Decentralized Task and Path Planning for Multi-Robot Systems

Yuxiao Chen, Ugo Rosolia, and Aaron D. Ames

***Abstract*—We consider a multi-robot system with a team of collaborative robots and multiple tasks that emerges over time. We propose a fully decentralized task and path planning (DTPP) framework consisting of a task allocation module and a localized path planning module. Each task is modeled as a Markov Decision Process (MDP) or a Mixed Observed Markov Decision Process (MOMDP) depending on whether full states or partial states are observable. The task allocation module then aims at maximizing the expected pure reward (reward minus cost) of the robotic team. We fuse the Markov model into a factor graph formulation so that the task allocation can be decentrally solved using the max-sum algorithm. Each robot agent follows the optimal policy synthesized for the Markov model and we propose a localized forward dynamic programming scheme that resolves conflicts between agents and avoids collisions. The proposed framework is demonstrated with high fidelity ROS simulations and experiments with multiple ground robots.**

I. Introduction

The planning and control of multi-robot systems is an important problem in robotics [1], [2], and its applications can be seen in transportation, logistics robots in manufacturing and e-commerce, rescue missions post disasters, and multi-robot exploration tasks. The planning and control of the robotic agents is a core functionality of the multi-robot system, including the high-level task planning and the low-level path planning and control. Take the famous Kiva warehouse robot as an example [3], the task planning layer determines which robot shall pick up which package, then the path planning layer plans the specific trajectory for each robot in a grid, and the control module tracks the trajectory. Comparing to single robot operations, the core challenges of multi-robot systems are task allocation among multiple robot agents, and the trajectory planning that resolves conflicts between the robot agents.

The problem of task allocation for multi-robot systems has been studied extensively in the literature [4], [5], [6]. Existing task allocation methods include auction or market based methods [7], [8], [9], and optimization-based methods such as mixed-integer programming [10] and generic optimization algorithms [11], [12]. The drinking philosopher problem is utilized for coordination of multiple agents in [13].

Another aspect of task allocation methods is whether they are centralized or decentralized. In the Kiva case, both the task assignment and the path planning are performed centrally, yet this may not be available if the multi-robot system operates in an environment without powerful sensing and computation capabilities. In general, the market/auction-based methods can be solved decentrally [14], but the problem structure needs to be simple enough so that each agent can act as bidders and place their bids on the tasks, which may be difficult when some tasks require multiple agents to cooperate. The optimization-based methods allow for more complicated problem structures, yet may be difficult to solve in a decentralized manner. In [15], the authors proposed the consensus-based bundle algorithm that allows for tasks requiring two agents to complete. However, it requires the agents to enumerate the possible bundles of task allocation and then resolve the conflict to achieve consensus, which does not scale as the number of agents grows. One powerful algorithm is the max-sum algorithm, which is based on the generalized distributive law (GDL) [16]. Other instances of GDL algorithms include the max-product, the sum-product, and the min-sum, and they have been widely used in problems such as belief propagation and factor graph optimization. The distributed nature of max-sum allows it to be used in decentralized optimizations, including task allocation [17], and coordination [18]. The proposed approach DTPP is based on the max-sum algorithm, and we shall show how max-sum is used to optimize the expected team reward in a decentralized manner.

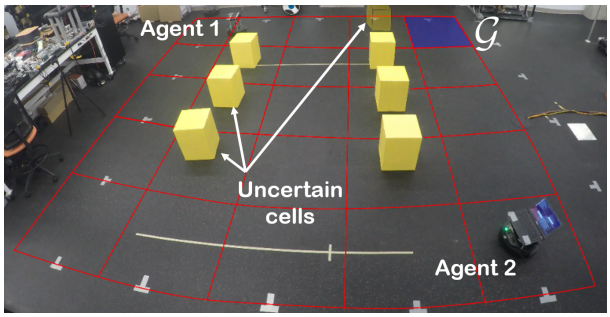

Fig. 1: Experiment with a multi-robot system consisting of Turtlebots in a grid world with obstacles

Motion planning is studied both in the continuous domain and discrete domain. In the continuous domain, more emphasis is put on feasibility and safety rather than optimality, such as the velocity space methods [19], [20], and control barrier functions [21]. The discrete multiagent motion planning problem deals with multiple agents on a graph [22], and was shown in [23] to be NP-hard.

The multi-robot system planning problem has been extended to the case with temporal logic specifications. STAP [24] decomposes linear temporal logic (LTL) formulae into subtasks for a multi-robot system to perform simultaneous task allocation and planning, and [25] focused on syntactically co-safe LTL. Multi-agent planning is also considered under the stochastic setup, such as Markov Decision Processes (MDP)

Manuscript received: Oct. 15, 2020; Revised: Jan 16, 2021, Year; Accepted: Feb 14, 2021.

This paper was recommended for publication by Editor Nancy Amato upon evaluation of the Associate Editor and Reviewers' comments. This work was supported by AFOSR award FA9550-19-1-0302 and NSF award 1932091.

The authors are with the Department of Mechanical and Civil Engineering, Caltech, Pasadena, CA, 91106, USA. Emails: {chenyx, urosolia, ames}@caltech.edu

Digital Object Identifier (DOI): see top of this page.

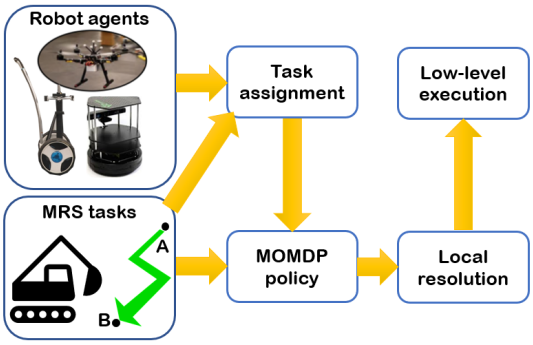

Fig. 2: Overview of system structure

and Partially Observed Markov Decision Processes (POMDP) [26], [27]. Dec-POMDP focuses on multiagent decentralized decision-making modelled as POMDP [28], yet its worst-case complexity is NEXP-complete. Existing works aiming for better scalability include online tree search [29] and Monte-Carlo methods [30].

Contribution We propose the DTPP framework that aims at maximizing the expected pure reward of a multi-robot system, where each robot's transition is modeled as an MOMDP. The overall system structure is shown in Fig. 2. The task allocation module takes in the set of robot agents and multi-robot tasks and determines which task each robot agent commits to. Each robot agent then picks its action based on the MOMDP policy of the task it commits to, which may be modified by the local resolution module if there is a potential conflict with other robot agents. The agents share a common belief over the unobserved states by communicating their observations.

The contributions are threefold.

- in Section II we introduce the multi-robot tasks which are capable of describing tasks for multiple robot agents, potentially requiring coordination.
- Section III presents the task allocation module that solves the task allocation with the highest expected pure reward for the multi-robot system, defined as the expected reward minus the expected cost. The algorithm is based on the max-sum algorithm, which is fully decentralized.
- Section IV presents a local resolution module that resolves potential conflicts between agents using a forward dynamic programming (DP) approach.

Simulation and experiment results are shown in Section V.

II. Multi-robot system agents and tasks

We consider the problem of multi-robot operations consisting of multiple robot agents and multiple tasks that appear over time. The tasks are confined within an environment and can be accomplished within a finite time horizon, such as surveillance over a region, pickup and place, and collecting objects. Tasks that requires infinite time to complete, such as visit two points infinitely often, are not in the scope of this paper. We focus on the high-level task and path planning for the robotic agents, which contains two subproblems to solve: (1) task assignment of multiple tasks among the multiple robot agents (2) path planning for the robot agents.

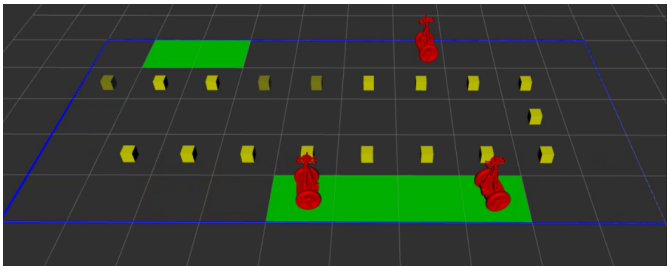

Fig. 3: High-fidelity ROS simulation of 3 segway robots with 3 uncertain regions, the green regions represents the goal regions of the multi-robot tasks

Robot agent modeling. The key idea is to extend the planning methods developed for a single robot to a multi-robot system. We use *Mixed Observable Markov Decision Processes* (MOMDP) to model a single robot planning problem, which is a tuple $(\mathcal{S}, \mathcal{E}, \mathcal{A}, \mathcal{O}, T_s, T_e, O, J)$, where

- $\mathcal{S} = \{1, \ldots, |\mathcal{S}|\}$ is a set of *fully observable states*;
- $\mathcal{E} = \{1, \ldots, |\mathcal{E}|\}$ is a set of *partially observable states*;
- $\mathcal{A} = \{1, \ldots, |\mathcal{A}|\}$ is a set of *actions*;
- $\mathcal{O} = \{1, \ldots, |\mathcal{O}|\}$ is the set of *observations* for the partially observable state $e \in \mathcal{E}$;
- $T_s : \mathcal{S} \times \mathcal{A} \times \mathcal{E} \times \mathcal{S} \to [0, 1]$ is the *observable state transition probability function* where $T_s(s, a, e, s')$ is the probability of the transition from s to s' under action a and partially observable state e.
- $T_e : \mathcal{S} \times \mathcal{E} \times \mathcal{A} \times \mathcal{E} \to [0, 1]$ is the *partially observable state transition probability function* where $T_e(s, e, a, e')$ is the probability of the transition from e to e' given the action a and the current observable state s
- $O : \mathcal{S} \times \mathcal{E} \times \mathcal{A} \times \mathcal{O} \to [0, 1]$ is the *observation function* where $O(s, e, a, o)$ describes the probability of observing the measurement $o \in \mathcal{O}$, given the current state of the system (s, e) and the action a applied at the previous time step
- $J : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \times \mathcal{O} \to \mathbb{R}$ is the *cost function* where $J(s, a, s', o)$ is the cost associated with the transition from s to s' under action a with observation o.

It is assumed that there exists an idle action, denoted as IDLE $\in \mathcal{A}$, which keeps the robot agent at the current state and incurs no cost.

Remark 1. Note that the MOMDP is only given a cost function, this is because all the robot agents incur the same cost as they move in the environment, but the rewards for accomplishing the tasks differ in different tasks, and the same MOMDP is used to describe all tasks.

Remark 2. When all states are observable, the MOMDP is reduced to a Markov Decision Process (MDP).

The MOMDP is shared by all agents in the multi-robot system where each robot agent selects an action at each time step, and the actions are executed simultaneously. A multi-robot system is then abstracted as a tuple $(\mathcal{I}, \text{MOMDP})$, where $\mathcal{I} = [1, 2, ..., N]$ is the indices of the robot agents. The overall cost for the multi-robot system is then the summation of the individual costs based on J. A collision happens when two robot agents are at the same state the same time, and we shall

use the local resolution scheme presented in Section IV to prevent collisions.

In the example used in this paper, the observed state is the position of the robot agent, the partially observable state is whether some regions of the environment are blocked with obstacles or free to pass, such as the half transparent obstacles in Fig. 3. The robot agent can get stochastic observations of the uncertain state, which gets more deterministic as the robot gets closer to the region.

Multi-robot tasks. A multi-robot task is a tuple $(\mathcal{G}, \mathcal{J}, t_0, t_f, \mathcal{R})$, where $\mathcal{G}$ is the goal set of the task, $\mathcal{J} \subseteq \mathcal{I}$ is the set of robot agents involved in this task, which is understood as the candidates for completing the task. t_0 is the starting time of the task and t_f is the ending time of the task. By t_f, the reward shall be collected based on the arrival of robot agents to the goal set of the task. The reward function $\mathcal{R} : \{0,1\}^{|\mathcal{J}|} \to \mathbb{R}$ maps the arrival status of the robot agents to reward value. In the homogeneous case, $\mathcal{R}$ is simply a function of the number of robot agents arriving at the goal set before t_f, while in the heterogeneous case, different robot agents can incur different reward. For simplicity, we focus on the homogeneous case for the remainder of this paper. Let $(r_0, r_1, r_2...)$ be the compact form of $\mathcal{R}$, where r_i denotes the reward with i agents arriving. Albeit simple, the reward function $\mathcal{R}$ can be quite expressive, here are a few examples.

Example 1. $(r_0 = 0, r_1 = 5, r_2 = 5)$ can represent a surveillance task, one robot arriving at the goal set is sufficient, additional robots arriving would not incur additional reward.

Example 2. $(r_0 = 0, r_1 = 5, r_2 = 8)$ can represent the task of moving a pile of sand with the total weight of 8 kg, yet one robot can only carry 5 kg. Therefore, $r_1 = 5$ and $r_2 = 8$.

Example 3. $(r_0 = 0, r_1 = 0, r_2 = 8)$ can represent the task of moving a box that weighs 8 kg, yet one robot can only carry 5 kg. Therefore, one robot arriving cannot move the box, and two robots arriving shall collect the full reward.

We shall show in Section III how the reward function is combined with the MOMDP to maximize the expected pure reward. The tasks are broadcast to the robot agents when they appear and the agents have no information about the tasks in advance. This paper is concerned with the problem of optimally assigning tasks to the robot agents and planning their actions to achieve the highest cumulative reward.

III. Dynamic Task Allocation with Max-Sum

The pure reward of a multi-robot system. Given the MOMDP that describes the robot transition dynamics and a multi-robot task, ideally, one would construct the product MDP/MOMDP for the whole multi-robot system and plan the joint action, yet this is usually not implementable due to the doubly exponential complexity [31]. Instead, we use one single MDP/MOMDP for a task assuming the agents' evolution is independent of each other and let all agents committed to the task run the same policy in parallel. Obviously, the assumption is not true in practice as we use the local resolution scheme to prevent collisions between agents. However, when the multi-robot system is scattered with a relatively low density, i.e., the interactions between agents are not frequent, this assumption can be quite close to reality.

Remark 3. To reflect the potential influence of the local resolution, a higher probability of staying at the current state is assigned to the MOMDP introduced in Section II.

MOMDP policy synthesis. We use an optimal quantitative approach to synthesize the policy for the MOMDP where the policy optimizes the cost function over all policies that maximize the probability of satisfying the specification, which is to reach the goal set before the terminal time. Given a MOMDP, a goal set $\mathcal{G}$, and a horizon t_f (t_0 is set to 0 for notational simplicity), the optimal quantitative synthesis problem is the following:

$$\begin{aligned} \pi^\star = \arg\max_{\pi} \quad & \mathbb{E}^{\pi}\left[\sum_{t=0}^{t_f-1} -J(s_t, a_t, s_{t+1}, o_{t+1})\right] \\ \text{subject to} \quad & \pi \in \arg\max\nolimits_{\kappa} \mathbb{P}^{\kappa}\left[\bigvee_{t=0}^{t_f} s_t \in \mathcal{G}\right], \end{aligned} \tag{1}$$

which solves for the policy that minimizes the expected cost among all policies that maximize the probability of reaching the goal set $\mathcal{G}$ before t_f. Problem (1) is using a point-based strategy as in [32], [33], see Appendix A for more detail. $\pi : \mathcal{S} \times \mathcal{B}_{\mathcal{E}} \to \mathcal{A}$ is a policy for the MOMDP that maps the current state and the belief vector b to an action, where $b \in \mathcal{B}_{\mathcal{E}} \doteq \{b \in \mathbb{R}^{|\mathcal{E}|}_{\geq 0} \mid \mathbb{1}^\intercal b = 1\}$. The solution of (1) consists of two value functions V_J and $V_{\mathcal{G}}$, and the optimal policy $\pi^\star$. The two value functions have clear physical meanings:

$$\begin{aligned} V_J(t, s, b) &= \mathbb{E}^{\pi^\star}[\textstyle\sum_{\tau=t}^{t_f} -J(s_\tau, a_\tau, s_{\tau+1}, o_{\tau+1}) | e_t \sim b] \\ V_{\mathcal{G}}(t, s, b) &= \mathbb{P}^{\pi^\star}[\textstyle\bigvee_{\tau=t}^{t_f} s_\tau \in \mathcal{G} | e_t \sim b], \end{aligned} \tag{2}$$

that is, given the time, state, and belief vector, V_J represents the expected negative cost-to-go, and $V_{\mathcal{G}}$ represents the probability of reaching the goal set before t_f. in the MDP case, V_J and $V_{\mathcal{G}}$ would be functions of only t and s.

With V_J and $V_{\mathcal{G}}$, the expected pure reward of a task can then be approximated given the robot agents committed to the task. In the homogeneous agent case, this is simply

$$\mathbb{E}[R] = \sum\nolimits_{i=0}^{|\mathcal{J}|} r_i P_c[i] + \sum\nolimits_{j \in \mathcal{J}} V_J(t, s_t^j, b), \tag{3}$$

where $\mathcal{J} \subseteq \mathcal{I}$ is the set of robot agents committed to the task, $P_c[i]$ is the cumulative probability of exactly i agents arriving at the goal set by t_f, and is calculated as

$$p_i = \sum\nolimits_{c^j \in \mathbb{B}, j \in \mathcal{J} | \sum_j c^j = i} V_{\mathcal{G}}(t, s_t^j, b)^{c^j} (1 - V_{\mathcal{G}}(t, s_t^j, b))^{1-c^j}.$$

For example, suppose $\mathcal{J} = \{1, 2\}$ and $p_j = V_{\mathcal{G}}(t, s_t^j, b), j = 1, 2$ are the probabilities of the two agents reaching the goal set by t_f from their current state and time, respectively, which are directly obtained from $V_{\mathcal{G}}$. Then P_c is calculated as

$$\begin{aligned} P_c[0] &= (1 - p_1)(1 - p_2) \\ P_c[1] &= p_1(1 - p_2) + (1 - p_1)p_2, \\ P_c[2] &= p_1 p_2. \end{aligned}$$

Given a multi-robot system with a set $\mathcal{K}$ of multiple tasks, let $M^i(t)$ be the set of all tasks that involve robot agent i at time t plus $\emptyset$, $m^i \in M^i(t)$ be the commitment variable where $m^i = k$ indicates that robot agent i is committed to task k, and $m^i = \emptyset$ indicates that robot agent i is not committed to any task. When $m^i = \emptyset$, it is assumed that the robot agent would stay still and incurs zero cost. The expected pure reward for each task then can be computed with (3). We let $F_k(\{m^i\}_{\mathcal{J}_k})$ denote the expected pure reward of task k as a function of the commitment of the robot agents in the candidate set $\mathcal{J}_k$. Note that each $\mathcal{J}_k \subseteq \mathcal{I}$ and they may have overlaps, i.e., one agent can be included in the candidate set of multiple tasks. The simplest choice is to take $\mathcal{J}_k = \mathcal{I}$ for all $k \in \mathcal{K}$, but in practice, one can exclude some robot agents with little chance of completing the task (e.g. agents that are too far away), which accelerates the computation.

Factor graph and the max-sum algorithm. It can be easily verified that the total expected pure reward is $\sum_{k\in\mathcal{K}} F_k(\{m^i\}_{\mathcal{J}_k})$, which is a function of $\{m^i\}_{\mathcal{I}}$. Note that the expected reward for each task is calculated based on the commitment of the robot agents, and the expected cost of each agent is summed up except the ones that are not committed to any tasks, which incurs zero cost. The dynamic task assignment problem solves for the commitment of the robot agents that leads to the largest expected pure reward:

$$\max_{\{m^i\}_{\mathcal{I}}} \sum_{k\in\mathcal{K}} F_k(\{m^i\}_{\mathcal{J}_k}). \tag{4}$$

The task assignment is "dynamic" because (4) changes over time, and is solved in every time step.

To this point, (4) is in the form of a factor graph, which is a bipartite graph representing the factorization of a function. It contains two types of nodes, variable nodes and factor nodes. In our case, the variable nodes are the commitment m^i of the robot agents, and the factor nodes are the expected pure reward F_k of each tasks. As an example, Fig. 4 shows the factor graph with 4 robot agents and 3 multi-robot tasks, where $\mathcal{J}_1 = \{1, 4\},\ \mathcal{J}_2 = \{1, 2\},\ \mathcal{J}_3 = \{1, 3, 4\}$.

We then use the max-sum algorithm to solve the task assignment problem, similar to [17]. The max-sum algorithm seeks to maximize the sum of all factors via exchanging messages between the factor nodes and the variable nodes. To be specific, two types of messages are exchanged: the q messages from variables to factors, and the r messages from factors to variables:

$$\begin{aligned} q_{i\to k}(m^i) &= \alpha_{ik} + \sum\nolimits_{n\in M^i\setminus k} r_{n\to i} \\ r_{k\to i}(m^i) &= \max_{\mathcal{J}_k\setminus i}[F_k(\{m^i\}_{\mathcal{J}_k}) + \sum\nolimits_{n\in\mathcal{J}_k\setminus i} q_{n\to k}(m^n)]. \end{aligned} \tag{5}$$

where α_{ik} is for normalization. All the messages are exchanged locally and no central coordination is needed. Once the messages converge, the optimal solution can be solved as

$$m^{i\star} = \arg\max \sum_{k\in M^i} r_{k\to i}(m^i). \tag{6}$$

The max-sum algorithm is guaranteed to converge for acyclic graphs. Although there is no convergence guarantee on cyclic graphs, multiple empirical studies show that the solution quality is decent without convergence. Moreover, there exist variations of max-sum that return suboptimal solutions with a bounded optimality gap [34].

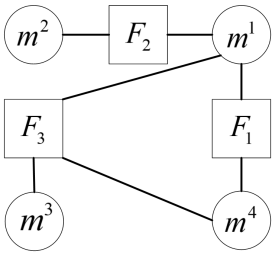

Fig. 4: Factor graph of 4 robot agents and 3 tasks

Algorithm 1 Task Allocation with max-sum

procedure TASK_ALLOCATION($\{s_t^i\}_{\mathcal{I}}$, $\mathcal{K}$, b_t, Maxiter)
  for $k \in \mathcal{K}$ **do**
    for $\{m^i\}_{\mathcal{J}^k}$ in $\{M^i\}_{\mathcal{J}^k}$ **do**
      Calculate value table entry $F_k(\{m^i\}_{\mathcal{J}^k})$
    end for
  end for
  iter$\leftarrow 0$
  while iter<Maxiter **do**
    for $i \in \mathcal{I}$ **do**
      Update q messages with (5)
    end for
    for $k \in \mathcal{K}$ **do**
      Update r messages with (5)
    end for
    if r and q messages do not change **then**
      Break
    end if
    iter++
  end while
  for $i \in \mathcal{I}$ **do**
    Calculate $m^{i\star}$ with (6)
  end for
  return $\{m^i\}_{\mathcal{I}}$
end procedure

IV. Local Path Planning

One key assumption we made is that the evolution of the robot agents is independent of each other, which decomposes the multi-robot system planning problem into multiple single-agent planning problems. As pointed out in previous sections, this is not true in practice due to the collision avoidance constraint. We shall present a local resolution scheme to coordinate adjacent robot agents and avoid a collision. The first step is to construct the adjacency graph for the multi-robot system.

Definition 1. Given an MOMDP, two states s, s' are adjacent if $\exists a, a' \in \mathcal{A}, e \in \mathcal{E}, s'' \in \mathcal{S}$ such that $T_s(s, a, e, s'') > 0$, $T_s(s', a', e, s'') > 0$, that is, two states are adjacent if there exist actions for the two states under which their possible successor states intersect.

Forward dynamic programming for local conflict resolution. Given the multi-robot system, two agents $i, j \in \mathcal{I}$ are adjacent if their current state s_t^i, s_t^j are adjacent. Let G be the adjacency graph with the nodes being the robot agents $\mathcal{I}$, and two nodes are connected if they are adjacent. G is divided into connected subgraphs, and for each subgraph, if it only contains one node, the robot agent simply follows the policy of the task it committed to; if it contains more than one node, the local resolution scheme is used to resolve the conflict.

Note that any subgraph only needs to consider the nodes within the subgraph since, by construction, the nodes will not collide with nodes outside the subgraph. Let $\bar{\mathcal{I}}$ be the robot agents within one connected subgraph, the local resolution problem at time t is the following:

$$\begin{aligned}\max_{\{a^i_{t:t+T}\}_{\bar{\mathcal{I}}}} \sum_{i\in\bar{\mathcal{I}}} \mathbb{E}\Bigg[&\sum_{\tau=t}^{t+T-1} -J(s^i_\tau, a^i_\tau, s^i_{\tau+1}, o^i_{\tau+1}) + \\ &V^i_J(t+T, s^i_{t+T}, b_t) + \delta R^i V^i_{\mathcal{G}}(t+T, s^i_{t+T}, b_t)\Bigg] \\ \text{s.t. } \forall i \in \bar{\mathcal{I}}, t \le \tau \le t+T-1, s^i_{\tau+1} &\sim \sum_e T_s(s^i_\tau, a^i_\tau, e) b_t(e) \\ \forall i, j \in \bar{\mathcal{I}}, i \ne j, \forall \tau \in \{t, ..., t+T\}, &\mathbb{P}(s^i_\tau = s^j_\tau) = 0,\end{aligned} \tag{7}$$

where T is the look-ahead horizon of the forward dynamic programming (DP), b is the belief vector at time t. Since we don't have access to future observations, b_t is assumed to be constant over the horizon. δR^i is the discrete derivative of the task reward that agent i is committed to, i.e., the reward difference agent i would make if it arrives at the goal set, which can be computed given the reward function $\mathcal{R}$ of the task. V^i_J and $V^i_{\mathcal{G}}$ are the two value functions associated with the task that agent i commits to. (7) is a sequential decision making problem with running reward $-J$ and terminal reward $\sum_{i\in\bar{\mathcal{I}}} R^i V^i_{\mathcal{G}} + V^i_J$, which is the expected reward at the terminal state.

Algorithm 2 Forward DP for local resolution

procedure LOC_RES($\{s^i_t, \delta R^i, V^i_J, V^i_{\mathcal{G}}\}_{\bar{\mathcal{I}}}$, b_t)
  Initialize the search tree $\mathcal{T}$ with $\{s^i_t\}_{\bar{\mathcal{I}}}$
  for $\tau = t, ...t+T-1$ **do**
    Expand $\mathcal{T}$ with all action combinations
    Trim collision nodes and dominated nodes from $\mathcal{T}$
  end for
  Add terminal reward to the leaf nodes
  return $\{a^i_t\}_{\bar{\mathcal{I}}}$ associated with the optimal leaf node
end procedure

Algorithms. The forward DP algorithm is summarized in Algorithm 2, where the search tree consists of nodes that store the collective state distribution of all agents in $\bar{\mathcal{I}}$ and the current cumulated reward and edges that store the joint actions. The trimming procedure removes nodes that contain possible collisions and nodes whose cumulated reward is smaller than another node sharing the same state distribution. Compared to backward DP, since the DP horizon, T is typically chosen to be small, forward DP saves computation time because not all states in the state space are explored. Algorithm 2 runs in a receding horizon fashion, i.e., only the first step of the action sequence is executed, and the algorithm replans in every time step.

To implement Algorithm 2 in a decentralized setting, one can simply select a node within the subgraph as the host and perform Algorithm 2 and share the result with other nodes in the subgraph.

Algorithm 3 summarizes all the modules of the DTPP, where $\mathcal{I}_j[1]$ is the only element in $\mathcal{I}_j$ when $|\mathcal{I}_j| = 1$. Besides the procedures introduced in Algorithm 1 and 2, other procedures involved are

- OBTAIN_PARTITION takes the current state of all agents and calculates the adjacency graph, then returns node sets in all connected subgraphs, denoted as $\{\mathcal{I}_j\}$
- POLICY evaluates the optimal policy of the task that the agent commits to
- EXECUTE executes the action and obtain the next state and observation
- UPDATE_BELIEF updates the belief with the new state and observation obtained from executing the action
- UPDATE_TASK updates the task set, removing expired tasks and adding new tasks should there be any.

Note that the belief gets updated sequentially by all the agents after executing their actions, and this piece of information is shared among the whole multi-robot system.

Algorithm 3 multi-robot system planning

Input: ($\mathcal{I}$, MOMDP) $\mathcal{K}_0$, b_0, $\{s^i_0\}_{\mathcal{I}}$, Maxiter
$t \leftarrow 0$, $\mathcal{K} \leftarrow \mathcal{K}_0$
while Not Terminate **do**
  $\{m^i\}_{\mathcal{I}}$=TASK_ALLOCATION$\{s^i_t\}_{\mathcal{I}}$, $\mathcal{K}$, b_t, Maxiter
  $\{\mathcal{I}_j\}$=OBTAIN_PARTITION($\{s^i_t\}_{\mathcal{I}}$, MOMDP)
  for $\mathcal{I}_j \in \{\mathcal{I}_j\}$ **do**
    if $|\mathcal{I}_j| == 1$ **then**
      $i \leftarrow \mathcal{I}_j[1]$ the robot agent index in
      if $m^i == \emptyset$ **then**
        $a^i_t \leftarrow$ IDLE
      else
        $a^i_t \leftarrow$ POLICY(t, m^i, s^i_t, b_t)
      end if
    else
      for $i \in \mathcal{I}_j$ **do**
        Obtain $\delta R^i, V^i_J, V^i_{\mathcal{G}}$ from task set $\mathcal{K}$
      end for
      $\{a^i_t\}_{\mathcal{I}_j} \leftarrow$ LOC_RES$(\{s^i_t, \delta R^i, V^i_J, V^i_{\mathcal{G}}\}_{\mathcal{I}_j}, b_t)$
    end if
  end for
  for $i \in \mathcal{I}$ **do**
    $o^i_{t+1}, s^i_{t+1} \leftarrow$ EXECUTE(i, a^i_t)
    $b_t \leftarrow$ UPDATE_BELIEF$(s^i_t, a^i_t, s^i_{t+1}, o^i_{t+1}, b_t)$
  end for
  $b_{t+1} \leftarrow b_t$, $t \leftarrow t+1$
  $\mathcal{K} \leftarrow$ UPDATE_TASK$(\mathcal{K}, t)$
end while

V. RESULTS

We demonstrate the proposed DTPP framework with a grid world example both in simulation and in experiments.

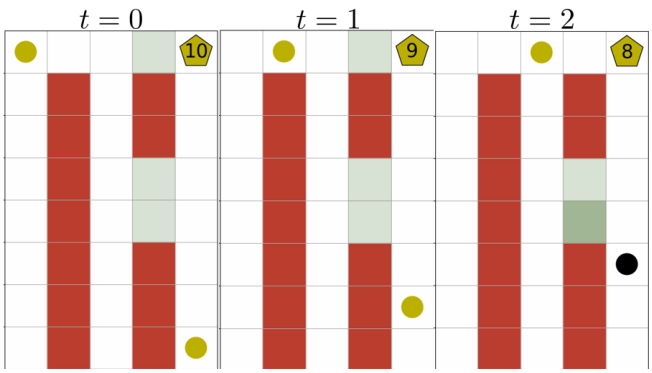

Fig. 5: Simulation with two agents and one task

Multi-robot system setup. The MOMDP is setup as a grid world. Each robot agent can choose from 5 actions: $\mathcal{A} = \{N, S, W, E, \text{IDLE}\}$, which make the robot move north, south, west, east, and stay still. The observable state is the agents' position within the grid world, and the partially observable state is the obstacle status of several uncertain cells, which may be clear or occupied by an obstacle. $\mathcal{E} = \mathbb{B}^{N_u}$, where N_u is the number of uncertain cells, and $|\mathcal{E}| = 2^{N_u}$. The state transition is assumed to be deterministic, i.e., for any state s and action a, there is only one possible successor state.

Remark 4. The state transition is deterministic when executing the actions, however, the agent is assumed to have 10% chance of staying still with $a \in \{N, S, W, E\}$ when solving for the quantitative optimal policy. This is to account for the possible influence of the local resolution and is particularly important for $V_{\mathcal{G}}$, the probability of reaching the goal set. Without the change of probability, the agent might think that it has 100% chance of reaching the goal yet fail to do so due to the local resolution preventing it from executing the action according to the policy. Under this change, the agent will be more certain that it can reach the goal as it gets closer to the goal.

The observation $\mathcal{O}$ space is the same as $\mathcal{E}$, and for each of the uncertain cells, we have

$$\forall i \in \{1, ..., N_u\}, \mathbb{P}(o_i = e_i) = \begin{cases} 1 & d \leq 1 \\ 0.8 & d = 2 \\ 0.5 & d > 2 \end{cases},$$

where o_i and e_i are the ith entry of o and e, the observed state and actual state of the uncertain cells, d is the Manhattan distance from s to the ith uncertain cell. T_e is set so that each uncertain cell has a 0.05 chance of changing its current state (from obstacle to free or the other way) when no agents are 2 steps or closer to it. The cost function J gives penalty 1 to all actions but IDLE.

Fig. 5 shows a sample simulation on a 7×5 grid world. The red blocks are the known obstacles and the green blocks are the uncertain cells with the transparency equal to the belief of it being an obstacle. The two yellow circles are the agents and the yellow pentagon denotes the goal region of the task, with the number showing the time left before t_f. The task's reward function $\mathcal{R}$ has a compact form of $(0, 50, 50)$, which means that one agent reaching shall earn a reward of 50, and two agents reaching will not increase the reward. The evolution of the value functions is shown in Table I.

TABLE I: Evolution of the value functions in the example shown in Fig. 5

	$t = 0$	$t = 1$	$t = 2$
$\bigvee_t^{t_f} s_t^1 \in \mathcal{G}$	0.727	0.800	0.999
$\bigvee_t^{t_f} s_t^2 \in \mathcal{G}$	0.947	0.962	0.974
$\bigvee_t^{t_f} s_t^1 \in \mathcal{G} \vee s_t^2 \in \mathcal{G}$	0.985	0.992	0.99997
V_J^1	-3.636	-2.89	-2.22
V_J^2	-7.616	-6.56	-5.49

At $t = 0$, both agents are assigned to the task because neither of them has 100% chance of reaching the goal in time. The algorithm decides to put two agents on the task to increase the probability that at least one of them reaches the goal, leading to a higher expected pure reward. At $t = 2$, the agent on the top figured out that the uncertain cell blocking its path to the goal is clear, significantly increasing its probability of reaching the goal from 80% to 99.9%, the algorithm then decided that the lower agent stays idle to save the cost.

To demonstrate the applicability of DTPP on real robotic systems, we ran high-fidelity simulations with multiple Segway robots and performed experiments with Turtlebots.

Segway simulation. In the Segway simulation, each Segway follows a nonlinear model with 7 states: $x = [X, Y, \theta, \dot{\theta}, v, \psi, \dot{\psi}]^\intercal$, where X, Y are the longitudinal and lateral coordinates, θ and $\dot{\theta}$ are the yaw angle and yaw rate, ψ and $\dot{\psi}$ are the pitch angle and pitch rate, and v is the forward velocity. The input is the wheel torques.

The high-level planning follows Algorithm 3 which sends high-level commands to the low-level controller, which runs a Model Predictive Controller (MPC) that generates torque command to the Segway. The high-level command consists of three parts: the desired waypoint $x^\star$, the state constraint $\mathcal{C}$, and the terminal state constraint $\mathcal{C}_f$. In the grid world case, $\mathcal{C}$ is simply the union of the current grid box and the next grid box to transition to, and $\mathcal{C}_f$ is the next grid box. The MPC then solves the following optimization to obtain the torque input:

$$\begin{aligned} \min_{u_{t:t+T-1}} & \sum_{\tau=t}^{t+T-1} x_\tau^\intercal Q x_\tau + u_\tau^\intercal R u_\tau + x_{t+T}^\intercal Q_f x_{t+T} \\ \text{s.t.} & \forall \tau = t, ...t+T-1, x_{\tau+1} = f(x_\tau, u_\tau), \\ & \forall \tau = t, ...t+T-1, x_\tau \in \mathcal{C}, u_\tau \in \mathcal{U}, x_{t+T-1} \in \mathcal{C}_f, \end{aligned} \tag{8}$$

where T is the horizon of the MPC, $\mathcal{U}$ is the set of available input, and f is the robot dynamics. The MPC uses sequential quadratic programming to accelerate the computation so that it can be implemented in real-time.

Fig. 6 shows one scenario of the simulation. At the beginning, Agent 2 was assigned to an existing task while the new task with the goal region in the middle appears, and agent 3 committed to the new task. Then in the second frame, the task that agent 2 committed to expires, and DTPP decided to let agent 2 commit to the new task, and agent 3 turned idle. Then in the third frame, the local resolution module made sure that agent 1 and agent 2 do not collide and let agent 1 enter the goal region first. In the last frame, agent 1 did not stop after

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/LRA.2021.3068103, IEEE Robotics and Automation Letters

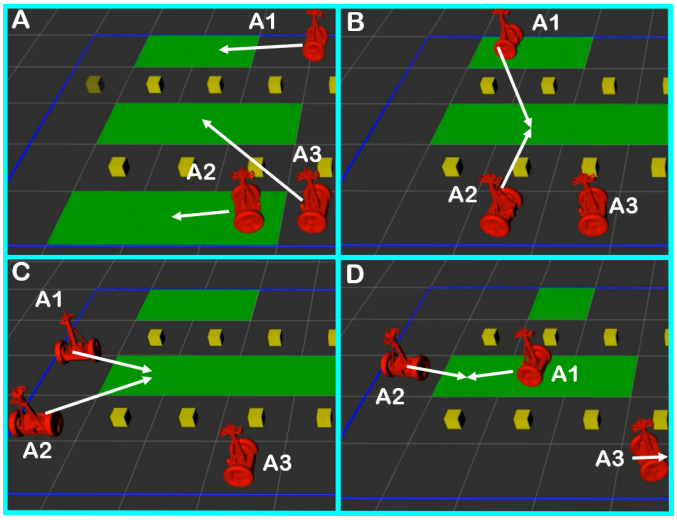

Fig. 6: Simulation with three segway robots

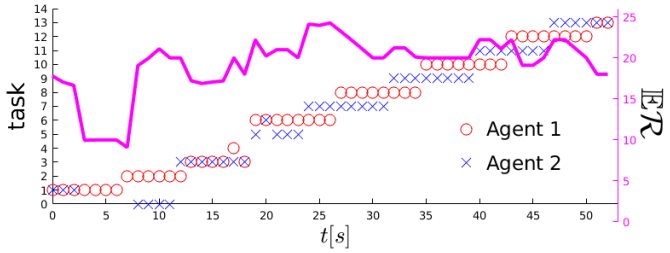

Fig. 7: Task assignment and expected pure reward in the Turtlebot experiment

entering the goal region, but kept moving to make room for agent 2, and agent 3 committed to a new task out of the frame.

Turtlebot experiment. We conducted experiments using two Turtlebots with differential driving capabilities. Since the model is simply a Dubin's car model with velocity and yaw rate inputs, we use a simple PID controller as the low-level controller for the Turtlebots. The experiment is performed on a 5×5 grid with 3 uncertain cells, of which one is a free box, and the other two are filled with obstacles. The two Turtlebots are presented with randomly generated multi-robot tasks with random goal regions and horizons ranging from 5 to 9 time steps. The high-level time step would increase by 1 if all agents reach the desired grid box planned by the high-level planner.

Fig. 1 shows a situation similar to that described by Fig. 5, where the robot agents figured out that the uncertain region is free of obstacle and the max-sum algorithm decides to let one robot agent go IDLE to save cost. A video containing the experiment and simulation result can be found in link.

Fig. 7 shows the task assignment result given by the max-sum algorithm and the expected pure reward of the multi-robot system consisting of two Turtlebots during the experiment. We kept the number of tasks at every time instance to be 2, and the task reward for most tasks (randomly generated except the first 4 tasks) is $(r_0 = 0, r_1 = 10, r_2 = 18)$. The two robots were usually assigned to different tasks to increase the expected pure reward, yet there are instances when they were assigned to the same task. The magenta curve shows the expected pure reward (up to the largest t_f of the existing tasks), which would change as new tasks emerged and the robot agents were assigned to new tasks.

Computation time The main benefit of DTPP is that it avoids the product MOMDP. Table II shows the computation time for the policy synthesis for the single agent MOMDP and the product MOMDP with multiple agents, the horizon is fixed to 8. Due to the double exponential complexity, the product MOMDP does not scale.

TABLE II: Computation time for MOMDP policy synthesis

Grid size	3×2	3×3	10×5	15×15
Single Agent MOMDP	78ms	80ms	3.61s	24.85s
Product MOMDP with 2 agents	8.97s	18.47s	NA	NA
Product MOMDP with 3 agents	198.67s	NA	NA	NA

We record the computation time for max-sum and forward DP with randomly generated initial positions of the agents on a 15×15 map, shown in Table III.

TABLE III: Computation time for the max-sum and the forward DP

Number of agents	3	4	5	6	7	8	9
Max-sum time (ms)	0.92	1.93	4.41	9.59	22.1	50.4	116.5
Forward DP time (ms)	2.09	2.22	5.80	9.60	17.2	38.0	55.2

The computation time for max-sum roughly grows quadratically with the number of agents, which is due to the fact that all agents are candidates for all tasks. Given a pre-screening process that picks out the nearby agents for each task, the complexity is $\mathcal{O}(1)$ with distributed computation and can be implemented online. The forward DP scheme can also be implemented with distributed computation, yet its complexity varies greatly with the scenario. For example, when a large group of agents are close to each other, the forward DP takes more time.

VI. Conclusion

We propose the decentralized task and path planning (DTPP) framework that is capable of task allocation and high-level path planning for a multi-robot system in a fully decentralized manner. Each robot agent is modeled as a Mixed Observed Markov Decision Process (MOMDP) assuming the independent evolution of the robot states. The task allocation is solved by representing the total pure reward as a factor graph and solved with the max-sum algorithm, which allows for collaboration between agents. Potential conflicts between robot agents are resolved by a local forward dynamic programming scheme, which guarantees no collision between agents.

References

[1] T. Arai, E. Pagello, L. E. Parker *et al.*, "Advances in multi-robot systems," *IEEE Transactions on robotics and automation*, vol. 18, no. 5, pp. 655–661, 2002.

[2] Z. Yan, N. Jouandeau, and A. A. Cherif, "A survey and analysis of multi-robot coordination," *International Journal of Advanced Robotic Systems*, vol. 10, no. 12, p. 399, 2013.

[3] P. R. Wurman, R. D'Andrea, and M. Mountz, "Coordinating hundreds of cooperative, autonomous vehicles in warehouses," *AI magazine*, vol. 29, no. 1, pp. 9–9, 2008.

[4] A. Khamis, A. Hussein, and A. Elmogy, "Multi-robot task allocation: A review of the state-of-the-art," in *Cooperative Robots and Sensor Networks 2015*. Springer, 2015, pp. 31–51.

[5] M. J. Matarić, G. S. Sukhatme, and E. H. Østergaard, "Multi-robot task allocation in uncertain environments," *Autonomous Robots*, vol. 14, no. 2-3, pp. 255–263, 2003.

[6] X. Bai, W. Yan, and M. Cao, "Clustering-based algorithms for multivehicle task assignment in a time-invariant drift field," *IEEE Robotics and Automation Letters*, vol. 2, no. 4, pp. 2166–2173, 2017.

[7] F. Tang and L. E. Parker, "A complete methodology for generating multi-robot task solutions using asymtre-d and market-based task allocation," in *Proceedings 2007 IEEE international conference on robotics and automation*. IEEE, 2007, pp. 3351–3358.

[8] D. P. Bertsekas, "Auction algorithms." *Encyclopedia of optimization*, vol. 1, pp. 73–77, 2009.

[9] X. Bai, W. Yan, M. Cao, and D. Xue, "Distributed multi-vehicle task assignment in a time-invariant drift field with obstacles," *IET Control Theory & Applications*, vol. 13, no. 17, pp. 2886–2893, 2019.

[10] M. Darrah, W. Niland, and B. Stolarik, "Multiple uav dynamic task allocation using mixed integer linear programming in a sead mission," in *Infotech@ Aerospace*, 2005, p. 7164.

[11] A. R. Mosteo and L. Montano, "Simulated annealing for multi-robot hierarchical task allocation with flexible constraints and objective functions," in *Workshop on Network Robot Systems: Toward Intelligent Robotic Systems Integrated with Environments". IROS*, 2006.

[12] C. Liu and A. Kroll, "A centralized multi-robot task allocation for industrial plant inspection by using a* and genetic algorithms," in *International Conference on Artificial Intelligence and Soft Computing*. Springer, 2012, pp. 466–474.

[13] Y. E. Sahin and N. Ozay, "From drinking philosophers to wandering robots," *arXiv preprint arXiv:2001.00440*, 2020.

[14] W. E. Walsh and M. P. Wellman, "A market protocol for decentralized task allocation," in *Proceedings International Conference on Multi Agent Systems (Cat. No. 98EX160)*. IEEE, 1998, pp. 325–332.

[15] H.-L. Choi, A. K. Whitten, and J. P. How, "Decentralized task allocation for heterogeneous teams with cooperation constraints," in *Proceedings of the 2010 American Control Conference*. IEEE, 2010, pp. 3057–3062.

[16] S. M. Aji and R. J. McEliece, "The generalized distributive law," *IEEE transactions on Information Theory*, vol. 46, no. 2, pp. 325–343, 2000.

[17] K. S. Macarthur, R. Stranders, S. D. Ramchurn, and N. R. Jennings, "A distributed anytime algorithm for dynamic task allocation in multi-agent systems," 2011.

[18] A. Farinelli, E. Zanotto, E. Pagello *et al.*, "Advanced approaches for multi-robot coordination in logistic scenarios," *Robotics and Autonomous Systems*, vol. 90, pp. 34–44, 2017.

[19] S.-H. Ji, J.-S. Choi, and B.-H. Lee, "A computational interactive approach to multi-agent motion planning," *International Journal of Control, Automation, and Systems*, vol. 5, no. 3, pp. 295–306, 2007.

[20] V. J. Lumelsky and K. Harinarayan, "Decentralized motion planning for multiple mobile robots: The cocktail party model," *Autonomous Robots*, vol. 4, no. 1, pp. 121–135, 1997.

[21] Y. Chen, A. Singletary, and A. D. Ames, "Guaranteed obstacle avoidance for multi-robot operations with limited actuation: a control barrier function approach," *IEEE Control Systems Letters*, vol. 5, no. 1, pp. 127–132, 2020.

[22] G. Wagner and H. Choset, "M*: A complete multirobot path planning algorithm with performance bounds," in *2011 IEEE/RSJ international conference on intelligent robots and systems*. IEEE, 2011, pp. 3260–3267.

[23] J. Yu and S. M. LaValle, "Structure and intractability of optimal multi-robot path planning on graphs," in *Twenty-Seventh AAAI Conference on Artificial Intelligence*, 2013.

[24] P. Schillinger, M. Bürger, and D. V. Dimarogonas, "Simultaneous task allocation and planning for temporal logic goals in heterogeneous multi-robot systems," *The international journal of robotics research*, vol. 37, no. 7, pp. 818–838, 2018.

[25] M. Kloetzer and C. Mahulea, "Multi-robot path planning for syntactically co-safe ltl specifications," in *2016 13th International Workshop on Discrete Event Systems (WODES)*. IEEE, 2016, pp. 452–458.

[26] F. Faruq, D. Parker, B. Laccrda, and N. Hawes, "Simultaneous task allocation and planning under uncertainty," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 3559–3564.

[27] P. Nilsson, S. Haesaert, R. Thakker, K. Otsu, C.-I. Vasile, A.-A. Agha-Mohammadi, R. M. Murray, and A. D. Ames, "Toward specification-guided active mars exploration for cooperative robot teams," 2018.

[28] C. Amato, G. Chowdhary, A. Geramifard, N. K. Üre, and M. J. Kochenderfer, "Decentralized control of partially observable markov decision processes," in *52nd IEEE Conference on Decision and Control*. IEEE, 2013, pp. 2398–2405.

[29] S. Paquet, L. Tobin, and B. Chaib-Draa, "An online pomdp algorithm for complex multiagent environments," in *Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, 2005, pp. 970–977.

[30] C. Amato and F. Oliehoek, "Scalable planning and learning for multiagent pomdps," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 29, no. 1, 2015.

[31] C. Belta, B. Yordanov, and E. A. Gol, *Formal methods for discrete-time dynamical systems*. Springer, 2017, vol. 89.

[32] J. D. Isom, S. P. Meyn, and R. D. Braatz, "Piecewise linear dynamic programming for constrained pomdps." in *AAAI*, vol. 1, 2008, pp. 291–296.

[33] J. Pineau, G. Gordon, S. Thrun *et al.*, "Point-based value iteration: An anytime algorithm for pomdps," in *IJCAI*, vol. 3, 2003, pp. 1025–1032.

[34] A. Rogers, A. Farinelli, R. Stranders, and N. R. Jennings, "Bounded approximate decentralised coordination via the max-sum algorithm," *Artificial Intelligence*, vol. 175, no. 2, pp. 730–759, 2011.

[35] S. Summers and J. Lygeros, "Verification of discrete time stochastic hybrid systems: A stochastic reach-avoid decision problem," *Automatica*, vol. 46, no. 12, pp. 1951–1961, 2010.

[36] V. Krishnamurthy, *Partially observed Markov decision processes*. Cambridge University Press, 2016.

APPENDIX

Here we briefly describe the approximate solution to the following cost optimal qualitative control problem

$$\begin{aligned} &\underset{\pi}{\text{maximum}} \quad \mathbb{E}^{\pi}\left[\sum_{t=0}^{t_f} -J(s_t, a_t, s_{t+1}, o_{t+1})\right] \\ &\text{subject to} \quad \pi \in \underset{\kappa}{\operatorname{argmax}}\, \mathbb{P}^{\kappa}[\bigvee_{t=0}^{t_f} s_t \in \mathcal{G}]. \end{aligned} \tag{9}$$

Notice that the above qualitative constraint on the probability of satisfying the specifications can be rewritten as

$$\begin{aligned} \mathbb{P}^{\kappa}[\bigvee_{t=0}^{t_f} s_t \in \mathcal{G}] &= \mathbb{P}^{\kappa}\big[\exists k \in \{0, \dots, t_f\} : s_k \in \mathcal{G}\big] \\ &= \mathbb{E}^{\kappa}\left[\sum_{t=0}^{t_f}\left(\prod_{\tau=0}^{t-1} \mathbb{1}_{\mathcal{S}\setminus\mathcal{G}}(s_\tau)\right)\mathbb{1}_{\mathcal{G}}(s_t)\right]. \end{aligned}$$

Furthermore, leveraging the result form [35, Lemma 4], we have that the value function associated with the above reachability problem is given by the following recursion

$$\begin{aligned} V^{\kappa}_{\mathcal{G}}(t, s, b) &= \mathbb{1}_{\mathcal{G}}(s) + \mathbb{1}_{\mathcal{Q}\setminus\mathcal{G}}(s)\mathbb{E}^{\kappa}[V^{\kappa}_{\mathcal{G}}(t+1, s', b')] \\ &= \begin{cases} 1 = \sum_{i=1}^{|\mathcal{E}|} b(i) & \text{If } s \in \mathcal{G} \\ \mathbb{E}^{\kappa}[V^{\kappa}_{\mathcal{G}}(t+1, s', b')] & \text{Else} \end{cases} \end{aligned} \tag{10}$$

with $V^{\kappa}_{\mathcal{G}}(t_f, s, \cdot) = 1$ if $s \in \mathcal{G}$ and $V^{\kappa}_{\mathcal{G}}(t_f, s, \cdot) = 0$ if $s \notin \mathcal{G}$. Notice that $V^{\kappa}_{\mathcal{G}}(t_f, s, \cdot)$ is a linear function for all $s \in \mathcal{S}$ and, consequently, $V^{\kappa}_{\mathcal{G}}(t, s, \cdot) : \mathcal{B}_{\mathcal{E}} \to \mathbb{R}$ is piecewise-affine by standard POMDP arguments [36, Theorem 7.4.1].

Finally, we have that as the value function (10) is piecewise-affine we can rewrite the quantitative problem (9) as a constrained POMDP, which we approximated using modified version of the algorithm presented [32]. We use another value function V_J to keep track of the expected reward-to-go, which is minimized among all policies that maximize $V_{\mathcal{G}}$. In particular, compared to the algorithm presented in [32], we propagate only a single belief point per constraint. This strategy, while being sub-optimal, allows us to reduced the computational burden associated with the algorithm presented in [32].