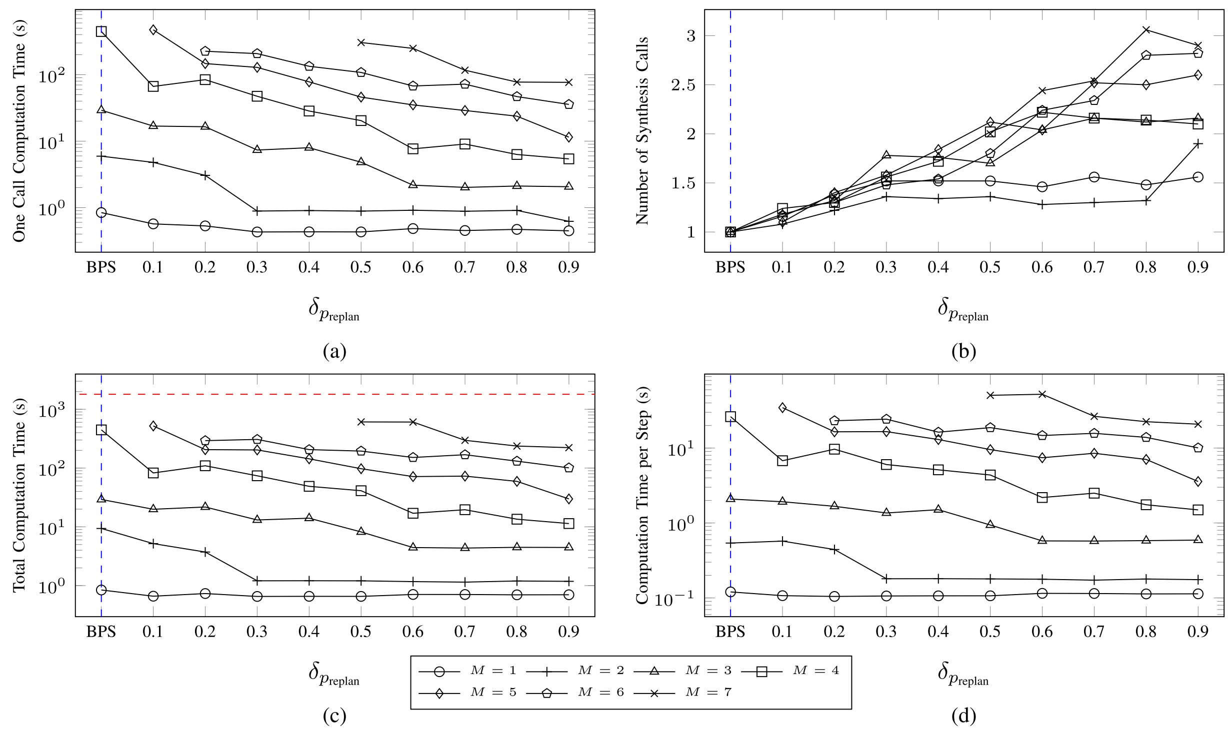


Fig. 6. Performance results for the kitchen domain as the bound $\delta_{p_{\text{replan}}}$ increases. Different plots correspond to tests with different numbers M of obstacles. Missing data points in a plot indicate time-out. The red dashed line is the plot of time = 1800 s (time-out). The blue dashed line passes through the data points generated by BPS. All the results are averaged over 50 independent runs. (a) Average computation time of one synthesis call. (b) Average number of synthesis calls. (c) Average total computation time. (d) Average computation time per step.

from 0.1 to 0.9) in the kitchen domain with various numbers of obstacles. BPS computes a full conditional plan that covers all observation branches and is equivalent to OPCPS with $\delta_{p_{\text{replan}}} = 0$.

Fig. 6(a)–(d) shows the average computation time of one synthesis call, the average number of synthesis calls, the average total computation time, and the average computation time per step as the bound $\delta_{p_{\text{replan}}}$ increases, respectively. As shown in Fig. 6(a) (semilog scale) and (b), the computation time of one synthesis call decreases very quickly, while the number of calls to partial conditional plan synthesis [see Fig. 6(b)] does not increase much as $\delta_{p_{\text{replan}}}$ increases. Therefore, the total computation time [see Fig. 6(c)] keeps decreasing as $\delta_{p_{\text{replan}}}$ increases. In addition, as we can see from Fig. 6(c) (semi-log scale), BPS can only scale up to four obstacles within 1800 s, while OPCPS with replanning probability bound $\delta_{p_{\text{replan}}} = 0.9$ can scale up to seven obstacles. With a small bound $\delta_{p_{\text{replan}}} = 0.1$, we observe a big performance gain compared to BPS: for the test case with $M = 4$ obstacles, the speedup is around five times, and for the test case with $M = 5$ obstacles, BPS times out, while OPCPS with $\delta_{p_{\text{replan}}} = 0.1$ can solve this test in around 9 min. Therefore, OPCPS achieves better performance than BPS in the tests by computing *partial* conditional plans to approximate *full* conditional plans. The results of the average computation time per step [Fig. 6(d)] also show the same trend. These results suggest that for domains where replanning is easy, increasing the replanning probability bound usually leads to better scalability.

B. Success Rate

For all the previous performance tests, the constructed partial conditional plans by OPCPS with different bounds $\delta_{p_{\text{replan}}}$ always achieve the safe-reachability objective (success rate = 100%) because the robot can move in four directions. When the robot enters a region surrounded by obstacles in three directions, the robot can always move back to its previous position, which means that replanning is always possible. However, in some domains such as autonomous driving and robot chefs, when the robot commits to an action and finds something wrong, it is difficult or impossible to reverse the action effects and replan. To evaluate how OPCPS performs in these scenarios, we test OPCPS in the kitchen domain with different numbers M of obstacles ($M \leq 4$ since BPS times out for tests with more than four obstacles), but we disable the robot's *move-north* action. Therefore, when the robot performs *move-south* and enters a region surrounded by obstacles in three directions, replanning fails. However, the robot still satisfies the safety requirement, due to the safety guarantee of OPCPS.

Fig. 7 shows the success rate as the bound $\delta_{p_{\text{replan}}}$ increases. For all the tests, the success rate is always greater than $1.0-\delta_{p_{\text{replan}}}$ (all data points are above the plot of success rate = $1.0 - \delta_{p_{\text{replan}}}$). This matches Theorem 1: the failure rate of a valid partial conditional plan is bounded by the replanning probability. Moreover, as the bound $\delta_{p_{\text{replan}}}$ decreases to 0, OPCPS produces a valid full conditional plan with 100% success rate. These results suggest that for some domains where we anticipate that replanning is difficult, users can

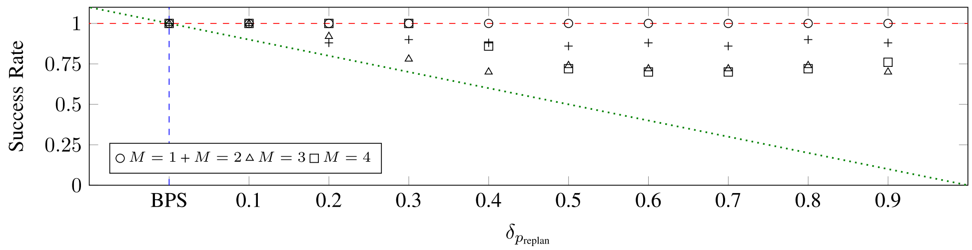


Fig. 7. Success rate as $\delta_{p_{\text{replan}}}$ increases. The green dotted line shows the plot of success rate $= 1.0 - \delta_{p_{\text{replan}}}$. The red dashed line is the plot of success rate $= 1.0$. The blue dashed line passes through the data points generated by BPS.

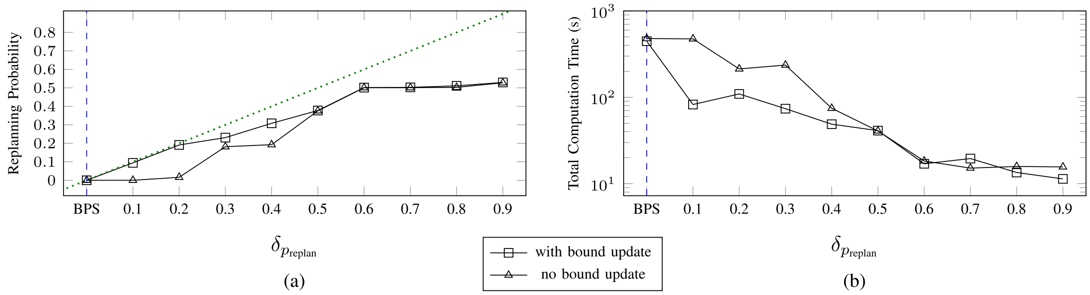


Fig. 8. Replanning probability and total computation time as the bound $\delta_{p_{\text{replan}}}$ increases ($M = 4$). The green dotted line shows the plot of replanning probability $= \delta_{p_{\text{replan}}}$. The blue dashed line passes through the data points generated by BPS. (a) Average replanning probability. (b) Average total computation time.

decrease the bound $\delta_{p_{\text{replan}}}$ and allocate computational resources for a high success rate.

Note that the replanning probability bound is a conservative upper bound of the failure rate since it pessimistically assumes all the uncovered observation branches that require replanning will fail, which is a rare case in practice. As we can see from Fig. 7, even with a high replanning probability bound $\delta_{p_{\text{replan}}} = 0.9$, the failure rate is at most 30%, which is much smaller than the given bound $\delta_{p_{\text{replan}}} = 0.9$.

C. Gains From Updating Replanning Probability Bound

As we discussed in Section IV-B, updating the replanning probability bound during partial conditional plan generation is important for avoiding unnecessary computation and improving efficiency. To evaluate the gains from this bound update step, we test OPCPS with and without the bound update in the kitchen domain with $M = 4$ obstacles.

Fig. 8(a) and (b) (semilog scale) shows the average replanning probability of the constructed partial conditional plans and the average total computation time as the bound $\delta_{p_{\text{replan}}}$ increases, respectively. As shown in Fig. 8(a), with both settings (with and without the bound update) OPCPS constructs a partial conditional plan with a replanning probability smaller than $\delta_{p_{\text{replan}}}$. However, OPCPS without the bound update constructs a partial conditional plan with a lower replanning probability than that constructed by OPCPS with the bound update. Therefore, OPCPS without the bound update performs unnecessary computation and constructs a partial conditional plan with more branches and thus spends more time than OPCPS with the bound update, as shown in Fig. 8(b). For the tests with $\delta_{p_{\text{replan}}} = 0.1, 0.2, 0.3$ that take more time to solve than those with $\delta_{p_{\text{replan}}} > 0.3$, OPCPS with the bound update achieves a 2–5 times speedup.

D. Gains From Caching

To evaluate the gains from caching, we compare the performance of OPCPS with caching against BPS and OPCPS without caching in the kitchen domain. For the kitchen domain with the number of obstacles ranging from 5 to 7, we compare the results from OPCPS with and without caching only since BPS is not able to solve these problems within the time limit. We evaluated OPCPS with or without caching in the kitchen domain with different replanning probability thresholds. In Fig. 9, we present a complete benchmark for the performance evaluation of OPCPS with replanning probability bound $\delta_{p_{\text{replan}}} = 0.5$. We can see that OPCPS with caching performs much better than BPS and OPCPS. In fact, OPCPS with caching is 2.5 times faster on average. Our experimental results demonstrate that OPCPS with caching gains computational efficiency by reusing previously computed conditional plans.

However, it is often a question whether or not the better performance of OPCPS with caching holds with different replanning probability bounds $\delta_{p_{\text{replan}}}$. Because of the huge computational times involved (e.g., $\delta_{p_{\text{replan}}} = 0.7$ with $5, 6, 7$ obstacles require 72–96 CPU hours), we present a spot check in Table I for assessing performance gains from caching with different values of $\delta_{p_{\text{replan}}}$. From Table I, we observe similar trends for different replanning probability thresholds, e.g., $\delta_{p_{\text{replan}}} = 0.9, 0.8, 0.7, 0.6$. Even when we choose a higher replanning probability, OPCPS with caching is 40%–57% faster than OPCPS without caching both in average and worst case runs.

E. Physical Validation

We conducted several physical validations using the mobile manipulator Fetch [44], which is equipped with a single 7-DOF arm, as well as a base-mounted laser scanner and a

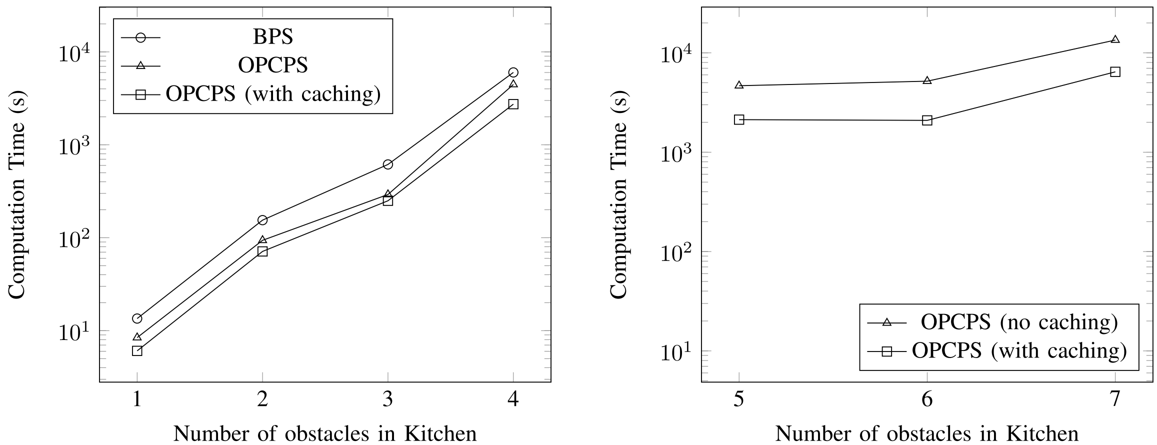


Fig. 9. Performance comparison between BPS, OPCPS with and without caching.

TABLE I

SPOT CHECK FOR ASSESSING PERFORMANCE GAINS FROM CACHING WITH DIFFERENT VALUES OF $\delta_{p_{\text{replan}}}$. THE RESULTS SHOWED THAT THE PERFORMANCE OF OPCPS IS SIGNIFICANTLY IMPROVED WITH CACHING FOR BOTH AVERAGE AND WORST CASE CONDITIONS. ALL RESULTS ARE AVERAGED OVER 50 INDEPENDENT RUNS

$\delta_{p_{\text{replan}}}$	Obstacle	Average Runtime (sec)		Worst-case Runtime (sec)	
		With Cache	Without Cache	With Cache	Without Cache
0.9	1	14	24	15	25
0.8	2	9	22	16	43
0.7	5	446	1047	1246	2863
0.6	6	2209	4031	3476	6015
0.7	7	1090	2542	2744	6001

(a) (b) (c) (d) (e)

Fig. 10. Physical validation of OPCPS for the domain shown in Fig. 1. (a) Initial state. (b) Move south. (c) Move west. (d) Move west. (e) Pick up the target.

head-mounted 3-D camera for perception. These validations were initially attempted in simulation using Gazebo [45], where the Fetch can be simulated over different environments. Both the simulated and the real-world robots are fully controlled via the robot operating system (ROS) [46]. The software control architecture of the robot includes a simultaneous localization and mapping (SLAM) system [47]. The SLAM utilizes the laser information to incrementally create a 2-D map of the surroundings, which is used to provide a global localization of the robot [48]. For navigation purposes, the robot is equipped with a *move* action that takes the robot to a given position and orientation with respect to a global reference.

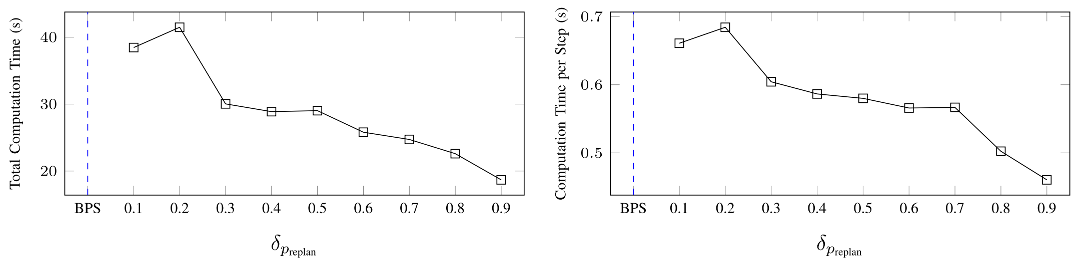
We validate OPCPS on the Fetch for the domain shown in Fig. 1. The setup is similar to the kitchen domain. The Fetch needs to pick up a target object (the blue can on the table) while avoiding collisions with uncertain obstacles such as floor signs and file cabinets, which can be placed in different locations. The POMDP's state space consists of locations of the robot and objects. We use the Vicon tracking system [49] to detect object locations, which is often accurate but can still produce false negative and false positive due to occlusion or inappropriate Vicon marker configurations on objects. We estimate the false negative and false positive probabilities by counting the false negative and false positive events during 100 Vicon detections. The POMDP's probabilistic observation function is defined based on the false negative and false positive probabilities.

To test the effects of different replanning probability bounds, we only allow the Fetch to move in three directions (west, east, and south), similar to the setup of the success rate experiments. Sometimes, the Fetch may fail to move its base when given a *move* action command and stay in the same place. We estimate the failure probability of these move actions by counting the failure events during 100 *move* action executions. The POMDP's probabilistic transition function is defined based on this failure probability. Fig. 10(a) shows the initial state. There are two uncertain obstacles (a wet-floor sign and a file cabinet). We test OPCPS with two bounds $\delta_{p_{\text{replan}}} = 0.9$ and $\delta_{p_{\text{replan}}} = 0.1$.

With $\delta_{p_{\text{replan}}} = 0.9$, after observing no obstacle in the south direction, the Fetch decides to move south [see Fig. 10(b)] because the partial conditional plan constructed with a high replanning probability bound does not cover the case where the Fetch is surrounded by obstacles and the wall. Then, replanning fails, but the Fetch still satisfies the safety requirement

(a) (b) (c) (d)

Fig. 11. Physical validation of OPCPS with caching for the lab domain. The Fetch requires to reach the table in (d) while avoiding white cabinets. The red rectangle in (a) and the blue rectangle in (d) represent the target and the start locations, respectively. The black line is the traversed path, while following the policy generated by OPCPS with caching. (a) Initial state. (b) Move north. (c) Move north. (d) Final state: goal reached.



(a) (b)

Fig. 12. Performance results for the tag domain as the replanning probability bound $\delta_{p_{\text{replan}}}$ increases. All the results are averaged over 50 independent runs. (a) Average total computation time. (b) Average total computation time per step.

as shown in Fig. 10(b), due to the safety guarantee provided by OPCPS.

However, with $\delta_{p_{\text{replan}}} = 0.1$, after observing no obstacles in the south direction, the Fetch decides to move west [see Fig. 10(c)] because the partial conditional plan constructed with a low replanning probability bound covers the case where the robot is surrounded by obstacles. In order to avoid this situation, the Fetch needs to move west and gather more information. Then, the Fetch observes an obstacle in the south direction and decides to move west again [see Fig. 10(d)]. Next, the Fetch observes no obstacle in the south direction, and now, it can move south. Unlike the case shown in Fig. 10(b) where the robot is surrounded by two obstacles and the wall, in the situation shown in Fig. 10(d), if there is another obstacle in the south direction, the Fetch can still move west since there are only two obstacles. Finally, the Fetch moves to the table and picks up the target object [see Fig. 10(e)].

We also validate OPCPS with caching on a Fetch robot in the same lab domain. The Fetch needs to reach a goal location while avoiding collisions with uncertain obstacles such as file cabinets. In this experiment, there are two uncertain obstacles (white cabinets) in the lab domain. The executions of the policy generated by OPCPS with caching are shown in Fig. 11(a)–(d). As shown in the figures, the Fetch successfully reached the goal location [near the table in Fig. 11(d)] following the policy generated by OPCPS with caching. Our physical experiments show that the assumptions made in this work can correspond to realistic settings and that the behavior of the real robot is intuitive and correct.

F. Tag Domain

To further demonstrate the advantage of OPCPS over BPS, we evaluate OPCPS on a classic POMDP domain [3]. The task for the robot is to search for and tag a moving agent in a grid with 29 locations. The agent follows a fixed strategy that intentionally moves away from the robot. Both the robot and the agent can move in four directions or stay. The robot's location is fully observable, whereas the agent's location is unobservable unless the robot and the agent are in the same location.

This tag domain is challenging for BPS because of a large number of observations ($|\mathcal{O}| = 30$) and, more importantly, a huge planning horizon for computing a full conditional plan. However, computing a full conditional plan is unnecessary since replanning is easy in this domain. Fig. 12(a) and (b) shows that the average total computation time and the average computation time per step for the reachability provide as the bound $\delta_{p_{\text{replan}}}$ increases. These results show a similar trend to the previous kitchen domain tests: with a small bound $\delta_{p_{\text{replan}}} = 0.1$, we observe a big performance gain compared to BPS. BPS cannot solve this test within the 1800-s time limit, whereas OPCPS with $\delta_{p_{\text{replan}}} = 0.1$ can solve this test in around 40 s and the computation time per step is less than 1 s. We also perform a spot check for assessing performance gains from caching in this domain as well. With $\delta_{p_{\text{replan}}} = 0.4$, we observe a significant performance gain compared to OPCPS without caching. In this setting, OPCPS without caching takes 658 s on average and 1541 s on worst cases, whereas OPCPS with caching takes 254 s on average and 611 s on worst case to solve this test.

VI. Discussion

We presented a new approach, called OPCPS, to policy synthesis for POMDPs with safe-reachability objectives. We introduce the notion of a *partial conditional plan* to improve computational efficiency. Rather than explicitly enumerating all possible observations to construct a *full* conditional plan, OPCPS samples a subset of all observations to ensure bounded replanning probability. Our theoretical and empirical results show that the failure rate of a valid partial conditional plan is bounded by the replanning probability. Moreover, OPCPS guarantees that the robot still satisfies the safety requirement when replanning fails. Compared to our previous BPS method [6], OPCPS with a proper replanning probability bound scales better in the tested domains and can solve problems that are beyond the capabilities of BPS within the time limit. The results also suggest that for domains where replanning is easy, increasing the replanning probability bound usually leads to better scalability, and for domains where replanning is difficult or impossible in some states, we can decrease the replanning probability bound and allocate more computation time to achieve a higher success rate. Our results also indicate that by updating the replanning probability bound during partial conditional plan generation, we can quickly detect whether the current partial conditional plan satisfies the bound and avoid unnecessary computation. Moreover, compared to OPCPS without caching, OPCPS with caching reuses constructed partial conditional plans for sampled belief states and greatly improves the computational efficiency as shown in the results.

In this work, we focus on discrete POMDPs. While many robot applications can be modeled using this discrete representation, discretization often suffers from the curse of dimensionality. Investigating how to deal with continuous POMDPs [8], [9], [11], [20] directly is a promising future direction. OPCPS constructs partial conditional plans by sampling observations according to the probability of occurrence (Algorithm 3, line 44), which does not consider the importance of observations [13]. How to extend OPCPS to handle critical observations is another important ongoing question.

ACKNOWLEDGMENT

The authors would like to thank the reviewers for their insightful comments. They also thank Bryce Willey and Constantinos Chamzas for their assistance in the physical experiments.

REFERENCES

[1] R. D. Smallwood and E. J. Sondik, "The optimal control of partially observable Markov processes over a finite horizon," *Operations Res.*, vol. 21, no. 5, pp. 1071–1088, Oct. 1973.

[2] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra, "Planning and acting in partially observable stochastic domains," *Artif. Intell.*, vol. 101, nos. 1–2, pp. 99–134, May 1998.

[3] J. Pineau, G. Gordon, and S. Thrun, "Point-based value iteration: An anytime algorithm for POMDPs," in *Proc. IJCAI*, 2003, pp. 1025–1030.

[4] K. Chatterjee, M. Chmelik, R. Gupta, and A. Kanodia, "Qualitative analysis of POMDPs with temporal logic specifications for robotics applications," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2015, pp. 325–330.

[5] P. Cai, Y. Luo, D. Hsu, and W. S. Lee, "HyP-DESPOT: A hybrid parallel algorithm for online planning under uncertainty," *Int. J. Robot. Res.*, Jul. 2020, doi: 10.1177/0278364920937074.

[6] Y. Wang, S. Chaudhuri, and L. E. Kavraki, "Bounded policy synthesis for POMDPs with safe-reachability objectives," in *Proc. AAMAS*, 2018, pp. 238–246.

[7] S.-K. Kim, R. Thakker, and A.-A. Agha-Mohammadi, "Bi-directional value learning for risk-aware planning under uncertainty," *IEEE Robot. Autom. Lett.*, vol. 4, no. 3, pp. 2493–2500, Jul. 2019.

[8] J. Hoey and P. Poupart, "Solving POMDPs with continuous or large discrete observation spaces," in *Proc. IJCAI*, 2005, pp. 1332–1338.

[9] J. M. Porta, N. Vlassis, M. T. J. Spaan, and P. Poupart, "Point-based value iteration for continuous POMDPs," *J. Mach. Learn. Res.*, vol. 7, pp. 2329–2367, Nov. 2006.

[10] H. Kurniawati, D. Hsu, and W. Sun Lee, "SARSOP: Efficient point-based POMDP planning by approximating optimally reachable belief spaces," in *Proc. Robot., Sci. Syst. IV*, Jun. 2008. [Online]. Available: http://www.roboticsproceedings.org/rss04/p9.html

[11] H. Bai, D. Hsu, and W. S. Lee, "Integrated perception and planning in the continuous space: A POMDP approach," *Int. J. Robot. Res.*, vol. 33, no. 9, pp. 1288–1302, Aug. 2014.

[12] A. Somani, N. Ye, D. Hsu, and W. S. Lee, "DESPOT: Online POMDP planning with regularization," in *Proc. NIPS*, 2013, pp. 1772–1780.

[13] Y. Luo, H. Bai, D. Hsu, and W. S. Lee, "Importance sampling for online planning under uncertainty," *Int. J. Robot. Res.*, vol. 38, nos. 2–3, pp. 162–181, Mar. 2019.

[14] A. Undurti and J. P. How, "An online algorithm for constrained POMDPs," in *Proc. IEEE Int. Conf. Robot. Autom.*, May 2010, pp. 3966–3973.

[15] M. Svoreňová *et al.*, "Temporal logic motion planning using POMDPs with parity objectives: Case study paper," in *Proc. HSCC*, 2015, pp. 233–238.

[16] K. Chatterjee, M. Chmelík, and J. Davies, "A symbolic SAT-based algorithm for almost-sure reachability with small strategies in POMDPs," in *Proc. AAAI*, 2016, pp. 3225 –3232.

[17] K. Chatterjee, M. Chmelík, R. Gupta, and A. Kanodia, "Optimal cost almost-sure reachability in POMDPs," *Artif. Intell.*, vol. 234, pp. 26–48, May 2016.

[18] L. P. Kaelbling and T. Lozano-Pérez, "Integrated task and motion planning in belief space," *Int. J. Robot. Res.*, vol. 32, nos. 9–10, pp. 1194–1227, Aug. 2013.

[19] D. Hadfield-Menell, E. Groshev, R. Chitnis, and P. Abbeel, "Modular task and motion planning in belief space," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Sep. 2015, pp. 4991–4998.

[20] K. M. Seiler, H. Kurniawati, and S. P. N. Singh, "An online and approximate solver for POMDPs with continuous action space," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2015, pp. 2290–2297.

[21] L. P. Kaelbling and T. Lozano-Pérez, "Implicit belief-space pre-images for hierarchical planning and execution," in *Proc. ICRA*, May 2016, pp. 5455–5462.

[22] Y. Wang, S. Chaudhuri, and L. E. Kavraki, "Online partial conditional plan synthesis for POMDPs with safe-reachability objectives," in *Proc. WAFR*, 2018, pp. 127–143.

[23] J. D. Isom, S. P. Meyn, and R. D. Braatz, "Piecewise linear dynamic programming for constrained POMDPs," in *Proc. AAAI*, 2008, pp. 291–296.

[24] D. Kim, J. Lee, K.-E. Kim, and P. Poupart, "Point-based value iteration for constrained POMDPs," in *Proc. IJCAI*, 2011, pp. 1968–1974.

[25] P. Poupart, A. Malhotra, P. Pei, K.-E. Kim, B. Goh, and M. Bowling, "Approximate linear programming for constrained partially observable Markov decision processes," in *Proc. AAAI*, 2015, pp. 3342–3348.

[26] P. Santana, S. Thiébaux, and B. Williams, "RAO*: An algorithm for chance-constrained POMDP's," in *Proc. AAAI*, 2016, pp. 3308–3314.

[27] J. Marecki and P. Varakantham, "Risk-sensitive planning in partially observable environments," in *Proc. AAMAS*, 2010, pp. 1357–1368.

[28] P. Hou, W. Yeoh, and P. Varakantham, "Solving risk-sensitive POMDPs with and without cost observations," in *Proc. AAAI*, 2016, pp. 3138–3144.

[29] D. Hadfield-Menell, S. Milli, P. Abbeel, S. J. Russell, and A. Dragan, "Inverse reward design," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 30, 2017, pp. 6765–6774.

[30] Y. Chow, O. Nachum, E. Duenez-Guzman, and M. Ghavamzadeh, "A Lyapunov-based approach to safe reinforcement learning," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 31, 2018, pp. 8092–8101.

[31] H.-T.-L. Chiang, A. Faust, M. Fiser, and A. Francis, "Learning navigation behaviors End-to-End with AutoRL," *IEEE Robot. Autom. Lett.*, vol. 4, no. 2, pp. 2007–2014, Apr. 2019.

[32] G. Flaspohler, N. A. Roy, and J. W. Fisher, III., "Belief-dependent macro-action discovery in POMDPs using the value of information," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 33, 2020. [Online]. Available: https://papers.nips.cc/paper/2020/hash/7f2be1b45d278ac18804b79207a24c53-Abstract.html

[33] S. Bhattacharya, S. Badyal, T. Wheeler, S. Gil, and D. Bertsekas, "Reinforcement learning for POMDP: Partitioned rollout and policy iteration with application to autonomous sequential repair problems," *IEEE Robot. Autom. Lett.*, vol. 5, no. 3, pp. 3967–3974, Jul. 2020.

[34] N. P. Garg, D. Hsu, and W. S. Lee, “Learning to grasp under uncertainty using POMDPs,” in *Proc. Int. Conf. Robot. Autom. (ICRA)*, May 2019, pp. 2751–2757.
[35] C. H. Papadimitriou and J. N. Tsitsiklis, “The complexity of Markov decision processes,” *Math. Oper. Res.*, vol. 12, no. 3, pp. 441–450, 1987.
[36] A. Paz, *Introduction to Probabilistic Automata*. New York, NY, USA: Academic, 1971.
[37] O. Madani, S. Hanks, and A. Condon, “On the undecidability of probabilistic planning and related stochastic optimization problems,” *Artif. Intell.*, vol. 147, nos. 1–2, pp. 5–34, Jul. 2003.
[38] K. Chatterjee, M. Chmelík, and M. Tracol, “What is decidable about partially observable Markov decision processes with ω-regular objectives,” *J. Comput. Syst. Sci.*, vol. 82, no. 5, pp. 878–911, 2016.
[39] M. Mundhenk, J. Goldsmith, C. Lusena, and E. Allender, “Complexity of finite-horizon Markov decision process problems,” *J. ACM*, vol. 47, no. 4, pp. 681–720, Jul. 2000.
[40] S. Ross, J. Pineau, S. Paquet, and B. Chaib-Draa, “Online planning algorithms for POMDPs,” *J. Artif. Intell. Res.*, vol. 32, pp. 663–704, Jul. 2008.
[41] L. De Moura and N. Bjørner, “Z_3: An efficient SMT solver,” in *Proc. TACAS*, 2008, pp. 337–340.
[42] A. Biere, A. Cimatti, E. M. Clarke, O. Strichman, and Y. Zhu, “Bounded model checking,” *Adv. Comput.*, vol. 58, pp. 117–148, 2003.
[43] N. T. Dantam, Z. K. Kingston, S. Chaudhuri, and L. E. Kavraki, “An incremental constraintbased framework for task and motion planning,” *Int. J. Robot. Res.*, vol. 37, no. 10, pp. 1134–1151, Sep. 2018.
[44] M. Wise, M. Ferguson, D. King, E. Diehr, and D. Dymesich, “Fetch and freight: Standard platforms for service robot applications,” in *Proc. Workshop Auton. Mobile Service Robots, Held Int. Joint Conf. Artif. Intell.*, 2016.
[45] N. Koenig and A. Howard, “Design and use paradigms for Gazebo, an open-source multi-robot simulator,” in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, vol. 3, Sep./Oct. 2004, pp. 2149–2154.
[46] M. Quigley *et al.*, “ROS: An open-source robot operating system,” in *Proc. ICRA Workshop Open Source Softw.*, 2009.
[47] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*. Cambridge, MA, USA: MIT Press, 2005.
[48] G. Grisetti, C. Stachniss, and W. Burgard, “Improved techniques for grid mapping with rao-blackwellized particle filters,” *IEEE Trans. Robot.*, vol. 23, no. 1, pp. 34–46, Feb. 2007.
[49] *Vicon*. Accessed: Jun. 2019. [Online]. Available: https://www.vicon.com/

Yue Wang received the Ph.D. degree in computer science from Rice University, Houston, TX, USA, in 2018.

He is currently a Research Scientist at Facebook, Menlo Park, CA, USA. His research interests include robotics, formal methods, and task and motion planning/synthesis for robotic applications in adversarial and/or partially observable environments.

Abdullah Al Redwan Newaz (Member, IEEE) received the B.Sc. degree in mechanical engineering from the Rajshahi University of Engineering and Technology, Rajshahi, Bangladesh, in 2011, and the M.S. and Ph.D. degrees in information science from the Japan Advanced Institute of Science and Technology, Nomi, Japan, in 2014 and 2017, respectively.

He was a Post-Doctoral Researcher with Nagoya University, Nagoya, Japan, and Rice University, Houston, TX, USA, from 2017 to 2018 and from 2018 to 2020, respectively. He is currently a Post-Doctoral Research Associate with North Carolina Agricultural and Technical State University, Greensboro, NC, USA. His research interests include autonomous systems, applied machine learning, motion planning under uncertainty, optimal control, policy synthesis, model checking, and related domains.

Juan David Hernández (Senior Member, IEEE) received the B.Sc. degree in electronic engineering from Pontifical Xavierian University, Cali, Colombia, in 2009, the M.Sc. degree in robotics and automation from the Technical University of Madrid, Madrid, Spain, in 2012, and the Ph.D. degree in technology (robotics) from the University of Girona, Girona, Spain, in 2017.

He worked as a Robotics Research Engineer at the Netherlands Organisation for Applied Scientific Research (TNO), The Hague, The Netherlands, from 2017 to 2018. He was a Post-Doctoral Research Associate with Rice University, Houston, TX, USA, from 2018 to 2019. He was a Senior Engineer for simulation of autonomous systems at Apple Inc., Sunnyvale, CA, USA, from 2019 to 2020. He is currently a Lecturer (Assistant Professor) with Cardiff University, Cardiff, U.K., where he is part of the Centre for AI, Robotics and Human-Machine Systems (IROHMS). His research is focused on motion planning algorithms and human–robot collaboration.

Dr. Hernández is a Senior Member of the IEEE Robotics and Automation Society.

Swarat Chaudhuri received the bachelor’s degree in computer science from IIT Kharagpur, Kharagpur, India, in 2001, and the Ph.D. degree in computer science from the University of Pennsylvania, Philadelphia, PA, USA, in 2007.

He held faculty positions at Rice University, Houston, TX, USA, and Pennsylvania State University, State College, PA, USA. He is currently an Associate Professor of computer science with The University of Texas at Austin, Austin, TX, USA. His research lies in the intersection of programming languages (PLs) and machine learning (ML). Specifically, he studies ways in which PL and ML techniques can be brought together to build robust and trustworthy intelligent systems targeting complex tasks, such as software development and robot control.

Dr. Chaudhuri was a recipient of the National Science Foundation CAREER Award, the ACM SIGPLAN John Reynolds Doctoral Dissertation Award, and the Morris and Dorothy Rubinoff Dissertation Award from the University of Pennsylvania.

Lydia E. Kavraki (Fellow, IEEE) received the Ph.D. degree in computer science from Stanford University, Stanford, CA, USA, in 1996.

She is currently the Noah Harding Professor of computer science, a Professor of bioengineering, a Professor of electrical and computer engineering, and a Professor of mechanical engineering with Rice University, Houston, TX, USA. She is also the Director of the Ken Kennedy Institute, Rice University. Work in her group has produced the Open Motion Planning Library (OMPL), an open-source library of motion planning algorithms. The library links directly with the Robot Operating System (ROS) and MoveIt, and it is heavily used in industry and in academia. She has authored more than 220 peer-reviewed journal and conference publications and is one of the authors of the widely used robotics textbook titled *Principles of Robot Motion* (MIT Press). Her research interests span robotics, artificial intelligence (AI), and biomedicine. In robotics and AI, she develops algorithms for motion planning for high-dimensional systems with kinematic and dynamic constraints, integrated frameworks for reasoning under sensing and control uncertainty, novel methods for learning and for using experiences, and ways to instruct robots at a high level and collaborate with them. In biomedicine, she develops computational methods and tools to model protein structure and function, understand biomolecular interactions, aid the process of medicinal drug discovery, and help integrate biological and biomedical data for improving human health.

Dr. Kavraki is a member of the National Academy of Medicine (NAM), the Academy of Medicine, Engineering, and Science of Texas (TAMEST), the International Academy of Medical and Biological Engineering (IAMBE), and the Academy of Athens. She is also a fellow of the Association for Computing Machinery (ACM), the American Association for the Advancement of Science (AAAS), the Association for the Advancement of Artificial Intelligence (AAAI), and the American Institute for Medical and Biological Engineering (AIMBE). She received the Association for Computing Machinery (ACM) Grace Murray Hopper Award, the ACM Athena Lecturer Award, the ACM/AAAI Allen Newell Award, and the Robotics Pioneer Award from the IEEE Robotics and Automation Society.