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

Shayegan Omidshafiei, Ali-akbar Agha-mohammadi, Christopher Amato, Shih-Yuan Liu, Jonathan P. How*

Abstract

The focus of this paper is 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, but representing and solving Dec-POMDPs is often intractable for large problems. To allow for a high-level representation that is natural for multi-robot problems and scalable to large discrete and continuous problems, this paper extends the Dec-POMDP model to the Decentralized Partially Observable Semi-Markov Decision Process (Dec-POSMDP). The Dec-POSMDP formulation allows asynchronous decision-making by the robots, which is crucial in multi-robot domains. We also present an algorithm for solving this Dec-POSMDP which is much more scalable than previous methods since it can incorporate closed-loop belief space macro-actions in planning. These macro-actions are automatically constructed to produce robust solutions. The proposed method's performance is evaluated on a complex multi-robot package delivery problem under uncertainty, showing that our approach can naturally represent multi-robot problems and provide high-quality solutions for large-scale problems.

1. Introduction

Given the low cost of hardware, it is becoming more cost-effective to deploy multiple robots to complete a single 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. Ideally, high-quality controllers for each robot would be automatically generated based on a high-level domain specification while considering uncertainty in the domain. Such methods (such as those based on Markov decision processes (Puterman 1994) and partially observable Markov decision processes (Kaelbling et al. 1998)) have proven to be effective in single-robot domains, but similar methods have only begun to be considered in multi-robot problems.

Challenges: A general representation of the multi-robot coordination problem is the Decentralized Partially Observable Markov Decision Process (Dec-POMDP) (Bernstein et al. 2002). Dec-POMDPs have a broad set of applications including networking problems, multi-robot exploration, and surveillance (Bernstein et al. 2001, Emery-Montemerlo et al. 2005,

* S. Omidshafiei, A. Agha, S-Y. Liu, and J.P. How are with the Laboratory for Information and Decision Systems (LIDS), MIT, Cambridge, MA. C. Amato is with the Dept. of Computer Science at the University of New Hampshire, Durham, NH. Work for this paper was completed while all authors were at MIT. {shayegan,aliagha,camato,syliu,jhow}@mit.edu

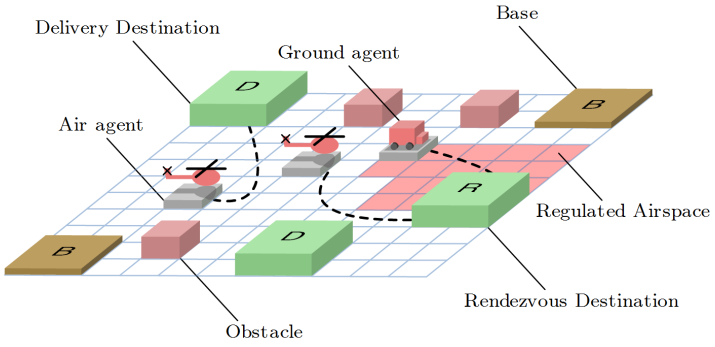


Fig. 1. Package delivery domain with key elements labeled.

Winstein and Balakrishnan 2013, Ure et al. 2013, N. Nigam and Vian 2012). Unfortunately, current Dec-POMDP solution methods are limited to small discrete domains and require synchronized decision-making.

In this paper, we extend promising recent work on incorporating macro-actions, temporally extended actions, (Amato et al. 2014, 2015a,b) to solve continuous and large-scale problems which were infeasible for previous methods. We present a method for both formally representing multi-robot coordination problems and automatically generating local planners based on the specification. While these local planners can be a set of hand-coded macro-actions, we also present an algorithm for automatically generating macro-actions that can then be sequenced to solve the problem. The result is a principled method for coordination in probabilistic multi-robot domains.

Macro-actions (MAs) have provided increased scalability in single-agent MDPs (Sutton et al. 1999) and POMDPs (Agha-mohammadi et al. 2014a, He et al. 2011, Lim et al. 2011), but are nontrivial to extend to multi-robot settings. Some of the challenges in extending MAs to decentralized settings are:

- In the decentralized setting, synchronized decision-making is problematic (or even impossible) as some robots must remain idle while others finish their actions. The resulting solution quality would be poor (or not implementable), resulting in the need for MAs that can be chosen asynchronously by the robots (an issue that has not been considered in the single-robot literature).
- Incorporating principled asynchronous MA selection is a challenge, because 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 varying 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.

MA-based Dec-POMDPs alleviate the above problems by no longer attempting to solve for a policy at the primitive action level, but instead considering temporally-extended actions, or MAs. This also addresses scalability issues, as the size of the action space is considerably reduced.

Proposed method: In this paper, we extend the Dec-POMDP to the Decentralized Partially Observable *Semi*-Markov Decision Process (Dec-POSMDP), which formalizes the use of closed-loop MAs. The Dec-POSMDP represents the theoretical basis for asynchronous decision-making in Dec-POMDPs. We also automatically design MAs using graph-based planning techniques. The resulting MAs are closed-loop and the completion time and success probability can be characterized analytically, allowing them to be directly integrated into the Dec-POSMDP framework. As a result, our framework can generate efficient decentralized plans which take advantage of estimated completion times to permit asynchronous decision-making. 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 design efficient discrete search algorithms for solving it, and demonstrate the performance of the method for the complex problem of multi-robot package delivery under uncertainty (Fig. 1).

2. Problem Statement

A Dec-POMDP (Bernstein et al. 2002) 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 continuous states, actions, and observation spaces. The agents share a single reward function based on the actions of all agents, making the problem cooperative, but their 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.

Formally, the Dec-POMDP problem considered in this paper[1] 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: $\bar{R} : \bar{\mathbb{S}} \times \bar{\mathbb{U}} \rightarrow \mathbb{R}$, the immediate reward for being in joint state $\bar{s} \in \bar{\mathbb{S}}$ and taking the joint action $\bar{u} \in \bar{\mathbb{U}}$.
- $\bar{\Omega}$ 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 $\bar{s}' \in \bar{\mathbb{S}}$.

The *full action-observation history* (obtained observations and taken actions) for the i-th robot is defined as follows,

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

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

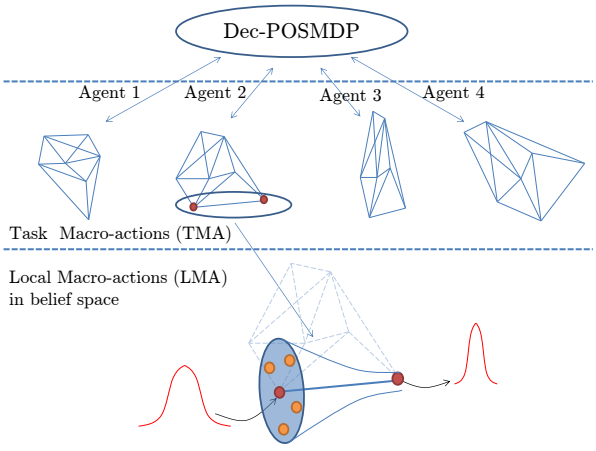


Fig. 2. Hierarchy of the proposed planner. In the highest level, a decentralized planner assigns a TMA to each robot. Each TMA encompasses a specific task (e.g. picking up a package). Each TMA in turn is constructed as a set of local macro-actions (LMAs). Each LMA (the lower layer) is a feedback controller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

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,

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

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 Section 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) $\bar{b} = p(\bar{s})$,

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

Then, a solution to a Dec-POMDP formally can be defined as the optimal policy

$$\bar{\eta}^* = \arg\max_{\bar{\eta}} \bar{V}^{\bar{\eta}}. \tag{4}$$

The Dec-POMDP problem stated in (4) is undecidable (as is the infinite-horizon POMDP problem even in discrete settings (Madani et al. 1999)). 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 macro-actions which can be executed in an asynchronous manner (Amato et al. 2014). 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* (Sutton et al. 1999) to multi-robot domains while dealing with the lack of synchronization between robots. The options framework is a formal model of MAs (Sutton et al. 1999) that has been very successful in aiding representation and solutions in single robot domains (Kober et al. 2013, Lim et al. 2011). Unfortunately, this method requires a full, discrete model of the system (including macro-action policies of all robots, sensors and dynamics at a low level).

As an alternative, we propose the Decentralized Partially Observable Semi-Markov Decision Process (Dec-POSMDP) framework which only requires 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. 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.

Open-loop versus closed-loop: To clarify the concepts, before describing TMAs, we distinguish between open-loop and closed-loop MAs. In general, MAs refer to temporally-extended actions (Theocharous and Kaelbling 2004). 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 we discuss in the next section, for a seamless incorporation of MAs in Dec-POMDP planning, we need to design closed-loop MAs, which is a challenge in partially-observable settings.

Task Macro-Action (TMA): 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. In this paper, we utilize information roadmaps (Agha-mohammadi et al. 2014a) as a substrate to generate such TMAs. We start by discussing the structure of feedback controllers in partially-observable domains.

Belief within macro-actions: 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 (Kumar and Varaiya 1986) that making decisions based on belief $b^{(i)}$ is equivalent to making decisions based on the history $H^{\pi^{(i)}}$ in a POMDP.

Feedback in belief space: 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., $(b_{k+1}, u_{k+1}) = \mathcal{L}(b_k, u_k, z_{k+1}) = (\tau(b_k, u_k, z_{k+1}), \mu(\tau(b_k, u_k, z_{k+1})))$.

Local macro-action (LMA): 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 Agha-mohammadi et al. (2014a) 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 funnel, the LMA drives the belief toward a target belief that is referred to as a *milestone* herein.

Linear LMAs: 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 (Agha-mohammadi et al. 2014a).

Chaining/Graphing LMAs: 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).

Constructing TMAs by Graphing Linear LMAs: Alg. 1 recaps the construction of a TMA using 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 Agha-mohammadi et al. (2014a)). An edge controller is a finite-time controller whose role is to take the mean of 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 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 are 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 system ends up in B^0. For more details on this procedure see Omidshafiei et al. (2014), Agha-mohammadi et al. (2014a).

Edge rewards and probabilities: 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. Finally, by $\mathcal{T}^{ij} = \mathcal{T}(B^i, \mathcal{L}^{ij})$ we denote the time it takes for LMA $\mathcal{L}^{ij}$ to complete its execution starting from B^i.

3.1. 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 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

Algorithm 1: TMA Construction (Offline)

Procedure : `ConstructTMA`$(b, \mathbf{v}^{goal}, \mathcal{M})$
input : Initial belief b, mean of goal belief $\mathbf{v}^{goal}$, task POMDP $\mathcal{M}$;
output : TMA policy π^*, success probability of TMA $P(B^{goal}|b_0;\pi^*)$, value of taking TMA $V(b_0;\pi^*)$;
Sample a set of LMA parameters $\{\theta^j\}_{j=1}^{n-2}$ from the state space of $\mathcal{M}$, where θ^{n-2} includes $\mathbf{v}^{goal}$;
Corresponding to each θ^j, construct a milestone B^j in belief space;
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;
Connect milestones using LMAs $\mathcal{L}^{ij}$;
and compute the LMA rewards, execution time, and transition probabilities by simulating LMAs offline;
Solve the LMA graph DP in (5) to construct TMA π^*;
Compute the associated success probability $P(B^{goal}|b;\pi^*)$, completion time $T^g(B^{goal}|b;\pi^*)$, and value $V(b;\pi^*)$;
return TMA policy π^*, success probability $P(B^{goal}|b;\pi^*)$, completion time $T^g(B^{goal}|b;\pi^*)$, and value $V(b;\pi^*)$

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 structure of our proposed TMAs allows 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.

TMA value and policy: 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, etc. 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,

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

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

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 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 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.

Success probability of TMA: For a given optimal TMA π^*, the associated optimal value $V^{\pi^*}(B^i)$ from any node B^i is computed via solving (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 (Agha-mohammadi et al. 2014a).

Completion time of TMA: 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 $\bar{T}^g$ is a column 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 $P(B^j|B^i;\pi)$.

Therefore, 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, and reward can be computed offline.

4. Environmental state and Updated Model

We also extend TMAs to the multi-robot setting where there is an environmental state that is locally observable by robots and can be affected by other robots.

Decision epochs: Robots can choose TMAs at *decision epochs*. Since TMA selection is asynchronous, a decision epoch is considered to be a timestep in which any robot finishes its current TMA and chooses a subsequent TMA. 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.

Time notation: k denotes the epoch number and t_k denotes 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).

Environment state: 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 for allowing 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 Section 9) is "there is a package in this base". An robot can only get this example measurement if the robot is in the base (hence it is partial). There is a limited set of TMAs available at each x^e, which is denoted by $\mathbb{T}(x^e)$.

Environmental observation: 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.

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 $b_k^{(i)} \notin B^{(i),goal}$.

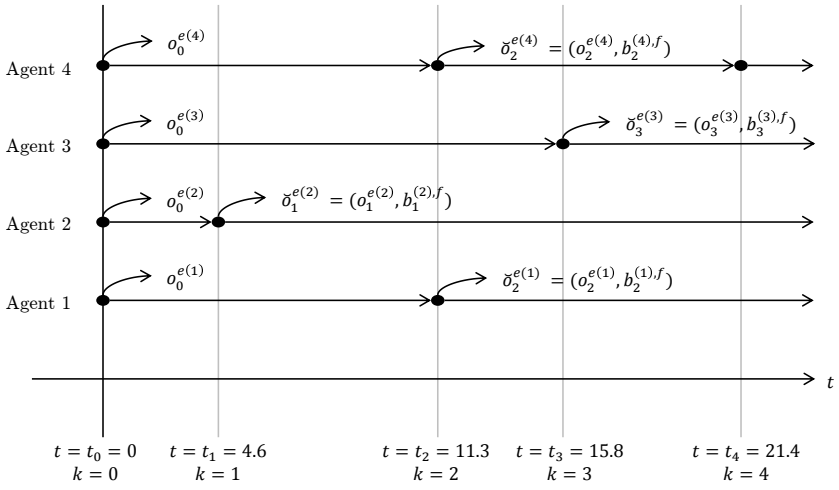


Fig. 3. Decision epochs t_k are defined as timesteps when any robot finishes its current TMA. At a decision epoch, each robot receives a TMA-observation $\breve{o}_k$, consisting of its belief and latest environment observation. In some scenarios, two robots may finish their TMAs at the same decision epoch, though this is a rare event due to TMA completion times being continuous.

Belief at epochs: 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.

Extended transition probabilities: 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.

Extended one-step reward: 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

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

In multi-robot planning domains, often computing R^e is 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.

Joint policy: 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.

Joint value: 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 general continuous problems.

TMA history: 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

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

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.

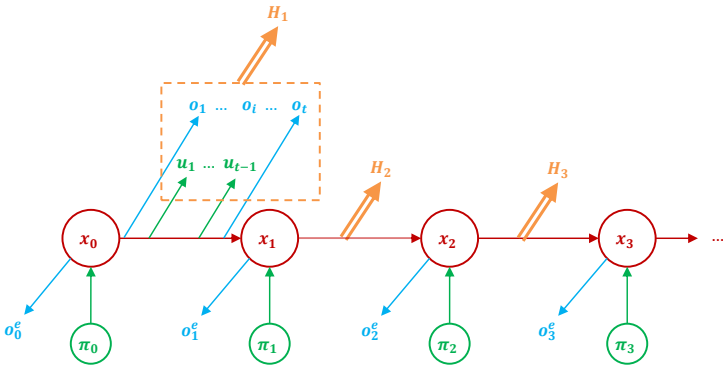


Fig. 4. The i-th robot moves from state x_0 through a sequence of TMAs, $\pi_0, \pi_1, \cdots, \pi_k$. Under each TMA it receives primitive actions $u_0, u_1, \cdots, u_t$ and perceives primitive observations $o_0, o_1, \cdots, o_t$. At the end of each TMA (as well as beginning of the mission), it also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

Compressed TMA history: 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, i.e., for the i-th 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

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

without losing any information (Kumar and Varaiya 1986). 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 $\Xi^{(i)}$.

Decentralized Policy: 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 $\bar{\xi} = \{\xi^{(1)}, \cdots, \xi^{(n)}\}$.

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

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

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 macro-actions in the semi-Markov setting, where macro-actions 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 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 macro-action $\bar{\pi}$ 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,

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

Below, we describe the elements of the Dec-POSMDP in more detail.

Generalized reward of a joint TMA: 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 macro-action $\bar{\pi}$ by evaluating the set of decentralized TMAs until at least one of them stops

$$\bar{R}^{\tau}(\bar{b}, x^e; \bar{\pi}) = \mathbb{E}\left[\sum_{t=0}^{\tau-1} \gamma^t \bar{R}(\bar{x}_t, x^e_t, \bar{u}_t)|p(\bar{x}_0) = \bar{b}, x^e_0 = x^e; \bar{\pi}\right] \tag{14}$$

where

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

The extended definition of reward shown 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.

Transition probabilities under MAs: It can be shown that the probability of transitioning from configuration $(\bar{b}, x^e)$ to configuration $(\bar{b}', x^{e'})$ after k steps under the joint TMA $\bar{\pi}$ can be solved using dynamic programming (see Appendix

11.2 for derivation),

$$P(\bar{b}', x^{e'}, k|\bar{b}, x^e; \bar{\pi}) = P(x_k^{e'}, \bar{b}'_k|x_0^e, \bar{b}_0; \bar{\pi}) \tag{16}$$

$$= \sum_{x_{k-1}^e, \bar{b}_{k-1}} \left[P(x_k^{e'}|x_{k-1}^e; \bar{\pi}(\bar{b}_{k-1})) P(\bar{b}'_k|x_{k-1}^e, \bar{b}_{k-1}; \bar{\pi}(\bar{b}_{k-1})) P(x_{k-1}^e, \bar{b}_{k-1}|x_0^e, \bar{b}_0; \bar{\pi}(\bar{b}_0)) \right]. \tag{17}$$

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

Joint value of the decentralized policy: 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,

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

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

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 11.1.

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

$$\bar{V}^*(\bar{b}, x^e) = \max_{\bar{\pi}} \{ \bar{R}^{\tau}(\bar{b}, x^e; \bar{\pi}) + \sum_{k=1}^{\infty} \gamma^{t_k} \sum_{\bar{b}', x^{e'}} P(\bar{b}', x^{e'}, k|\bar{b}, x^e; \bar{\pi}) \bar{V}^*(\bar{b}', x^{e'}) \}, \quad \forall\, \bar{b}, x^e \tag{20}$$

$$\bar{\pi}^* = \bar{\phi}^*(\bar{b}, x^e) = \arg\max_{\bar{\pi}} \{ \bar{R}^{\tau}(\bar{b}, x^e; \bar{\pi}) + \sum_{k=1}^{\infty} \gamma^{t_k} \sum_{\bar{b}', x^{e'}} P(\bar{b}', x^{e'}, k|\bar{b}, x^e; \bar{\pi}) \bar{V}^*(\bar{b}', x^{e'}) \}, \quad \forall\, \bar{b}, x^e. \tag{21}$$

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

6. Solving Dec-POSMDP using Finite State Automata and Policy iteration

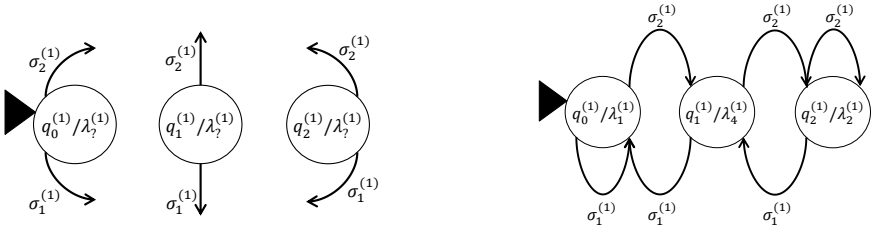
To solve the dynamic programming problem in Eq. (20), we rely on Finite State Automata (FSA) that induce the optimal joint policy $\bar{\phi}^*$.

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 $\pi \in \mathbb{T}$.

- $\delta : Q \times \Sigma \rightarrow Q$ is the deterministic transition function.
- $\lambda : Q \rightarrow \Lambda$ is the output function.
- $q_0 \in Q$ is the initial node.

Fig. 5 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.



(a) (b)

Fig. 5. Example FSA for a single robot, with the initial node indicated by the black arrow. On the left, an "empty" controllers with no parameters generated is shown (output functions λ and node-to-node transition function δ have not been solved for). On the right, the FSA with generated parameters is shown.

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. 6, 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 for robot i, $\phi^{(i)}$ 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 Section 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 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 11.3 for derivation),

$$\bar{V}^{\bar{\phi}}(\bar{b}, x^e, \bar{q}) = \bar{R}^{\tau}(\bar{b}, x^e, \bar{\lambda}(\bar{q}); \bar{\phi}) + \sum_{k=1}^{\infty} \gamma^{t_k} \sum_{\bar{o}^e} P(\bar{o}^e | \bar{b}, x^e, \bar{\lambda}(\bar{q}); \bar{\phi}) P(k | \bar{b}, x^e, \bar{\lambda}(\bar{q}), \bar{o}^e; \bar{\phi}) \bar{V}^{\bar{\phi}}(\bar{b}'_{\bar{o}^e, \bar{\lambda}(\bar{q})}, x^{e'}_{\bar{o}^e, \bar{\lambda}(\bar{q})}, \bar{\delta}(\bar{q}, \bar{o}^e)). \tag{22}$$

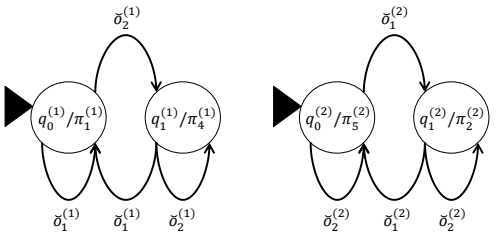


Fig. 6. Finite-state automata for two robots, each with two nodes, five possible actions and two observations. Note that superscripts indicate robot indices, while subscripts indicate indices of the nodes/TMAs/observations.

In Eq. (22),

$$P(k|\bar{b}, x^e, \bar{\lambda}(\bar{q}), \bar{o}^e) = P(k|\bar{b}'_{\bar{o}^e,\bar{\lambda}(\bar{q})}, x^{e'}_{\bar{o}^e,\bar{\lambda}(\bar{q})}, \bar{b}, x^e, \bar{\lambda}(\bar{q})) \tag{23}$$

$$= \frac{P(\bar{b}'_{\bar{o}^e,\bar{\lambda}(\bar{q})}, x^{e'}_{\bar{o}^e,\bar{\lambda}(\bar{q})}, k|\bar{b}, x^e, \bar{\lambda}(\bar{q}))}{\sum_k P(\bar{b}'_{\bar{o}^e,\bar{\lambda}(\bar{q})}, x^{e'}_{\bar{o}^e,\bar{\lambda}(\bar{q})}, k|\bar{b}, x^e, \bar{\lambda}(\bar{q})} \tag{24}$$

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

In Eq. (22), we have 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. (46) with the deterministic update 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).

Generating Parameters for FSAs: Now that evaluating a set of controllers is defined, we must 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 (Madani et al. 1999), but for the (primitive action) Dec-POMDP case, we can represent a policy within any ϵ of the optimal value with finite-sized controllers (Bernstein et al. 2009).

Here, we extend these ideas to the macro-action 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.

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. Also, $\mathcal{F} = \{\mathcal{F}^{(1)}, \ldots, \mathcal{F}^{(n)}\}$, and . 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 (though 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.

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 $\mathcal{F}^+_{k+1}$;

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.

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 $\hat{\mathcal{F}}^{(i)}$;

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

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:

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

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

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 $\bar{\phi}^{new}$;

$|Q^{(i)}|$ nodes in the controller, there will be $|\mathbb{T}^{(i)}||Q^{(i)}|^{|\mathbb{O}^{(i)}|}$ 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 Bernstein et al. (2009).

7. Solving Dec-POSMDP using Masked Monte Carlo Search

Each policy $\bar{\phi}$ is a discrete controller, represented by a graph. Each node of the discrete controller corresponds to a MA. Note that different nodes in the controller could use the same MA. Each edge in this graph is a $\breve{o}$. An example discrete controller for a package delivery domain is illustrated in Fig. 7.

The above transformation from continuous Dec-POMDP to Dec-POSMDP enables discrete domain algorithms to be used. However, for realistic problems with many MAs, even the Dec-POSMDP may be intractable to solve. 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 Section 9, MMCS allows solving problems with extremely large search spaces which would otherwise be intractable if solved through exhaustive search. 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 and an optimal solution is sought, an alternative hierarchically optimal search algorithm called G-DICE is detailed in Section 8.

Because the infinite-horizon problem is undecidable (Madani et al. 1999), infinite-horizon methods typically focus on producing approximate solutions given a computational budget (MacDermed and Isbell 2013, Amato et al. 2010). A Monte

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** $\bar{\phi}_{MMCS}$;

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.

The MMCS algorithm detailed in Alg. 5. MMCS uses a discrete controller 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 completing a TMA π_k, seeing an observation $\breve{o}_k$, and following the appropriate edge to the next TMA, π_{k+1}. Fig. 7 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 Amato and Zilberstein (2009) as follows,

$$P[\bar{V}^{\bar{\phi}^*}(\bar{b}) - \bar{V}^{\bar{\phi}}(\bar{b}) \geq \epsilon] \leq \delta. \tag{25}$$

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).

8. Solving Dec-POSMDP using Graph-based Direct Cross Entropy

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 (Oliehoek et al. 2008). 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 approach uses importance sampling to solve the optimization problem,

$$x^* = \arg\max_x V(x) \tag{26}$$

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

$$\bar{\phi}^* = \arg\max_{\bar{\phi}} \bar{V}^{\bar{\phi}}(\bar{b}). \tag{27}$$

To solve Eq. (26), the CE method performs sample-and-update on a probability distribution $f(x;\theta)$ over the decision variable x, where θ is a parameter vector. Using an initial parameter vector θ_0, samples are generated from $f(x;\theta_0)$ and are used to evaluate $V(x)$. The N_b best samples are used to generate a maximum likelihood estimate of the parameter vector at the next timestep, θ_1. Finally, smoothing is applied to deter convergence of the parameters to a local minima,

$$\theta_{k+1} = \alpha\theta_{k+1} + (1-\alpha)\theta_k \tag{28}$$

where $\alpha \in [0,1]$ is the learning rate. This process is repeated until convergence of the value function, $V(x)$.

We extend the CE method to the notion of FSAs introduced in Section 6, resulting in a new algorithm called Graph-based Direct Cross Entropy (G-DICE), which is outlined in Alg. 6. We do so by sampling the TMA output function $\lambda^{(i)}(q^{(i)})$ from distribution $f(\pi^{(i)}|q^{(i)};\theta_k^{(i)(\pi|q)})$, where parameter vector $\theta_k^{(i)(\pi|q)}$ is estimated using the CE method. Similarly, the transition function $\delta^{(i)}(q^{(i)},\breve{o}^{(i)})$ is sampled from $f(q^{(i)'}|q^{(i)},\breve{o}^{(i)};\theta_k^{(i)(q'|q,\breve{o})})$, where $\theta_k^{(i)(q'|q,\breve{o})}$ is a second parameter vector to be estimated. Although probabilistic sampling is used for generating the FSA output and transition functions, the functions themselves are deterministic in nature, thereby resulting in 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 $|\bar{\breve{\mathbb{O}}}|$ 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, $\bar{V}_b$, is initialized to $-\infty$ (Alg. 6, Line 5), and the parameter vectors are initialized (either using a priori knowledge from previous runs, or more typically sampled from a uniform distribution). 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 13) and are evaluated using Eq. (22). The best-joint-policy-so-far, $\bar{\phi}_b$, is retained (Alg. 6, Line 18). 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 for a maximum likelihood update of the parameter vectors (Alg. 6, Line 21 and Line 23). The parameter vectors are then smoothed (Alg. 6, Line 22 and Line 24) 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 $\bar{\phi}_b \approx \bar{\phi}^*$.

G-DICE is an anytime algorithm offering several advantages to the tree-based approach utilized previously for Dec-POMDPs (Oliehoek et al. 2008). First, the use of graphs allows solving of infinite-horizon Dec-POSMDPs, since loops in the policy graph allow it to be executed indefinitely. Second, 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 (Seuken 2007).

9. 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 (Agha-mohammadi et al. 2014b, Banker 2013). 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.

9.1. Application Domain: Constrained Package Delivery

Though our method is general and can be used for many decentralized planning problems, 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. 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. 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 (quadcopters) and ground robots (trucks). Air robots handle pickup of packages from bases, and can also deliver packages to two delivery locations, $Dest_1$ 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.

Algorithm 6: G-DICE

Procedure : G-DICE($\bar{\mathbb{T}}, \bar{\bar{\mathbb{O}}}, \mathbb{I}, N_n, N_k, N_s, N_b, \alpha$)
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
output : best joint policy $\bar{\phi}_b$
Initialize graph with N_n nodes and $|\bar{\bar{\mathbb{O}}}|$ edges per node;
$\bar{V}_b \leftarrow -\infty$;
for $i = 1$ *to* $|\mathbb{I}|$ **do**
  Initialize $\theta_0^{(i)(\pi|q)} \ \forall q$;
  Initialize $\theta_0^{(i)(q'|q,\breve{o})} \ \forall q, \breve{o}$;
for $k = 0$ *to* N_k **do**
  $\bar{\phi}_{list} \leftarrow \varnothing$;
  for $s = 1$ *to* N_s **do**
    for $i = 1$ *to* $|\mathbb{I}|$ **do**
      $\phi^{(i)} \leftarrow$ sample π from $f(\pi|q; \theta_k^{(i)(\pi|q)}) \ \forall q$ and sample q' from $f(q'|q, \breve{o}; \theta_k^{(i)(q'|q,\breve{o})}) \ \forall q, \breve{o}$;
    $\bar{V}^{\bar{\phi}} \leftarrow$Evaluate($\bar{\phi}$);
    $\bar{\phi}_{list} \leftarrow \bar{\phi}_{list} \cup \bar{\phi}$;
    if $\bar{V}^{\bar{\phi}} > \bar{V}_b$ **then**
      $\bar{V}_b \leftarrow \bar{V}^{\bar{\phi}}$;
      $\bar{\phi}_b \leftarrow \bar{\phi}$;
  $\bar{\phi}_{b,list} \leftarrow$ best N_b policies in $\bar{\phi}_{list}$;
  for $i = 1$ *to* $|\mathbb{I}|$ **do**
    $\theta_{k+1}^{(i)(\pi|q)} \leftarrow$ maximum likelihood estimate of $\theta^{(i)(\pi|q)}$ using $\bar{\phi}_{b,list} \ \forall q$;
    $\theta_{k+1}^{(i)(\pi|q)} \leftarrow \alpha\theta_{k+1}^{(i)(\pi|q)} + (1-\alpha)\theta_k^{(i)(\pi|q)}$;
    $\theta_{k+1}^{(i)(q'|q,\breve{o})} \leftarrow$ maximum likelihood estimate of $\theta^{(i)(q'|q,\breve{o})}$ using $\bar{\phi}_{b,list} \ \forall q, \breve{o}$;
    $\theta_{k+1}^{(i)(q'|q,\breve{o})} \leftarrow \alpha\theta_{k+1}^{(i)(q'|q,\breve{o})} + (1-\alpha)\theta_k^{(i)(q'|q,\breve{o})}$;
return $\bar{\phi}_b$;

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.

A robot at any base location can, therefore, observe the 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 the 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.

We assume one ground robot and two air robots are used, 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.

9.2. 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 $i \in \{i_{a,1}, i_{a,2}\}$.

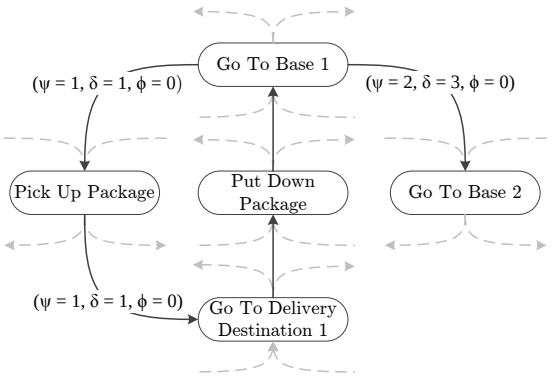


Fig. 7. Partial view of a single robot's policy obtained using MMCS for the package delivery domain. In this discrete policy controller, nodes represent TMAs and edges represent e-states. Greyed out edges represent connections to additional nodes which have been excluded to simplify the figure.

- *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 Section 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 Section 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. 10(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 of the TMA. This information can then be directly used by a policy search algorithm to perform evaluation of policies that use this macro-action. Given a goal location, the 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. 8 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. 9 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. 9(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. 10(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.

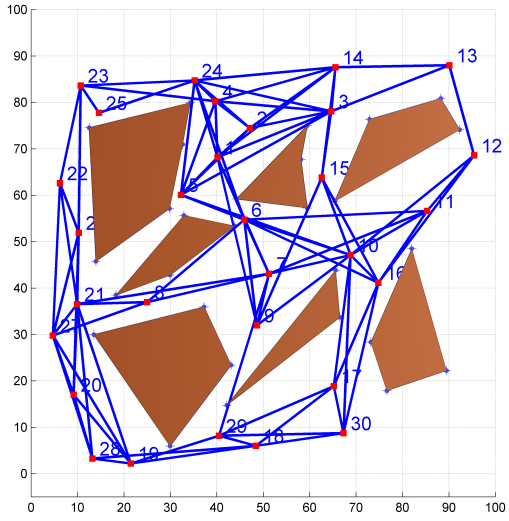
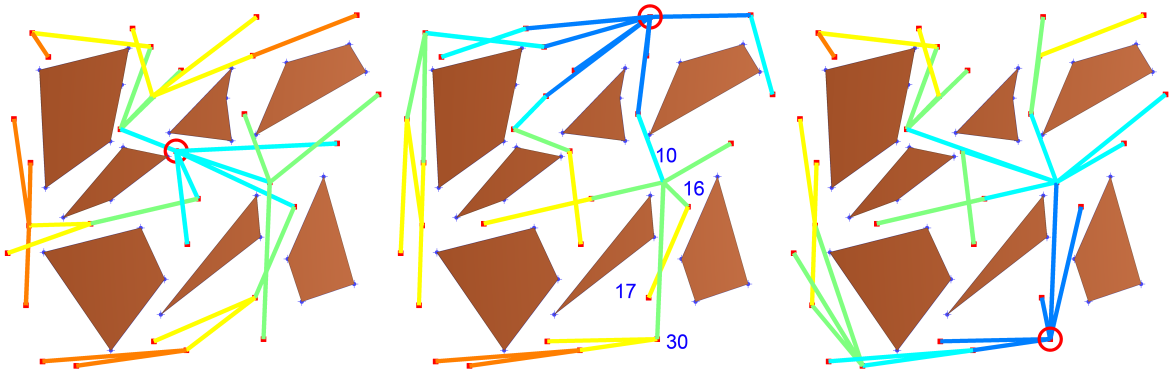


Fig. 8. A graph of LMAs over 30 nodes (red). Connecting edges are indicated in blue, and obstacles in brown.



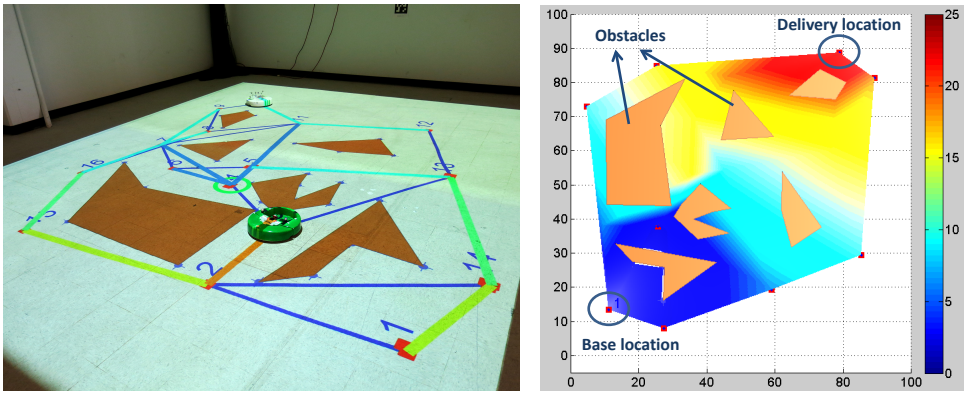
(a) (b) (c)

Fig. 9. 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.

9.3. Closed-Loop Policy Controller Results - MMCS

A portion of a policy for a single air robot in the package delivery domain is illustrated in Fig. 7. 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. 7 (for all possible TMAs) as well as edges (for all possible environment observations).

Fig. 11 compares a uniform random Monte Carlo search to MMCS in the package delivery domain. Results for the Monte Carlo search were generated by repeatedly sampling random, valid policies and retaining the policy with the highest



(a) t (b)

Fig. 10. On the left, the cost-to-go for an example domain is shown, projected on robots in a lab environment. On the right, we show the value of "Go to $Dest_1$" TMA over its belief space (only the 2D mean part is shown).

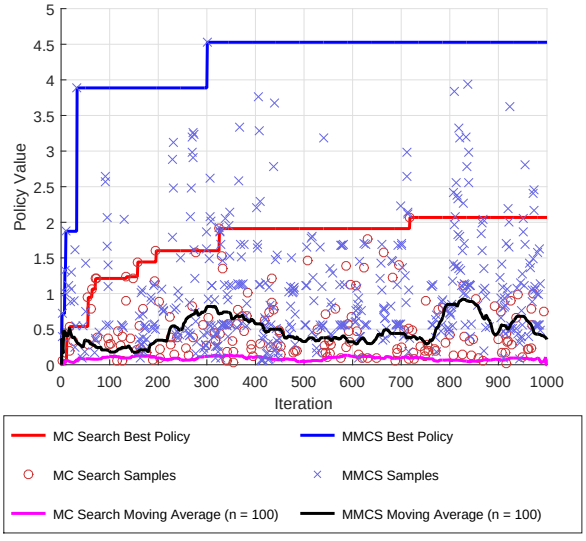
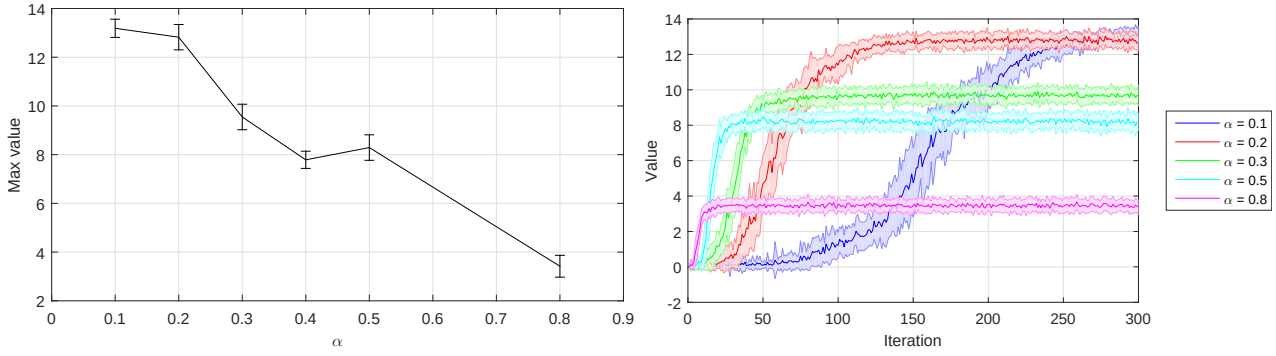


Fig. 11. Policy differences between MC and MMCS. Moving average of policy values (over 100 samples) are also indicated.

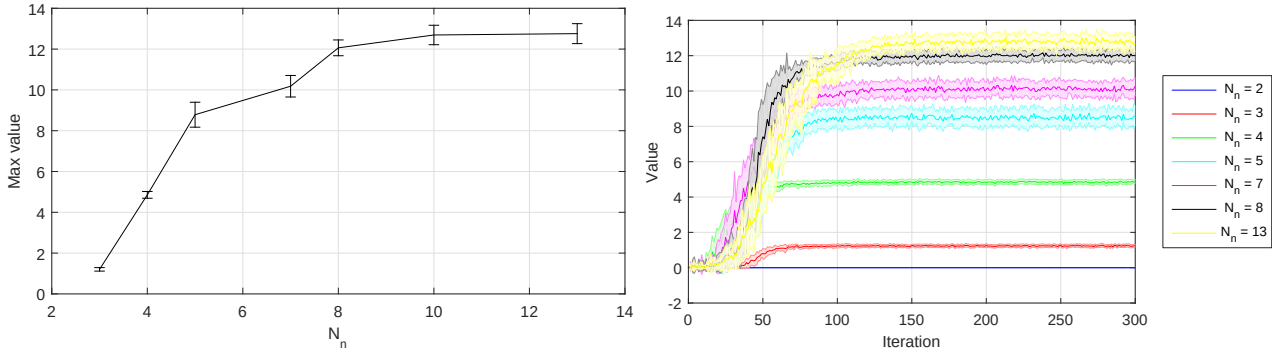
expected value. Both approaches used a policy controller with a fixed number of nodes ($n_{nodes} = 13$). Results indicate that given a fixed computational budget (quantified by the number of search iterations), MMCS outperforms standard Monte Carlo search in terms of expected value. Specifically, as seen in Table 14(b), 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. 11 illustrates MMCS's ability to exploit previous policy samples to bias towards well-performing regions of the policy space, while using random sampling for further exploration of the space.

Fig. 11 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.



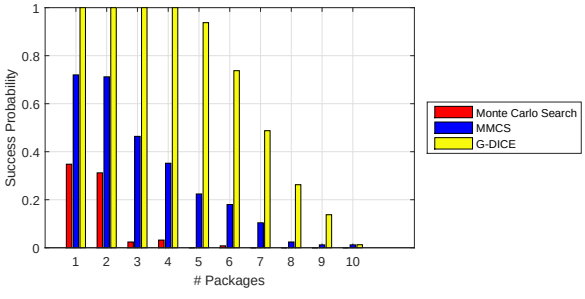
(a) (b)

Fig. 12. On the left, the max joint value obtained for the package delivery domain using G-DICE, for varying learning rates, α. Note that this is the MEAN of the max value, not the absolute max (which is what table II provides). On the right, the effect of learning rate on policy value convergence is shown.



(a) (b)

Fig. 13. On the left, the max joint value obtained for the package delivery domain using G-DICE, for varying learning policy controller sizes, N_n. On the right, the effect of varying policy controller size on policy value convergence is shown.



(a)

Algorithm	Policy Value	Samples
Monte Carlo Search	2.068	1000
MMCS	4.528	300
G-DICE	14.443	3000
Exhaustive Search	—	5.622e+17

(b)

Fig. 14. The left plot shows success probability of delivering the specified number of packages *or more* within a fixed time horizon. The table shows a comparison of the maximum values obtained by the search algorithms.

9.4. Closed-Loop Policy Controller Results - G-DICE

MMCS is a greedy algorithm designed for providing 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 a probabilistically optimal algorithm which outperforms MMCS in policy quality. As seen in Table 14(b), the value of the package delivery policy generated by G-DICE (14.443) is more than 3 times the

MMCS policy value (4.528). G-DICE policy values with varying learning rate are shown in Fig. 12(a). Higher learning rate 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. 12(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 research efforts have focused on developing adaptive learning rates or removing them altogether for algorithms such as stochastic gradient descent (Schaul et al. 2013). 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, however, motivates future work in the area of developing adaptive learning rates for G-DICE.

Fig. 13(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. 13(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. The behaviors exhibited here 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. 13(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 policy 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 more intuitively quantify performance of G-DICE, MMCS, and Monte Carlo policies in the package delivery domain, Fig. 14(a) 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, 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.

9.5. Hardware Experiments: Constrained Package Delivery with Health Awareness

We extend the simulated package delivery domain to a hardware domain, with some modifications (Fig. 15). 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) (Omidshafiei et al. 2015), 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. 15 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 quadcopters (circled in red) performing package delivery with no communication between them. The quadcopters 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 large enough processing time constraint. Fig. 16(a) illustrates a blue package generated at the base, corresponding to (blue) delivery destination 3 (shown in Fig. 15). 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. 16(b) illustrates a quadcopter which has just visited a base and picked up a package. The package is shown as a square beneath the quad 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. 16(b) also shows the planned trajectory of the quad (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. 18 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 quadcopter (Fig. 17(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. 17(a), the quadcopter has 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 2. 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 2: 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 the healing station (shown in Fig. 15) 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 obtained later). Joint pickup and delivery as well as ground robot TMAs have been removed (although solo TMAs remain), although 2 additional air robots have been added for a total of 4, in contrast to the simulation domain which only had 2.

Fig. 19(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. 19(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. 19(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 2, the robot’s health belief can sometimes shift back towards higher health due to increased sensing noise when damaged. Fig. 17(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 continuous 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.

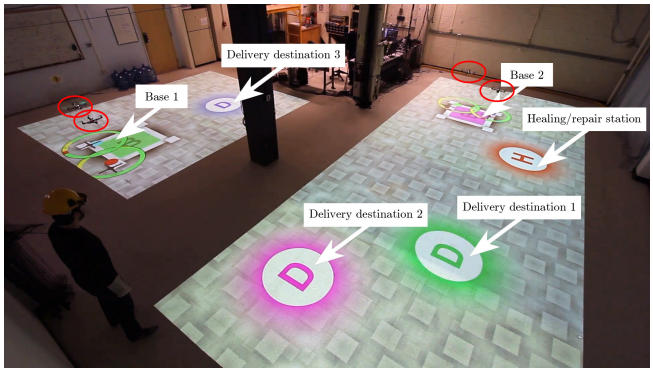
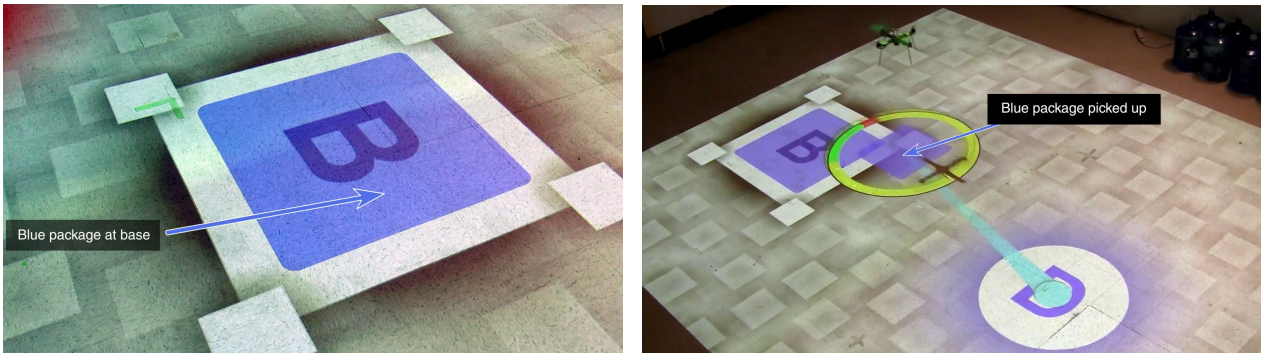


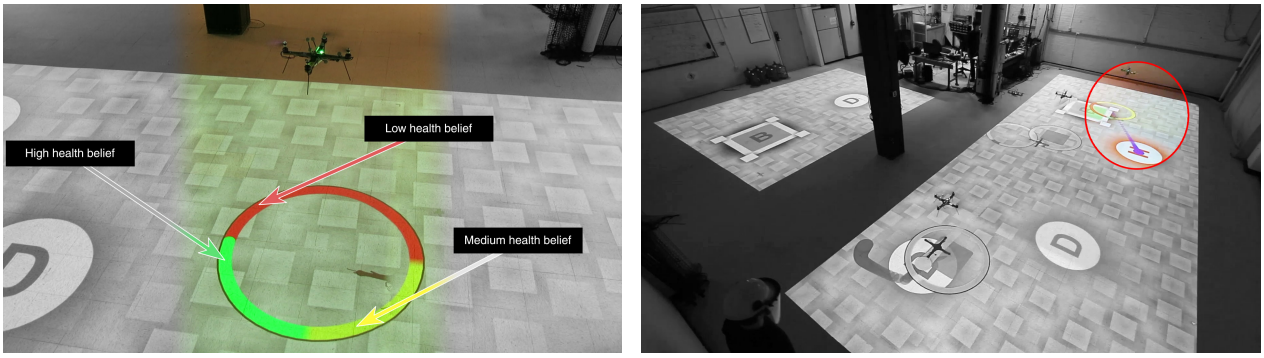
Fig. 15. Overview of constrained package delivery hardware domain.



(a) (b)

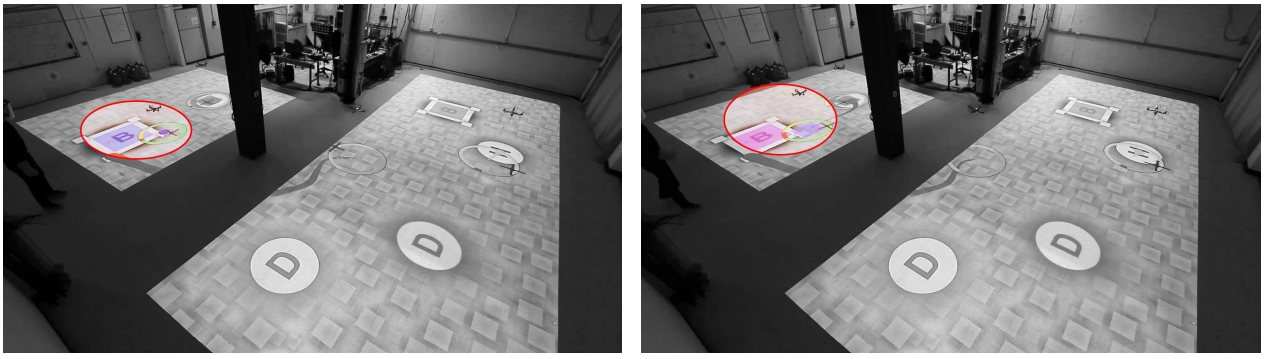
Fig. 16. On the left, visualization of blue package generated at base is shown. On the right, a quadcopter is carrying a blue package to the correct delivery destination.

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.



(a) (b)

Fig. 17. Health belief visualization is done using a “belief ring" (shown on the left), indicating the probability of the robot being in each health state based on noisy observations of its health. On the right, we see a quadcopter going to the repair station due to low health belief.



(a) (b)

Fig. 18. Visualization of quadcopter package pickup. On the left, the base has a blue package as the quadcopter approaches. On the right, the quadcopter has picked up the blue package (now shown underneath it), and the base displays the next package in the queue (pink).

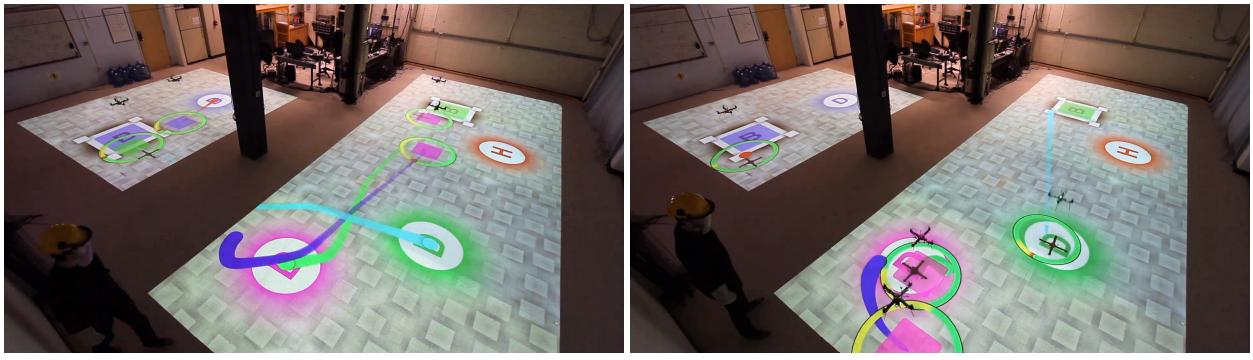
10. Conclusion

This paper proposed a layered framework to exploit the macro-action abstraction in decentralized planning for a group of robots acting 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 two algorithm, MMCS and G-DICE, for solving Dec-POSMDPs and compared their performance on a multi-robot package delivery problem under uncertainty, with a health-aware hardware experiment also implemented.

The Dec-POSMDP represents a general 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. Future work include formal analysis of the G-DICE algorithm, investigation of adaptive learning rates for G-DICE, and investigation of node pruning to yield compact policy graphs without requiring user-selection of the graph size.

Acknowledgment

This work supported by ONR MURI grant N000141110688 and Boeing Research & Technology.



(a) (b)

Fig. 19. On the left, the mission starts and quadcopters plan paths to their appropriate delivery destinations. On the right, quadcopters are completing package delivery and deciding which base to return to.

References

Ali-akbar Agha-mohammadi, Suman Chakravorty, and Nancy Amato. FIRM: Sampling-based feedback motion planning under motion uncertainty and imperfect measurements. *International Journal of Robotics Research (IJRR)*, 33(2):268–304, 2014a.

Ali-akbar Agha-mohammadi, N. Kemal Ure, Jonathan P. How, and John Vian. Health aware stochastic planning for persistent package delivery missions using quadrotors. In *International Conference on Intelligent Robots and Systems (IROS)*, Chicago, September 2014b.

Christopher Amato and Shlomo Zilberstein. Achieving goals in decentralized POMDPs. In *International Conference on Autonomous Agents and Multiagent Systems*, pages 593–600, 2009.

Christopher Amato, Daniel S. Bernstein, and Shlomo Zilberstein. Optimizing fixed-size stochastic controllers for POMDPs and decentralized POMDPs. *Journal of Autonomous Agents and Multi-Agent Systems*, 21(3):293–320, 2010.

Christopher Amato, George D. Konidaris, and Leslie P. Kaelbling. Planning with macro-actions in decentralized POMDPs. In *International Conference on Autonomous Agents and Multiagent Systems*, 2014.

Christopher Amato, George D. Konidaris, Ariel Anders, Gabriel Cruz, Jonathan P. How, and Leslie P. Kaelbling. Policy search for multi-robot coordination under uncertainty. In *Proceedings of Robotics: Science and Systems (RSS)*, 2015a.

Christopher Amato, George D. Konidaris, Gabriel Cruz, Christopher A. Maynor, Jonathan P. How, and Leslie P. Kaelbling. Planning for decentralized control of multiple robots under uncertainty. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2015b.

Steve Banker. Amazon and drones – here is why it will work, December 2013. URL `http://www.forbes.com/sites/stevebanker/2013/12/19/amazon-drones-here-is-why-it-will-work/`.

Daniel S. Bernstein, Shlomo Zilberstein, Richard Washington, and John L. Bresina. Planetary rover control as a Markov decision process. In *Proceedings of the International Symposium on Artificial Intelligence, Robotics and Automation in Space*, 2001.

Daniel S. Bernstein, Robert Givan, Neil Immerman, and Shlomo Zilberstein. The complexity of decentralized control of Markov decision processes. *Mathematics of Operations Research*, 27(4):819–840, 2002.

Daniel S. Bernstein, Christopher Amato, Eric A. Hansen, and Shlomo Zilberstein. Policy iteration for decentralized control of Markov decision processes. *Journal of Artificial Intelligence Research*, 34:89–132, 2009.

Rosemary Emery-Montemerlo, Geoff Gordon, Jeff Schneider, and Sebastian Thrun. Game theoretic control for robot teams. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1163–1169, 2005.

Ruijie He, Emma Brunskill, and Nicholas Roy. Efficient planning under uncertainty with macro-actions. *Journal of Artificial Intelligence Research*, 40:523–570, February 2011.

L. P. Kaelbling, M. L. Littman, and A. R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101:99–134, 1998.

Jens Kober, J. Andrew Bagnell, and Jan Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238 – 1274, September 2013. ISSN 1741-3176. .

P. R. Kumar and P. P. Varaiya. *Stochastic Systems: Estimation, Identification, and Adaptive Control*. Prentice-Hall, Englewood Cliffs, NJ, 1986.

Zhan Lim, Lee Sun, and Daniel J. Hsu. Monte carlo value iteration with macro-actions. In J. Shawe-Taylor, R.S. Zemel, P.L. Bartlett, F. Pereira, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 24*, pages 1287–1295. Curran Associates, Inc., 2011. URL `http://papers.nips.cc/paper/4477-monte-carlo-value-iteration-with-macro-actions.pdf`.

Liam C MacDermed and Charles Isbell. Point based value iteration with optimal belief compression for Dec-POMDPs. In *Advances in Neural Information Processing Systems*, pages 100–108, 2013.

Omid Madani, Steve Hanks, and Anne Condon. On the undecidability of probabilistic planning and infinite-horizon partially observable Markov decision problems. In *Proceedings of the Sixteen Conference on Artificial Intelligence (AAAI)*, pages 541–548, 1999.

I. Kroo N. Nigam, S. Bieniawski and John Vian. Control of multiple uavs for persistent surveillance: Algorithm and flight test results. 20(5):1236–1251, September 2012.

Frans A. Oliehoek, Julian F.P. Kooi, and Nikos Vlassis. The cross-entropy method for policy search in decentralized POMDPs. *Informatica*, 32:341–357, 2008.

Shayegan Omidshafiei, Ali akbar Agha-mohammadi, Christopher Amato, and Jonathan P. How. Technical report: Decentralized control of partially observable markov decision processes using belief space macro-actions. Technical report, Department of Aeronautics and Astronautics, Massachusetts Institute of Technology, September 2014. URL `http://acl.mit.edu/papers/ShayO-DecPOSMDP.pdf`.

Shayegan Omidshafiei, Ali akbar Agha-mohammad, Yu Fan Chen, N. Kemal Ure, Jonathan How, John Vian, and Rajeev Surati. MAR-CPS: Measurable Augmented Reality for Prototyping Cyber-Physical Systems. 2015.

Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley-Interscience, 1994.

Tom Schaul, Sixin Zhang, and Yann LeCun. No More Pesky Learning Rates. In *International Conference on Machine Learning (ICML)*, 2013.

Sven Seuken. Memory-bounded dynamic programming for dec-pomdps. In *In Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI*, pages 2009–2015, 2007.

Richard S Sutton, Doina Precup, and Satinder Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1):181–211, 1999.

Georgios Theocharous and Leslie Pack Kaelbling. Approximate planning in POMDPs with macro-actions. In *Advances in Neural Information Processing Systems 16 (NIPS03)*, 2004. URL `http://people.csail.mit.edu/lpk/papers/theochar-nips03.pdf`.

Nazim Kemal Ure, Girish Chowdhary, Jonathan P How, and John Vian. *Planning Under Uncertainty*, chapter Multi-Agent Planning for Persistent Surveillance. MIT Press, 2013.

Keith Winstein and Hari Balakrishnan. TCP ex Machina: Computer-Generated Congestion Control. In *SIGCOMM*, 2013.

11. Appendix

11.1. 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

$$\bar{V}^{\bar{\phi}}(\bar{b}_0, x_0^e) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \bar{R}(\bar{x}_t, x_t^e, \bar{u}_t) | \bar{b}_0, x_0^e; \bar{\phi}\right] \tag{29}$$

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

$$= \mathbb{E}\left[\sum_{k=0}^{\infty} \sum_{t=t_k}^{t_{k+1}-1} \gamma^t \bar{R}(\bar{x}_t, x_t^e, \bar{u}_t) | \bar{b}_0, x_0^e; \bar{\phi}\right] \tag{30}$$

$$= \sum_{k=0}^{\infty} \mathbb{E}\left[\sum_{t=t_k}^{t_{k+1}-1} \gamma^t \bar{R}(\bar{x}_t, x_t^e, \bar{u}_t) | \bar{b}_0, x_0^e; \bar{\phi}\right] \tag{31}$$

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

$$= \sum_{k=0}^{\infty} \sum_{\bar{b}_{t_k}, x_{t_k}^e} \mathbb{E}\left[\sum_{t=t_k}^{t_{k+1}-1} \gamma^t \bar{R}(\bar{x}_t, x_t^e, \bar{u}_t) | \bar{b}_{t_k}, x_{t_k}^e; \bar{\phi}\right] P(\bar{b}_{t_k}, x_{t_k}^e | \bar{b}_0, x_0^e; \bar{\phi}). \tag{32}$$

Now, making a change of time indices,

$$= \sum_{k=0}^{\infty} \sum_{\bar{b}_{t_k}, x_{t_k}^e} \gamma^{t_k} \mathbb{E}\left[\sum_{t=0}^{t_{k+1}-1-t_k} \gamma^t \bar{R}(\bar{x}_t, x_t^e, \bar{u}_t) | \bar{b}_{t_k}, x_{t_k}^e; \bar{\phi}\right] P(\bar{b}_{t_k}, x_{t_k}^e | \bar{b}_0, x_0^e; \bar{\phi}). \tag{33}$$

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

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

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

$$= \sum_{k=0}^{\infty} \sum_{\bar{b}_{t_k}, x_{t_k}^e} \gamma^{t_k} \bar{R}^{\tau}(\bar{b}_{t_k}, x_{t_k}^e, \bar{\pi}_{t_k}) P(\bar{b}_{t_k}, x_{t_k}^e | \bar{b}_0, x_0^e; \bar{\phi}) \tag{35}$$

$$= \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^{t_k} \bar{R}^{\tau}(\bar{b}_{t_k}, x_{t_k}^e, \bar{\pi}_{t_k}) | \bar{b}_0, x_0^e; \bar{\phi}\right]. \tag{36}$$

11.2. 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.

$$P(\bar{b}', x^{e'}, k|\bar{b}, x^e; \bar{\pi}) = P(x_k^{e'}, \bar{b}'_k|x_0^e, \bar{b}_0; \pi) \tag{37}$$

$$= \sum_{x_{k-1}^e, \bar{b}_{k-1}} P(x_k^{e'}, \bar{b}'_k|x_{k-1}^e, \bar{b}_{k-1}, x_0^e, \bar{b}_0; \bar{\pi}) P(x_{k-1}^e, \bar{b}_{k-1}|x_0^e, \bar{b}_0; \bar{\pi}) \tag{38}$$

$$= \sum_{x_{k-1}^e, \bar{b}_{k-1}} P(x_k^{e'}, \bar{b}'_k|x_{k-1}^e, \bar{b}_{k-1}; \pi) P(x_{k-1}^e, \bar{b}_{k-1}|x_0^e, \bar{b}_0; \bar{\pi}) \tag{39}$$

$$= \sum_{x_{k-1}^e, \bar{b}_{k-1}} P(x_k^{e'}, \bar{b}'_k|x_{k-1}^e, \bar{b}_{k-1}; \pi(\bar{b}_{k-1})) P(x_{k-1}^e, \bar{b}_{k-1}|x_0^e, \bar{b}_0; \bar{\pi}) \tag{40}$$

$$= \sum_{x_{k-1}^e, \bar{b}_{k-1}} P(x_k^{e'}|\bar{b}'_k, x_{k-1}^e, \bar{b}_{k-1}; \pi(\bar{b}_{k-1})) P(\bar{b}'_k|x_{k-1}^e, \bar{b}_{k-1}; \pi(\bar{b}_{k-1})) P(x_{k-1}^e, \bar{b}_{k-1}|x_0^e, \bar{b}_0; \bar{\pi}) \tag{41}$$

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

11.3. FSA Evaluation

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

$$\bar{V}^{\bar{\phi}}(\bar{b}, x^e, \bar{q}) = \bar{R}^{\tau}(\bar{b}, x^e; \bar{\phi}) + \sum_{k=1}^{\infty} \gamma^{t_k} \sum_{\bar{b}', x^{e'}} P(\bar{b}', x^{e'}, k | \bar{b}, x^e; \bar{\phi}) \bar{V}^{\bar{\phi}}(\bar{b}', x^{e'}, \bar{q}')) \tag{43}$$

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 $\bar{\lambda}(\bar{q})$,

$$= \bar{R}^{\tau}(\bar{b}, x^e, \bar{\lambda}(\bar{q}); \bar{\phi}) + \sum_{k=1}^{\infty} \gamma^{t_k} \sum_{\bar{b}', x^{e'}} P(\bar{b}', x^{e'}, k | \bar{b}, x^e, \bar{\lambda}(\bar{q}); \bar{\phi}) \bar{V}^{\bar{\phi}}(\bar{b}', x^{e'}, \bar{q}')). \tag{44}$$

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

$$= \bar{R}^{\tau}(\bar{b}, x^e, \bar{\lambda}(\bar{q}); \bar{\phi}) + \sum_{k=1}^{\infty} \gamma^{t_k} \sum_{\bar{b}', x^{e'}, \bar{o}^e} P(\bar{b}', x^{e'}, k, \bar{o}^e | \bar{b}, x^e, \bar{\lambda}(\bar{q}); \bar{\phi}) \bar{V}^{\bar{\phi}}(\bar{b}', x^{e'}, \bar{\delta}(\bar{q}, \bar{o}^e)) \tag{45}$$

$$= \bar{R}^{\tau}(\bar{b}, x^e, \bar{\lambda}(\bar{q}); \bar{\phi}) + \sum_{k=1}^{\infty} \gamma^{t_k} \sum_{\bar{o}^e} P(\bar{o}^e | \bar{b}, x^e, \bar{\lambda}(\bar{q}); \bar{\phi}) \sum_{\bar{b}', x^{e'}} P(\bar{b}', x^{e'}, k | \bar{b}, x^e, \bar{\lambda}(\bar{q}), \bar{o}^e) \bar{V}^{\bar{\phi}}(\bar{b}', x^{e'}, \bar{\delta}(\bar{q}, \bar{o}^e)) \tag{46}$$

$$= \bar{R}^{\tau}(\bar{b}, x^e, \bar{\lambda}(\bar{q}); \bar{\phi}) + \sum_{k=1}^{\infty} \gamma^{t_k} \sum_{\bar{o}^e} P(\bar{o}^e | \bar{b}, x^e, \bar{\lambda}(\bar{q}); \bar{\phi}) P(k | \bar{b}, x^e, \bar{\lambda}(\bar{q}), \bar{o}^e; \bar{\phi}) \bar{V}^{\bar{\phi}}(\bar{b}'_{\bar{o}^e, \bar{\lambda}(\bar{q})}, x^{e'}_{\bar{o}^e, \bar{\lambda}(\bar{q})}, \bar{\delta}(\bar{q}, \bar{o}^e)). \tag{47}$$