Decentralized Control of Multi-Robot Partially Observable Markov Decision Processes using Belief Space Macro-actions

Shayegan Omidshafiei[1], Ali-akbar Agha-mohammadi[2], Christopher Amato[3], Shih-Yuan Liu[1], Jonathan P. How[1], John Vian[4]

Abstract

This work focuses on solving general multi-robot planning problems in continuous spaces with partial observability given a high-level domain description. Decentralized Partially Observable Markov Decision Processes (Dec-POMDPs) are general models for multi-robot coordination problems. However, representing and solving Dec-POMDPs is often intractable for large problems. This work extends the Dec-POMDP model to the Decentralized Partially Observable Semi-Markov Decision Process (Dec-POSMDP) to take advantage of high-level representations that are natural for multi-robot problems and to facilitate scalable solutions to large discrete and continuous problems. The Dec-POSMDP formulation uses task macro-actions created from lower-level local actions that allow asynchronous decision-making by the robots, which is crucial in multi-robot domains. This transformation from Dec-POMDPs to Dec-POSMDPs with a finite set of automatically-generated macro-actions allows use of efficient discrete-space search algorithms to solve them. The paper presents algorithms for solving Dec-POSMDPs, which are more scalable than previous methods since they can incorporate closed-loop belief space macro-actions in planning. These macro-actions are automatically constructed to produce robust solutions. The proposed algorithms are then evaluated on a complex multi-robot package delivery problem under uncertainty, showing that our approach can naturally represent realistic problems and provide high-quality solutions for large-scale problems.

Keywords

Decentralized partially observable Markov decision processes, multi-agent planning, multi-agent systems, planning with macro-actions, hierarchical planning

1 Introduction

Given the low cost of hardware, it is becoming more cost-effective to deploy multiple robots to complete a single task or set of tasks. Unfortunately, control and coordination of multi-robot systems in real-world settings is difficult. For instance, many real-world multi-robot coordination problems operate in continuous spaces and robots often possess partial and noisy sensors. In addition, asynchronous decision-making is often needed due to stochastic action effects and the lack of perfect communication. Furthermore, in settings such as underwater robotics or space exploration, communication between robots may be expensive or inherently infeasible due to lack of infrastructure. Preferably, high-quality controllers for each robot would be automatically generated based on a high-level domain specification while considering uncertainty in the domain. Methods following this principle, such as those based on Markov decision processes (**?**) and partially observable Markov decision processes (**?**), have proven to be effective in single-robot domains. Similar methods have only begun to be considered in multi-robot problems. This paper presents scalable multi-robot decision-making in partially-observable settings using belief space macro-actions.

[1]Laboratory for Information and Decision Systems (LIDS), MIT, Cambridge, MA

[2]Qualcomm Research Center, San Diego, CA (work done while author was at LIDS, MIT)

[3]Department of Computer Science at the University of New Hampshire, Durham, NH

[4]Boeing Research & Technology, Seattle, WA

Corresponding author:

Shayegan Omidshafiei, 32 Vassar Street, 32-D631, Cambridge, MA, 02139, USA

Email: shayegan@mit.edu

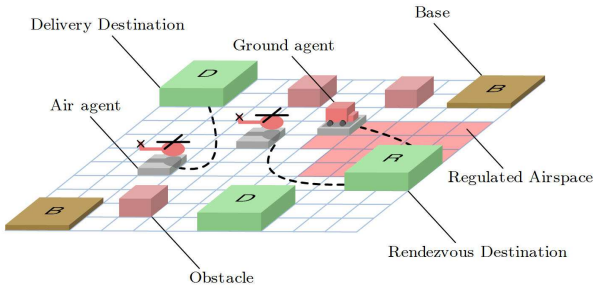


Figure 1. Package delivery domain with key elements labeled.

1.1 Related Work

Decision-making in a team setting can be performed in a centralized or a decentralized manner. Centralized planners have been utilized in a variety of multi-robot domains (**?????**), but rely on full sharing of all information through an assumed low-cost, low-latency, and high-reliance network. These frameworks, however, are difficult to make robust against communication infrastructure failures. Distributed planning algorithms (**???**), where local communication is used for consensus on robot policies, can be considered a middle-ground between centralized and decentralized decision-making and do not assume perfect communication with a central planning robot. However, if perfect communication between robots is assumed and consensus on joint robot state is desired, such algorithms can suffer from long convergence times which can prevent them from being implemented in real-time planning scenarios. In such scenarios, distributed, integer-programming based auction algorithms (**?**) are more robust to communication failures. However, they impose synchronized communication during their consensus (conflict resolution) phase, requiring artificial delays and extra communication to ensure synchronization.

Other promising works on multi-robot coordination focus on single-step problems (e.g., task allocation). A hierarchical decomposition approach to task allocation for homogeneous robots has been proposed in **?**, where partitioning is done based on local communication groups formed by neighboring robots. A highly-scalable hierarchical approach is also presented in **?**, which exploits the locality and sparsity of tasks to partition the joint problem into a set of independent sub-problems. Relevant work in hierarchical planning is presented in **?**, where regression planning is used to seek a set of high-level actions (with tight integration to motion models) which ensure the agent has high probability of reaching its goal state. Their work targets planning under partial observability, handling uncertainty by operating in belief space (as in our approach), but is focused on single-agent domains.

A general representation of the multi-robot coordination problem is the Decentralized Partially Observable Markov Decision Process (Dec-POMDP) (**?**). Dec-POMDPs have a broad set of applications including networking problems, multi-robot exploration, and surveillance (**?????**). Many authors have considered scalable Dec-POMDP methods, for instance **?** which exploit independence between agents to allow scalable planning with dozens of agents. Existing approaches rely on completely discrete domains and are not scalable for large (or continuous) state or observation spaces, whereas our approach can be applied to domains with underlying continuous states, actions, and observations (by treating them *within* the macro-actions). By using automatically-generated macro-actions, our high-level planning problem is discrete, yet its resulting policy is executed directly on a continuous underlying domain. This subtle point makes our framework distinct from purely discrete approaches. Furthermore, due to the use of durative macro-actions, our approach also supports asynchronous decision-making.

Macro-actions (MAs), or temporally-extended actions, have provided increased scalability in single-agent MDPs (**?**) and POMDPs (**???**). **?** introduced the *options* framework integrating discrete-time MAs in MDPs, resulting in a discrete-time semi-MDP (SMDP), a framework in which durations between action transitions are random variables (**??**).

Authors such as **?** considered parallelization of temporally-extended actions for single agents (managing parallel processes) and multiple agents (acting in parallel). This work also formalized the notion of decision epochs, time-steps at which decision-making is conducted in durative action processes, allowing treatment of asynchronous actions in settings with observable states. In our work, a specific subset of such decision epochs are used to allow asynchronous decision-making in partially-observable settings.

Multi-agent factored MDPs were considered in **??**, where a weighted linear combination of value functions was used to approximate the joint value function, and a message passing scheme was used for online coordination of agents. Concurrent MDPs (CoMDPs) were introduced in **?**, in which parallel (but instantaneous) probabilistic actions are executed. This work also introduced action pruning heuristics which speed up policy search by removing provably-suboptimal concurrent action combinations. Consideration for deterministic, integer-valued action durations is made in **?**, with a further extension for probabilistically-varying durations in **?**. In graph-based planning, Temporal Graphplan (TGP) (**?**) introduced durative actions and associated

mutual exclusion reasoning to recursively seek a plan satisfying goal propositions.

Usage of durative actions in partially-observable settings was considered in **?**, resulting in the Partially Observable semi-MDP (POSMDP), where primitive noisy observations are received within each durative action. Many authors have considered POSMDP-related models. For example, **?** considered planning with MAs in POMDPs, where belief simplex discretization on a uniform grid is used for MA-reward calculation using reinforcement learning. Further work on single-agent MA-based planning was conducted in **?**, where multi-modal belief updates using a modified Kalman Filter (similar to a Gaussian sum filter) were used in conjunction with a forward search algorithm for efficient policy search.

In this paper, we extend promising recent work on incorporating MAs (**???**) to solve continuous and large-scale problems which were infeasible for previous methods. A framework for formally representing asynchronous multi-robot coordination problems is developed. While the framework can use a set of hand-coded MAs, we also present an algorithm for automatically generating MAs that can then be sequenced to solve the problem. The result is a principled method for coordination in probabilistic multi-robot domains.

1.2 Challenges

Several factors make incorporation of MAs into multi-robot settings non-trivial. In particular, MAs will often require variable amounts of time for completion. In the decentralized setting, synchronized decision-making is not possible or problematic with these variable time MAs as some robots must remain idle while others finish their actions. The resulting solution would be difficult to implement on physical systems and would yield poor quality results. MAs that can be chosen asynchronously by the robots are proposed in this paper to overcome these limitations. Some of the challenges in extending variable time MAs to decentralized settings are:

- Incorporating principled asynchronous MA selection. It is not clear how to choose optimal MAs for one robot while other robots are still executing. Hence, a novel formal framework is needed to represent Dec-POMDPs with asynchronous decision-making and MAs that may last non-deterministic amounts of time.
- Designing these variable-time MAs also requires characterizing the stopping time and probability of terminating at every goal state of the MAs. Novel methods are needed that can provide this characterization.
- Given such a framework integrating multi-robot durative actions under partial-observability, algorithms for finding suitable team policies still need to be defined.

MA-based Dec-POMDPs alleviate scalability problems by no longer attempting to solve for a policy at the primitive action level, but instead considering temporally-extended actions, or MAs, at a high level. MAs also help address scalability issues, as the size of the action space is often considerably reduced to a smaller set of high-level decisions. Recent work (**?**) has considered use of MAs in Dec-POMDPs. This approach uses finite-state automata for policy representation, with the assumption that full specification of all MAs is provided by a domain expert. Our work allows automatic generation of these MAs, and also presents algorithms for solving the MA-based Dec-POMDPs. We also provide comparisons with **?**'s approach in Sec. 7.5.

1.3 Summary of Contributions

This paper extends the Dec-POMDP to the Decentralized Partially Observable *Semi*-Markov Decision Process (Dec-POSMDP), which formalizes the use of closed-loop MAs in multi-robot settings. The Dec-POSMDP represents the theoretical basis for asynchronous decision-making in Dec-POMDPs. In the proposed framework, we automatically design MAs using an existing graph-based planning technique, and present derivations necessary for integration of MAs into the Dec-POMDP framework. The resulting MAs are closed-loop with probabilistic completion durations and success rates, allowing them to be directly integrated into the Dec-POSMDP framework.

The reduction of Dec-POMDPs to Dec-POSMDPs with a finite, automatically-generated set of MAs allows use of discrete-space search algorithms to solve them. As a result, our framework can generate efficient decentralized plans which take advantage of estimated MA completion times to permit asynchronous decision-making under uncertainty. The proposed Dec-POSMDP framework enables solutions for large domains (in terms of state/action/observation space) with long horizons, which are otherwise computationally intractable to solve. We leverage the Dec-POSMDP framework and introduce a Monte Carlo-based discrete search algorithm and an entropy-based optimization algorithm for solving it. The performance of the methods is demonstrated for a constrained variant of the multi-robot package delivery under uncertainty problem (Fig. 1), and compared against the recent MA-based MDHS algorithm (**?**).

This work extends our earlier conference papers (**??**). Compared to both papers, this work provides a much more rigorous definition of the Dec-POSMDP, as well

as the transformation from the Dec-POMDP to the Dec-POSMDP. In addition, proofs for several properties of the Dec-POSMDP, such as joint value of the decentralized policy and the extended transition probabilities under MAs, are presented. This work also presents new hardware experiment results for a health-aware multi-robot package delivery domain.

1.4 Paper Organization

The paper is structured as follows. Section 2 introduces the decentralized decision-making under uncertainty problem as a Dec-POMDP. Section 3 motivates the need for MAs in decentralized decision-making, and introduces them in a principled manner. Automatic generation of MAs is described, and in Section 4 several definitions are presented to allow use of MAs in asynchronous decision-making. In Section 5, the Dec-POSMDP framework is formally defined. Section 6 introduces policy representation using finite state automata, and search algorithms for Dec-POSMDP policies. Section 7 presents both simulated and hardware experiments for multi-robot planning using Dec-POSMDPs. The constrained package delivery domain is introduced, where a heterogeneous team of robots are tasked with delivering packages from bases to delivery destinations. In the simulated experiments, MAs allowing joint robot actions (e.g., two robots picking up a single package) are also considered. In the hardware experiments, the robots consider a belief over their health state (e.g., fuel or damage), and an MA allowing robot repair is included. The experiments include comparisons of policies resulting from the algorithms introduced in Section 6. Finally, Section 8 concludes the paper and details future work.

2 Problem Statement

A Dec-POMDP (**?**) is a sequential decision-making problem where multiple agents (e.g., robots) operate under uncertainty based on different streams of observations. At each step, every robot chooses an action (in parallel) based purely on its local observations, resulting in an immediate reward and an observation for each individual robot, based on stochastic (Markovian) models over state, action, and observation spaces. The team shares a single reward function based on the actions of all agents, making the problem cooperative. However, the agents' local views mean that execution is decentralized.

We define a notation aimed at reducing ambiguities when discussing single robots and multi-robot teams. A generic parameter p related to the i-th robot is noted as $p^{(i)}$, whereas a joint parameter for a team of n robots is noted as $\bar{p} = \{p^{(1)}, p^{(2)}, \cdots, p^{(n)}\}$. Environment parameters or those referring to graphs are indicated without parentheses, for instance v^i may refer to a parameter of a graph node, and w^{ij} to a parameter of a graph edge starting at node i and ending at node j.

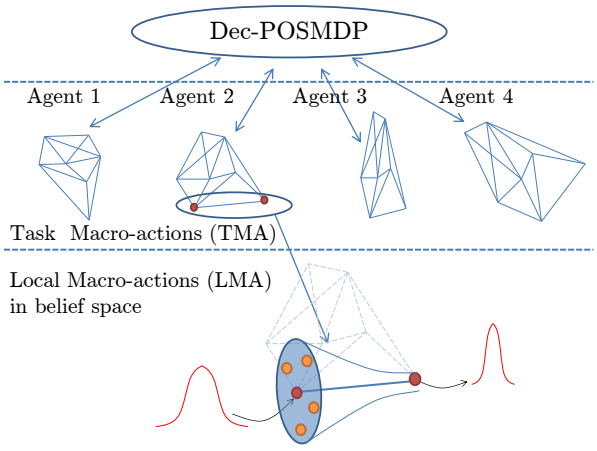


Figure 2. Hierarchy of the proposed planner. In the highest level, a decentralized planner assigns a TMA to each robot. Each TMA encompasses a specific task (e.g. picking up a package). Each TMA in turn is constructed as a set of local MAs (LMAs). Each LMA (the lower layer) is a feedback controller that acts as a funnel. LMAs funnel a large set of beliefs to a small set of beliefs (termination belief of the LMA).

Formally, the Dec-POMDP problem considered in this paper* is described by the following elements:

- $\mathbb{I} = \{1, 2, \cdots, n\}$ is a finite set of robots' indices.
- $\bar{\mathbb{S}}$ is a continuous set of joint states. The joint state space can be factored as $\bar{\mathbb{S}} = \bar{\mathbb{X}} \times \mathbb{X}^e$ where $\mathbb{X}^e$ denotes the environmental state and $\bar{\mathbb{X}} = \times_i \mathbb{X}^{(i)}$ is the joint state space of robots, with $\mathbb{X}^{(i)}$ being the state space of the i-th robot. $\bar{\mathbb{S}}$ is a super-state which contains both the robots' states as well as the environment-state, where $\mathbb{X}^{(i)}$ is a continuous space and $\mathbb{X}^e$ is assumed to be a finite set.
- $\bar{\mathbb{U}}$ is a continuous set of joint actions, which can be factored as $\mathbb{U} = \times_i \mathbb{U}^{(i)}$, where $\mathbb{U}^{(i)}$ is the set of actions for the i-th robot.
- State transition probability density function is denoted as $p(\bar{s}'|\bar{s}, \bar{u})$, that specifies the probability density of transitioning from state $\bar{s} \in \bar{\mathbb{S}}$ to $\bar{s}' \in \bar{\mathbb{S}}$ when joint action $\bar{u} \in \bar{\mathbb{U}}$ is taken by the robots.
- $\bar{R}$ is a joint reward function: $\bar{R} : \bar{\mathbb{S}} \times \bar{\mathbb{U}} \to \mathbb{R}$, the immediate reward for being in joint state $\bar{s} \in \bar{\mathbb{S}}$ and taking the joint action $\bar{u} \in \bar{\mathbb{U}}$.

*The standard (and more general) Dec-POMDP definition does not assume factored state spaces or environmental observations (**?**).

- $\bar{\Omega}$ is a continuous set of observations obtained by all robots. It is factored as $\bar{\Omega} = \bar{\mathbb{Z}} \times \bar{\mathbb{Z}}^e$, where $\bar{\mathbb{Z}} = \times_i \mathbb{Z}^{(i)}$ and $\bar{\mathbb{Z}}^e = \times_i \mathbb{Z}^{e(i)}$. The set $\mathbb{Z}^{(i)} \times \mathbb{Z}^{e(i)}$ is the set of observations obtained by the i-th robot. Environmental observation $o^{e(i)} \in \mathbb{Z}^{e(i)}$ is the observation that is a function of the environmental state $x^e \in \mathbb{X}^e$. We assume the set of environmental observations $\mathbb{Z}^{e(i)}$ is a finite set for any robot i.
- Observation probability density function $h(\bar{o}|\bar{s}', \bar{u})$ encodes the probability of seeing observations $\bar{o} \in \bar{\Omega}$ given joint action $\bar{u} \in \bar{\mathbb{U}}$ and the resulting joint state $\bar{s}' \in \bar{\mathbb{S}}$.

In a Dec-POMDP, robots make decisions based on their action-observation histories. The *full action-observation history* (obtained observations and taken actions) for the i-th robot is defined as follows,

$$\breve{H}_t^{(i)} = \{\breve{o}_0^{(i)}, u_0^{(i)}, \breve{o}_1^{(i)}, u_1^{(i)}, \cdots, \breve{o}_{t-1}^{(i)}, u_{t-1}^{(i)}, \breve{o}_t^{(i)}\} \quad (1)$$

where $\breve{o}^{(i)} \in \Omega^{(i)}$. For applications where robots operate in bursts of time where environment state x^e is not modified, we can represent the above more compactly as the *action-observation* history,

$$H_t^{(i)} = \{o_0^{(i)}, u_0^{(i)}, o_1^{(i)}, u_1^{(i)}, \cdots, o_{t-1}^{(i)}, u_{t-1}^{(i)}, o_t^{(i)}\} \quad (2)$$

where $o^{(i)} \in \mathbb{Z}^{(i)}$.

Note that the final observation $o_t^{(i)}$ after action $u_{t-1}^{(i)}$ is included in the action-observation history. Due to interactivity between multiple agents and the environment, environmental observations are of particular importance in the decentralized setting and are further detailed in Sec. 4.

The solution of a Dec-POMDP is a collection of decentralized policies $\bar{\eta} = \{\eta^{(1)}, \eta^{(2)}, \cdots, \eta^{(n)}\}$. In general, each robot does not have access to the observations of other robots, so each policy depend only on local information. Also, it is beneficial for agents to remember history, since the full state is not directly observed. As a result, $\eta^{(i)}$ maps the individual full action-observation history of the i-th robot to its next action: $u_t^i = \eta^{(i)}(\breve{H}_t^{(i)})$.

We can define the value associated with a given policy $\bar{\eta}$ starting from an initial joint belief (or state distribution) $\bar{b} = p(\bar{s})$,

$$\bar{V}^{\bar{\eta}}(\bar{b}) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \bar{R}(\bar{s}_t, \bar{u}_t)|\bar{\eta}, p(\bar{s}_0) = \bar{b}\right]. \quad (3)$$

The solution to a Dec-POMDP is then formally defined as the optimal policy, the policy with the highest value starting at the initial joint belief,

$$\bar{\eta}^* = \arg\max_{\bar{\eta}} \bar{V}^{\bar{\eta}}. \quad (4)$$

The Dec-POMDP problem stated in Eq. (4) is undecidable, as is the infinite-horizon POMDP problem (even in discrete settings) (**?**). In fact, no Dec-POMDP solution methods currently exist for problems with continuous state spaces.

In discrete settings, recent work has extended the Dec-POMDP model to incorporate MAs which can be executed in an asynchronous manner (**?**). In planning with MAs, decision-making occurs in a two layer manner (see Fig. 2). A higher-level policy will return a MA for each robot and the selected MA will return a primitive action to be executed. This approach is an extension of the *options framework* (**?**) to multi-robot domains while dealing with the lack of synchronization between robots. The options framework is a formal model of MAs (**?**) that has been very successful in aiding representation and solutions in single robot domains (**??**). However, the method in (**?**) requires full specification of MAs for all robots, including sensors and dynamics at a low level. An extension in (**?**) assumes the MA-level model is given (including all MAs), and presents a heuristics-based policy search algorithm. Our approach extends this by allowing automatic generation of these MAs, defining (and presenting equations to calculate) the probabilistic characteristics needed to integrate MAs into Dec-POMDPs, and formalizing the resulting hierarchical planning framework.

Our proposed framework, called the Decentralized Partially Observable Semi-Markov Decision Process (Dec-POSMDP), requires only a high-level model of the problem. The Dec-POSMDP provides a high-level discrete planning formalism which can be defined on top of continuous spaces. As such, we can approximate continuous multi-robot coordination problems with a tractable Dec-POSMDP formulation. In the next section, before defining our more general Dec-POSMDP model, we first discuss the form of MAs that allow for efficient planning within our framework.

3 Hierarchical Graph-based Macro-actions

This section introduces a mechanism to generate complex MAs based on a graph of lower-level (simpler) MAs, which is a key point in solving Dec-POSMDPs without explicitly computing success probabilities, times, and rewards of MAs in the decentralized planning level. We refer to the generated complex MAs as Task MAs (TMAs). When utilizing TMAs, we deal with the continuous spaces within them and leave the high-level decentralized framework with a finite set of TMAs and environmental observations, allowing discrete-space search algorithms to be used to solve them.

To clarify concepts, we first distinguish open-loop and closed-loop MAs before detailing TMAs. In general, MAs

refer to temporally-extended actions (**?**). An l-step long open-loop MA is a sequence of pre-defined actions such as $u_{0:l} = \{u_0, u_1, \cdots, u_l\}$. However, a closed-loop MA is policy $\pi(\cdot)$ which maps histories to actions, $u_t = \pi(H_t)$. As discussed in the next section, for seamless incorporation of MAs in Dec-POMDP planning, we need to design closed-loop MAs, which is a challenge in partially-observable settings.

3.1 Background

Generating a closed-loop MA that accomplishes a single-robot task such as "pick-up-a-package", "deliver-a-package", "open-a-door", and so on, itself requires solving a POMDP problem. This section details utilization of Feedback-based Information Roadmaps (FIRM) (**?**) as a substrate to generate such TMAs. FIRM makes several assumptions regarding the system model, namely that it is reasonably approximated by its linearization and that its process and measurement models have independent, zero-mean Gaussian noise. Our framework is designed for a class of systems that includes most physical robotic systems, where the Gaussian assumption is, for many tasks with standard observation models, reasonable and commonly made by other robotics frameworks (**???**). This makes FIRM a reasonably non-restrictive candidate for TMA generation within Dec-POSMDPs. Moreover, as discussed in Sec. 3.2, TMAs designed by frameworks other than FIRM can also be utilized in our setting, therefore allowing application to more general classes of problems.

We start by discussing the structure of feedback controllers in partially-observable domains. A macro-action $\pi^{(i)}$ for the i-th robot maps the histories $H^{\pi^{(i)}}$ of actions and observations that have occurred to the next action the robot should take, $u = \pi^{(i)}(H^{\pi^{(i)}})$. Note that the environmental observations are only obtained when the MA terminates. We compress this history into a belief $b^{(i)} = p(x^{(i)}|H^{\pi^{(i)}})$, with joint belief for the team denoted by $\bar{b} = \{b^{(1)}, b^{(2)}, \cdots, b^{(n)}\}$. It is well-known (**?**) that making decisions based on belief $b^{(i)}$ is equivalent to making decisions based on the history $H^{\pi^{(i)}}$ in a POMDP.

For any given robot, a feedback controller in the partially-observable environment comprises a Bayesian filter $b_{k+1} = \tau(b_k, u_k, z_{k+1})$ that evolves the belief and a separated controller $u_{k+1} = \mu(b_{k+1})$ that generates control signals based on the current belief. Therefore, a feedback controller $\mathcal{L}$ in belief space can be viewed as a function that maps the current belief b_k, control u_k, and observation z_{k+1} to the pair of next belief b_{k+1} and control u_{k+1}; i.e., $(b_{k+1}, u_{k+1}) = \mathcal{L}(b_k, u_k, z_{k+1}) = (\tau(b_k, u_k, z_{k+1}), \mu(\tau(b_k, u_k, z_{k+1})))$.

A local MA we consider herein is a feedback controller that is effective (has basin of attraction) locally in a region of the state/belief space. Many controllers that rely on linearization fall into this category as the linearization assumption is valid locally around the linearization point. The goal of LMA in the partially-observable setting is to drive the system's belief to a particular belief. In **?**, it has been shown that in Gaussian belief space, utilizing a combination of Kalman filter and linear controllers, the system's belief can be steered toward certain probability distributions. In other words, LMAs act like a funnel in belief space (see Fig. 2). Starting from a belief in the mouth of the funnel, the LMA drives the belief toward a target belief that is referred to as a *milestone* herein.

Since LMAs act locally on belief space, we can locally linearize the system and design corresponding simple LMAs. In this paper, we assume the belief space is Gaussian and thus mean vector $\hat{x}^+$ and covariance matrix P^+ characterize the belief, denoted as $b \equiv (\hat{x}^+, P^+)$. For a given mean value $\mathbf{v}$, we linearize the nonlinear process and measurement equations to get a stationary linear system with Gaussian noises. Associated with this linear system, we design a stationary Kalman filter and a linear separated controller, $\mu(b) = -L(\hat{x}^+ - \mathbf{v})$. Thus, the utilized LMA is parametrized by feedback gain matrix L and point $\mathbf{v}$; i.e., $\mu(b; \theta)$, where $\theta = (L, \mathbf{v})$. It can be shown that under the appropriate choice of L and mild observability conditions, this linear LMA acts as a funnel in belief space that drives the belief toward the milestone $\check{b} \equiv (\mathbf{v}, \check{P})$, where $\check{P}$ is the solution of the Riccati equation corresponding to the Kalman filter (**?**).

A chain of funnels is a sequence of funnels where the target belief of each funnel falls into the mouth (or pre-image) of the next funnel in the chain. A richer way of combining funnels is via *graphication* (to form a graph of funnels). An *information roadmap* is defined as a graph of funnels, where each node of this graph is a milestone and each edge is an LMA funnel (see Fig. 2) (**?**).

Alg. 1 recaps the construction of a TMA using FIRM, by constructing a graph of linear LMAs. To construct a graph of LMAs, we sample a set of parameters $\{\theta^j = (L^j, \mathbf{v}^j)\}$ (Alg. 1, Line 4) and generate corresponding LMAs $\{\mathcal{L}^j\}$ as described above. Associated with the j-th LMA, we compute the j-th milestone $\check{b}^j$. We define the j-th node of our LMA graph as an ϵ-neighborhood around the milestone; i.e., $B^j = \{b : \|b - \check{b}^j\| \leq \epsilon\}$ (Alg. 1, Line 5).

The set of all nodes is denoted $\mathbb{V} = \{B^j\}$. We connect B^j to its k-nearest neighbors via their corresponding LMAs. If neighboring nodes i and j are so far from each other that the j-th LMA $\mathcal{L}^j$ cannot take the belief from i to j (since the linearization used to construct $\mathcal{L}^j$ is not valid around B^i), we utilize an edge controller (as detailed in **?**). An edge controller is a finite-time controller whose role is to take the mean of the distribution close enough to the node B^j via tracking a trajectory that connects $\mathbf{v}^i$ to $\mathbf{v}^j$ in the state space. Once the distribution mean gets close enough

Algorithm 1: TMA Construction (Offline)

1 **Procedure** : `ConstructTMA`$(b, \mathbf{v}^{goal}, \mathcal{M})$
2 **input** : Initial belief b, mean of goal belief $\mathbf{v}^{goal}$, task POMDP $\mathcal{M}$;
3 **output** : TMA policy π^*, success probability of TMA $P(B^{goal}|b_0;\pi^*)$, value of taking TMA $V(b_0;\pi^*)$;
4 Sample a set of LMA parameters $\{\theta^j\}_{j=1}^{n-2}$ from the state space of $\mathcal{M}$, where θ^{n-2} includes $\mathbf{v}^{goal}$;
5 Corresponding to each θ^j, construct a milestone B^j in belief space;
6 Add to them the $(n-1)$-th node as the singleton initial milestone $B^n = \{b\}$, and the n-th node as the constraint milestone B^0;
7 Connect milestones using LMAs $\mathcal{L}^{ij}$;
8 Compute LMA rewards, completion time distribution, and transition probabilities by simulating LMAs offline;
9 Solve the LMA graph DP in Eq. (5) to construct TMA π^*;
10 Compute associated success probability $P(B^{goal}|b;\pi^*)$, completion time distribution for $T^g(B^{goal}|b;\pi^*)$, and value $V(b;\pi^*)$;
11 **return** TMA policy π^*, success probability $P(B^{goal}|b;\pi^*)$, completion time distribution for $T^g(B^{goal}|b;\pi^*)$, and value $V(b;\pi^*)$

to the target node, the system's control is handed over to the LMA associated with the target node, $\mathcal{L}^j$. We denote the concatenation of the edge controller and the funnel utilized to take the belief from B^i to B^j by $\mathcal{L}^{ij}$, which is defined as the (i,j)-th graph edge. The set of all edges is denoted by $\mathbb{L} = \{\mathcal{L}^{ij}\}$. We denote the set of available LMAs at B^i by $\mathbb{L}(i)$. One can view a TMA as a graph whose nodes are $\mathbb{V} = \{B^j\}$ and whose edges are LMAs $\mathbb{L} = \{\mathcal{L}^{ij}\}$ (Fig. 2).

To incorporate the lower-level state constraints (e.g., obstacles) and control constraints, we consider B^0 as a hypothetical node, hitting which represents violation of constraints. We add B^0 to the set of nodes $\mathbb{V}$. Therefore, taking any $\mathcal{L}^{ij}$ there is a chance that the system ends up in B^0. For more details on this procedure see **??**.

We can simulate the behavior of LMA $\mathcal{L}^{ij}$ at B^i offline (Alg. 1, Line 8) and compute the probability of landing in any given node B^r, which is denoted by $P(B^r|B^i,\mathcal{L}^{ij})$. Similarly, we can compute the reward of taking LMA $\mathcal{L}^{ij}$ at B^i offline, which is denoted by $R(B^i,\mathcal{L}^{ij})$ and defined as the sum of one-step rewards under this LMA. We denote $\mathcal{T}^{ij} = \mathcal{T}(B^i,\mathcal{L}^{ij}) \sim P(\mathcal{T}^{ij})$, where $P(\mathcal{T}^{ij})$ is the completion time distribution for LMA $\mathcal{L}^{ij}$. Calculating this distribution can be done in a variety of ways, and analytic calculation may be feasible for simple systems with Gaussian noise assumptions. Our experiments in Sec. 7 use a Monte Carlo simulation approach to estimate this completion distribution, which is subsequently used for discounting rewards in the joint policy evaluation.

3.2 Utilizing TMAs in the Decentralized Setting

In utilizing TMAs in the decentralized setting the following properties of the TMAs need to be available to the high-level decentralized planner: *(i)* TMA policy and value from any given initial belief, *(ii)* TMA completion time distribution from any given initial belief, and *(iii)* TMA success probability from any given initial belief. What makes computing these properties challenging is the requirement that they need to be calculated for *every* possible initial belief. Every belief is needed because when one robot's TMA terminates, the other robots might be in any belief while still continuing to execute their own TMA. This information about the progress of robots' TMAs is needed for nontrivial asynchronous TMA selection.

In the following, we discuss how the graph-based structures of the proposed TMAs allow us to compute a closed-form equation for the success probability, value, and time. As a result, when evaluating the high-level decentralized policy, these values can be efficiently retrieved for any given start and goal states. This is particularly important in decentralized multi-robot planning since the state/belief of the j-th robot is not known a priori when the TMA of the i-th robot terminates.

A TMA policy $\pi \in \mathbb{T}$ is defined as a policy that is found by performing dynamic programming on the graph of LMAs. Consider a graph of LMAs that is constructed to perform a simple task (such as open-the-door, pick-up-a-package, move-a-package). An important feature of this graph is that it is multi-query, meaning that it may be valid for any starting and goal belief. Depending on the goal belief of the task, we can solve the dynamic programming problem on the LMA graph that leads to a policy which achieves the goal while trying to maximize the accumulated reward and taking into account the probability of hitting failure set B^0. Formally, we need to solve the following DP,

$$V^{\pi^*}(B^i) = \max_{\mathcal{L}\in\mathbb{L}(i)} \Big\{ R(B^i,\mathcal{L}) + \sum_j P(B^j|B^i,\mathcal{L}) V^*(B^j) \Big\}, \forall i \tag{5}$$

$$\pi^*(B^i) = \operatorname*{argmax}_{\mathcal{L}\in\mathbb{L}(i)} \Big\{ R(B^i,\mathcal{L}) + \sum_j P(B^j|B^i,\mathcal{L}) V^*(B^j) \Big\}, \forall i \tag{6}$$

where $V^{\pi^*}(\cdot)$ is the optimal value defined over the graph nodes with $V(B^{goal})$ set to zero and $V(B^0)$ set to a suitable negative reward for violating constraints. The resulting optimal TMA is $\pi^*(\cdot)$. Primitive actions can be retrieved from each TMA via a two-stage computation: the TMA

first picks the best LMA at each milestone and the LMA generates the next primitive action based on the perceived observations until the belief reaches the next milestone; i.e., $u_{k+1} = \mathcal{L}(b_k, u_k, z_{k+1}) = \pi^*(B)(b_k, u_k, z_{k+1})$ where B is the last visited milestone and $\mathcal{L} = \pi^*(B)$ is the best LMA chosen by the TMA at milestone B.

For a given optimal TMA π^*, the associated optimal value $V^{\pi^*}(B^i)$ from any node B^i is computed via solving Eq. (5). Also, using Markov chain theory we can analytically compute the probability $P(B^{goal}|B^i;\pi^*)$ of reaching the goal node B^{goal} under the optimal TMA π^* starting from any node B^i in the offline phase (**?**).

Similarly, we can compute the time it takes for the TMA to go from B^i to B^{goal} under any TMA π as follows,

$$T^g(B^i;\pi) = \mathcal{T}(B^i;\pi) + \sum_j P(B^j|B^i;\pi)T^g(B^j;\pi),\ \forall i \tag{7}$$

where $T^g(B;\pi)$ denotes the time it takes for TMA π to take the system from B to the TMA's goal, whereas $\mathcal{T}(B^i;\pi)$ is the one-step time associated with taking the first LMA within the TMA when starting from B^i. Defining $\mathcal{T}^i = \mathcal{T}(B^i;\pi)$ and $\bar{\mathcal{T}} = (\mathcal{T}^1, \mathcal{T}^2, \cdots, \mathcal{T}^n)^T$ we can write Eq. (7) in its matrix form as,

$$\bar{T}^g = \bar{\mathcal{T}} + \bar{P}\bar{T}^g \Rightarrow \bar{T}^g = (I - \bar{P})^{-1}\bar{\mathcal{T}} \tag{8}$$

where $\bar{T}^g \sim P(\bar{T}^g)$ is a random vector with its i-th element equal to $T^g(B^i;\pi)$ and $\bar{P}$ is a matrix with its (i,j)-th entry equal to $P(B^j|B^i;\pi)$. Analytic representation of the completion time distribution for general domains is a formidable task. Although the policy evaluation equations in this work are defined over probabilistic completion times, offline Monte Carlo simulations are used to characterize this distribution for each TMA within our sample-based solution methods (Sec. 6). In that sense, the solution approaches rely on an approximation of the completion time distribution, which is then used to accordingly evaluate the policy. Discussion of this process is presented in Sec. 6.4.

Using the formulation presented in this section, a TMA can be used in a higher-level planning algorithm as a MA whose success probability, execution time distribution, and reward can be computed offline. Note that despite the usage of FIRM for TMA generation, our high-level planning framework is agnostic of FIRM's internals. It relies only on specification of the TMA's value, completion time distribution, and success probability. As long as these characteristics can be calculated, use of TMAs generated using frameworks other than FIRM is also possible and may allow treatment of more general classes of problems (e.g., where the Gaussian belief assumptions of FIRM may not apply).

4 Asynchronous Decision-Making with Macro-actions

This section introduces foundational concepts for the proposed Dec-POSMDP framework, allowing TMAs to be extended to the multi-robot setting in a manner similar to prior work on concurrent, durative actions (**????**). Specifically, we formalize the notion of decision epochs and introduce an environmental state that is partially-observable by each robot and can be affected by other robots.

4.1 Decision Epochs and Environmental States

Robots can choose TMAs at *decision epochs*. Since TMAs are durative, their selection is asynchronous and a decision epoch is considered to be a timestep at which any robot finishes its current TMA and chooses a subsequent TMA. The notion of decision epochs was formally introduced in **?**, where 3 forms of termination schemes were considered for concurrent action sets: 1) T_{any} (where the decision epoch occurs at the termination of any action, at which point all other actions terminate), 2) T_{all} (where the decision epoch occurs at termination of the final action in a set of actions, and all other threads wait in the meantime), and 3) $T_{continue}$ (used in our framework, where the decision epoch occurs at the termination of any action, while allowing remaining actions to continue running).

We can define the set of the first k decision epochs as an ordered set $t_{0:k} = (t_0, \cdots, t_k)$, which consists of all the timesteps at which any one (or multiple) of the robots completed a TMA. Refer to Fig. 3 for an illustrated example of epochs. If we define $\bar{\pi} = \{\pi^{(1)}, \cdots, \pi^{(n)}\}$ as the collection of TMAs currently assigned to the robots, $\bar{\pi}$ only changes at epochs (since, per definition, an epoch occurs when a robot completes its current TMA and chooses a new one). We can formally define the timestep at the k-th epoch by t_k, where $t_k = \min_i \min_t\{t > t_{k-1} : b_t^{(i)} \in B^{(i),goal}\}$, where $t_0 = 0$ and $b_t^{(i)}$ is the belief of robot i at timestep t.

The epoch number is denoted by k, with t_k being the time associated with it. Below we rely on the notation $(\cdot)_k = (\cdot)_{t_k}$ as a convention to unclutter the expressions. For instance, b_k and b_{t_k} both refer to the same belief (at epoch k or time t_k).

We denote the environment state (e-state) at the k-th epoch as $x_k^e \in \mathbb{X}^e$. The e-state encodes the information in the environment that can be manipulated and observed by different robots. It can be interpreted as a state which is shared amongst the robots, and can be manipulated (typically at a cost) to allow implicit communication between robots. We assume x_k^e is only locally (partially) observable. An example for x_k^e in the package delivery application (presented in Sec. 7) is "there is a package in

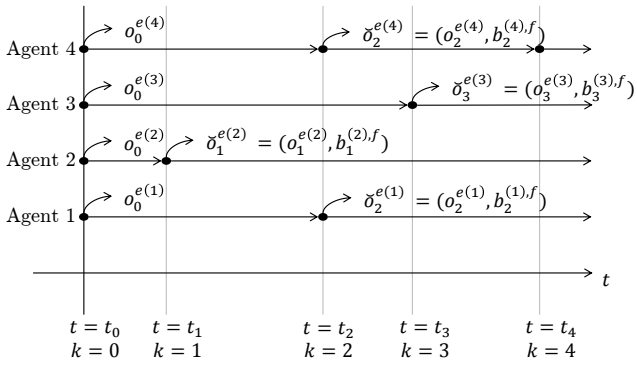


Figure 3. Decision epochs t_k are defined as timesteps when any robot finishes its current TMA. At a decision epoch, each robot receives a TMA-observation $\breve{o}_k^e$, consisting of its belief and latest environment observation.

this base". A robot can only get this example measurement if the robot is in the base (hence the entire process is partially observable). There is a limited set of TMAs available at each x^e, which is denoted by $\mathbb{T}(x^e)$.

Each time a robot i completes a TMA, it receives a partial observation of the environmental state x^e, denoted by $o^{e^{(i)}}$. Though the e-state is shared, noisy observations of it are made independently by each robot, thus allowing each robot to have a different observation model for it. The proposed framework assumes that low-level controllers operate in a closed-loop manner until their termination. Note that, if needed, low-level observations of the environment (for instance, landmark positions) are still possible within the TMA. A primary assumption here is that TMAs for remaining agents continue until termination, at which point they receive respective e-observations. Although this is somewhat limiting, note that this is a reasonable (and previously-made) assumption for reducing problem complexity in POSMDPs (**?**). Finally, treatment of e-observations as discrete is necessary for feasibly solving this planning problem. Existing Dec-POMDP solvers also assume discrete observations, and treatment of continuous observations remains future work for the field (**??**).

We denote the e-state observation likelihood function for robot i as $p(o_k^{e^{(i)}}|x_k^e, b_k^{(i)})$. Note that at every epoch k, only robots whose TMA has completed at that specific epoch, i.e., $b_k^{(i)} \in B^{(i),goal}$, will receive an environmental observation. The remaining robots (which continue to execute their TMAs) do not receive an environmental observation, i.e., $o_k^{e^{(i)}} = null$, for all i where $b_k^{(i)} \notin B^{(i),goal}$.

To incorporate environmental variables into the formulation, we need to consider all robots simultaneously since the e-state x^e is not local to a given robot and can be manipulated by other robots. Thus, we define $\bar{b}_k := (b_k^{(1)}, \cdots, b_k^{(n)})$ as the collection of beliefs of all robots at the k-th epoch, where $b_t^{(i)}$ is the belief of the i-th robot at time step t. Similarly, we define $\bar{B}_k^{goal} = (B_k^{goal,(1)}, \cdots, B_k^{goal,(n)})$ as the collection of goal regions in belief space at the k-th epoch for all robots.

4.2 Updated Model

Accordingly for $\pi \in \mathbb{T}(x^e)$, we extend single robot transition probabilities $P(B^{goal}|b;\pi)$ to take the e-state into account and define the extended transition probability as $P(\bar{B}_{k+1}^{goal}, x_{k+1}^{e'}|\bar{b}_k, x_k^e; \bar{\pi}_k)$, which denotes the probability of getting to the joint goal region $\bar{B}_{k+1}^{goal}$ and e-state $x_{k+1}^{e'}$ starting from joint belief $\bar{b}_k$ and e-state x_k^e under the joint TMA policy $\bar{\pi}_k$ at the k-th epoch.

The extended reward $\bar{R}(\bar{b}, x^e, \bar{u})$ encodes the reward obtained by the entire team, where $\bar{b} = \{b^{(1)}, \cdots, b^{(n)}\}$ is the joint belief and $\bar{u}$ is the joint action defined previously.

We assume the joint reward is a multi-linear function of a set of reward functions $\{R^{(1)}, \cdots, R^{(n)}\}$ and R^e, where $R^{(i)}$ only depends on the i-th robot's state and R^e depends on all the robots. In other words, we have

$$\bar{R}(\bar{x}, x^e, \bar{u}) = g(R^{(1)}(x^{(1)}, x^e, u^{(1)}), R^{(2)}(x^{(2)}, x^e, u^{(2)}), \cdots, R^{(n)}(x^{(n)}, x^e, u^{(n)}), R^e(\bar{x}, x^e, \bar{u})). \quad (9)$$

In multi-robot planning domains, computing R^e is often computationally less expensive than computing $\bar{R}$, which is the property we exploit in designing the higher-level decentralized algorithm. This is due to R^e essentially being an environment reward, generated for a certain e-state and configuration of robots. On the other hand, $\bar{R}$ is related to the robots' sequences of joint actions and their impact on the environment.

Similarly, the joint policy $\bar{\phi} = \{\phi^{(1)}, \cdots, \phi^{(n)}\}$ is the set of all decentralized policies, where $\phi^{(i)}$ is the decentralized policy associated with the i-th robot. In the next section, we discuss how these decentralized policies can be computed based on the Dec-POSMDP formulation.

Finally, joint value $\bar{V}^{\bar{\phi}}(\bar{b}, x_0^e) = \bar{V}(\bar{b}, x_0^e; \bar{\phi})$ encodes the value of executing the collection $\bar{\phi}$ of decentralized policies starting from e-state x_0^e and initial joint belief $\bar{b}$. The optimal joint policy $\bar{\phi}^*$ is defined as the policy which results in the maximum joint value $\bar{V}^*(\bar{b}, x_0^e; \bar{\phi}^*)$.

5 The Dec-POSMDP Framework

In this section, we formally introduce the Dec-POSMDP framework. We discuss how to transform a continuous Dec-POMDP to a Dec-POSMDP using a finite number of TMAs, allowing discrete domain algorithms to generate a decentralized solution for continuous problems.

In the decentralized setting, each robot has to make a decision based on its individual action-observation history.

In utilizing TMAs, this history includes the chosen TMAs $\{\pi_k^{(i)}\}$, the action-observation histories $\{H_k^{(i)}\}$ under chosen TMAs, and the environmental observations $\{o_k^{e(i)}\}$ received at the termination of TMAs for all epochs. In other words, the TMA history for the i-th robot at the beginning of the k-th epoch is defined as

$$\xi_k^{(i)} = (o_0^{e(i)}, \pi_0^{(i)}, H_1^{(i)}, o_1^{e(i)}, \pi_1^{(i)}, H_2^{(i)}, o_2^{e(i)}, \pi_2^{(i)}, \dots, o_{k-1}^{e(i)}, \pi_{k-1}^{(i)}, H_{k-1}^{(i)}, o_k^{e(i)}) \quad (10)$$

which includes the chosen TMAs $\pi_{1:k-1}^{(i)}$, the action-observation histories under chosen TMAs $H_{1:k-1}^{(i)}$, and the environmental observations $o_{1:k}^{e(i)}$ received at the termination of TMAs.

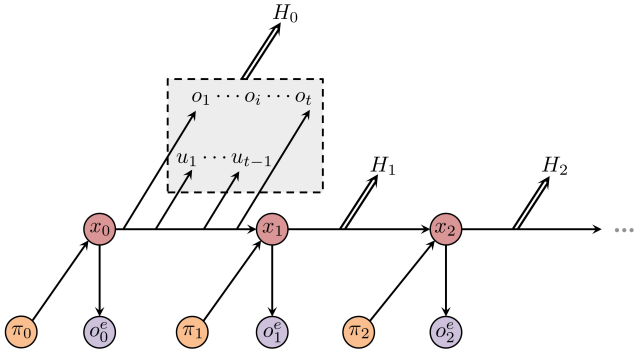


Figure 4. The i-th robot moves from state x_0 through a sequence of TMAs, $\pi_0, \pi_1, \cdots, \pi_k$. Under each TMA it receives primitive actions $u_0, u_1, \cdots, u_{t-1}$ and perceives primitive observations $o_0, o_1, \cdots, o_t$. At the end of each TMA (as well as beginning of the mission), it also obtains environmental observations $o_0^e, \cdots, o_k^e$.

The TMA history, as introduced above, is a full representation of a robot's decisions and observations, both high-level and low-level. However, in large-scale problems, it can be memory-intensive to retain this full history for each robot. Due to the special structure of the proposed TMAs, we can record the robot's final belief b^f at its TMA's termination. Final belief b_k^f compresses the entire history H_k under the TMA π_k. Explicitly, for each robot we define $b_k^f = p(s_k|H_k)$. We define the TMA-observation as $\breve{o}_k = (b_k^f, o_k^e)$ for $k > 0$ and $\breve{o}_0 = o_0^e$. As a result, we can compress the TMA history as

$$\xi_k^{(i)} = \{\breve{o}_0^{(i)}, \pi_0^{(i)}, \breve{o}_1^{(i)}, \pi_1^{(i)}, \cdots, \breve{o}_{k-1}^{(i)}, \pi_{k-1}^{(i)}, \breve{o}_k^{(i)}\} \quad (11)$$

without losing any information (**?**). From this point on, the term "TMA history" refers to this compressed definition of TMA history. The space of all possible TMA histories for the i-th robot is denoted by $\Xi^{(i)}$.

We denote the high-level decentralized policy for the i-th robot by $\phi^{(i)} : \Xi^{(i)} \to \mathbb{T}^{(i)}$, where $\xi^{(i)} \in \Xi^{(i)}$ is the TMA history for the i-th robot. Accordingly, we can define the joint policy $\bar{\phi} : \bar{\Xi} \to \bar{\mathbb{T}}$ as the collection of decentralized policies for all robots, i.e., $\bar{\phi} = \{\phi^{(1)}, \phi^{(2)}, \dots, \phi^{(n)}\}$. Similarly, $\bar{\pi} \in \bar{\Pi}$ is the joint TMA $\bar{\pi} = \{\pi^{(1)}, \pi^{(2)}, \dots, \pi^{(n)}\}$ and $\bar{\xi} \in \bar{\Xi}$ is the joint TMA history $\bar{\xi} = \{\xi^{(1)}, \cdots, \xi^{(n)}\}$.

The value of joint policy $\bar{\phi}$ is given

$$\bar{V}^{\bar{\phi}}(\bar{b}) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \bar{R}(\bar{s}_t, \bar{u}_t) | \bar{\phi}, \{\bar{\pi}\}, p(\bar{s}_0) = \bar{b}\right], \quad (12)$$

but it is unclear how to evaluate this equation without a full (discrete) low-level model of the domain. Even in that case, it would often be intractable to calculate the value directly. Therefore, we now formally define the Dec-POSMDP problem, which has the same goal as the Dec-POMDP problem (finding the optimal policy), and present methods for solving it.

The primary difference between the Dec-POMDP and Dec-POSMDP case is that we seek the optimal policy for choosing MAs in the semi-Markov setting, where MAs with probabilistic completion duration are used.

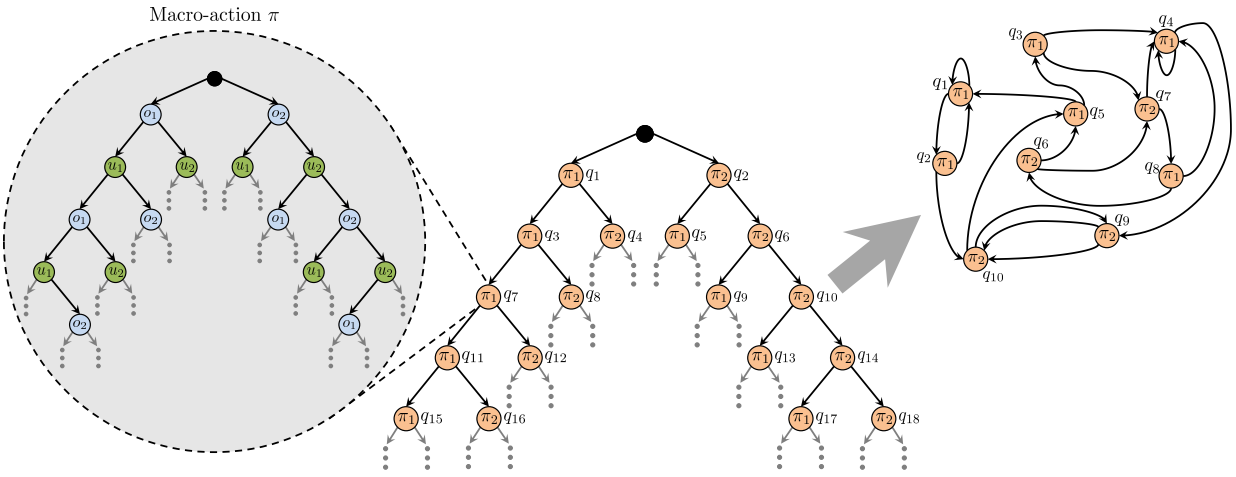
Definition 1. ***(Dec-POSMDP)*** *The Dec-POSMDP framework is described by the following elements:*

- $\mathbb{I} = \{1, 2, \cdots, n\}$ is a finite set of robots' indices.
- $\mathbb{B}^{(1)} \times \mathbb{B}^{(2)} \times \ldots \times \mathbb{B}^{(n)} \times \mathbb{X}^e$ is the underlying state space for the proposed Dec-POSMDP, where $\mathbb{B}^{(i)}$ is the set of belief milestones of i-th robot's TMAs (i.e., $\mathbb{T}^{(i)}$).
- $\bar{\mathbb{T}} = \mathbb{T}^{(1)} \times \mathbb{T}^{(2)} \ldots \times \mathbb{T}^{(n)}$ is the space of high-level actions (i.e., MAs) in Dec-POSMDP, where $\mathbb{T}^{(i)}$ is the set of TMAs for the i-th robot.
- $P(\bar{b}', x^{e'}, k|\bar{b}, x^e; \bar{\pi})$ denotes the transition probability under TMAs $\bar{\pi}$ from a given $\bar{b}, x^e$ to $\bar{b}', x^{e'}$ as described below.
- $\bar{R}^\tau(\bar{b}, x^e; \bar{\pi})$ denotes the generalized reward of taking a joint MA $\bar{\pi}$ at $\bar{b}, x^e$ as described further below.
- $P(\bar{\breve{o}}|\bar{b}, x^e)$ denotes the observation likelihood model, where $\bar{\breve{o}} = \{\breve{o}^{(1)}, \breve{o}^{(2)}, \cdots, \breve{o}^{(n)}\}$.
- $\bar{\mathbb{O}} = \{\bar{\breve{o}}\}$ is the set of all joint TMA-observations.

The solution to the Dec-POSMDP is the joint policy which optimizes the joint value,

$$\bar{\phi}^* = \arg\max_{\bar{\phi}} \bar{V}^{\bar{\phi}}(\bar{b}). \quad (13)$$

Fig. 5 provides a visual summary of the hierarchical Dec-POSMDP decision-making scheme. Solving the Dec-POSMDP results in the high-level joint policy, $\bar{\phi}$, dictating the TMA each robot should take based on its MA-history (the history of executed TMAs and received MA-observations). The selected TMA, $\pi^{(i)}$, for each robot i



(a) The selected TMA, $\pi^{(i)}$, for each robot i is itself a low-level policy mapping the robot's history of action-observations $H^{(i)}$ to its next low-level (primitive) action, $u^{(i)} \in \mathbb{U}$.

(b) High-level tree-based policy, $\phi^{(i)}$, with nodes $q^{(i)}$ representing the TMAs the robot should execute and edges representing the TMA-observation the robot receives after the TMA execution.

(c) Graph-based controllers (or finite state automata) are used to compress policy representation, reducing the size of the policy search space and allowing execution in infinite-horizon domains.

Figure 5. Overview of hierarchical decision-making using the Dec-POSMDP. The above illustrates policy, $\phi^{(i)}$, for the i-th robot. Each robot's policy is represented as a high-level graph-based controller (right), which can be considered a compression of a tree-based policy over MAs (middle). Each TMA is, in turn, a low-level policy (left).

is itself a low-level policy mapping the robot's history of action-observations $H^{(i)}$ to its next low-level (primitive) action, $u^{(i)} \in \mathbb{U}$ (Fig. 5(a)). The result is a sequential decision-making process eventually leading to low-level actions and observations (Fig. 4). This approach's efficacy comes from solving the problem in a hierarchical manner — first selecting the TMA for each robot, and then selecting the low-level action (which then executes in a continuous domain). Below, we describe the elements of the Dec-POSMDP in more detail.

Consider a joint TMA $\bar{\pi} = \{\pi^{(1)}, \pi^{(2)}, \cdots, \pi^{(n)}\}$ for the entire team. Incorporating terminal conditions for different robots in a multi-robot system, we can generalize the one-step reward of a joint primitive action $\bar{u}$ to a truncated reward of a joint MA $\bar{\pi}$ by evaluating the set of decentralized TMAs until at least one of them stops

$$\bar{R}^{\tau}(\bar{b}, x^e; \bar{\pi}) = \mathbb{E}\left[\sum_{t=0}^{\tau-1} \gamma^t \bar{R}(\bar{x}_t, x_t^e, \bar{u}_t) | p(\bar{x}_0) = \bar{b}, x_0^e = x^e; \bar{\pi}\right] \tag{14}$$

where,

$$\tau = \min_i \min_t \{t : b_t^{(i)} \in B^{(i),goal}\}. \tag{15}$$

Note that τ is itself a random variable, since the TMA completion times are also random variables (as discussed in Sec. 3.2). The expectation in Eq. (14) is, therefore, taken over the TMA completion durations as well. In practice (as discussed in Sec. 6), sampling-based approaches are used to estimate this expectation.

The extended definition of the reward in Eq. (14) can be considered the joint, discounted reward obtained by the team as a result of a chain of primitive actions $\bar{u}$ under TMA $\bar{\pi}$. Note that the upper bound of the summation is $\tau - 1$ to prevent double counting of rewards across multiple TMAs.

The probability of transitioning from configuration $(\bar{b}, x^e)$ to configuration $(\bar{b}', x^{e'})$ after k epochs under the joint TMA $\bar{\pi}$ can be solved using dynamic programming (see Appendix 9.2 for derivation),

$$\begin{aligned} &P(\bar{b}', x^{e'}, k | \bar{b}, x^e; \bar{\pi}) \\ &= P(x_k^{e'}, \bar{b}'_k | x_0^e, \bar{b}_0; \bar{\pi}) \qquad (16) \\ &= \sum_{x_{k-1}^e, \bar{b}_{k-1}} [P(x_k^{e'} | x_{k-1}^e; \bar{\pi}(\bar{b}_{k-1})) \\ &\qquad P(\bar{b}'_k | x_{k-1}^e, \bar{b}_{k-1}; \bar{\pi}(\bar{b}_{k-1})) \\ &\qquad P(x_{k-1}^e, \bar{b}_{k-1} | x_0^e, \bar{b}_0; \bar{\pi}(\bar{b}_0))]. \qquad (17) \end{aligned}$$

Note that recursive Eq. (17) implicitly uses the TMA completion time distribution of $\bar{T}^g$ in solving for epoch-based transition probabilities.

Joint value $\bar{V}^{\bar{\phi}}(\bar{b}, x^e)$ then encodes the value of executing the collection $\bar{\phi}$ of decentralized policies starting from e-state x_0^e and initial joint belief $\bar{b}_0$. The below equation describes the value transformation from low-level, primitive actions u to the scope of high-level TMAs π, which is vital for allowing us to efficiently perform

evaluation,

$$\bar{V}^{\bar{\phi}}(\bar{b}_0, x_0^e) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \bar{R}(\bar{x}_t, x_t^e, \bar{u}_t) | \bar{b}_0, x_0^e; \bar{\phi}\right] \quad (18)$$

$$= \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^{t_k} \bar{R}^\tau(\bar{b}_{t_k}, x_{t_k}^e; \bar{\pi}_{t_k}) | \bar{b}_0, x_0^e; \bar{\phi}\right] \quad (19)$$

where $t_k = \min_i \min_t \{t > t_{k-1} : b_t^{(i)} \in B^{(i),goal}\}$, $t_0 = 0$, and $\bar{\pi} = \bar{\phi}(\bar{b}, x^e)$. Details of this derivation can be found in Appendix 9.1. As mentioned earlier, note that the expectation above is taken over the epoch durations t_k as well, as they are random variables (due to the probabilistic nature of TMA completion times).

Denoting the optimal joint policy as $\bar{\phi}^*$, the dynamic programming formulation corresponding to the defined joint value function over MAs is

$$\bar{V}^*(\bar{b}, x^e) = \max_{\bar{\pi}} \{\bar{R}^\tau(\bar{b}, x^e; \bar{\pi}) + \sum_{t_k, \bar{b}', x^{e'}} \gamma^{t_k} P(\bar{b}', x^{e'}, t_k | \bar{b}, x^e; \bar{\pi}) \bar{V}^*(\bar{b}', x^{e'})\}, \quad \forall\, \bar{b}, x^e \quad (20)$$

$$\bar{\pi}^* = \bar{\phi}^*(\bar{b}, x^e) = \arg\max_{\bar{\pi}} \{\bar{R}^\tau(\bar{b}, x^e; \bar{\pi}) + \sum_{t_k, \bar{b}', x^{e'}} \gamma^{t_k} P(\bar{b}', x^{e'}, t_k | \bar{b}, x^e; \bar{\pi}) \bar{V}^*(\bar{b}', x^{e'})\}, \quad \forall\, \bar{b}, x^e. \quad (21)$$

The significant reduction from the continuous Dec-POMDP to the Dec-POSMDP over a finite number of TMAs is a key factor in solving large Dec-POMDP problems. In the following section we discuss how to compute a decentralized policy based on the Dec-POSMDP formulation.

6 Dec-POSMDP Policy Representation and Solution Methods

This section introduces Dec-POSMDP policy representation, defines policy iteration for the MA case, and presents algorithms for solving them. Existing Dec-POMDP algorithms assume synchronized decision-making over discrete time steps. Extending these methods to our framework is challenging since Dec-POSMDPs are inherently asynchronous. While related works in single-agent domains have targeted MA-integration, multi-agent domains introduce asynchronous decision-making that must be considered by the associated search algorithms.

6.1 Policy Representation using Finite State Automata

To solve the dynamic programming problem in Eq. (20), we rely on Finite State Automata (FSA) that induce the optimal joint policy $\bar{\phi}^*$.

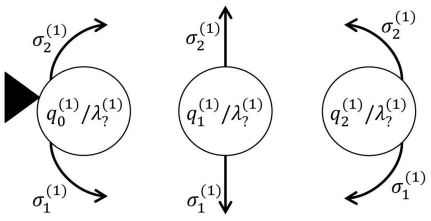
Definition 2. ***(FSA)*** *The Finite State Automata $\mathcal{F} = \{Q, \Sigma, \Lambda, \delta, \lambda, q_0\}$ is a discrete controller, represented by a graph, which is described by the following elements:*

- $Q = \{q\}$ are the FSA nodes.
- Σ is the input alphabet space, where inputs are $\sigma \in \Sigma$. In the scope of Dec-POSMDPs, these are environment observations o^e.
- Λ is the output alphabet space, with outputs $\lambda \in \Lambda$. In the scope of Dec-POSMDPs, these are TMAs $\pi \in \mathbb{T}$.
- $\delta : Q \times \Sigma \to Q$ is the deterministic transition function.
- $\lambda : Q \to \Lambda$ is the output function.
- $q_0 \in Q$ is the initial node.

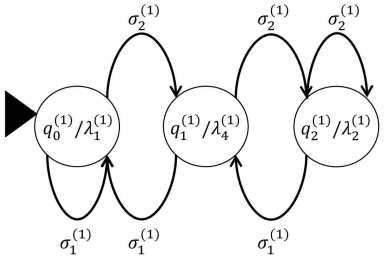
Fig. 6 illustrates an example FSA with three nodes. Given some sequence of inputs, $\{\sigma_0, \ldots, \sigma_t\}$ where $\sigma_i \in \Sigma$, the FSA will begin in q_0, output $\lambda(q_0) \in \Lambda$ based on the initial node, transition based on the first input and initial node $\delta(q_0, \sigma_0) \in Q$, and continue this process for the specified number of steps. In this example, the initial node is $q_0^{(1)}$, so the FSA outputs TMA $\lambda_1^{(1)}$. Then, when $\sigma_0 = \sigma_1^{(1)}$, the FSA transitions to the same initial node ($q_0^{(1)}$) and when $\sigma_0 = \sigma_2^{(1)}$, the FSA transitions to the second node, $q_1^{(1)}$. The TMA associated with the current node will then be executed and the process will continue.

The FSA can be considered a controller which takes observations as inputs and outputs TMAs for the robot to execute at each step. That is, we can define an FSA for each robot i, where the input alphabet Σ is given by the set of observations for that robot, $\mathbb{O}^{(i)} = \{o^{e(i)}\}$, and the output alphabet Λ is given by the available actions $\mathbb{T}^{(i)}$. Once the other parameters (which we now notate per robot as $Q^{(i)}, \delta^{(i)}, \lambda^{(i)}$, and $q_0^{(i)}$) have been specified for each robot, a set of controllers is defined as shown in Fig. 7, which choose actions based on the current node and transition based on the observations that have been seen.

A given FSA for robot i, $\mathcal{F}^{(i)}$, induces a policy $\phi^{(i)}$. In other words, we parametrize policy, $\phi^{(i)}$, for the robot by an FSA $\mathcal{F}^{(i)}$, i.e., $\pi^{(i)} = \phi^{(i)}(\xi^{(i)}; \mathcal{F}^{(i)})$, where $\xi^{(i)}$ is the TMA history defined in Sec. 5. The FSA generates mappings from histories to actions in the same way as discussed above for the general FSA case. That is, execution proceeds with $\lambda^{(i)}(q^{(i)}) = \pi^{(i)}$ followed by a transition in the controller $q' = \delta^{(i)}(q^{(i)}, o^{e(i)}) \in Q^{(i)}$, an action selected at the next node, and so on. Note that although the domain is stochastic, the policy is deterministic. The policy



(a) "Empty" controllers with no parameters generated (output functions λ and node-to-node transition function δ have not been solved for).



(b) FSA with solved parameters.

Figure 6. Example FSA for a single robot, with the initial node indicated by the black arrow.

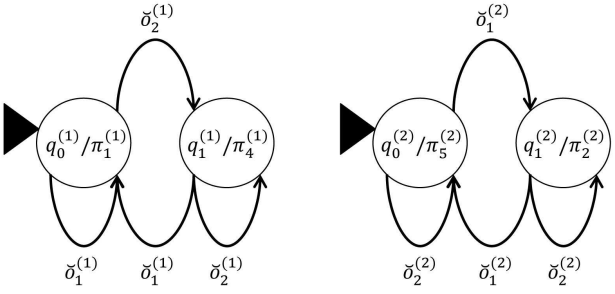


Figure 7. FSAs for two robots, each with two nodes, five possible TMAs and two observations. Note that superscripts indicate robot indices, while subscripts indicate indices of the nodes/TMAs/observations.

can proceed for an infinite number of steps due to loops in the controller.

A set of such FSAs (one for each robot) can be defined, parameterized by joint output functions $\bar{\pi} = \bar{\lambda}(\bar{q}) = \{\lambda^{(1)}(q^{(1)}), \ldots, \lambda^{(n)}(q^{(n)})\}$ and joint transition (input) functions $\bar{\delta}(\bar{q}, \bar{o}^e) = \{\delta^{(1)}(q^{(1)}, o^{e(1)}), \ldots, \delta^{(n)}(q^{(n)}, o^{e(n)})\}$. The FSAs can be evaluated at a given initial joint belief $\bar{b}$ and initial joint node $\bar{q}$ over epochs k as follows (see Appendix 9.3 for derivation),

$$\bar{V}^{\bar{\phi}}(\bar{b}, x^e, \bar{q}) = \bar{R}^{\tau}(\bar{b}, x^e, \bar{\lambda}(\bar{q}); \bar{\phi}) + \sum_{t_k, \bar{o}^e} \gamma^{t_k} P(\bar{o}^e|\bar{b}, x^e, \bar{\lambda}(\bar{q}); \bar{\phi}) P(t_k|\bar{b}, x^e, \bar{\lambda}(\bar{q}), \bar{o}^e; \bar{\phi}) \bar{V}^{\bar{\phi}}(\bar{b}'_{\bar{o}^e, \bar{\lambda}(\bar{q})}, x^{e'}_{\bar{o}^e, \bar{\lambda}(\bar{q})}, \bar{\delta}(\bar{q}, \bar{o}^e)). \quad (22)$$

In Eq. (22),

$$P(t_k|\bar{b}, x^e, \bar{\lambda}(\bar{q}), \bar{o}^e) \quad (23)$$

$$= P(t_k|\bar{b}'_{\bar{o}^e, \bar{\lambda}(\bar{q})}, x^{e'}_{\bar{o}^e, \bar{\lambda}(\bar{q})}, \bar{b}, x^e, \bar{\lambda}(\bar{q})) \quad (24)$$

$$= \frac{P(\bar{b}'_{\bar{o}^e, \bar{\lambda}(\bar{q})}, x^{e'}_{\bar{o}^e, \bar{\lambda}(\bar{q})}, t_k|\bar{b}, x^e, \bar{\lambda}(\bar{q}))}{\sum_{t_k} P(\bar{b}'_{\bar{o}^e, \bar{\lambda}(\bar{q})}, x^{e'}_{\bar{o}^e, \bar{\lambda}(\bar{q})}, t_k|\bar{b}, x^e, \bar{\lambda}(\bar{q}))} \quad (25)$$

where $P(\bar{b}'_{\bar{o}^e, \bar{\lambda}(\bar{q})}, x^{e'}_{\bar{o}^e, \bar{\lambda}(\bar{q})}, t_k|\bar{b}, x^e, \bar{\lambda}(\bar{q}))$ can be calculated using Eq. (17).

In Eq. (22), we marginalized out updated joint belief $\bar{b}'$ and updated e-state $x^{e'}$. We also utilize deterministic belief update $\bar{b}'_{\bar{o}^e, \lambda(\bar{q})}$ and e-state update $x^{e'}_{\bar{o}^e, \lambda(\bar{q})}$ which are calculated given previous joint belief $\bar{b}$, joint action $\bar{\lambda}(\bar{q})$, joint observation $\bar{o}^e$. Since these are deterministic updates, they have allowed replacement of belief update probability in Eq. (50) with the deterministic counterpart in Eq. (22).

Using Eq. (22), we can calculate the value of the policy starting at the initial node for each robot, $\bar{q}_0$, as $\bar{V}^{\bar{\pi}}(\bar{b}_0, x^e_0, \bar{q}_0)$ for any given initial belief and e-state ($\bar{b}_0$ and x^e_0).

6.2 Policy Iteration for Dec-POSMDPs

Now that evaluating a set of controllers is defined, we can discuss how to generate the parameters for these controllers. The sets of TMAs and observations are domain-specific and fixed, but the size of the controllers (i.e., the number of nodes), the action selection and transition functions, and the initial node must be specified in order to induce a policy. Note that if we do not limit the controller sizes or the action and transition functions, we will be able to represent any deterministic policy by using these Moore machines (since we could potentially create a separate node for each reachable belief up to an arbitrary depth). Unfortunately, as is shown in the POMDP case, an optimal policy may require an infinite number of controller nodes (**?**), but for the (primitive action) Dec-POMDP case, we can represent a policy within any ϵ of the optimal value with finite-sized controllers (**?**).

We now extend these ideas to the MA case. The resulting policy iteration algorithm incrementally grows the size of each robot's controller by adding nodes for each TMA and controller transitions for each combination of observations and then removes nodes that a) cannot satisfy the initial

conditions of the reachable belief states or b) have provably lower value. Like the primitive action case, we can prove that an ϵ-optimal policy (given the set of TMAs) can be found with this policy iteration method. While this approach has strong theoretical guarantees, we will discuss more scalable algorithms in the next section.

An overview of the method is given in Algorithm 2. The method takes some initial set of FSAs and a value threshold, ϵ, as input. Here, $R^{\tau}_{\max} = \max_i R^{\tau(i)}$, the maximum generalized reward obtained by any single agent in any epoch before any other agents complete their TMAs in the same epoch. The initial FSAs matter in practice, but not theoretically since these controllers will be improved sufficiently that *any* initial controllers would result in an ϵ-optimal solution. The algorithm iterates through these steps of improvement by repeatedly adding additional nodes to the controllers (through the ExhaustiveBackup detailed in Algorithm 4) and removing nodes that have provably suboptimal value (through Prune which is detailed in Algorithm 3). The controllers are evaluated as described above, with $\bar{V}^{\mathcal{F}} = \bar{V}^{\bar{\phi}}$ which also specifies initial nodes given by the FSAs $\mathcal{F}$. Because pruning any robot's FSA could result in additional pruning by some other robot, the procedure continues until no robots can prune their controllers.

Pruning is described in more detail in Algorithm 3 and Table 1. This procedure takes the set of FSAs and a specific robot, i, as input. It iterates through the nodes of i's FSA to see if they are dominated. Dominated nodes can be removed from the controller and the incoming links to the dominated nodes can instead be moved to the (possibly set of) nodes that dominate them. The test for dominance is detailed in Table 1, which gives the linear program for determining if a particular node, $q^{(i)}$, has lower value than some distribution over other nodes for all beliefs, external states and nodes of the other robots. Defining $\bar{\phi}^{(-i)} = \{\phi^{(1)}, \phi^{(2)}, \cdots, \phi^{(i-1)}, \phi^{(i+1)}, \cdots, \phi^{(n)}\}$, we can partition the joint policy as $\bar{\phi} = \{\phi^{(i)}, \bar{\phi}^{(-i)}\}$. Similarly, we can partition $\bar{q} = \{q^{(i)}, \bar{q}^{(-i)}\}$, where $\bar{q}^{(-i)} = \{q^{(1)}, q^{(2)}, \cdots, q^{(i-1)}, q^{(i+1)}, \cdots, q^{(n)}\}$. If ϵ is nonpositive, there is some distribution of nodes that always has value that is (at least $-\epsilon$) higher than $q^{(i)}$. Therefore, $q^{(i)}$ can be replaced by the distribution $x(\hat{q}^{(i)})$ any time there was a transition to $q^{(i)}$. If there are no transitions to $q^{(i)}$ (it is a new node that has been added to the controller after the last backup), it can simply be removed. The procedure outputs the resulting pruned FSA for robot i.

An exhaustive backup is summarized in Algorithm 4. Here, exhaustive backups are performed separately on each robot's FSA. An individual backup exhaustively generates all next-step FSA from the current-step FSA. That is, new nodes are added for each (macro-)action and each possible transition into the current FSA for each observation. If there are currently $|Q^{(i)}|$ nodes in the controller, there will be

Algorithm 2: Policy iteration

Procedure : `Policy Iteration`$(\mathcal{F}_0, \epsilon)$;
input : $\mathcal{F}_0$, initial FSA set;
output : $\mathcal{F}^*$, ϵ-optimal FSA set;
$k \leftarrow 0$;
while $\frac{\gamma^{t_{k+1}} |R^{\tau}_{\max}|}{1-\gamma} \geq \epsilon$ **do**
  $\mathcal{F}^{-}_{k+1} \leftarrow$ExhaustiveBackup$(\mathcal{F}_k)$;
  Compute $\bar{V}^{\mathcal{F}^{-}_{k+1}}(\bar{b}, x^e)$ for all $\bar{b}, x^e$;
  do
    $\mathcal{F}^{+}_{k+1} \leftarrow \mathcal{F}^{-}_{k+1}$;
    foreach *robot* i **do**
      $\mathcal{F}^{(i)+}_{k+1} \leftarrow$Prune$(\mathcal{F}^{+}_{k+1}, i)$;
      Update $V^{\mathcal{F}_{k+1}}$;
  while $|\mathcal{F}^{+}_{k+1}| \neq |\mathcal{F}^{-}_{k+1}|$;
  $k \leftarrow k+1$;
return $\mathcal{F}^{+}_{k+1}$;

Algorithm 3: Pruning

Procedure : `Prune`$(\mathcal{F}, i)$;
input : $\mathcal{F}$, FSA and robot index, i;
output : $\hat{\mathcal{F}}^{(i)}$, updated FSA for robot i;
foreach $q^{(i)} \in Q^{(i)}$ **do**
  dominated, $x(\hat{q}^{(i)}) \leftarrow$ CheckDominated($\mathcal{F}$,$q^{(i)}$);
  if *dominated* **then**
    $Q^{(i)} \leftarrow Q^{(i)} / q^{(i)}$;
    ReplaceTransitions($\delta^{\mathcal{F}^{(i)}}$, $q^{(i)}$, $x(\hat{q}^{(i)})$);
return $\hat{\mathcal{F}}^{(i)}$;

CheckDominated($\mathcal{F}$,$q^{(i)}$):
For a given $q^{(i)} \in Q^{\mathcal{F}^{(i)}}$ and variables ϵ and $x(\hat{q}^{(i)})$
Maximize ϵ, given:

$$V(q^{(i)}, \bar{q}^{(-i)}, \bar{b}, x^e) + \epsilon \leq \sum_{\hat{q}^{(i)}} x(\hat{q}^{(i)}) V(\hat{q}^{(i)}, \bar{q}^{(-i)}, \bar{b}, x^e) \; \forall \; \bar{q}^{(-i)}, \bar{b}, x^e \quad (26)$$

$$\sum_{\hat{q}^{(i)}} x(\hat{q}^{(-i)}) = 1 \text{ and } x(\hat{q}^{(i)}) \geq 0 \; \forall \hat{q}^{(i)} \quad (27)$$

Table 1. CheckDominated: The linear program that determines if robot i node $q^{(i)}$ is dominated by the distribution of other nodes for that robot, $\hat{q}^{(i)}$. Node $q^{(i)}$ is dominated if ϵ is nonpositive.

Algorithm 4: Exhaustive Backup

Procedure : `Exhaustive Backup`($\bar{\phi}$);
input : $\bar{\phi}$, policy;
output : $\bar{\phi}^{new}$, updated policy;
foreach *robot* i **do**
  $\phi^{(i)} \leftarrow$ IndividualBackUp($\phi^{(i)}$);
  $\bar{\phi}^{new} \leftarrow \{\phi^{(1)}, \phi^{(2)}, \cdots, \phi^{(n)}\}$;
return $\bar{\phi}^{new}$;

$|\mathbb{T}^{(i)}||Q^{(i)}|^{|\mathbb{O}^{(i)}|}$ nodes in the next-step FSA (since there are $|\mathbb{T}^{(i)}|$ different actions to choose and some node in the current controller must be transitioned to for each of the $|\mathbb{O}^{(i)}|$ observations). Exhaustive backups can be thought of as an exhaustive search through controller space, where all possible controllers will be generated if an infinite number of exhaustive backups are performed. Pruning is used to combat the exponential growth in the number of FSA nodes at each iteration while preserving the solution quality of the FSA set. Through discounting, we can show that the resulting controllers will be ϵ-optimal using a simple extension of the proof in **?**.

6.3 Solving Dec-POSMDPs using Masked Monte Carlo Search

The above transformation from continuous Dec-POMDP to Dec-POSMDP enables discrete domain algorithms to be used, such as the policy iteration approach in the previous section. However, for realistic problems with many MAs, policy iteration will not be able to converge to an ϵ-optimal solution due to large memory and time requirements. In this section, we propose an efficient method, referred to as Masked Monte Carlo Search (MMCS), to generate a decentralized multi-robot solution to solve the Dec-POSMDP.

As demonstrated in Sec. 7, MMCS allows solving problems with extremely large search spaces which would otherwise be intractable to solve. It uses an informed Monte Carlo policy sampling scheme to achieve this, exploiting results from previous policy evaluations to narrow the search space. MMCS utilizes greedy search of the policy space to solve extremely large multi-robot problems with relatively low computational overhead. However, the version of MMCS presented does not have probabilistic guarantees, although it can be extended to include them. For situations where computational budget is less restrictive, an improved algorithm called G-DICE is detailed in Sec. 6.4.

Because the infinite-horizon problem is undecidable (**?**), infinite-horizon methods typically focus on producing approximate solutions given a computational budget (**??**). A Monte Carlo approach can be implemented by repeatedly sampling from the policy space randomly and retaining the policy with the highest expected value as an approximate solution. The search can be stopped at any point and the latest iteration of the approximate solution can be utilized.

Algorithm 5: MMCS

1 **Procedure** : `MMCS`($\mathbb{T}, \breve{\mathbb{O}}, \mathbb{I}, K$)
2 **input** : TMA space $\mathbb{T}$, environmental observation space $\breve{\mathbb{O}}$, robots $\mathbb{I}$, number of best policies to check K
3 **output** : joint policy $\bar{\phi}_{MMCS}$
4 **foreach** *robot* $i \in \mathbb{I}$ **do**
5 $masked^{(i)} \leftarrow$ setToFalse();
6 $\phi^{(i)}_{MMCS} \leftarrow null$;
7 **for** $iter_{MMCS} = 1$ *to* $iter_{max,MMCS}$ **do**
8 **for** $iter_{MC} = 1$ *to* $iter_{max,MC}$ **do**
9 $\bar{\phi}_{new} \leftarrow \bar{\phi}_{MMCS}$;
10 **foreach** *robot* $i \in \mathbb{I}$ **do**
11 **foreach** $(\pi^{(i)}, \breve{o}^{(i)}) \in \mathbb{T} \times \breve{\mathbb{O}}$ **do**
12 **if** *not* $masked^{(i)}(\pi^{(i)}, \breve{o}^{(i)})$ **then**
13 $\phi^{(i)}_{new}(\pi^{(i)}, \breve{o}^{(i)}) \leftarrow$sample($\mathbb{T}(\pi^{(i)}, \breve{o}^{(i)})$);
14 $\bar{\phi}_{list}$.append($\bar{\phi}_{new}$);
15 $\bar{V}^{\bar{\phi}}_{list}(\bar{b}, x^e)$.append(evalPolicy($\bar{\phi}_{new}$));
16 $\bar{\phi}_{list} \leftarrow$getBestKPolicies($\bar{\phi}_{list}, \bar{V}^{\bar{\phi}}_{list}(\bar{b}, x^e), K$);
17 $(masked, \bar{\phi}_{MMCS}) \leftarrow$ createMask($\bar{\phi}_{list}, \bar{V}^{\bar{\phi}}_{list}(\bar{b}, x^e)$);
18 **return** $\bar{\phi}_{MMCS}$;

The MMCS algorithm detailed in Alg. 5. MMCS uses the same discrete controller represenation as policy iteration to represent the policy, $\phi^{(i)}$, of each robot. The nodes of the discrete controller are TMAs, $\pi \in \mathbb{T}$, and edges are observations, $\breve{o} \in \breve{\mathbb{O}}$. Each robot i transitions in the discrete controller by executing a TMA π_k, receiving an observation $\breve{o}_k$, and following the appropriate edge to the next TMA, π_{k+1}. Fig. 8 shows part of a single robot's policy.

MMCS initially generates a set of valid policies by randomly sampling the policy space (Alg. 5, Line 13), while adhering to the constraint that the termination milestone of a TMA node in the discrete controller must intersect the set of initiation milestones of its child TMA node. The joint value, $\bar{V}^{\bar{\phi}}(\bar{b}, x^e)$, of each policy given an initial joint belief $\bar{b}$ and e-state x^e is calculated using Monte Carlo simulations (Alg. 5, Line 15).

MMCS identifies the TMA transitions that occur most often in the set of K-best policies, in a process called 'masking' (Alg. 5, Line 17). It includes these transitions in future iterations of the policy search by explicitly checking for the existence of a mask for that transition, preventing the transition from being re-sampled (Alg. 5, Line 12). Note

that the mask is not permanent, as re-evaluation of the mask using the K-best policies occurs in each iteration (Alg. 5, Line 17).

The above process is repeated until a computational budget is reached, after which the best policy, $\bar{\phi}_{MMCS}$, is selected (Alg. 5, Line 17). Though 'masking' places focus on promising policies, it is done in conjunction with random sampling, allowing previously unexplored regions of the policy space to be sampled.

MMCS is designed for efficient search of the policy space for large problems. The algorithm can be extended to allow probabilistic guarantees on policy quality by allowing probabilistic resampling (Alg. 5, Line 13) based on the history of policy values, rather than using a binary mask. Error bounds on joint policy $\bar{\phi}$ could then be adopted from **?** as follows,

$$P[\bar{V}^{\bar{\phi}^*}(\bar{b}) - \bar{V}^{\bar{\phi}}(\bar{b}) \geq \epsilon] \leq \delta. \tag{28}$$

That is, MMCS can be extended straightforwardly to ensure that with probability of at least $1-\delta$ we can construct controllers with value within ϵ of optimal (for a fixed size).

6.4 Solving Dec-POSMDPs using Graph-based Direct Cross Entropy Method

Dec-POSMDPs can be posed as a combinatorial optimization problem, where decision variables are the joint policies of the robots. Combinatorial optimization algorithms have already seen recent applications to Dec-POMDPs due to their ability to find near-optimal solutions for otherwise intractable problems. The sampling-based Cross-Entropy (CE) method has been shown as a candidate for solving Dec-POMDPs, where specifically the Direct Cross-Entropy (DICE) algorithm is a promising approach for tree-based policy search (**?**). We extend the CE method to a graph-based variant called G-DICE that can be used to search the space of joint policies in order to solve the Dec-POSMDP.

The CE method uses a probabilistic sampling approach for solving the optimization problem,

$$x^* = \arg\max_x V(x) \tag{29}$$

which is analogous to the Dec-POSMDP problem of finding the optimal joint policy,

$$\bar{\phi}^* = \arg\max_{\bar{\phi}} \bar{V}^{\bar{\phi}}(\bar{b}). \tag{30}$$

To solve Eq. (29), the CE method maintains a sampling distribution $f(x;\theta)$, parameterized by θ, which it uses to sample solutions x in the search space. The CE method iteratively updates $f(x;\theta)$ such that samples drawn from it get progressively closer to the global optimum. This process is summarized as follows:

1. Generate a set of samples $\mathbf{X}$ from $f(x;\theta_k)$, where k is the iteration number and θ_0 is the initial parameter vector.
2. Use the best N_b samples $\mathbf{X_b}$ to calculate the Maximum Likelihood Estimate (MLE) of the parameter vector, $\theta_{k+1} = \text{argmax}_\theta f(\mathbf{X_b};\theta)$. Note that samples with value $V(x)$ less than $V_{w,k}$ (the worst-performing sample from the previous iteration's best N_b samples) are rejected to counter convergence towards local optima.
3. Apply smoothed update to θ_{k+1} (to further counter convergence towards local optima)

$$\theta_{k+1} \leftarrow \alpha\theta_{k+1} + (1-\alpha)\theta_k, \tag{31}$$

   where $\alpha \in (0,1]$ is called the learning rate.
4. Repeat until convergence, return best sample x_b.

This process minimizes the KL-divergence between indicator function $I_{(V(x)\geq V_{w,k})}$ and $f(x;\theta_k)$,

$$D_{KL}(I_{(V(x)\geq V_{w,k})} || f(x;\theta_k)), \tag{32}$$

and empirically causes $f(x;\theta_k)$ to converge towards a Dirac delta distribution centered at an optima (**?**).

We extend the CE method to the notion of FSAs introduced in Sec. 6.1, resulting in a new algorithm called Graph-based Direct Cross Entropy (G-DICE), outlined in Alg. 6.

Recall that the FSA for each robot, i, is characterized by two functions, the output function $\lambda^{(i)}(q^{(i)})$ and the transition function $\delta^{(i)}(q^{(i)}, \breve{o}^{(i)})$. G-DICE, therefore, maintains two probability distributions at each FSA node $q^{(i)}$:

1. TMA output function $f(\pi^{(i)}|q^{(i)};\theta_k^{(i)(\pi|q)})$, parameterized by $\theta_k^{(i)(\pi|q)}$.
2. FSA transition function $f(q^{(i)'}|q^{(i)},\breve{o}^{(i)};\theta_k^{(i)(q'|q,\breve{o})})$, parameterized by $\theta_k^{(i)(q'|q,\breve{o})}$.

For a simple implementation, a categorical underlying sampling distribution can be used (although more complex distributions may benefit certain domains). For each robot, both parameters are updated using the CE method. The result is a policy dictating deterministic selection of TMAs $\pi^{(i)}$ and node transitions based on each robot's action observation history.

To compute the graph-based policy controller using G-DICE, we first create an initial graph with N_n nodes and $|\breve{\mathbb{O}}|$ outgoing edges from each node (Alg. 6, Line 4), where N_n is chosen empirically based on system memory limitations and desired accuracy of the joint policy. The best-joint-value-so-far, $\bar{V}_b$, and worst-joint-value-so-far, $\bar{V}_w$, are initialized to $-\infty$ (Alg. 6, Lines 5-6). The parameter vectors are initialized (either using a priori

Algorithm 6: G-DICE

1 **Procedure** : G-DICE($\bar{\mathbb{T}}, \bar{\bar{\mathbb{O}}}, \mathbb{I}, N_n, N_k, N_s, N_b, \alpha$)
2 **input** : TMA space $\bar{\mathbb{T}}$, environmental observation space $\bar{\bar{\mathbb{O}}}$, robots $\mathbb{I}$, number of nodes in graph N_n, number of iterations N_k, number of samples per iteration N_s, number of best samples retained N_b, learning rate α
3 **output** : best joint policy $\bar{\phi}_b$
4 For each robot, initialize policy graph with N_n nodes and $|\bar{\bar{\mathbb{O}}}|$ edges per node;
5 $\bar{V}_b \leftarrow -\infty$;
6 $\bar{V}_{w,0} \leftarrow -\infty$;
7 **for** $i = 1$ *to* $|\mathbb{I}|$ **do**
8 Initialize $\theta_0^{(i)(\pi|q)} \ \forall q$;
9 Initialize $\theta_0^{(i)(q'|q,\breve{o})} \ \forall q, \breve{o}$;
10 **for** $k = 0$ *to* $N_k - 1$ **do**
11 $\bar{\phi}_{list} \leftarrow \varnothing$;
12 **for** $s = 1$ *to* N_s **do**
13 **for** $i = 1$ *to* $|\mathbb{I}|$ **do**
14 $\phi^{(i)} \leftarrow$ sample π from $f(\pi|q; \theta_k^{(i)(\pi|q)}) \ \forall q$ and sample q' from $f(q'|q, \breve{o}; \theta_k^{(i)(q'|q,\breve{o})})$ $\forall q, \breve{o}$;
15 $\bar{V}^{\bar{\phi}} \leftarrow$Evaluate($\bar{\phi}$);
16 **if** $\bar{V}^{\bar{\phi}} \geq \bar{V}_{w,k}$ **then**
17 $\bar{\phi}_{list} \leftarrow \bar{\phi}_{list} \cup \bar{\phi}$;
18 $\bar{V}_{list} \leftarrow \bar{V}_{list} \cup \bar{V}^{\bar{\phi}}$;
19 **if** $\bar{V}^{\bar{\phi}} > \bar{V}_b$ **then**
20 $\bar{V}_b \leftarrow \bar{V}^{\bar{\phi}}$;
21 $\bar{\phi}_b \leftarrow \bar{\phi}$;
22 $\bar{\phi}_{b,list}, \bar{V}_{b,list} \leftarrow$ best N_b policies in $\bar{\phi}_{list}$;
23 $\bar{V}_{w,k+1} \leftarrow \min(\bar{V}_{b,list})$;
24 **for** $i = 1$ *to* $|\mathbb{I}|$ **do**
25 $\theta_{k+1}^{(i)(\pi|q)} \leftarrow$ maximum likelihood estimate of $\theta^{(i)(\pi|q)}$ using $\bar{\phi}_{b,list} \ \forall q$;
26 $\theta_{k+1}^{(i)(\pi|q)} \leftarrow \alpha\theta_{k+1}^{(i)(\pi|q)} + (1-\alpha)\theta_k^{(i)(\pi|q)}$;
27 $\theta_{k+1}^{(i)(q'|q,\breve{o})} \leftarrow$ maximum likelihood estimate of $\theta^{(i)(q'|q,\breve{o})}$ using $\bar{\phi}_{b,list} \ \forall q, \breve{o}$;
28 $\theta_{k+1}^{(i)(q'|q,\breve{o})} \leftarrow \alpha\theta_{k+1}^{(i)(q'|q,\breve{o})} + (1-\alpha)\theta_k^{(i)(q'|q,\breve{o})}$;
29 **return** $\bar{\phi}_b$;

knowledge from previous runs, or more typically such that their associated distributions are uniform).

In each iteration, batches of N_s policies are sampled using the current parameter vectors $\theta_k^{(i)(\pi|q)}$ and $\theta_k^{(i)(q'|q,\breve{o})}$ (Alg. 6, Line 14). The policies are then evaluated using Eq. (22). Policies with value lower than the previous iteration's worst value, $\bar{V}_{w,k}$, are rejected (Alg. 6, Lines 17-18) and the best-policy-so-far, $\bar{\phi}_b$, is noted (Alg. 6, Line 21). This process can be performed very efficiently through parallelization, since each sampled joint policy $\bar{\phi}$ is evaluated independently.

Following this, the best N_b policies from the list of sampled policies, $\bar{\phi}_{list}$, are used to calculated MLEs of the parameter vectors (Alg. 6, Line 25 and Line 27). The parameter vectors are then smoothed (Alg. 6, Line 26 and Line 28) and the process is repeated until N_k iterations are completed. Since the parameters and the best-policy-so-far, $\bar{\phi}_b$, are updated in each iteration, G-DICE can be stopped at any point to produce an approximation of the optimal joint policy as $\bar{\phi}_b \approx \bar{\phi}^*$.

A key advantage of the Dec-POSMDP framework is probabilistic treatment of TMA completion times. As discussed in Sec. 4.1 and Sec. 5, this is a direct consequence of the extension of primitive actions to TMAs. Each TMA targets a goal belief milestone B^{goal}, but reaches this set in a non-deterministic length of time. Thus, in the evaluation step of G-DICE (Alg. 6, Line 15), the expected joint value over TMA completion times is calculated. Due to the difficulty of generating closed-form expressions for TMA completion probabilities, this evaluation is done using a Monte Carlo approach where parallel executions of the TMAs are used to generate a set of duration samples. This same Monte Carlo-based policy evaluation is also used for the MMCS algorithm.

G-DICE is an anytime algorithm offering several advantages to the tree-based approach utilized previously for Dec-POMDPs (**?**). First, the use of graphs allows solving of infinite-horizon Dec-POSMDPs, since loops in the policy graph allow it to be executed indefinitely. Additionally, the number of nodes in the graph, N_n, can be fixed a priori in order to impose memory bounds on the algorithm, since out-of-memory issues are prevalent in problem domains with long time horizons (**?**).

7 Experiments

In this section, we consider a package delivery under uncertainty scenario involving a team of heterogeneous robots, an application which has recently received particular attention (**??**). The overall objective in this problem is to retrieve and deliver packages from base locations to delivery locations using a group of robots. This domain is complicated due to several imposed constraints, including decentralization (no online communication between robots), failure probabilities for TMAs, partial and noisy observability of packages, and probabilistic distributions of TMA completion times. Hence, the problem cannot be solved by standard task allocation and planning algorithms, and instead use of the Dec-POSMDP framework is necessary for finding a near-optimal solution.

We first introduce the application domain and formalize it in the Dec-POSMDP notation, including detailing of all domain TMAs. We then analyze characteristics of an example TMA, finally using both the MMCS and G-DICE algorithms to generate closed-loop policy controllers for the robots. We do not report results from policy iteration due to its limited scalability, however we do include comparisons with **?**'s approach.

7.1 Application Domain: Constrained Package Delivery

This domain was chosen due to its extreme complexity. For instance, the experiments conducted use a policy controller with 13 nodes, resulting in a policy space with cardinality $5.622e+17$ in the package delivery domain, making it computationally intractable to solve using exhaustive search. Additional challenges stem from the presence of different sources of uncertainty (wind, actuator, sensor), obstacles and constraints in the environment, and different types of tasks (pick-up, drop-off, etc.). Also, the presence of joint tasks such as joint pickup of large packages introduces a significant multi-robot coordination component. Note that the e-state is not fully observable by the agents in this domain, and the Dec-POSMDP allows treatment of both partial and noisy e-state observations. Specifically, in the package delivery setting, only partial observations of package characteristics are received by agents. In the health-aware hardware experiments (Sec. 7.6), observations of agent health states (which are treated at the e-state level) are both noisy and partial.

This problem is formulated as a Dec-POMDP in continuous space, and hence current Dec-POMDP methods are not applicable. Even if we could modify the domain to allow their use, discretizing the state/action/observation space in this problem would lead to poor solution quality or computational intractability.

Fig. 1 illustrates the package delivery domain. Robots are classified into two categories: air robots (quadrotors) and ground robots (trucks). Air robots handle pickup of packages from bases, and can also deliver packages to two destinations, $Dest_1$ and $Dest_2$. An additional delivery location $Dest_r$ exists in a regulated airspace, where air robots cannot fly. Packages destined for $Dest_r$ must be handed off to a ground robot at a rendezvous location. The ground robot is solely responsible for deliveries in this regulated region. Rewards are given to the team only when a package is dropped off at its correct delivery destination.

Packages are available for pickup at two bases in the domain. Each base contains a maximum of one package, and a stochastic generative model is used for allocating packages to bases. Each package has a designated delivery location, $\delta \in \Delta = \{d_1, d_2, d_r\}$. Additionally, each base has a size descriptor for its package, $\psi \in \Psi = \{\varnothing, 1, 2\}$, where $\psi = \varnothing$ indicates no package at the base, $\psi = 1$ indicates a small package, and $\psi = 2$ indicates a large package. Small packages can be picked up by a single air robot, whereas large packages require cooperative pickup by two air robots. The descriptors of package destinations and sizes will implicitly impact the policy of the decentralized planner.

To allow coordination of cooperative TMAs, an e-state stating the availability of nearby robots, $\phi \in \Phi = \{0, 1\}$, is observable at any base or at the rendezvous location, where $\phi = 1$ signifies that another robot is at the same milestone (or location) as the current robot and $\phi = 0$ signifies that all other robots are outside the current robot's milestone.

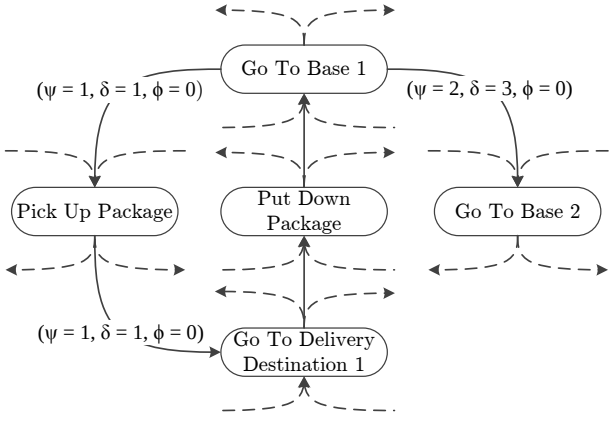


Figure 8. Partial view of a single robot's policy obtained using MMCS for the package delivery domain. In this discrete policy controller, nodes represent TMAs and edges represent e-states. Greyed out edges represent connections to additional nodes which have been excluded to simplify the figure.

A robot at any base location can, therefore, observe the local e-state $x^e = (\psi, \delta, \phi) \in \Psi \times \Delta \times \Phi$, which contains details about the availability and the size of the package at its current base (if it exists), the delivery destination of the package, and availability of nearby robots (for performing cooperative tasks). A robot at the rendezvous location can observe $x^e = \phi \in \Phi$ for guidance of rendezvous TMAs.

The considered system includes one ground robot and two air robots, with two base locations $Base_1$ and $Base_2$. Air robots are initially located at $Base_1$ and the ground robot is at $Dest_r$. If $b^{(i)} \in Base_j$, we say the i-th robot is at the j-th base.

7.2 TMA Analysis

Each TMA is defined with an associated initiation and termination set. The available TMAs in this domain are:

- Go to base $Base_j$ for $j \in \{1, 2\}$
  - *Robots involved:* 1 air robot.

  - *Initiation set:* Air robot i is available, where $i \in \{i_{a,1}, i_{a,2}\}$.
  - *Termination set:* robot i available and its belief is $b^{(i)} \in B^{h_j}$.
- Go to delivery destination $Dest_j$ for $j \in \{1, 2, r\}$
  - *Robots involved:* 1 (any type)
  - *Initiation set:* robot i available, can be at any location.
  - *Termination set:* robot i available and its belief is $b^{(i)} \in B^{d_j}$.
- Joint go to delivery destination $Dest_j$ for $j \in \{1, 2\}$
  - *Robots involved:* 2 air robots.
  - *Initiation set:* Air robots $i_{a,1}$ and $i_{a,2}$ are available.
  - *Termination set:* robots $i_{a,1}$ and $i_{a,2}$ are available and their beliefs are $b^{(i_{a,1})} \in B^{d_j}$, $b^{(i_{a,2})} \in B^{d_j}$, where $j \in \{1, 2\}$.
- Pick up package
  - *Robots involved:* 1 air robot
  - *Initiation set:* Air robot i is available, where $i \in \{i_{a,1}, i_{a,2}\}$. $x^e = \{\psi = 1, \delta \in \Delta, \phi \in \Phi\}$
  - *Termination set:* Ground robot i is carrying package and is unavailable.
- Joint pick up package
  - *Robots involved:* 2 air robots.
  - *Initiation set:* Air robots $i_{a,1}$ and $i_{a,2}$ are available. $x^e = \{\psi = 2, \delta \in \Delta, \phi = exact\}$
  - *Termination set:* robots $i_{a,1}$ and $i_{a,2}$ are carrying package and are unavailable.
- Put down package
  - *Robots involved:* 1 ground robot or 1 air robot.
  - *Initiation set:* robot $i \in \mathbb{I}$ is carrying package, $b^{(i)} \in B^{d_j}$, where $j \in \{1, 2\}$
  - *Termination set:* robot i is available.
- Joint put down package
  - *Robots involved:* 2 air robots.
  - *Initiation set:* Air robots $i_{a,1}$ and $i_{a,2}$ are available and jointly carrying a package. $b^{(i_1)} \in B^{d_j}$ and $b^{(i_2)} \in B^{d_j}$, where $j \in \{1, 2\}$.
  - *Termination set:* robots $i_{a,1}$ and $i_{a,2}$ are available and their beliefs are $b^{(i_{a,1})} \in B^{d_j}$, $b^{(i_{a,2})} \in B^{d_j}$, where $j \in \{1, 2\}$.
- Go to rendezvous location
  - *Robots involved:* 1 ground robot or 1 air robot
  - *Initiation set:* robot $i \in \mathbb{I}$ is available.
  - *Termination set:* robot i is available and its belief is $b^{(i)} \in B^{d_r}$.
- Place package on truck
  - *Robots involved:* 1 air robot, 1 ground robot.
  - *Initiation set:* Air robot $i_a \in \{i_{a,1}, i_{a,2}\}$ and ground robot i_g are available and located at rendezvous location, where $b^{(i_a)} \in B^{d_r}$ and $b^{(i_a)} \in B^{d_r}$.
  - *Termination set:* robots i_a and i_g are available and their beliefs are $b^{(i_a)} \in B^{d_j}$, $b^{(i_g)} \in B^{d_j}$, where $j \in \{1, 2\}$.
- Wait at current location
  - *Robots involved:* 1 ground robot or 1 air robot.
  - *Initiation set:* robot $i \in \mathbb{I}$ is available.
  - *Termination set:* robot i is available.

The robot's belief and the e-state affect the TMA's behavior. For instance, the behavior of the "Go to $Base_j$" TMA will be affected by the presence of obstacles for different robots.

To generate a closed-loop policy corresponding to each TMA, we follow the procedure explained in Sec. 3. For example, for the "Go to $Base_j$" TMA, the state of system consists of the air robot's pose (position and orientation). Thus, the belief space for this TMA is the space of all possible probability distributions over the system's pose. To follow the procedure in Sec. 3, we incrementally sample beliefs in the belief space, design corresponding LMAs, generate a graph of LMAs, and use dynamic programming on this graph to generate the TMA policy.

Fig. 11(b) shows performance of an example TMA ("Go to $Dest_1$"). The results show that the TMA is optimized to achieve the goal in the manner that produces the highest value, but its performance is robust to noise and tends to minimize the constraint violation probability. A key observation is that the value function is available over the entire space. The same is true for the success probability and completion time distribution of the TMA. This information can then be directly used by a policy search algorithm to perform evaluation of policies that use this TMA. Given a goal location, dynamic programming provides a feedback driving the robot to that goal, as well as cost-to-go starting from any other milestone node in the belief space. Fig. 9 illustrates an example graph of LMAs over a space. 30 milestones are represented on the graph, with a total of 178 edges connecting them. Obstacles in the space are also included and limit the connectivity of the nodes.

Fig. 10 illustrates the optimal path as well as the cost-to-go given a fixed goal node (circled in red), for all start nodes. The color of the edges correspond to the normalized cost-to-go for each starting node, with red edges being most costly and blue edges being least costly. For nodes near the goal location, edges are calculated to be less costly, as expected.

More interesting observations can be made by considering costs for related or nearby edges. For instance, in Fig. 10(b), the edge connecting node 17 to 16 ($e_{17,16}$) is more costly than the one connecting node 30 to 10 ($e_{30,10}$), despite it being shorter in length. To generate this graph, 50 particles were used to calculate the success probability and cost-to-go from each edge. Since edge $e_{17,16}$ passes near the middle of two adjacent obstacles, it was more likely for the simulator particles to collide with either of the obstacles. However, edge $e_{30,10}$ passes closer to one obstacle, leaving a large gap on its other side. Additionally, it does not run parallel to the obstacle boundaries, meaning the window of opportunity for collisions is smaller. These factors contribute to it having a lower-cost-to-go than its neighboring edge.

The results of the previous experiments can be used in a real-world setting with physical robotic platforms. For instance, Fig. 11(a) shows a projected overlay of the LMA graphs over 2 iRobot Create robots. As the robots move between nodes, or as their goal nodes change, the cost-to-go graph can be updated and re-visualized in real-time. The ability to visualize cost-to-go from all starting nodes was found to be highly beneficial in hardware tests as introduction of modeling errors or disturbance inputs into the system often resulted in straying of the robots to nearby nodes, after which the visuals were referenced for determining the robots' new trajectories to goal.

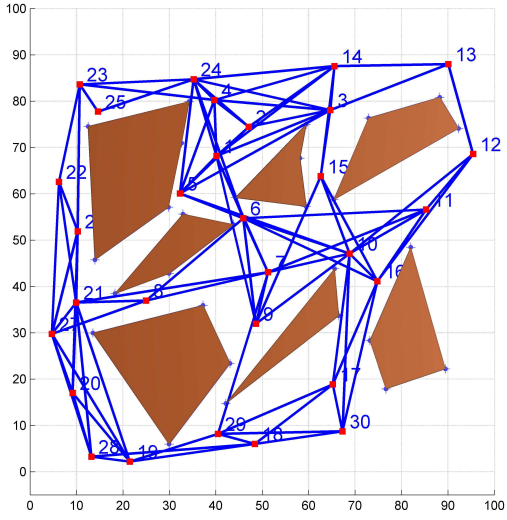
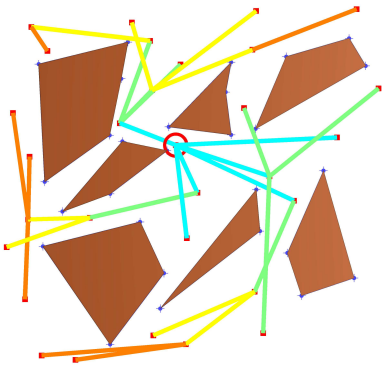
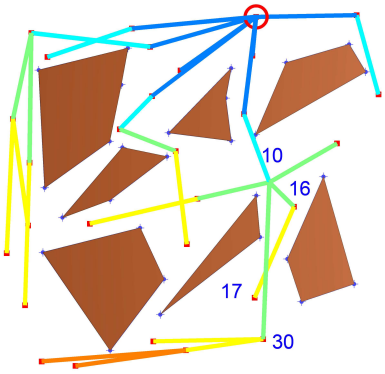


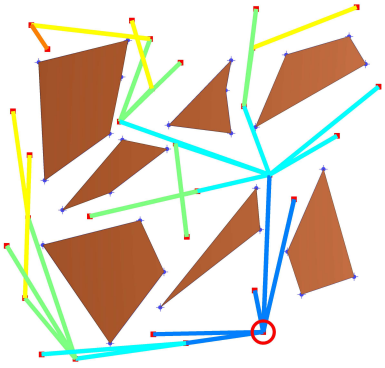
Figure 9. A graph of LMAs over 30 nodes (red). Connecting edges are indicated in blue, and obstacles in brown.



(a)



(b)



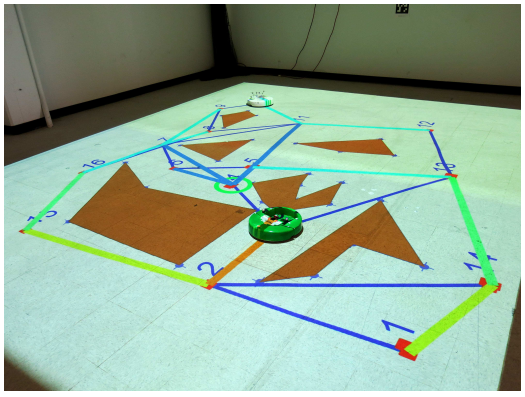
(c)

Figure 10. TMA policy for varying goal nodes, starting at any given node. The goal node is circled in red, and edge colors correspond to cost-to-go, where red edges are more costly than blue edges.

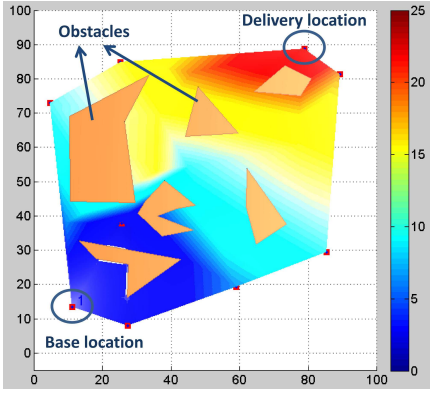
7.3 MMCS Policy Results

A portion of a policy for a single air robot in the package delivery domain is illustrated in Fig. 8. The policy involves going to $Base_1$ and observing x^e. Subsequently, the robot chooses to either pick up the package (alone or with another robot) or go to $Base_2$. The full policy controller includes more nodes than shown in Fig. 8 (for all possible TMAs) as well as edges (for all possible environment observations).

Fig. 12 compares a uniform random Monte Carlo search to MMCS in the package delivery domain. Results for the



(a) Cost-to-go for an example domain, projected on robots in a lab environment.



(b) Value of "Go to $Dest_1$" TMA over its belief space (only the 2D mean part is shown).

Figure 11. TMA cost-to-go and value visualization.

Table 2. Comparison of the maximum values obtained by the search algorithms.

Algorithm	Policy Value
Monte Carlo Search	2.07
MDHS (**?**)	2.67
MMCS	4.53
G-DICE	14.44
Exhaustive Search	—

Monte Carlo search were generated by repeatedly sampling random, valid policies and retaining the policy with the highest expected value. Both approaches used a policy controller with a fixed number of nodes ($n_{nodes} = 13$). Results indicate that given a fixed computational budget (quantified by the number of search iterations), MMCS outperforms standard Monte Carlo search in terms of expected value. Specifically, after 1000 search iterations in the package delivery problem, the expected value of the policy from MMCS is 118% higher than the one obtained by Monte Carlo search. Additionally, due to this problem's extremely large policy space, determination of an optimal joint value through exhaustive search is not possible. Fig. 12 illustrates MMCS's ability to exploit previous policy

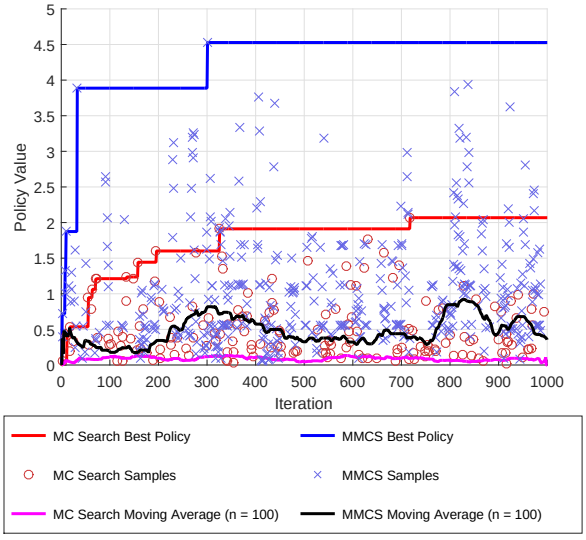
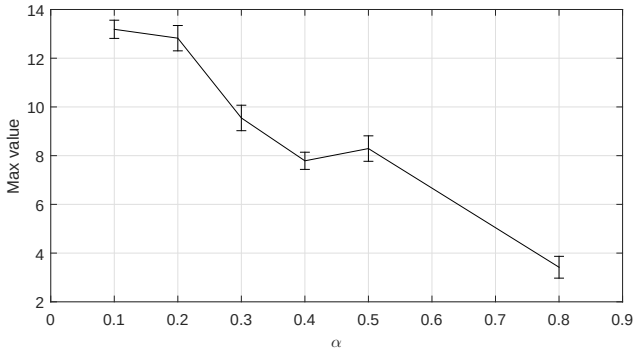
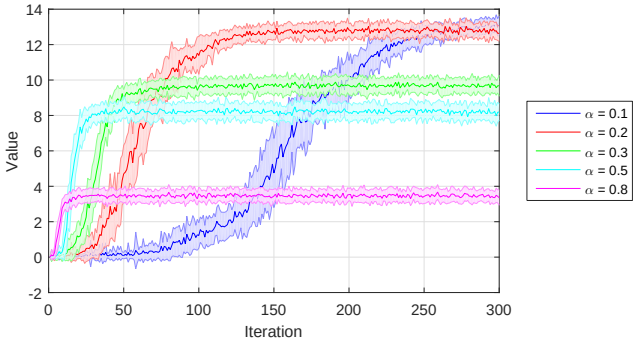


Figure 12. Policy differences between MC and MMCS. Moving average of policy values (over 100 samples) are also indicated.

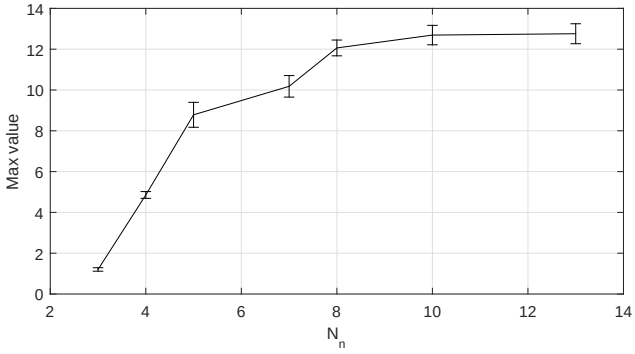


(a) Max joint value obtained for the package delivery domain using G-DICE, for varying learning rates, α. Note that the *mean* of the max values are shown here. The absolute max is indicated in Table 2.

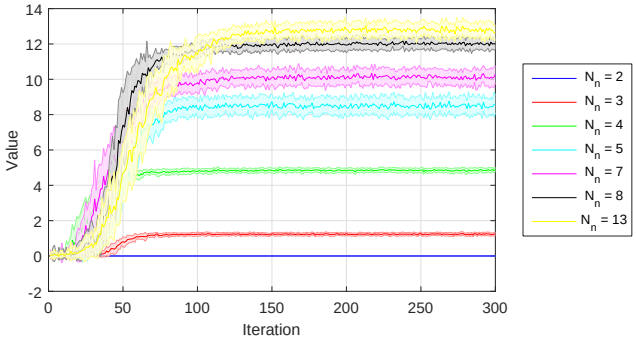


(b) Effect of learning rate on policy value convergence.

Figure 13. G-DICE results for varying learning rate.



(a) Max joint value obtained for the package delivery domain using G-DICE, for varying policy controller sizes, N_n.



(b) Effect of varying policy controller size on policy value convergence.

Figure 14. G-DICE results for varying policy controller sizes.

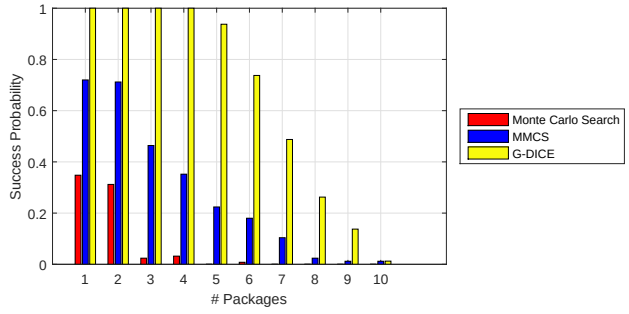


Figure 15. Success probability of delivering the specified number of packages *or more* within a fixed time horizon.

samples to bias towards well-performing regions of the policy space, while using random sampling for further exploration of the space.

Fig. 12 details policy values at each iteration of both the MMCS and Monte Carlo methods, allowing better understanding of MMCS' performance. Specifically, it can be observed that while the average policy value for the Monte Carlo method remains fairly constant, it shows patterns of fluctuations in the MMCS case. Peaks of high average policy value can be seen in the MMCS data, where good policies are sampled consistently. Though values of policies over successive iterations appear to be correlated, the process of resampling the mask (in Alg. 5, Line 17) allows exploration of other regions of the policy-space which may be more promising.

7.4 G-DICE Policy Results

MMCS is a greedy algorithm designed for finding good quality solutions with few samples, leveraging shared patterns in sampled policies to bias search towards promising regions of the policy space. However, the form of MMCS shown in Alg. 5 has no probabilistic guarantees, meaning that the solution returned in some domains may be a local optima.

On the other hand, G-DICE is an importance sampling-based algorithm which outperforms MMCS in policy quality. As seen in Table 2, the value of the package delivery policy generated by G-DICE (14.44) is more than 3 times the MMCS policy value (4.53). G-DICE policy values for varying learning rates are shown in Fig. 13(a). Higher learning rates typically result in lower joint policy value. G-DICE with a learning rate of $\alpha \approx 0.8$ results in a policy similar in value to the MMCS policy. With learning rate $0 \leq \alpha \leq 0.2$, the resulting G-DICE policies have fairly similar value.

Fig. 13(b) shows convergence characteristics for G-DICE with varying learning rates. Though lowering the learning rate causes higher joint policy value, it also increases the convergence time of the algorithm. This behavior is typical in stochastic search algorithms, and recent efforts have focused on developing adaptive learning rates or removing them altogether for algorithms such as stochastic gradient descent (**?**). Given a computational budget defined as the maximum number of G-DICE iterations, N_k, an appropriate learning rate can be determined empirically and is domain-specific. In the package delivery domain presented, a learning rate of $\alpha = 0.2$ was found to provide a balance between solution quality and convergence time. This analysis motivates future work in the area of developing adaptive learning rates for G-DICE.

Fig. 14(a) illustrates the effect of changing the number of graph nodes, N_n, on the joint policy value. Graphs with more nodes have higher fidelity and provide better policies in general, as they are able to store more accurate representation of action-observation histories for the robots. However, as Fig. 14(b) illustrates, graphs with more nodes require longer time for convergence. Additionally, once a large enough N_n is chosen, the value of the policy returned is near-optimal. In the package delivery example, $N_n \geq 10$ results in policies with similar values. These results suggest a trade-off in the graph size, similar to the one presented for learning rate. Memory limitations can be used to place an upper bound on the graph size, and empirical trials may suggest an appropriate graph size for a given problem.

An interesting observation in Fig. 14(b) is that although the $N_n = 2$ graph results in a zero-value policy (since there are not enough nodes to perform the chains of TMAs required for a successful package delivery), the $N_n = 3$ graph yields a policy with positive value. This is because the policy generated by G-DICE for $N_n = 3$ involves the robots performing a pickup, traveling to a delivery destination, and dropping off the package, a total of 3 TMAs. However, once the delivery is made, the combination of TMAs possible in a 3-node graph have been exhausted, and the robot does not have the ability to travel to the base location for another package pickup. Additionally, since only 1 TMA is allotted for delivery, only 1 of the delivery destinations can be fulfilled. Thus, the $N_n = 3$ policy involves the robot picking up the package, delivering it to one of the destinations, dropping it off, and remaining in the delivery destination. This behavior is a direct result of the constraints imposed on graph size, and serves as a check to ensure that the joint policies produced by G-DICE are intuitively valid.

To further quantify performance of G-DICE, MMCS, and Monte Carlo policies in the package delivery domain, Fig. 15 compares success probability of delivering a given minimum number of packages for both methods, within a fixed mission time horizon. Results were generated over 250 simulated runs with randomly-generated packages and initial conditions. Using the Monte Carlo policy, the probability of successfully delivering 5 or more packages given a fixed time horizon is near zero, whereas the MMCS policy successfully delivers a larger number of packages (in some cases up to 9). The G-DICE policy (with $N_n = 13$ and $\alpha = 0.2$) clearly outperforms the others, delivering up to 3 packages with probability 1.0, and up to 6 packages with probability of approximately 0.8. Similarly to MMCS, the probability of delivering 9 or more packages becomes near-zero, due to the mission time horizon constraining the number of deliveries possible.

7.5 Comparisons to MDHS

The MDHS algorithm (**?**) is a recent heuristic search method which searches over the space of FSAs. MDHS uses a Mealy machine representation for FSAs (rather than the Moore machine used in our approach), although the difference in implementation is trivial. A more substantial difference between our framework and the one used in **?** is that the latter assumes full specification of MAs by a domain expert, whereas our approach benefits from automatic construction of MAs. However, the hierarchical nature of our approach can be used to generate MAs for MDHS. This allows meaningful direct comparisons against MMCS and G-DICE in the package delivery domain.

MDHS uses upper and lower bound heuristics on policy value to target promising policy space regions during search. It begins search with an empty controller (as in Fig. 6(a)) and incrementally defines actions $\lambda^{(i)}$ and transitions $\delta^{(i)}$ for additional nodes. It uses the maximum-valued trajectory of randomly-sampled fillings of remaining unfilled nodes to estimate an upper bound for the policy value defined by each partially-filled FSA. Any policy with an upper bound greater than the current best policy value is added to a policy search list, while others are discarded (and their children pruned from the search space). It then repeats this process, incrementally defining further nodes for policies in the search list, estimating upper bounds using random fillings, and either expanding partial policies further or discarding them (and pruning the search space of their child policies). MDHS is an anytime algorithm which can be stopped at any point to return the latest best policy.

MDHS is executed for a policy with $N_n = 13$ nodes for the same number of iterations as the G-DICE results shown in Table 2. This results in a policy value of 2.67, in comparison to Monte Carlo, MMCS, and G-DICE policies with values 2.07, 4.53, and 14.44, respectively. The low value of the MDHS policy is explained due to the extremely large number of child policies which must be expanded for our package delivery domain, making it infeasible for the heuristics-based pruning approach to completely search the policy space. Specifically, the number of child policies to be expanded at each step are $|\mathbb{T}|(N_n)^{|\bar{\bar{\mathbb{O}}}|}$, which is a formidable number even after pruning policies with invalid sequences of MAs and observations. For instance, for the "go-to-base" MA, there are 12 valid subsequent observations possible, meaning there are approximately $3.9e + 15$ expanded policies.

Despite MDHS using upper bound heuristics to prune the policy space, it must still calculate this upper bound for each child policy during its expansion step, meaning that (for our domain at least) a significant amount of computation is spent on policies which are eventually pruned. This leads MDHS to perform only slightly better than a Monte Carlo search in our domain. MDHS performance could be improved with a better choice of heuristic, although this is left as future work in **?**. An advantage of MDHS is that it is a complete search algorithm with convergence guarantees, although the time required for convergence can be nearly infeasible for extremely large domains where pruning is ineffective. On the other hand, G-DICE is an effective search strategy for such domains, although its performance is sensitive to the choice of learning rate α. Such trade-offs must be considered carefully when selecting the appropriate search strategy for a given domain.

7.6 Hardware Experiments: Constrained Package Delivery with Health Awareness

We extend the simulated package delivery domain to a hardware domain, with some modifications (Fig. 16). The

Aerospace Controls Laboratory uses a combination of motion capture and augmented reality projection system referred to as MAR-CPS (Measurable Augmented Reality for Prototyping Cyber-Physical Systems) (**?**), allowing display of augmented visual elements directly on top of hardware platforms. Thus, visualizations including bases, delivery destinations, packages, and planned robot paths are shown in real-time during hardware experiments.

Fig. 16 provides an overview of the hardware domain, with 2 base locations (where package generation occurs) and 3 delivery destinations (color-coded as blue, green, and pink). The mission involves 4 quadrotors (circled in red) performing package delivery with no communication between them. A quaternion-based dynamical model, outlined in **?**, is used for the vehicles. The quadrotors have partial observability of the packages, only observing packages detected by their camera sensors. The policy is obtained using G-DICE, due to it outperforming MMCS when given a long enough processing time. Parameter values used for G-DICE were $N_k = 100$, $N_s = 100$, $N_b = 10$, $N_n = 13$, and $\alpha = 0.1$. Fig. 17(a) illustrates a blue package generated at the base, corresponding to blue delivery destination 3 (shown in Fig. 16). As in the simulation domain, the robots are only rewarded if their package is delivered to the correct destination. The color-coding visually indicates where each package is meant to be delivered, as opposed to where it is actually delivered.

Fig. 17(b) illustrates a quadrotor which has just visited a base and picked up a package. The package is shown as a square beneath the quadrotor as it flies, and its blue color indicates that its target is delivery destination 3. In this case, the base has regenerated another blue package (represented by a blue square over it), but in general a package destined for any of the 3 delivery targets can be generated. Fig. 17(b) also shows the planned trajectory of the quadrotor (shown as a cyan path leading to the blue delivery destination), which is updated in real-time due to the presence of obstacles and other robots in the domain. Fig. 19 illustrates the transition process which occurs when a robot picks up a blue package at the base, at which point the base displays the next package in the queue (in this case pink). The robot then continues onward to its delivery destination, while a teammate approaches the base to pick up the next package.

Health-awareness was also implemented in this hardware domain, where each robot has 3 health states - high, medium, and low. Since a noisy sensor is used for obtaining health observations, the robot also maintains a belief over health state. In the hardware setup, health belief is represented by a ring around each quadrotor (Fig. 18(a)). The health ring consists of 3 colors: green, yellow, and red, corresponding to high health, medium health, and low health belief, respectively. For instance, in Fig. Fig. 18(a), the quadrotor has approximately 25% belief that its health is high (green), 25% that it is medium (yellow), and 50% that it is low (red). As the robot's health drops, the health belief is updated in real-time based on noisy health observations from the model shown in Table 3. The observation model is designed such that at high health, the robot never mis-observes its health state as being low. On the other hand, at low health, there is 0.1 probability that the health state is reported as high (due to health sensors being more likely to malfunction at low health).

Table 3. Health observation model, denoting probabilities of receiving a certain health observation given the health state.

		Health State		
		L	M	H
Health Observation	L	0.6	0.2	0.0
	M	0.3	0.6	0.3
	H	0.1	0.2	0.7

A "repair" TMA has also been added to this domain, involving the robot going to a healing station (shown in Fig. 16) and repairing itself. As each robot's health decreases, its TMAs take longer to complete. Thus, due to discount factor γ in the Dec-POSMDP dynamic programming Eq. (20), the joint reward will be lowered if the robots choose not to repair themselves (due to delivery actions taking longer and rewards being delayed). Joint pickup and delivery as well as ground robot TMAs have been removed (although solo TMAs remain). 2 additional air robots have been added for a total of 4, in contrast to the simulation domain which only had 2.

Fig. 20(a) illustrates the experiment running with all visual elements in place. The robots start with high health, though based on their noisy observations, they all have some belief of being in medium health. All robots have picked up a package from a base and have planned a path to their respective delivery destinations. Fig. 20(b) shows the mission after a few timesteps, at which point two of the robots with the pink packages have reached their destination. Another robot (at the green destination) has just finished delivering its package, and has chosen a "go-to-base" TMA in order to pick up a subsequent package. The final robot, at the top-left of the image, has already arrived at the base and is preparing for the next package pickup. Package pickup and delivery continues in this manner, with robots deciding which base to return to after delivery. For bases near the center of the domain (such as the green delivery destination in Fig. 20(a)), this decision is complex due to the destination being equidistant from the bases.

Policy execution continues and in general the robots shift towards low health beliefs, although due to the observation model in Table 3, the robot's health belief can sometimes shift back towards higher health due to increased sensing

noise when damaged. Fig. 18(b) illustrates a robot with low health belief (primarily in the low-medium region with a red-yellow ring) which has arrived at a base but chooses instead to go to the healing station for repair.

The decision-making process in this domain is complex due to both the inclusion of probabilistic completion times for each TMA as well as health-awareness for the robots. Due to the joint reward being discounted, decision-making regarding which base to return to after delivery and when to perform a repair action is critical. An intuitive policy would involve the robots not wasting critical amounts of time visiting all the bases or destinations, since time-wasting causes further discounting of rewards. However, policy verification by a domain expert is also difficult due to complexities introduced by the impacts of health state on each robot's performance.

The experiments indicate the Dec-POSMDP framework combined with the G-DICE and MMCS algorithms allow solutions for tractable solving of complex multi-robot problems, such as package delivery, that would not be possible using traditional Dec-POMDP approaches.

8 Conclusion

This paper proposed a hierarchical framework to exploit macro-action abstraction in decentralized planning for a group of robots operating under uncertainty. We developed a set of equations that formally detail the implementation of macro-actions into Dec-POMDPs, resulting in a Dec-POSMDP that can be solved using discrete space search techniques. We also formulated three algorithms, policy iteration, MMCS, and G-DICE, for solving Dec-POSMDPs and compared the performance of two of them on a multi-robot package delivery problem under uncertainty, with a health-aware hardware experiment also implemented.

The Dec-POSMDP represents a formalism for probabilistic multi-robot coordination problems and our results show that high-quality solutions can be automatically generated from this high-level domain specification. We note that the framework currently uses FIRM (**?**) for automatic TMA generation, therefore assuming Gaussian beliefs within the TMAs. However, as discussed in Sec. 3, the high-level framework is agnostic to TMA internals, and it is possible to use other means of TMA generation with looser assumptions. A further limitation of the current framework is that e-observations are assumed to be discrete and received upon termination of TMAs, which is a reasonable assumption in the POSMDP setting (**?**). Consideration of continuous e-observations and rigorous investigation of continuous-time Dec-POSMDP equations remain future work. Additional future work includes investigation of adaptive learning rates for G-DICE, integration of more complex e-observations into the framework, and investigation of automatic node pruning in G-DICE to yield compact policy graphs.

Acknowledgements

This work was supported by Boeing Research & Technology and ONR MURI grant N000141110688. The authors gratefully acknowledge the anonymous reviewers for their highly insightful comments and feedback.

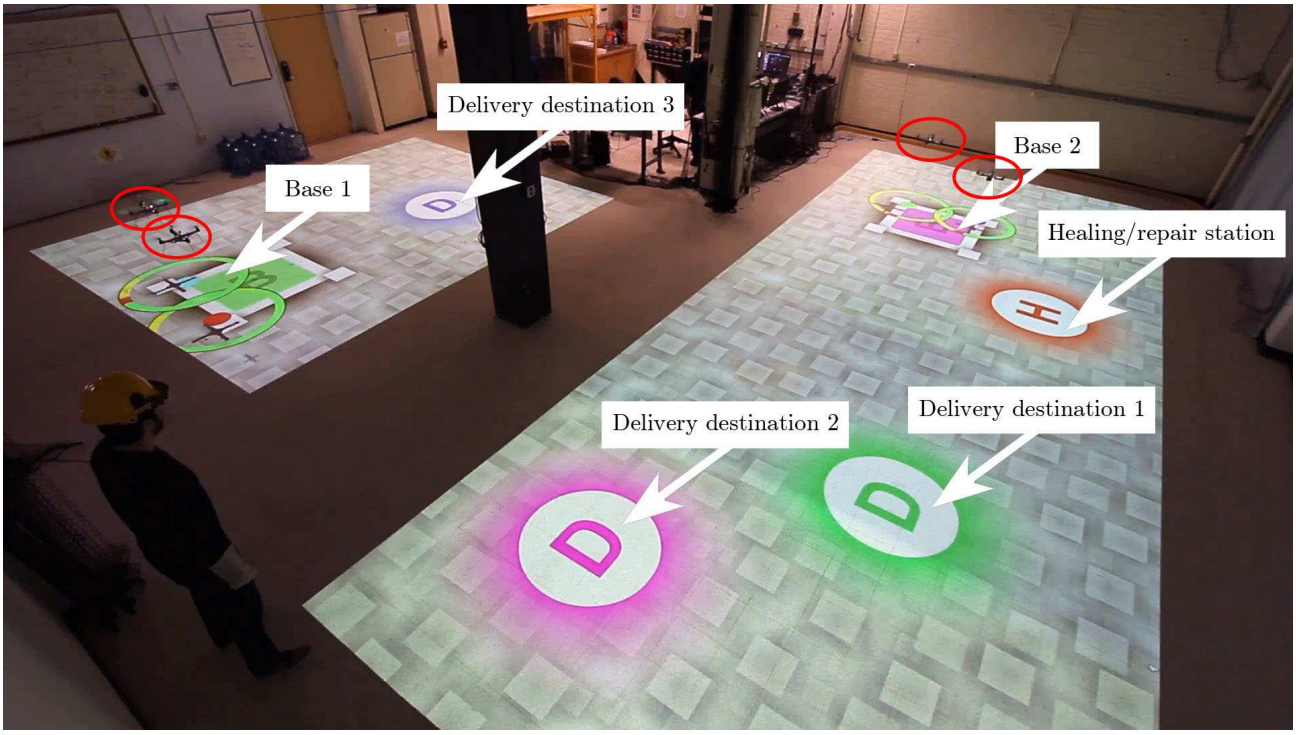
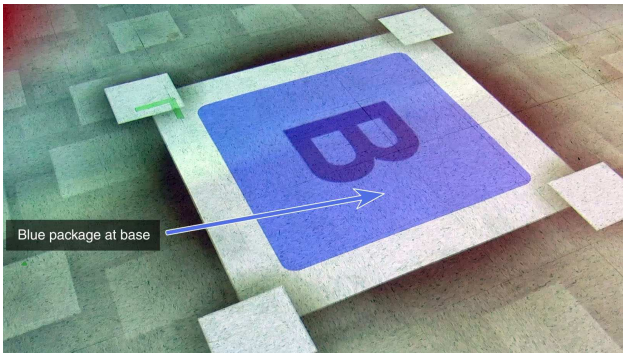
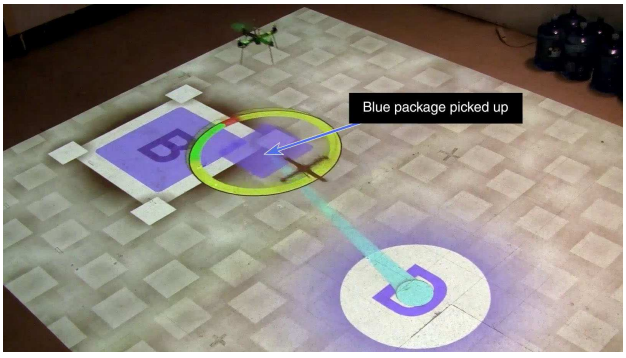


Figure 16. Overview of constrained package delivery hardware domain.

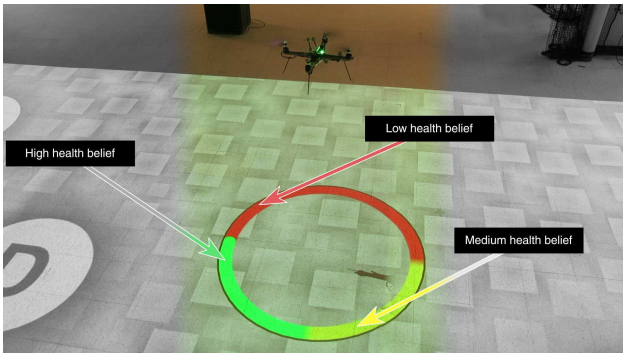


(a) Visualization of blue package generated at base.

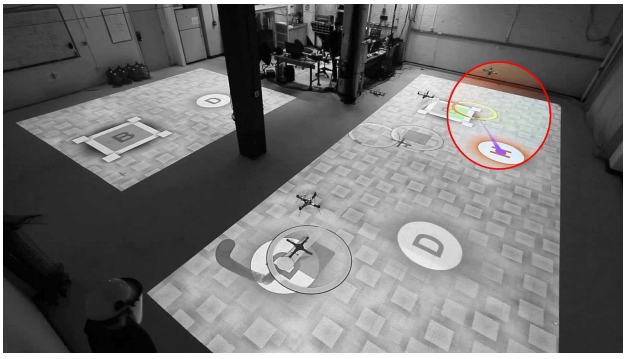


(b) A quadrotor is carrying a blue package to the corresponding blue delivery destination.

Figure 17. Package delivery visualization elements.

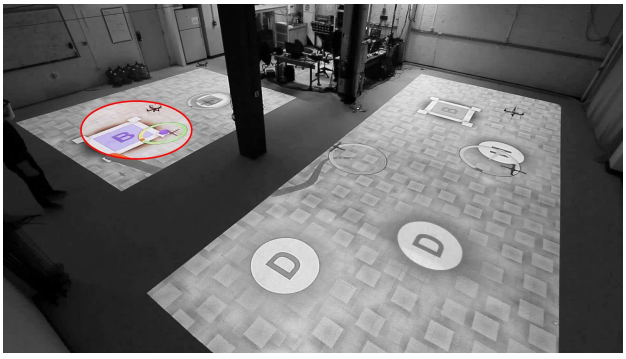


(a) Health belief visualization is done using a "belief ring", indicating the probability of the robot being in a given health state based on noisy observations of its health.

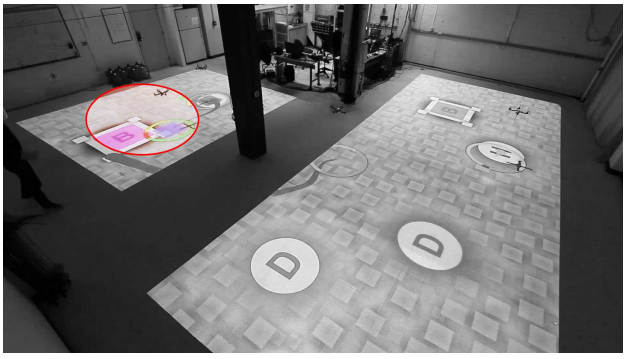


(b) Quadrotor going to the repair station due to low health belief.

Figure 18. Health belief and vehicle repair visualization.

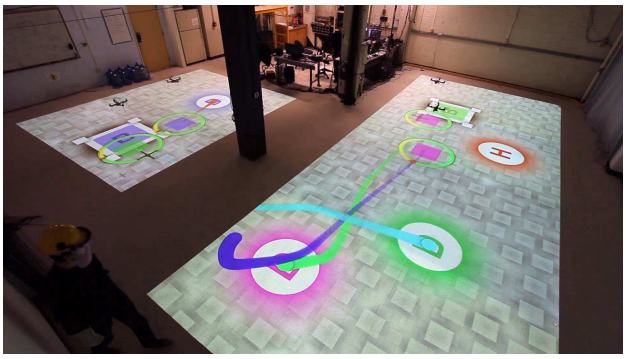


(a) The base has a blue package as the quadrotor approaches.

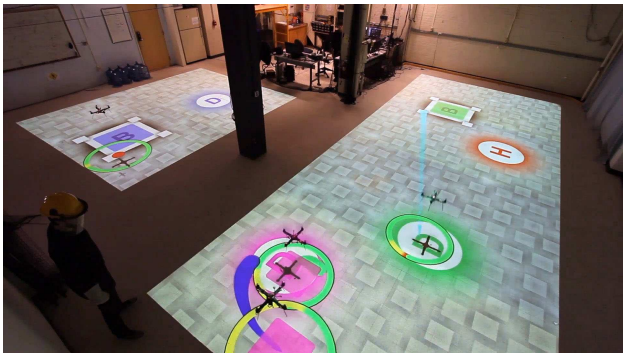


(b) The quadrotor has picked up the blue package (now shown underneath it), and the base displays the next package in the queue (pink).

Figure 19. Visualization of quadrotor package pickup.



(a) The mission starts and quadrotors plan paths to their appropriate delivery destinations.



(b) Quadrotors are completing package delivery and deciding which base to return to.

Figure 20. Full mission visualization at run-time.

9 Appendix

9.1 Joint Value of Decentralized Policy

The definition of the joint value $\bar{V}^{\bar{\phi}}$ of decentralized policy $\bar{\phi}$ is defined at the primitive reward level as

$$\bar{V}^{\bar{\phi}}(\bar{b}_0, x_0^e) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \bar{R}(\bar{x}_t, x_t^e, \bar{u}_t) | \bar{b}_0, x_0^e; \bar{\phi}\right] \quad (33)$$

which can be partitioned into joint rewards obtained per epoch, k, as follows

$$= \mathbb{E}\left[\sum_{k=0}^{\infty} \sum_{t=t_k}^{t_{k+1}-1} \gamma^t \bar{R}(\bar{x}_t, x_t^e, \bar{u}_t) | \bar{b}_0, x_0^e; \bar{\phi}\right] \quad (34)$$

$$= \sum_{k=0}^{\infty} \mathbb{E}\left[\sum_{t=t_k}^{t_{k+1}-1} \gamma^t \bar{R}(\bar{x}_t, x_t^e, \bar{u}_t) | \bar{b}_0, x_0^e; \bar{\phi}\right] \quad (35)$$

where $t_0 = 0$. We can then marginalize out the initial belief and e-state configuration $(\bar{b}_0, x_0^e)$, taking an expectation over only the k-th epoch's belief and e-state configuration $(\bar{b}_{t_k}, x_{t_k}^e)$ as follows

$$= \sum_{k=0}^{\infty} \sum_{\bar{b}_{t_k}, x_{t_k}^e} \mathbb{E}\left[\sum_{t=t_k}^{t_{k+1}-1} \gamma^t \bar{R}(\bar{x}_t, x_t^e, \bar{u}_t) | \bar{b}_{t_k}, x_{t_k}^e; \bar{\phi}\right] P(\bar{b}_{t_k}, x_{t_k}^e | \bar{b}_0, x_0^e; \bar{\phi}). \quad (36)$$

Now, making a change of time indices,

$$= \sum_{k=0}^{\infty} \sum_{\bar{b}_{t_k}, x_{t_k}^e} \gamma^{t_k} \mathbb{E}\left[\sum_{t=0}^{t_{k+1}-1-t_k} \gamma^t \bar{R}(\bar{x}_t, x_t^e, \bar{u}_t) | \bar{b}_{t_k}, x_{t_k}^e; \bar{\phi}\right] P(\bar{b}_{t_k}, x_{t_k}^e | \bar{b}_0, x_0^e; \bar{\phi}). \quad (37)$$

Noting from the definition of τ in Eq. (15) that $t_{k+1} - t_k - 1 = \tau - 1$

$$= \sum_{k=0}^{\infty} \sum_{\bar{b}_{t_k}, x_{t_k}^e} \gamma^{t_k} \mathbb{E}\left[\sum_{t=0}^{\tau-1} \gamma^t \bar{R}(\bar{x}_t, x_t^e, \bar{u}_t) | \bar{b}_{t_k}, x_{t_k}^e; \bar{\phi}\right] P(\bar{b}_{t_k}, x_{t_k}^e | \bar{b}_0, x_0^e; \bar{\phi}). \quad (38)$$

Finally, using the definition of generalized reward in Eq. (14), we can express the joint value at the TMA level,

$$= \sum_{k=0}^{\infty} \sum_{\bar{b}_{t_k}, x_{t_k}^e} \gamma^{t_k} \bar{R}^{\tau}(\bar{b}_{t_k}, x_{t_k}^e, \bar{\pi}_{t_k}) P(\bar{b}_{t_k}, x_{t_k}^e | \bar{b}_0, x_0^e; \bar{\phi}) \quad (39)$$

$$= \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^{t_k} \bar{R}^{\tau}(\bar{b}_{t_k}, x_{t_k}^e, \bar{\pi}_{t_k}) | \bar{b}_0, x_0^e; \bar{\phi}\right]. \quad (40)$$

9.2 Extended Transition probabilities under TMAs

We can calculate the probability that a set of controllers $\bar{\pi}$ will execute until any other configuration is reached for a given amount of time k.

$$P(\bar{b}', x^{e'}, k | \bar{b}, x^e; \bar{\pi}) = P(x_k^{e'}, \bar{b}'_k | x_0^e, \bar{b}_0; \bar{\pi}) \quad (41)$$

$$= \sum_{x_{k-1}^e, \bar{b}_{k-1}} P(x_k^{e'}, \bar{b}'_k | x_{k-1}^e, \bar{b}_{k-1}, x_0^e, \bar{b}_0; \bar{\pi}) P(x_{k-1}^e, \bar{b}_{k-1} | x_0^e, \bar{b}_0; \bar{\pi}) \quad (42)$$

$$= \sum_{x_{k-1}^e, \bar{b}_{k-1}} P(x_k^{e'}, \bar{b}'_k | x_{k-1}^e, \bar{b}_{k-1}; \bar{\pi}) P(x_{k-1}^e, \bar{b}_{k-1} | x_0^e, \bar{b}_0; \bar{\pi}) \quad (43)$$

$$= \sum_{x_{k-1}^e, \bar{b}_{k-1}} P(x_k^{e'}, \bar{b}'_k | x_{k-1}^e, \bar{b}_{k-1}; \bar{\pi}(\bar{b}_{k-1})) P(x_{k-1}^e, \bar{b}_{k-1} | x_0^e, \bar{b}_0; \bar{\pi}) \quad (44)$$

$$= \sum_{x_{k-1}^e, \bar{b}_{k-1}} P(x_k^{e'} | \bar{b}'_k, x_{k-1}^e, \bar{b}_{k-1}; \bar{\pi}(\bar{b}_{k-1})) P(\bar{b}'_k | x_{k-1}^e, \bar{b}_{k-1}; \bar{\pi}(\bar{b}_{k-1})) P(x_{k-1}^e, \bar{b}_{k-1} | x_0^e, \bar{b}_0; \bar{\pi}) \quad (45)$$

$$= \sum_{x_{k-1}^e, \bar{b}_{k-1}} P(x_k^{e'} | x_{k-1}^e; \bar{\pi}(\bar{b}_{k-1})) P(\bar{b}'_k | x_{k-1}^e, \bar{b}_{k-1}; \bar{\pi}(\bar{b}_{k-1})) P(x_{k-1}^e, \bar{b}_{k-1} | x_0^e, \bar{b}_0; \bar{\pi}) \quad (46)$$

9.3 FSA Evaluation

The FSAs can be evaluated at a given initial joint belief $\bar{b}$ and initial joint node $\bar{q}$ over epochs as follows,

$$\begin{aligned}\bar{V}^{\bar{\phi}}(\bar{b}, x^e, \bar{q}) \\ = \bar{R}^{\tau}(\bar{b}, x^e; \bar{\phi}) + \\ \sum_{t_k, \bar{b}', x^{e'}} \gamma^{t_k} P(\bar{b}', x^{e'}, t_k | \bar{b}, x^e; \bar{\phi}) \bar{V}^{\bar{\phi}}(\bar{b}', x^{e'}, \bar{q}'))\end{aligned} \tag{47}$$

where $\bar{q}' = \bar{\delta}(\bar{q}, \bar{o}^e) = \{\delta^{(1)}(q^{(1)}, o^{e(1)}), \ldots, \delta^{(n)}(q^{(n)}, o^{e(n)})\}$ is the *deterministic* next-node in the policy graph for all agents. Explicitly defining the policy in terms of TMA output function $\bar{\lambda}(\bar{q})$,

$$\begin{aligned}= \bar{R}^{\tau}(\bar{b}, x^e, \bar{\lambda}(\bar{q}); \bar{\phi}) \\ + \sum_{t_k, \bar{b}', x^{e'}} \gamma^{t_k} P(\bar{b}', x^{e'}, t_k | \bar{b}, x^e, \bar{\lambda}(\bar{q}); \bar{\phi}) \bar{V}^{\bar{\phi}}(\bar{b}', x^{e'}, \bar{q}')).\end{aligned} \tag{48}$$

Substituting $\bar{q}' = \bar{\delta}(\bar{q}, \bar{o}^e)$,

$$\begin{aligned}= \bar{R}^{\tau}(\bar{b}, x^e, \bar{\lambda}(\bar{q}); \bar{\phi}) \\ + \sum_{t_k, \bar{b}', x^{e'}, \bar{o}^e} P(\bar{b}', x^{e'}, t_k, \bar{o}^e | \bar{b}, x^e, \bar{\lambda}(\bar{q}); \bar{\phi}) \bar{V}^{\bar{\phi}}(\bar{b}', x^{e'}, \bar{\delta}(\bar{q}, \bar{o}^e))\end{aligned} \tag{49}$$

$$\begin{aligned}= \bar{R}^{\tau}(\bar{b}, x^e, \bar{\lambda}(\bar{q}); \bar{\phi}) \\ + \sum_{t_k, \bar{o}^e} \gamma^{t_k} P(\bar{o}^e | \bar{b}, x^e, \bar{\lambda}(\bar{q}); \bar{\phi}) \\ \sum_{\bar{b}', x^{e'}} P(\bar{b}', x^{e'}, t_k | \bar{b}, x^e, \bar{\lambda}(\bar{q}), \bar{o}^e) \bar{V}^{\bar{\phi}}(\bar{b}', x^{e'}, \bar{\delta}(\bar{q}, \bar{o}^e))\end{aligned} \tag{50}$$

$$\begin{aligned}= \bar{R}^{\tau}(\bar{b}, x^e, \bar{\lambda}(\bar{q}); \bar{\phi}) \\ + \sum_{t_k, \bar{o}^e} P(\bar{o}^e | \bar{b}, x^e, \bar{\lambda}(\bar{q}); \bar{\phi}) \\ P(k | \bar{b}, x^e, \bar{\lambda}(\bar{q}), \bar{o}^e; \bar{\phi}) \bar{V}^{\bar{\phi}}(\bar{b}'_{\bar{o}^e, \bar{\lambda}(\bar{q})}, x^{e'}_{\bar{o}^e, \bar{\lambda}(\bar{q})}, \bar{\delta}(\bar{q}, \bar{o}^e)).\end{aligned} \tag{51}$$