Journal of Artificial Intelligence Research 64 (2019) 817-859 Submitted 07/18; published 03/19

Modeling and Planning with Macro-Actions in Decentralized POMDPs

Christopher Amato CAMATO@CCS.NEU.EDU
Khoury College of Computer Sciences, Northeastern University
Boston, MA 02115 USA

George Konidaris GDK@CS.BROWN.EDU
Department of Computer Science, Brown University
Providence, RI 02912 USA

Leslie P. Kaelbling LPK@CSAIL.MIT.EDU
MIT Computer Science and Artificial Intelligence Laboratory
Cambridge, MA 02139 USA

Jonathan P. How JHOW@MIT.EDU
MIT Laboratory for Information and Decision Systems
Cambridge, MA 02139 USA

Abstract

Decentralized partially observable Markov decision processes (Dec-POMDPs) are general models for decentralized multi-agent decision making under uncertainty. However, they typically model a problem at a low level of granularity, where each agent's actions are primitive operations lasting exactly one time step. We address the case where each agent has macro-actions: temporally extended actions that may require different amounts of time to execute. We model macro-actions as *options* in a Dec-POMDP, focusing on actions that depend only on information directly available to the agent during execution. Therefore, we model systems where coordination decisions only occur at the level of deciding which macro-actions to execute. The core technical difficulty in this setting is that the options chosen by each agent no longer terminate at the same time. We extend three leading Dec-POMDP algorithms for policy generation to the macro-action case, and demonstrate their effectiveness in both standard benchmarks and a multi-robot coordination problem. The results show that our new algorithms retain agent coordination while allowing high-quality solutions to be generated for significantly longer horizons and larger state-spaces than previous Dec-POMDP methods. Furthermore, in the multi-robot domain, we show that, in contrast to most existing methods that are specialized to a particular problem class, our approach can synthesize control policies that exploit opportunities for coordination while balancing uncertainty, sensor information, and information about other agents.

1. Introduction

The Dec-POMDP (Bernstein, Givan, Immerman, & Zilberstein, 2002; Oliehoek & Amato, 2016) is a general framework for decentralized sequential decision-making under uncertainty and partial observability. Dec-POMDPs model problems where a team of agents shares the same objective function, but where each individual agent can only make noisy, partial observations of the environment. Solution methods for Dec-POMDPs aim to produce policies that optimize reward while considering uncertainty in action outcomes, sensors, and information about the other agents.

Although much research has been conducted on solution methods for Dec-POMDPs, solving large instances remains intractable. Advances have been made in optimal algorithms (see, for example, Amato, Chowdhary, Geramifard, Ure, & Kochenderfer, 2013; Amato, Dibangoye, & Zilberstein, 2009; Aras, Dutech, & Charpillet, 2007; Boularias & Chaib-draa, 2008; Dibangoye, Amato, Buffet, & Charpillet, 2013; Oliehoek, Spaan, Amato, & Whiteson, 2013; Dibangoye, Amato, Buffet, & Charpillet, 2016), but most approaches that scale well make very strong assumptions about the domain (e.g., assuming a large amount of independence between agents) (Dibangoye, Amato, Doniec, & Charpillet, 2013; Melo & Veloso, 2011; Nair, Varakantham, Tambe, & Yokoo, 2005) and/or have no guarantees about solution quality (Oliehoek, Whiteson, & Spaan, 2013; Seuken & Zilberstein, 2007b; Velagapudi, Varakantham, Sycara, & Scerri, 2011). One reason for this intractability is that actions are modeled as primitive (low-level) operations that last exactly one time step. The length of a single step can be adjusted (trading off solution quality for horizon length), but is always assumed to be the same for all agents. This allows synchronized action selection, but also requires reasoning about action selection and coordination at every time step.

In single-agent (i.e., MDP) domains, hierarchical approaches to learning and planning (Barto & Mahadevan, 2003), exemplified by the *options framework* (Sutton, Precup, & Singh, 1999), have explored using higher-level, temporally extended macro-actions (or *options*) to represent and solve problems, leading to significant performance improvements in planning (Silver & Ciosek, 2012; Sutton et al., 1999). We now extend these ideas to the multi-agent case by introducing a Dec-POMDP formulation with macro-actions modeled as options. The primary technical challenge here is that decision-making is no longer synchronized: each agent's options must be selected, and may complete, at different times. To permit coordination, agents must use their knowledge of option policies to reason about the progress of other agents and their impact on the world.

The use of macro-actions in the multi-agent case can incorporate the benefits of the single agent case, such as simpler and more efficient modeling of real systems (e.g., robots with actions that execute predefined controllers) (Stone, Sutton, & Kuhlmann, 2005), more efficient planning (Sutton et al., 1999), skill transfer (Konidaris & Barto, 2007), and skill-specific abstractions (Konidaris & Barto, 2009; Dietterich, 2000). Additional benefits can be gained by exploiting known structure in the multi-agent problem. For instance, in some cases macro-actions may only depend on locally observable information. One example is a robot navigating to a waypoint in a security patrol application. Only local information is required for navigation, but choosing which waypoint to navigate to next requires reasoning about the location and state of all the other robots. Macro-actions with independent execution allow coordination decisions to be made only when necessary (i.e., when choosing macro-actions) rather than at every time step. Furthermore, macro-actions can build on other macro-actions, allowing hierarchical planning. The resulting macro-action formulation allows asynchronous decision-making using actions with varying time durations.

We therefore focus on the case where the agents are given *local options* that depend only on information locally observable to the agent during execution. Our results show that high-quality solutions can be found for a typical Dec-POMDP benchmark as well as large problems that traditional Dec-POMDP methods cannot solve: a four agent meeting-in-a-grid problem and a domain based on robots navigating among movable obstacles (Stilman & Kuffner, 2005). Our macro-action-based methods can scale well in terms of the problem horizon and domain variables, but do not directly target scalability in terms of the number of agents (although such extensions are possible in the future). Incorporating macro-actions into Dec-POMDPs results in a scalable algorithmic framework for generating solutions for a wide range of probabilistic multi-agent systems.

One important application area for our approach is multi-robot systems. For single robots, automatic planning systems provide a flexible general-purpose strategy for constructing plans given high-level declarative domain specifications, even in the presence of substantial stochasticity and partial observability (Thrun, Burgard, & Fox, 2005). By incorporating macro-actions into Dec-POMDPs, we show that this strategy can be effectively extended to multi-robot systems: our methods naturally bridge Dec-POMDPs and multi-robot coordination, allowing principled decentralized methods to be applied to real domains. To solidify this bridge, we describe a process for creating a multi-robot macro-action Dec-POMDP (MacDec-POMDP) model, solving it, and using the solution to produce a set of executable SMACH (Bohren, 2010) finite-state machine task controllers. Our methods allow automatic off-line construction of robust multi-robot policies that support coordinated actions—including generating communication strategies that exploit the environment to share information critical to achieving the group's overall objective.

2. Background

We now describe the Dec-POMDP and options frameworks, upon which our work is based.

2.1 Decentralized Partially-Observable Markov Decision Processes

Dec-POMDPs (Bernstein et al., 2002) generalize POMDPs[1] (Kaelbling, Littman, & Cassandra, 1998) and MDPs[2] (Puterman, 1994) to the multi-agent, decentralized setting. As depicted in Figure 1, Dec-POMDPs model a team of agents that must cooperate to solve some task by receiving local observations and individually selecting and executing actions over a sequence of time steps. The agents share a single reward function that specifies their objective, but which is not typically observed during execution. Execution is decentralized because each agent must select its own action at each time step, without knowledge of the actions chosen or observations received by the other agents. Finally, the problem is partially observable because, while the formalism assumes that there exists a Markovian state at each time step, the agents do not have access to it. Instead, each agent receives a separate observation at each time step, which reflects its own partial and local view of the world.

More formally, a Dec-POMDP is defined by a tuple $\langle I, S, \{A_i\}, T, R, \{\Omega_i\}, O, h\rangle$, where:

- I is a finite set of agents.
- S is a finite set of states with designated initial state distribution b_0.
- A_i is a finite set of actions for each agent i with $A = \times_i A_i$ the set of joint actions.

[1] POMDPs are Dec-POMDPs where there is only one agent or the decision-making by the agents is centralized.

[2] MDPs are POMDPs where the state is fully observable.

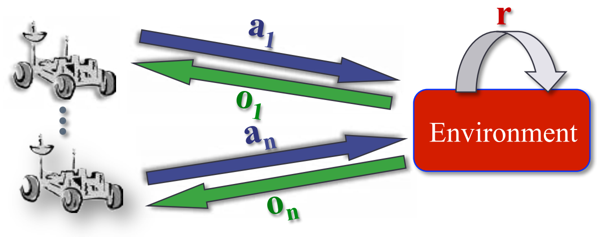


Figure 1: An n-agent Dec-POMDP. Each agent i receives observatoons o_i and executes actions a_i; all agents receive a single collective reward r.

- T is a state transition probability function, $T : S \times A \times S \rightarrow [0, 1]$, that specifies the probability of transitioning from state $s \in S$ to $s' \in S$ when actions $\vec{a} \in A$ are taken by the agents. Hence, $T(s, \vec{a}, s') = \Pr(s'|\vec{a}, s)$.
- R is a reward function: $R : S \times A \rightarrow \mathbb{R}$, giving the immediate reward for being in state $s \in S$ and taking actions $\vec{a} \in A$.
- Ω_i is a finite set of observations for each agent, i, with $\Omega = \times_i \Omega_i$ the set of joint observations.
- O is an observation probability function: $O : \Omega \times A \times S \rightarrow [0, 1]$, the probability of the agents receiving observations $\vec{o} \in \Omega$ given actions $\vec{a} \in A$ were taken which results in state $s' \in S$. Hence $O(\vec{o}, \vec{a}, s') = \Pr(\vec{o}|\vec{a}, s')$.
- h is the number of steps until the problem terminates, called the horizon.

Note that while the actions and observations are factored with one factor per agent, the state—which represents the state of the whole system—need not be.

The solution to a Dec-POMDP is a *joint policy*—a set of policies, one for each agent. In an MDP, a solution policy is represented directly as a mapping from states to actions. In partially observed settings, the agents do not have access to the state, and so must represent policies some other way. In POMDP settings it is typically possible to calculate the belief state—a probability distribution over the unobserved state—and represent the agent's policy as a mapping from belief state to actions. However, this is not possible in the Dec-POMDP setting, because each agent would need access to the histories of all the other agents to calculate a (centralized) belief state. We therefore represent the history of each agent explicitly: the action-observation history for agent i, $h_i^A = (a_i^0, o_i^0, \ldots, a_i^t, o_i^t)$, represents the actions taken and observations received at each step (up to step t); the set of such histories for agent i is H_i^A. Each agent's policies are then a function of the agent's history, and are either represented as a policy tree, where the vertices indicate actions to execute and the edges indicate transitions conditioned on an observation, or as a finite state controller which executes in a similar manner. An example of each is given in Figure 2.

The value of a joint policy, π, from state s is

$$V^\pi(s) = \mathrm{E}\left[\sum_{t=0}^{h-1} \gamma^t R(\vec{a}^t, s^t)|s, \pi\right],$$

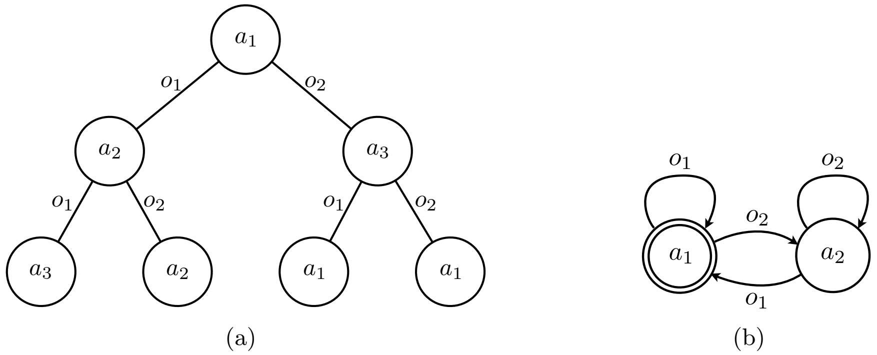


Figure 2: A single agent's policy represented as (a) a policy tree and (b) a finite-state controller with initial state shown with a double circle.

which represents the expected value of the immediate reward for the set of agents summed for each step of the problem given the action prescribed by the policy until the horizon is reached. In the finite-horizon case (which we consider in this paper), the discount factor, γ, is typically set to 1. An *optimal policy* beginning at state s is $\pi^*(s) = \text{argmax}_\pi V^\pi(s)$. The goal is to maximize the total cumulative reward, beginning at some initial distribution over states b_0.

Dec-POMDPs have been widely studied and there are number of significant advances in algorithms (e.g., see recent surveys Amato et al., 2013; Oliehoek, 2012; Oliehoek & Amato, 2016). Unfortunately, optimal (and boundedly optimal) methods (Amato et al., 2009; Bernstein, Amato, Hansen, & Zilberstein, 2009; Aras et al., 2007; Boularias & Chaib-draa, 2008; Dibangoye et al., 2013; Oliehoek et al., 2013) do not scale to large problems and approximate methods (Oliehoek et al., 2013; Seuken & Zilberstein, 2007b; Velagapudi et al., 2011; Wu, Zilberstein, & Chen, 2010a; Wu, Zilberstein, & Jennings, 2013) do not scale or perform poorly as problem size (including horizon) grows. Subclasses of the full Dec-POMDP model have been explored, but they make strong assumptions about the domain (e.g., assuming a large amount of independence between agents) (Dibangoye et al., 2013; Melo & Veloso, 2011; Nair et al., 2005). The key question is then: how can scalability with respect to horizon and domain variables be achieved while making minimal (and accurate) assumptions about the problems being solved?

Our solution to this question is the use of hierarchy in Dec-POMDPs. While many hierarchical approaches have been developed for multi-agent systems (e.g., Horling & Lesser, 2004), very few are applicable to multi-agent models based on MDPs and POMDPs. In this paper, the hierarchy will take the form of options replacing each agent's primitive actions. The result is a general framework for asynchronous decision making operating at multiple levels of granularity that fits many real-world problems. As a result, we target scalability with respect to the problem horizon an domain variables (actions, observations and states), but leave scalability with respect to the number of agents to future work (e.g., by combining the methods from this paper with those that scale in terms of the number of agents).

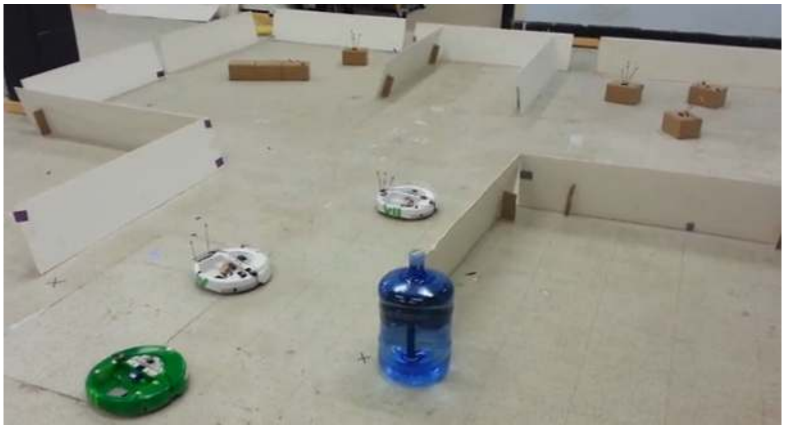


Figure 3: A multi-robot warehouse domain.

2.2 Multi-Robot Domains

Our work is motivated by multi-robot coordination domains. Consider the multi-robot warehousing problem (shown in Figure 3) that we present in the experiments. A team of robots is tasked with finding a set of large and small boxes in the environment and returning them to a shipping location. Large boxes require multiple robots to push. As a result, coordination is necessary not just for assigning robots to push specific boxes, but also because two robots are required to cooperate to push the larger box at the same time. There is stochasticity in the movements of robots and partial observability with respect to the location of the boxes and other robots: both can be only be detected when they are within range. We also consider cases where the robots can send communication signals to each other, but we do not define the meaning of the messages. Therefore, our planner must determine where the robots should navigate, what boxes they should push and what communication messages should be sent (if any) at each step of the problem to optimize the solution for the team. The robots must make these decisions based solely on the information they individually receive during execution (e.g., each robot's estimate of its own location as well as where and when boxes and other robots have been seen).

This multi-robot warehousing problem can be formalized as a Dec-POMDP.[3] In fact, any problem where multiple robots share a single overall reward or cost function can be formulated as a Dec-POMDP. Therefore, a Dec-POMDP solver could potentially automatically generate control policies (including policies over when and what to communicate) for very rich decentralized control problems, in the presence of uncertainty. Unfortunately, this generality comes at a cost: as mentioned above, Dec-POMDPs are typically infeasible to solve except for very small problems (Bernstein et al., 2002). By contrast, we will show that by considering macro-actions, we retain the ability to coordinate while allowing high-quality solutions to be generated for significantly larger problems than would have been possible using other Dec-POMDP-based methods. In this example, macro-actions could

[3] In fact, there is a common Dec-POMDP benchmark that can be thought of as a simple version of a warehouse problem (Seuken & Zilberstein, 2007a).

be navigating to a small or large box, pushing a box (alone or with another robot) to a destination, or communicating with another robot.

Macro-actions are a natural model for the modular controllers often sequenced to obtain robot behavior. The macro-action approach leverages expert-designed or learned controllers for solving subproblems (e.g., navigating to a waypoint or grasping an object), bridging the gap between traditional robotics research and work on Dec-POMDPs. This approach has the potential to produce high-quality general solutions for real-world heterogeneous multi-robot coordination problems by automatically generating control and communication policies.

2.3 The Options Framework

The options framework (Sutton et al., 1999) provides methods for learning and planning using high-level actions, or options, in Markov decision processes. In that setting, an option is defined by a tuple:

$$m = (\beta_m, \mathcal{I}_m, \pi_m),$$

consisting of a stochastic *termination condition*, $\beta_m : S \to [0,1]$, which determines the probability with which an option ceases to execute in each state; an *initiation set*, $\mathcal{I}_m \subset S$, which determines whether or not an option can be executed from a state; and a stochastic *option policy*, $\pi_m : S \times A \to [0,1]$, that maps states to action execution probabilities. An option describes a policy that an agent can choose to execute from any state in $\mathcal{I}_m$, which results in the execution of policy π_m until execution ceases according to β_m. The set of options is termed M.

For example, in the warehouse example above, an option-based macro-action may be navigating to a waypoint. In that case, the initiation set may be all states (it is available anywhere), the option policy may be a policy that navigates the robot to the waypoint location from any location and the termination condition may be the state that represents the waypoint location or a set of states within a given radius of the waypoint. There may also be terminal states for failure to reach the waypoint (e.g., states representing the robot getting stuck).

The resulting problem is known as a *Semi-Markov Decision Process*, or SMDP (Sutton et al., 1999). Note that we can create an option for a single-step action a by defining $\pi_m(s,a) = \beta_m(s) = 1, \forall s$, and $\mathcal{I}_m = S$. The option framework therefore generalizes the traditional MDP setting.

The goal is to generate a (possibly stochastic) policy, $\mu : S \times M \to [0,1]$, that selects an appropriate option given the current state. The Bellman equation for the SMDP is:

$$V^{\mu}(s) = \sum_{m} \mu(s,m) \left[R(s,m) + \sum_{s'} p(s'|s,m) V^{\mu}(s') \right],$$

where $p(s'|s,m) = \sum_{k=0}^{\infty} p_s^m(s',k)\gamma^k$ with $p_s^m(s',k)$ representing the probability that option m will terminate in state s' from state s after k steps and $R(s,m)$ is an expectation over discounted rewards until termination $\mathrm{E}[r^t + \gamma r^{t+1} + \ldots + \gamma^{k-1} r^{t+k}]$ (for executing option m starting at time t and terminating at time $t+k$). If the option-policies and the lower-level MDP is known, these quantities can be calculated from the underlying models. If these quantities are not known, learning can be used to generate a solution.

When the state is partially observable, these ideas can be directly transferred to the POMDP case. This can be done by representing the POMDP as a belief-state MDP. That is, given a current belief state, b, and a policy of option-based macro-actions, μ, the value can be calculated as:

$$V^{\mu}(b) = \sum_{m} \mu(b,m) \left[R(b,m) + \int_{b'} p(b'|b,m) V^{\mu}(b') \right],$$

where $\mu(b,m)$ now selects a policy based on the belief state, $R(b,m) = \sum_s b(s)R(s,m)$ and $p(b'|b,m) = \sum_{k=0}^{\infty} p_b^m(b',k)\gamma^k$ with $p_b^m(b',k)$ representing the probability that option m will terminate in belief state b' from belief b after k steps. Several POMDP methods have been developed that use option-based macro-actions (Theocharous & Kaelbling, 2003; He, Brunskill, & Roy, 2011; Lim, Sun, & Hsu, 2011).

Using either of these approaches directly is not possible in a decentralized multi-agent setting. First, the centralized information (a state or belief state) that prior approaches use for high-level action selection is not present during execution in the Dec-POMDP setting. Consequently, the action selection function, μ, must be reformulated for the decentralized case. Second, in the multi-agent case the inclusion of temporally extended actions means that action selection is no longer synchronized across agents—some agents' options would terminate while others are still executing. Therefore, it is not clear when macro-actions should be considered complete (i.e., up to which point rewards and transitions should be calculated), which complicates the definition of the reward and transition functions, R and p. We now introduce a framework that addresses these issues, thereby enabling the use of options in the Dec-POMDP setting.

3. Adding Options to Dec-POMDPs

We extend the Dec-POMDP model by replacing the local actions available to each agent with option-based macro-actions. Specifically, the action set of each agent i, which is denoted A_i above, is replaced with a finite set of options M_i. Then, $M = \times_i M_i$ the set of joint options, replacing A, the joint set of actions. We focus on *local options* for each agent i, each of which is defined by a tuple:

$$m_i = (\beta_{m_i}, \mathcal{I}_{m_i}, \pi_{m_i}),$$

where $\beta_{m_i} : H_i^A \rightarrow [0,1]$ is a stochastic termination condition; $\mathcal{I}_{m_i} \subset H_i^M$ is the initiation set; and $\pi_{m_i} : H_i^A \times A_i \rightarrow [0,1]$ is the option policy. Note that H_i^A is agent i's *primitive* action-observation history, while H_i^M is agent i's *macro-action-macro-observation* history (or *option history*, which is formally defined below). The different histories allow the agents to locally maintain the necessary information to know how to execute and terminate macro-actions (based on low-level actions and observations, typically beginning when an option is first executed) and initiate them (based on high-level history information that is maintained over a longer timeframe). Such local options model systems where the execution of a particular option, once selected, does not require coordination between agents, but can instead be completed by the agent on its own. Decision making that enables coordination between agents need only happen at the level of *which option to execute*, rather than inside the options themselves. Of course, other (non-local) forms of options that control and depend on multiple agents are possible, but we discuss the local form due to its simplicity and generality.

The macro-actions for the warehouse problem are discussed in 6.2.1, but, in short, macro-actions can be defined for navigation, pushing and communication. For example, there are macro-actions for navigating to each room that could contain boxes. For these macro-actions, the initiation set is all observations (they are available everywhere), the policy navigates the robot to the specified room using low-level observation information that is available to that robot (using low-level observation histories) and the termination condition consists of observations that are only possible inside the desired room (localization information within the room).

3.1 The MacDec-POMDP Model

We will refer to Dec-POMDPs with such macro-actions as *MacDec-POMDPs*. In the MacDec-POMDP, the agent and state spaces remain the same as in the Dec-POMDP definition, but macro-actions and macro-observations are added. Formally, a MacDec-POMDP is a tuple $\langle I, S, \{M_i\}, \{A_i\}, T, R, \{Z_i\}, \{\Omega_i\}, \zeta_i, O, h\rangle$, where:

- I, S, $\{A_i\}$, T, R, $\{\Omega_i\}$, O and h are the same as the Dec-POMDP definition (and represent the 'underlying' Dec-POMDP),
- M_i is a finite set of macro-actions for each agent i with $M = \times_i M_i$ the set of joint macro-actions,
- ζ_i is a finite set of macro-observations for each agent, i, with $\zeta = \times_i \zeta_i$ the set of joint macro-observations,
- Z_i is a macro-observation probability function for agent i: $Z_i : \zeta_i \times M_i \times S \to [0, 1]$, the probability of the agent receiving macro-observation $z_i \in \zeta_i$ given macro-action $m_i \in M_i$ has completed and the current state is $s' \in S$. Hence $Z_i(z_i, m_i, s') = \Pr(z_i|m_i, s')$.

Note that the macro-observations are assumed to be independently generated for each agent after that agent's macro-action has completed. This is reasonable since macro-action completion is asynchronous (making it uncommon that multiple macro-actions terminate at the same time) and are generated based on the underlying state (which could include information about the other agents).

In the MacDec-POMDP, we will not attempt to directly represent the transition and reward functions, but instead infer them by using the underlying Dec-POMDP model or a simulator[4]. That is, because we assume either a model or a simulator of the underlying Dec-POMDP is known, we can evaluate policies using macro-actions in the underlying Dec-POMDP by either knowing that underlying Dec-POMDP model or having a simulator that implements such a model. This evaluation using the Dec-POMDP model or simulator can be thought of as 'unrolling' each agent's macro-action and when any macro-action completes, selecting an appropriate next macro-action for that agent. As a result, a formal representation of higher-level transition and reward models is not necessary.

[4] In related work based on the ideas in this paper, we do generate such an explicit model that considers time until completion for any macro-action, resulting in the semi-Markovian Dec-POSMDP (Omidshafiei, Agha-mohammadi, Amato, & How, 2017).

3.2 Designing Macro-Observations

In the MacDec-POMDP, macro-observations are assumed to be given or designed. Determining the set of macro-actions that provides the necessary information, without unnecessarily adding problem variables remains an open question (as it is in the primitive observation case). In general, the high-level macro-observations can consist of any finite set for each agent, but some natural representations exist. For instance, the macro-observation may just be the particular terminal condition that was reached (e.g., the robot entered office #442). A lot of information is lost in this case, so macro-observations can also be action-observation histories, representing all the low-level information that took place during macro-action execution. When action-observation histories are used, initiation conditions of macro-actions can depend on the histories of macro-actions already taken and their results. Option policies and termination conditions will generally depend on histories that begin when the macro-action is first executed (action-observation histories). While defining the 'best' set of macro-observations is an open problem, there is some work on choosing them and learning the macro-observation probability functions (Omidshafiei, Liu, Everett, Lopez, Amato, Liu, How, & Vian., 2017a). In this paper, we assume they are defined based on the underlying state (as defined above). The macro-observation probability function can be adapted to depend on terminal conditions or local observations rather than states.

3.3 MacDec-POMDP Solutions

Solutions to MacDec-POMDPs map from option histories to macro-actions. An *option history*, which includes the sequence of macro-observations seen and macro-actions selected, is defined as $h_i^M = (z_i^0, m_i^1, \ldots, z_i^{t-1}, m_i^t)$. Here, z_i^0 may be a null macro-observation or an initial macro-observation produced from the initial belief state b_0. Note that while histories over primitive actions provide the number of steps that have been executed (because they include actions and observations at each step), an option history typically requires many more (primitive) steps to execute than the number of macro-actions listed.

We can then define policies for each agent, μ_i, for choosing macro-actions that depend on option histories. A (stochastic) local policy, $\mu_i : H_i^M \times M_i \to [0,1]$ then depends on these option histories and a joint policy for all agents is written as μ. The evaluation of such policies is more complicated than the Dec-POMDP case because decision-making is no longer synchronized. In cases when a model of macro-action execution (e.g., the option policy) and the underlying Dec-POMDP are available we can evaluate the high-level policies in a similar way to other Dec-POMDP-based approaches. Given a joint policy, the primitive action at each step is determined by the (high-level) policy, which chooses the macro-action, and the macro-action policy, which chooses the (primitive) action. This 'unrolling' uses the underlying Dec-POMDP to generate (primitive) transitions and rewards, but determines what actions to take from the macro-actions. The joint high-level and macro-action policies can then be evaluated as:

$$V^{\mu}(s) = \mathrm{E}\left[\sum_{t=0}^{h-1} \gamma^t R(\vec{a}^t, s^t) | s, \pi, \mu\right].$$

When the underlying Dec-POMDP and the macro-action policies are not available, we can use a simulator or a high-level model to execute the policies and return samples of the relevant values. Simulation is very similar to model-based evaluation, but uses Monte Carlo estimation as discussed in Section 5.

For example, we can evaluate a joint 2-agent policy μ which begins with macro-actions m_1 and m_2 at state s and executes for t steps as:

$$\begin{aligned}
V_t^{\mu}(m_1, m_2, s) = \sum_{o_1,o_2} O(o_1, o_2, a_1, a_2, s) \sum_{a_1,a_2} \pi_{m_1}(o_1, a_1)\pi_{m_2}(o_2, a_2) \Bigg[& R(a_1, a_2, s) + \\
& \sum_{s'} T(s', a_1, a_2, s) \sum_{o'_1,o'_2} O(o'_1, o'_2, a_1, a_2, s') \\
\Bigg(\beta_{m_1}(o'_1)\beta_{m_2}(o'_2) \sum_{m'_1,m'_2} \mu_1(o'_1, m'_1)\mu_2(o'_2, m'_2)V_{t-1}^{\mu}(s', m'_1, m'_2) & \qquad \text{(both terminate)} \\
+ \beta_{m_1}(o'_1)\big(1 - \beta_{m_2}(o'_2)\big) \sum_{m'_1} \mu_1(o'_1, m'_1)V_{t-1}^{\mu}(m'_1, m_2, s') & \qquad \text{(agent 1 terminates)} \\
+ \big(1 - \beta_{m_1}(o'_1)\big)\beta_{m_2}(o'_2) \sum_{m'_2} \mu_2(o'_2, m'_2)V_{t-1}^{\mu}(m_1, m'_2, s') & \qquad \text{(agent 2 terminates)} \\
+ \big(1 - \beta_{m_1}(o'_1)\big)\big(1 - \beta_{m_2}(o'_2)\big)V_{t-1}^{\mu}(m_1, m_2, s') \Bigg) \Bigg], & \qquad \text{(neither terminates)}
\end{aligned}$$

where single observations are used instead of longer histories for macro-action policies, π, and termination conditions, β. For simplicity, we also use observations based on the current state, $O(o_1, o_2, a_1, a_2, s)$, rather than the next state. The example can easily be extended to consider histories and the other observation function (as well as more agents). Also, note that macro-actions will be chosen from the policy over macro-actions μ based on the option history, which is not shown explicitly (after termination of a macro-action, a high-level macro-observation will be generated and the next specified macro-action will be chosen as described above). Note that agents' macro-actions may terminate at different times; the appropriate action is then chosen by the relevant agent's policy and evaluation continues. Because we are interested in a finite-horizon problem, we assume evaluation continues for h (primitive) steps.

Given that we can evaluate policies over macro-actions, we can then compare these policies. We can define a *hierarchically optimal policy* $\mu^*(s) = \text{argmax}_{\mu} V^{\mu}(s)$ which defines the highest-valued policy among those that use the given MacDec-POMDP. Because a hierarchically optimal policy may not include all possible history-dependent policies, it may have lower value than the optimal policy for the underlying Dec-POMDP (the *globally optimal policy*).[5]

A globally optimal policy can be guaranteed by including the primitive actions in the set of macro-actions for each agent and mapping the primitive observation function to the macro-observation function, because the same set of policies can be created from this primitive macro-action set as would be created in the underlying Dec-POMDP. However,

[5] Unlike flat Dec-POMDPs, stochastic policies may be beneficial in the macro-action case because full agent histories are no longer used. This remains an area of future work.

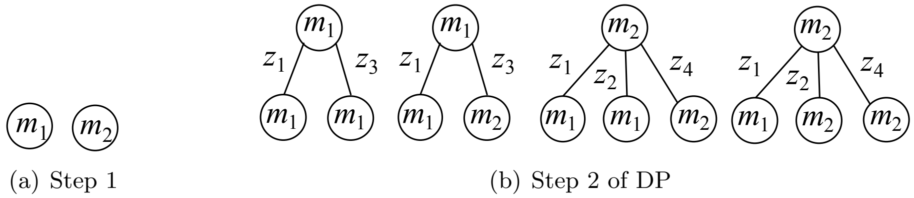


Figure 4: Policies for a single agent after (a) one step and (b) two steps of dynamic programming using macro-actions m_1 and m_2 and macro-observations z (some of which are not possible after executing a particular macro-action).

this typically makes little sense, because it is at least as hard as planning in the underlying Dec-POMDP directly.

4. Algorithms

Because Dec-POMDP algorithms produce policies mapping agent histories to actions, they can be extended to consider macro-actions instead of primitive actions by adjusting policy evaluation and keeping track of macro-action progress and termination. We discuss how macro-actions can be incorporated into three such algorithms; extensions can also be made to other approaches.

In these cases, deterministic polices are generated which are represented as policy trees (as shown in Figure 4). A policy tree for each agent defines a policy that can be executed based on local information. The root node defines the macro-action to choose in the known initial state, and macro-actions are specified for each legal macro-observation of the root macro-action (as seen in Figure 4(b)). In the figure, macro-observations that are not shown are not possible after the given macro-action has completed. Execution continues until the primitive horizon h is reached, meaning some nodes of the tree may not be reached due to the differing execution times of some macro-actions. Such a tree can be evaluated up to a desired horizon using the policy evaluation given above (i.e., evaluation using the underlying Dec-POMDP model or simulator). All the methods we discuss use some form of search through the policy space to generate high-quality macro-action-based policies.

4.1 Dynamic Programming

A simple exhaustive search method can be used to generate hierarchically optimal deterministic policies which use macro-actions. This algorithm is similar in concept to the dynamic programming algorithm used in Dec-POMDPs (Hansen, Bernstein, & Zilberstein, 2004), but full evaluation and pruning (removing dominated policies) are not used at each step (since these cannot naturally take place in the macro-action setting). Instead we can exploit the structure of macro-actions to reduce the space of policies considered. Due to the inspiration from dynamic programming for finite-horizon Dec-POMDPs (Hansen et al., 2004), we retain the name for the algorithm, but our algorithm is not a true dynamic programming

Algorithm 1 Option-based dynamic programming (O-DP)

```
function OptionDecDP($h$)
    $t \leftarrow 0$
    $PrimitiveHorizonBelowh \leftarrow true$
    $\mathcal{M}_t \leftarrow \emptyset$
    repeat
        $\mathcal{M}_{t+1} \leftarrow$ExhaustiveBackup($\mathcal{M}_t$)
        $PrimitiveHorizonBelowh \leftarrow$TestPolicySetsLength($\mathcal{M}_{t+1}$)
        $t \leftarrow t + 1$
    until $PrimitiveHorizonBelowh = false$
    return $\mathcal{M}_t$
end function
```

algorithm as a full evaluation is not conducted and built on at every step (as discussed below).

We can exhaustively generate all combinations of macro-actions by first considering each agent using any single macro-actions to solve the problem, as seen for one agent with two macro-actions (m_1 and m_2) in Figure 4(a). We can test all combinations of these 1-macro-action policies for the set of agents to see if they are guaranteed to reach (primitive) horizon h (starting from the initial state). If any combination of policies does not reach h with probability 1, we will not have a valid policy for all steps. Therefore, an exhaustive backup is performed by considering starting from all possible macro-actions and then for any legal macro-observation of the macro-action (represented as z in the figure), transitioning to one of the 1-macro-action policies from the previous step (see Figure 4(b)). This step creates all possible next (macro-action) step policies. We can check again to see if any of the current set of policies will terminate before the desired horizon and continue to grow the policies (exhaustively as described above) as necessary. When all policies are sufficiently long, all combinations of these policies can be evaluated as above (by flattening out the polices into primitive action Dec-POMDP policies, starting from some initial state and proceeding until h). The combination with the highest value at the initial state, s_0, is chosen as the (hierarchically optimal) policy.

Pseudocode for this approach is given in Algorithm 1. Here, $\mathcal{M}_t$ represents the set of (joint) macro-action policies generated for t (macro-action) steps. ExhaustiveBackup performs the generation of all possible next-step policies for each agent and TestPolicySetsLength checks to see if all policies reach the given horizon, h. $PrimitiveHorizonBelowh$ represents whether there is any tree that has a primitive horizon less than h. The algorithm continues until all policies reach h and the final set of policies $\mathcal{M}_t$ can be returned for evaluation.

This algorithm will produce a hierarchically optimal deterministic policy because it constructs all legal deterministic macro-action policies that are guaranteed to reach horizon h. This follows from the fact that macro-actions must last at least one step and all combinations of macro-actions are generated at each step until it can be guaranteed that additional backups will cause redundant policies to be generated. Our approach represents exhaustive search in the space of legal policies that reach a desired horizon. As such it is not a true dynamic programming algorithm, but additional ideas from dynamic programming for Dec-POMDPs (Hansen et al., 2004) can be incorporated. For instance, we could prune policies based on value, but this would require evaluating all possible joint policies at every state after each backup. This evaluation would be very costly as the policy would be flattened after each backup and all combinations of flat policies would be evaluated for all states for all possible reachable horizons. Instead, beyond just scaling in the horizon due to the macro-action length, another benefit of our approach is that only legal policies are generated using the initiation and terminal conditions for macro-actions. As seen in Figure 4(b), macro-action m_1 has two possible terminal states while macro-action m_2 has three. Furthermore, macro-actions are only applicable given certain initial conditions. For example, m_1 may not be applicable after observing z_4 and m_2 may not be applicable after z_1. This structure limits the branching factor of the policy trees produced and thus the number of trees considered.

4.2 Memory-Bounded Dynamic Programming

Memory-bounded dynamic programming (MBDP) (Seuken & Zilberstein, 2007b) can also be extended to use macro-actions as shown in Algorithm 2. MBDP is similar to the dynamic programming method above, but only a finite number of policy trees are retained (given by parameter *MaxTrees*) after each backup.

After an exhaustive backup has been performed (in either DP or MBDP), at most $|M_i| \times |\mathcal{M}_{i,t-1}|^{|\zeta_i|}$ new trees for each agent i given the previous policy set $\mathcal{M}_{i,t-1}$ is generated (although it will often be much less since many macro-actions may not be possible after a given macro-observation). The key addition in MBDP is that, next, a subset of t-step trees, $\hat{\mathcal{M}_t}$, is chosen by evaluating the full set of trees, $\mathcal{M}_t$, at states[6] that are generated by a heuristic policy (H_{pol} in the algorithm). The heuristic policy is executed for the first $h-t-1$ steps of the problem.[7] Heuristic policies can include centralized MDP or POMDP policies or random policies (or a combination of these), providing a set of possible states to consider at that depth. A set of $MaxTrees$ states is generated and the highest valued trees for each state are kept. This process of exhaustive backups and retaining $MaxTrees$ trees continues, using shorter and shorter heuristic policies until the all combinations of the retained trees reach horizon h. Again, the set of trees with the highest value at the initial state is returned.

This approach is potentially suboptiomal because a fixed number of trees are retained, and tree sets are optimized over states that are both assumed to be known and may never be reached. Nevertheless, since the number of policies retained at each step is bounded by $MaxTrees$, MBDP has time and space complexity linear in the horizon. As a result, MBDP and its extensions (Amato et al., 2009; Kumar & Zilberstein, 2010; Wu et al., 2010a) have been shown to perform well in many large Dec-POMDPs. The macro-action-based extension of MBDP uses the structure provided by the initiation and terminal conditions

[6] The original MBDP algorithm (Seuken & Zilberstein, 2007b) uses *beliefs* rather than states at lines 9 and 10 of the algorithm. Our algorithm could similarly use beliefs, but we discuss using states for simplicity.

[7] Note that h is a primitive (underlying Dec-POMDP) horizon, while t is a macro-action step. While backups will often result in increasing policy length by more than one primitive step, we conservatively use one step here, but recognize that more accurate calculations along with corresponding better state estimates are possible.

Algorithm 2 Option-based memory bounded dynamic programming (O-MBDP)

```
function OptionMBDP($MaxTrees$,$h$,$H_{pol}$)
    $t \leftarrow 0$
    $PrimitiveHorizonBelowh \leftarrow true$
    $\mathcal{M}_t \leftarrow \emptyset$
    repeat
        $\mathcal{M}_{t+1} \leftarrow$ExhaustiveBackup($\mathcal{M}_t$)
        $\hat{\mathcal{M}}_{t+1} \leftarrow \emptyset$
        for all $k \in MaxTrees$ do
            $s_k \leftarrow$ GenerateState($H_{pol}$,$h-t-1$)
            $\hat{\mu}_{t+1} \leftarrow \hat{\mathcal{M}}_{t+1} \cup \arg\max_{\mu_{t+1} \in \mathcal{M}_{t+1}} V^{\mu_{t+1}}(s_k)$
        end for
        $t \leftarrow t+1$
        $\mathcal{M}_t \leftarrow \hat{\mathcal{M}}_{t+1}$
        $PrimitiveHorizonBelowh \leftarrow$TestPolicySetsLength($\mathcal{M}_t$)
    until $PrimitiveHorizonBelowh = false$
    return $\mathcal{M}_t$
end function
```

as in the dynamic programming approach in Algorithm 1, but does not have to produce all policies that will reach horizon h as the algorithm no longer is seeking hierarchical optimality. Scalability can therefore be adjusted by reducing the $MaxTrees$ parameter (although solution quality may be reduced).

4.3 Direct Cross Entropy Policy Search

Another method for solving Dec-POMDPs that has been effective is a cross entropy method, called DICE (for DIrect Cross Entropy) (Oliehoek, Kooi, & Vlassis, 2008). Instead of using dynamic programming, this method searches through the space of policy trees by sampling. That is, it maintains sampling distributions (the probability of choosing an action) at each history of each agent. Policies are sampled based on these distributions and the resulting joint policies are evaluated. A fixed number of best-performing policies are retained and the sampling distributions are updated based on the action choice frequency of these policies (mixed with the current distributions). Policy sampling and distribution updates continue for a fixed number of iterations (or a convergence test such as one based on KL-divergence).

The macro-action version of DICE is described in Algorithm 3. The inputs are the number of iterations of the algorithm ($Iter$), the number of joint policies to sample at each iteration, N, the number of joint policies used for updating the sampling distributions, N_b, the learning rate, α, and the (primitive) horizon, h. The best value, V_{best}, is initialized to negative infinity and the sampling distributions are typically initialized to uniform action distributions.

In the macro-action case, sampling distributions that are based on option histories are used instead of primitive histories. Specifically, we maintain $\xi^{h_i^M}(m)$ for each option history h_i^M, of each agent, i, which represents the probability of selecting macro-action m after that agent observes history h_i^M. The algorithm then begins with an empty set

Algorithm 3 Option-based direct cross entropy policy search (O-DICE)

```
function OptionDICE($Iter$,$N$,$N_b$,$\alpha$,$h$)
    $V_{best} \leftarrow -\infty$
    $\xi \leftarrow$InitialDistribution
    for all $i \in Iter$ do
        $\mathcal{M} \leftarrow \emptyset$
        for $n \leftarrow 0$ to $N$ do
            $\mu \leftarrow$ Sample($\xi$)
            $\mathcal{M} \leftarrow \mathcal{M} \cup \{\mu\}$
            $V \leftarrow V^{\mu}(s_0)$
            if $V > V_{best}$ then
                $V_{best} \leftarrow V$
                $\mu_{best} \leftarrow \mu$
            end if
        end for
        $\mathcal{M}_{best} \leftarrow$KeepBestPols($\mathcal{M}, N_b$)
        $\xi_{new} \leftarrow$ Update($\xi$)
        $\xi_{new} \leftarrow \alpha\xi_{new} + (1-\alpha)\xi$
        $\xi \leftarrow \xi_{new}$
    end for
    return $\mu_{best}$
end function
```

of joint policies, $\mathcal{M}$, and samples N policies for each agent. Because macro-actions often have limited initial and terminal conditions, sampling is more complicated. It is done in a top down fashion from the first macro-action until the (primitive) horizon is reached, while taking into account the possible macro-observations after starting from the initial state and executing the policy to that point. This allows both the terminal conditions and initial sets to be used to create distributions over valid macro-actions based on the previous histories. These N policies for each agent are evaluated and the if a new best policy is found, the value and policy are stored in V_{best} and μ_{best}. The policies with the N_b highest values from the N are stored in $\mathcal{M}_{best}$ and ξ_{new} is updated for each agent's histories with $\xi_{new}^{h_i^M}(m) = 1/N_b \sum_{\mu \in \mathcal{M}_{best}} \mathbb{I}(\pi_i, h_i^M, m)$ where $\mathbb{I}(\mu_i, h_i^M, m)$ is an indicator variable that is 1 when macro-action m is taken by policy μ_i after history h_i^M. This ξ_{new} is mixed with the previous distribution, ξ, based on the learning rate, α, and the process continues until the number of iterations is exhausted. The best joint policy, μ_{best} can then be returned.

5. Simulation-Based Execution in MacDec-POMDPs

The MacDec-POMDP framework is a natural way to represent and generate behavior for realistic general problems such as multi-robot systems, but requiring full knowledge of both the high-level macro-action model and the low-level Dec-POMDP model is often impractical. To use the MacDec-POMDP model as described above, we would assume an abstract model of the system is given in the form of macro-action representations, which include the associated policies as well as initiation and terminal conditions. These macro-actions are controllers operating in (possibly) continuous time with continuous (low-level) actions and feedback, but their operation is discretized for use with the planner. This discretization represents an underlying discrete Dec-POMDP which consists of the primitive actions, states of the system, and the associated rewards. While the complexity of MacDec-POMDP solution methods primarily depends on the size of the MacDec-POMDP model, and not the size of the underlying Dec-POMDP (as only policies over macro-actions are needed with execution in the underlying Dec-POMDP being fixed), it is often difficult to generate and represent a full Dec-POMDP model for real-world systems.

We therefore extend this model to use a simulator rather than a full model of the problem, as shown in Figure 5. In many cases, a simulator already exists or is easier to construct than the full model. Our planner still assumes the set of macro-actions and macro-observations are known, but the policies of the macro-actions as well as the underlying Dec-POMDP are not explicitly known. Instead, we make the more realistic assumption that we can simulate the macro-actions in an environment similar to the real-world domain. As such, our proposed algorithms for generating policies over macro-actions remain the same (since constructing policies of macro-actions only requires knowledge of the set of macro-actions and their initiation and terminal conditions), but all evaluation is conducted in the simulator (through sampling) rather than through enumerating all reachable states to compute the Bellman equation. That is, by using policy search, we can decouple the process of finding solutions with the process of evaluating them. As a result, we assume the macro-action and macro-observation sets are discrete, but the underlying state, action and observation spaces can be continuous.

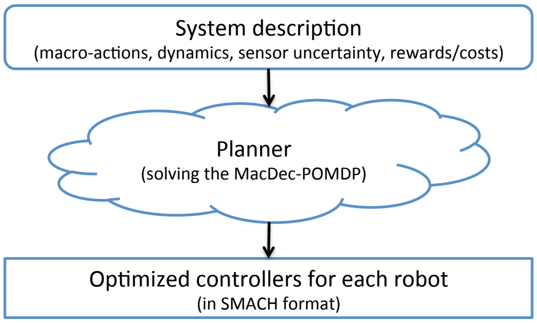

System description
(macro-actions, dynamics, sensor uncertainty, rewards/costs)
Planner
(solving the MacDec-POMDP)
Optimized controllers for each robot
(in SMACH format)

Figure 5: A high level system diagram for multi-robot problems where the system can be described formally or using a simulator, solutions are generated with our planning methods and the output is a set of controllers, one for each robot.

Specifically, a fixed policy can be evaluated by Monte Carlo sampling starting at an initial state (or belief state), choosing an action for each agent according to the policy, sampling an observation from the system, updating the current position in the policy (i.e., the current node in each agent's policy tree) and then continuing this process until some maximum time step has been reached. The value of the k-th sample-based trajectory starting at s_0 and using policy π is given by $V^{\pi,k}(s_0) = r_0^k + \ldots + \gamma^T r_T^k$, where r_t^k is the reward given to the team on the t-th step. After K trajectories, $\hat{V}^\pi(s_0) = \sum_{k=1}^K \frac{V^{\pi,k}(s_0)}{K}$.

As the number of samples increases, the estimate of the policy's value will approach the true value. This sample-based evaluation is necessary in large or continuous state spaces. Sample-based evaluation has been used in the Dec-POMDP case (Wu, Zilberstein, & Chen, 2010b; Liu, Amato, Liao, Carin, & How, 2015), but we extend the idea to the macro-action case where there is the added benefit of abstracting away the details of the macro-action policies.

In the multi-robot case, given the macro-actions, macro-observations and simulator, our off-line planners can automatically generate a solution which optimizes the value function with respect to the uncertainty over outcomes, sensor information, and other robots. The planner generates the solution in the form of a set of policy trees (as in Figure 4) which are parsed into a corresponding set of SMACH controllers (Bohren, 2010), one for each robot. SMACH controllers are hierarchical state machines for use in a ROS (Quigley, Conley, Gerkey, Faust, Foote, Leibs, Wheeler, & Ng, 2009) environment. Just like the policy trees they represent, each node in the SMACH controller represents a macro-action which is executed on the robot (e.g., navigation to a waypoint or wait for another robot) and each edge corresponds to a macro-observation. Our system can automatically generate SMACH controllers—which are typically designed by hand—for complex, general multi-robot systems.

6. Experiments

We test the performance of our macro-action-based algorithms in simulation, using existing benchmarks, a larger domain, and in a novel multi-robot warehousing domain.

6.1 Simulation Experiments

For the simulation experiments, we test on a common Dec-POMDP benchmark, a four agent extension of this benchmark, and a large problem inspired by robot navigation. Our algorithms were run on a single core 2.5 GHz machine with 8GB of memory. For option-based MBDP (O-MBDP), heuristic policies for the desired lengths were generated by producing 1000 random policies and keeping the joint policy with the highest value at the initial state. Sampling was used (10000 simulations) to determine if a policy will terminate before the horizon of interest.

6.1.1 An Existing Dec-POMDP Problem: Meeting in a Grid

The meeting-in-a-grid problem is an existing two-agent Dec-POMDP benchmark in which agents receive 0 reward unless they are both in one of two corners in a 3x3 grid (Amato et al., 2009). Agents can move up, down, left, right or stay in place, but transitions are noisy, so an agent may move to an adjacent square rather than its desired location. Each agent has full observability of its own location, but cannot observe the other agent (even when they share the same grid square). We defined two options for each agent: each one moving the agent to one of the two goal corners. Options are valid in any (local) state and terminate when they reach the appropriate goal corner. An agent stays in a corner on a step by choosing the appropriate option again. Macro-observations are the agent's location (they are the same as the primitive observations, but the agent only observes

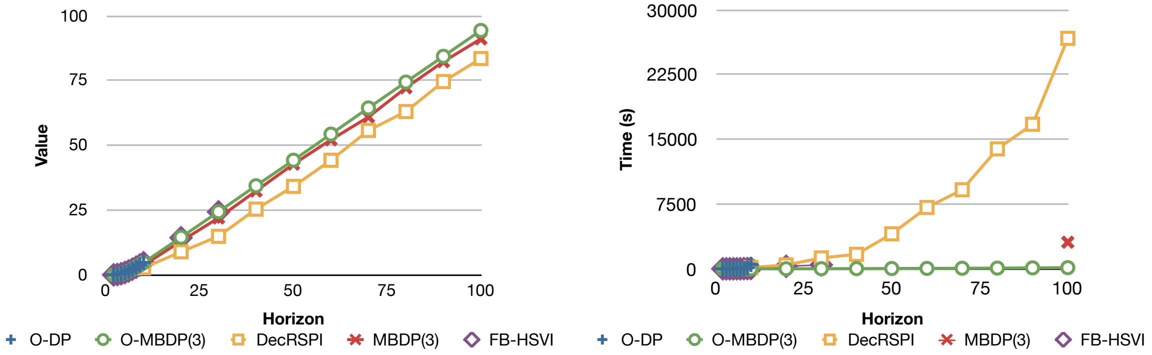


Figure 6: Value and time results for the meeting in a grid Dec-POMDP benchmark including leading Dec-POMDP approaches DecRSPI and MBDP as well as option-based DP and MBDP.

it's updated location after completion of a macro-action). It is clear that these options provide the important macro-actions for the agents and navigation is possible based on local information in this problem. While this is a very small problem, it allows for direct comparison with Dec-POMDP methods.

Results for this problem are split between Figure 6 and Table 1 because not all results are available for all algorithms. We compared against one leading optimal Dec-POMDP algorithm, feature-based heuristic search value iteration (FB-HSVI) (Dibangoye et al., 2016), and three leading approximate Dec-POMDP algorithms: MBDP with incremental policy generation (MBDP-IPG) (Amato et al., 2009), rollout sampling policy iteration (DecRSPI) (Wu et al., 2010a) and trial-based dynamic programming (TBDP) (Wu et al., 2010b). $MaxTrees = 3$ was used in both O-MBDP and MBDP-IPG (referred to as MBDP in the figure and table). Results for other algorithms are taken from their respective publications. As such, results were generated on different machines, but the trends should remain the same. The left figure shows that all approaches achieve approximately the same value, but option-based DP (O-DP) cannot solve horizons longer than 10 without running out of memory. Impressively, FB-HSVI is able to scale to horizon 30 by not explicitly representing a policy and maintaining a compressed distribution over agent histories and the state. Nevertheless, since FB-HSVI is an optimal method, it becomes intractable as the horizon grows (it would be an interesting area of future research to see how macro-actions could be combined with the compressed representation of FB-HSVI). The right figure shows the time required for different horizons. All approaches run quickly for small horizons, but DecRSPI required an intractable amount of time as the horizon grows. The table shows time and value results for larger horizons. Again, all approaches achieve similar values, but O-MBDP is much faster than MBDP-IPG or TBDP. The benefit of using a macro-action representation can be seen most directly by comparing O-MBDP and MBDP, which are both based on the same algorithm: there is a significant improvement in running time, while solution quality is maintained.

	Value		Time (s)	
	$h = 100$	$h = 200$	$h = 100$	$h = 200$
O-MBDP(3)	94.4	194.4	133	517
MBDP(3)	92.1	193.4	3084	13875
TBDP	92.8	192.1	427	1372

Table 1: Times and values for larger horizons on the meeting in a grid benchmark.

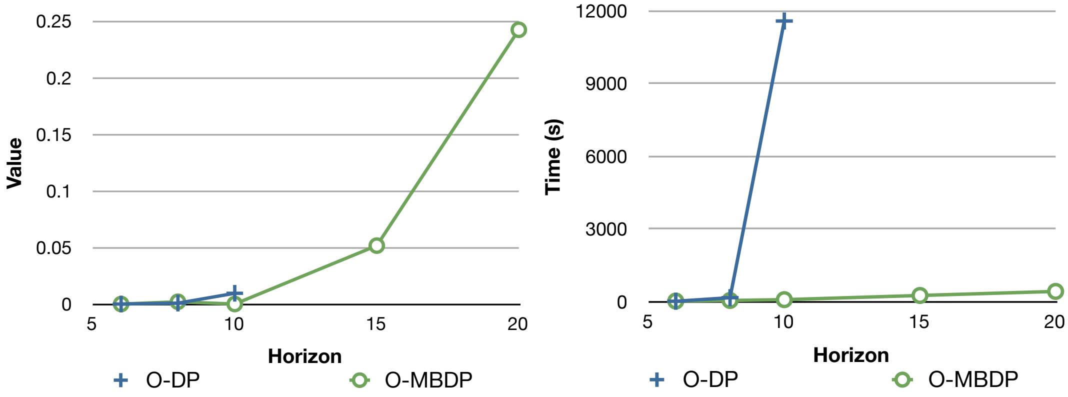


Figure 7: 4-agent meeting in a grid results showing (a) value and (b) running time on a 10×10 grid.

6.1.2 Larger Grids with More Agents

To test the scalability of these approaches, we consider growing the meeting-in-a-grid benchmark to a larger grid size and a larger number of agents. That is, agents still receive zero reward unless *all* agents are in one of the goal corners. The same options and macro-observations are used as in the 3x3 version of the problem. We generated results for several four-agent problems with random starting locations for each agent. We did not compare with current optimal or approximate Dec-POMDP methods because, while they may be theoretically applicable, current implementations cannot solve problems with more than two agents or the methods assume structure (e.g., factorization or independence) that is not present in our problem.

Results for option-based dynamic programming and MBDP on problems with a 10×10 grid are shown in Figure 7. Three trees were used for O-MBDP. It is worth noting that these are very large problems with 10^8 states. Also, the 4-agent version of the problem is actually much harder than the 2-agent problem in Section 6.1.1, because all 4 agents must be in the same square to receive any reward (rather than just 2) and the grid is much larger (10x10 rather than 3x3). Agents are randomly initialized, but for horizon 10, it may be impossible for all 4 agents to reach each other in the given time. By horizon 20 (the largest we solved), the agents can often reach each other, but just at the later horizons due to noise and the large grid. For instance, an optimal solution to a deterministic version of this problem (an upper bound for the stochastic problem we use) for horizon 20 is approximately

2. The dynamic programming method is able to solve problems with a long enough horizon to reach the goal (producing positive value), but higher horizons are not solvable. The MBDP-based approach is able to solve much larger horizons, requiring much less time than O-DP. O-MBDP is able to produce near-optimal values for horizons that are also solvable by O-DP, but results may be further from optimal as the horizon grows (as is often the case with MBDP-based approaches).

6.1.3 Two-Agent NAMO

We also consider a two-agent version of the problem of robots navigating among movable obstacles (Stilman & Kuffner, 2005). Here, as shown in Figure 8, both agents are trying to reach a goal square (marked by G), but there are obstacles in the way. Each robot can move in four directions (up, down, left and right) or use a 'push' action to attempt to move a box to a specific location (diagonally down to the left for the large box and into the corner for both small boxes). The push action fails and the robot stays in place when the robot is not in front of the box. Robots can move the small boxes (b_1 and b_2) by themselves, but must move the larger box (b_3) together. Observations are an agent's own location (but not the location of the other agent) and whether the large box or the same numbered box has been moved (i.e., agent 1 can observe box 1 and agent 2 can observe box 2). There is noise in both navigation and in box movement: movement is successful with probably 0.9 and pushing the small and large boxes is successful with probably 0.9 and 0.8, respectively. To encourage the robots to reach the goal as quickly as possible, there is a negative reward (-1) when any agent is not in the goal square.

Four options were defined for each agent. These consisted of 1) moving to a designated location to push the big box, 2) attempting to push the large box, 3) pushing the designated small box (box 1 for agent 1 and box 2 for agent 2) to the corner square, and 4) moving to the goal. The option of moving to the goal is only valid when at least one box has been moved and movement of any box is only valid if the large box and the agent's designated box has not yet been moved. Movement options terminate at the desired location and pushing options terminate with the box successfully or unsuccessfully moved. Macro-observations were the same as primitive observations (the agent's location and box movements). These options provide high-level choices for the agents to coordinate on this problem, while abstracting away the navigation tasks to option execution. Options for just moving to the small boxes could also be incorporated, but were deemed unnecessary because coordination is unnecessary for pushing the small boxes.

Results for option-based dynamic programming are given in Figure 9. Here, O-DP performs very well on a range of different problem sizes and horizons. Because negative reward is given until both agents are in the goal square, more steps are required to reach the goal as the problem size increases. The agents will stay in the goal upon reaching it, causing the value to plateau. As shown in the top figure, O-DP is able to produce this policy for the different problem sizes and horizons. The running times for each of the grid sizes (5×5 to 25×25) are shown in the bottom figure for the horizon 25 problem. Here, we see the running time increases for larger state spaces but the growth is sublinear.

A comparison with other Dec-POMDP algorithms (including O-MDBP) is shown in Table 2. For TBDP and GMAA-ICE* (a leading optimal Dec-POMDP algorithm) (Oliehoek

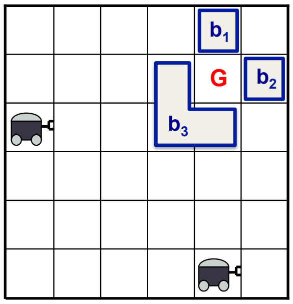


Figure 8: A 6x6 two-agent NAMO problem.

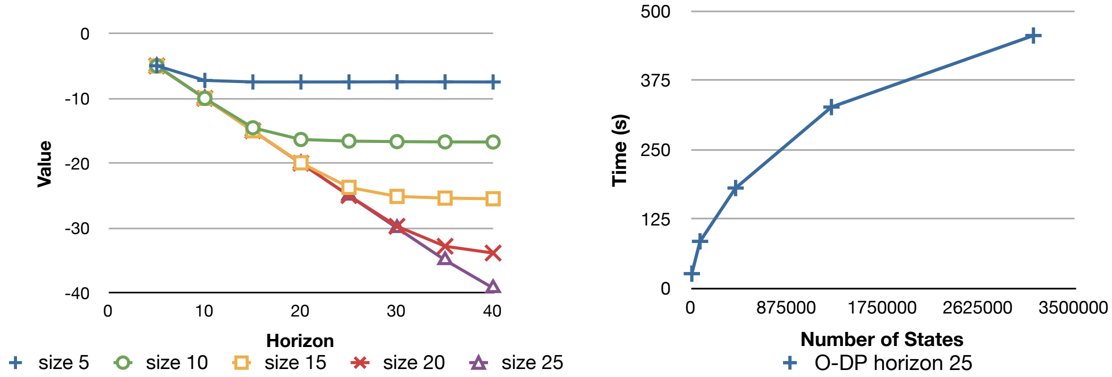


Figure 9: Value and time results for O-DP in the two-agent NAMO problem for various size grids (where size is the length of a single side)

et al., 2013), the grid size was increased while at least horizon 4 could be solved and then the horizon was increased until it reached 100. Results for these algorithms were provided by personal communication with the authors and run on other machines, but the trends remain the same. For O-MBDP, 20 trees were used because smaller numbers resulted in poor performance, but parameters were not exhaustively evaluated. The results show that TBDP is able to solve the 4×4 problem, but runs out of memory when trying to solve any 5×5 problems. GMAA*-ICE can solve larger problem sizes, but runs out of memory for larger horizons. GMAA*-ICE scales better with the increased state space because it is able to exploit the factorization of the problem, but is limited to very small horizons because it is solving the underlying Dec-POMDP optimally. The inability for current approaches to solve these problems is not surprising given their size. By contrast, O-DP is able to solve the 25×25 problem which has over 3 million states states while O-MBDP solves the 50×50 problem that has has 50 million states. O-MBDP is able to solve even larger problems, but we did not analyze its performance beyond the 50×50 problem.

[3] Larger problem sizes were not tested for GMAA*-ICE, but some may be solvable. Note that for any problem larger than 4×4 horizons beyond 4 are not solvable and the running time is already high for the 12×12 case.

	Num. of States	h	Value	Time (s)
O-DP	3.125×10^6	100	-42.7	40229
O-MBDP(20)	5×10^7	100	-93.0	4723
GMAA*-ICE[3]	$165,888$	4	-4	11396
TBDP	$2,048$	100	-6.4	1078

Table 2: Largest representative NAMO problems solvable by each approach. For GMAA*-ICE and TBDP problem size was increased until horizon 4 was not solvable.

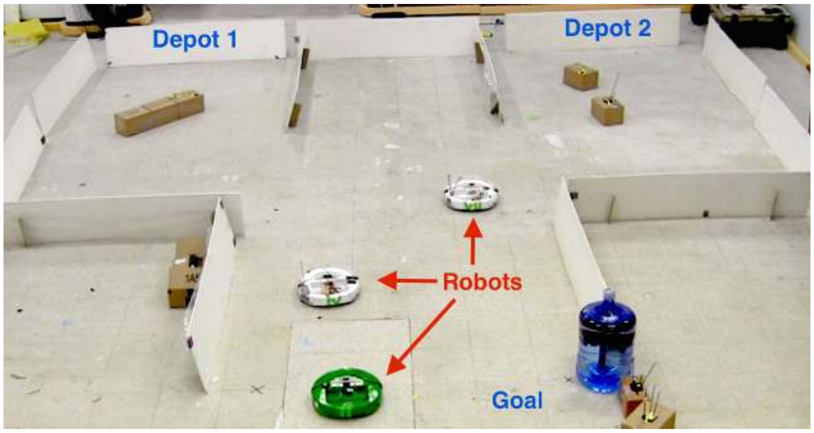


Figure 10: The multi-robot warehouse domain with depots and robots labeled.

6.2 Multi-Robot Experiments

We also tested our methods in a warehousing scenario using a collection of iRobot Creates (Figure 10) where we varied the communication capabilities available to the robots. The results demonstrate that our methods can automatically generate the appropriate motion and communication behavior while considering uncertainty over outcomes, sensor information and other robots.

6.2.1 The Warehouse Domain

We consider three robots in a warehouse that are tasked with finding and retrieving boxes of two different sizes: large and small. Robots can navigate to known depot locations (rooms) to retrieve boxes and bring them back to a designated drop-off area. The larger boxes can only be moved effectively by two robots (if a robot tries to pick up the large box by itself, it will move to the box, but fail to pick it up). While the locations of the depots are known, the contents (the number and type of boxes) are unknown. In our implementation, we assumed there were three boxes (one large and two small), each of which was equally likely to be in one of two depots. Our planner generates a SMACH controller for each of the robots off-line using our option-based algorithms. These controllers are then executed online in a decentralized manner.

In each scenario, we assumed that each robot could observe its own location, see other robots if they were within (approximately) one meter, observe the nearest box when in a depot and observe the size of the box if it is holding one (defining the resulting macro-observations). In the simulator used by the planner to evaluate solutions, the resulting state space includes the location of each robot (discretized into nine possible locations) and the location of each of the boxes (in a particular depot, with a particular robot or at the goal). In particular, there are $\prod_{i \in I} locAg_i \times \prod_{b \in B} locB_b$ states, where $locAg_i$ is the location of an agent and is discretized to a 3x3 grid and $locB_b$ represents the location of each of 3 boxes (at a depot, with a robot, at a goal, or with a pair of robots), with the size of $locB_b$ for all b is $numDepots + numAgents + numGoals + numAgents * numAgents$ where we set $numDepots = 2$, $numAgents = 3$ and $numGoals = 1$. The primitive actions are to move in four different directions as well as pickup, drop and communication actions. The macro-actions and macro-observations vary a bit for each scenario, but are detailed in the sections below. Note that this primitive state and action representation is used for evaluation purposes and not actually implemented on the robots (which just utilize the SMACH controllers). Higher fidelity simulators could also be used, but running time may increase if the simulations are computationally intensive (average solution times for the policies presented below were approximately one hour). The three-robot version of this scenario has 2,460,375 states, which is several orders of magnitude larger than problems typically solvable by Dec-POMDP approaches.[8] These problems are solved using the option-based MBDP algorithm initialized with a hand coded heuristic policy.

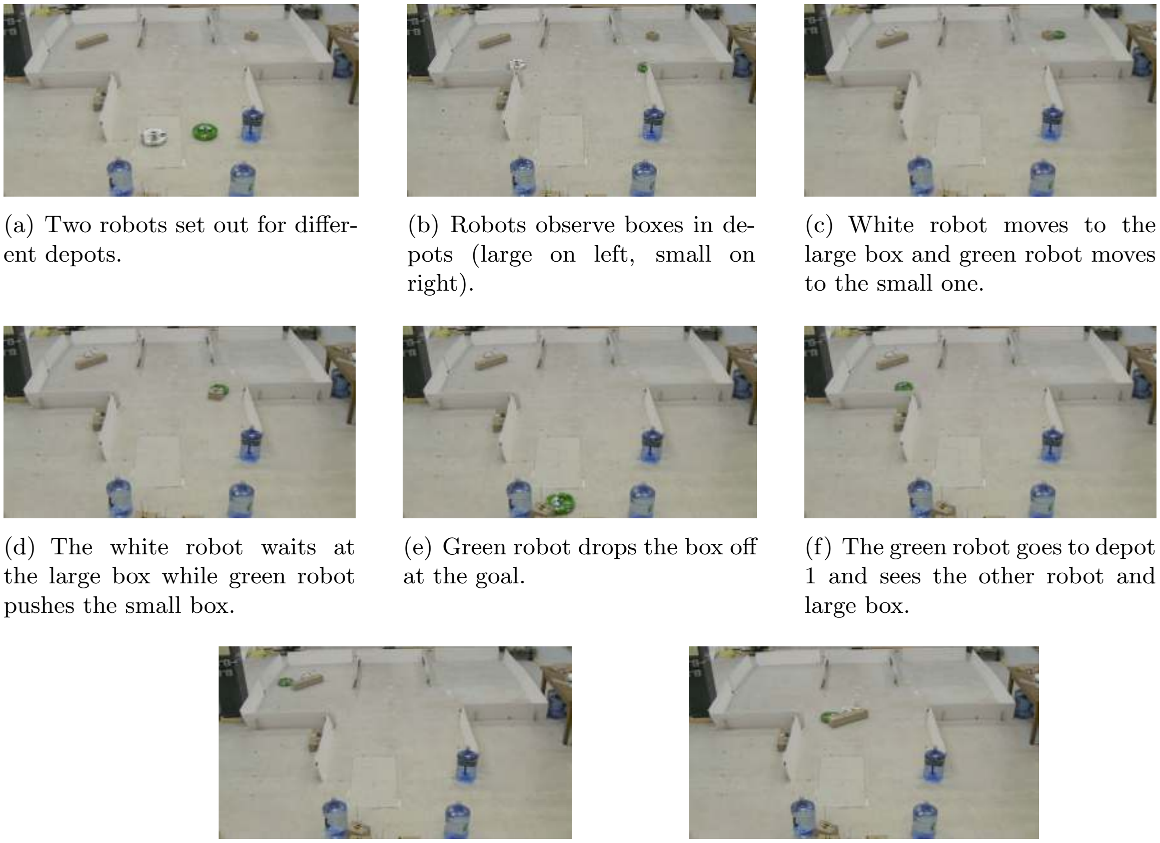
Navigation has a small amount of noise in the amount of time required to move to locations (reflecting the real-world dynamics): this noise increases when the robots are pushing the large box (reflecting the need for slower movements and turns in this case). Specifically, the robots were assumed to transition to the desired square deterministically with no boxes, with probability 0.9 with the small box and with probability 0.8 with the large box. Picking up boxes and dropping them was assumed to be deterministic. These noise parameters were assumed to be known in this work, but they could also be learned by executing macro-actions multiple times in the given initiation sets.[9] Note that the MacDec-POMDP framework is very general so other types of macro-actions and observations could also be used (including observation of other failures). More details about each scenario are given below.

6.2.2 Scenario 1: No Communication

In the first scenario, the robots cannot communicate with each other. Therefore, all cooperation is based on the controllers that are generated by the planner (which were generated offline) and observations of the other robots (when executing online). The macro-actions were: `Go to depot 1`, `Go to depot 2`, `Go to the drop-off area`, `Pick up the small box`, `Pick up the large box`, and `Drop off a box`.

[8] Our state representation technically has 1,259,712,000 states, since we also include observations of each agent (of which there are 8 in this version of the problem) in the state space.

[9] These parameters and controllers were loosely based on the actual robot navigation and box pushing. Other work has looked at more directly determining these models and parameters (Amato, Konidaris, Anders, Cruz, How, & Kaelbling, 2017; Omidshafiei et al., 2017).



(a) Two robots set out for different depots.

(b) Robots observe boxes in depots (large on left, small on right).

(c) White robot moves to the large box and green robot moves to the small one.

(d) The white robot waits at the large box while green robot pushes the small box.

(e) Green robot drops the box off at the goal.

(f) The green robot goes to depot 1 and sees the other robot and large box.

(g) Green robot moves to help the white robot.

(h) The green robot moves to the box and the two robots push it back to the goal.

Figure 11: Scenario 1 (no communication).

The depot macro-actions are applicable anywhere and terminate when the robot is within the walls of the appropriate depot. The drop-off and drop macro-actions are only applicable if the robot is holding a box, and the pickup macro-actions are only applicable when the robot observes a box. Picking up the small box was assumed to succeed deterministically, but the model could easily be adjusted if the pickup mechanism is less robust. The macro-observations are the basic ones defined above: the robot can observe it's own location (9 discrete positions), whether there is another robot present in the location, observe the nearest box when in a depot (small, large or none) and observe the size of the box if it is holding one (small, large or none). The macro-actions correspond to natural choices for robot controllers.

This case[10] (seen in Figure 11 along with a depiction of the executed policy in Figure 12) uses only two robots to more clearly show the optimized behavior in the absence of communication. The robots begin in the drop-off area and the policy generated by the planner begins by assigning one robot to go to each of the depots (seen in Figure 11(a)). The robots then observe the contents of the depots they are in (seen in Figure 11(b)). If

[10] All videos can be seen at `http://youtu.be/fGUHTHH-JNA`

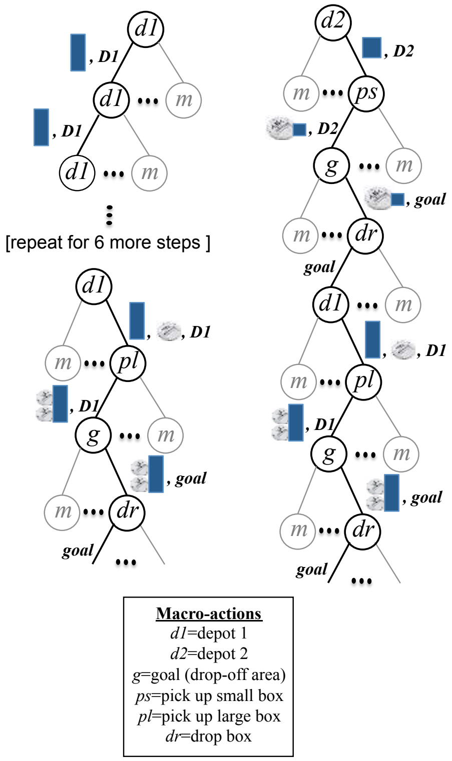
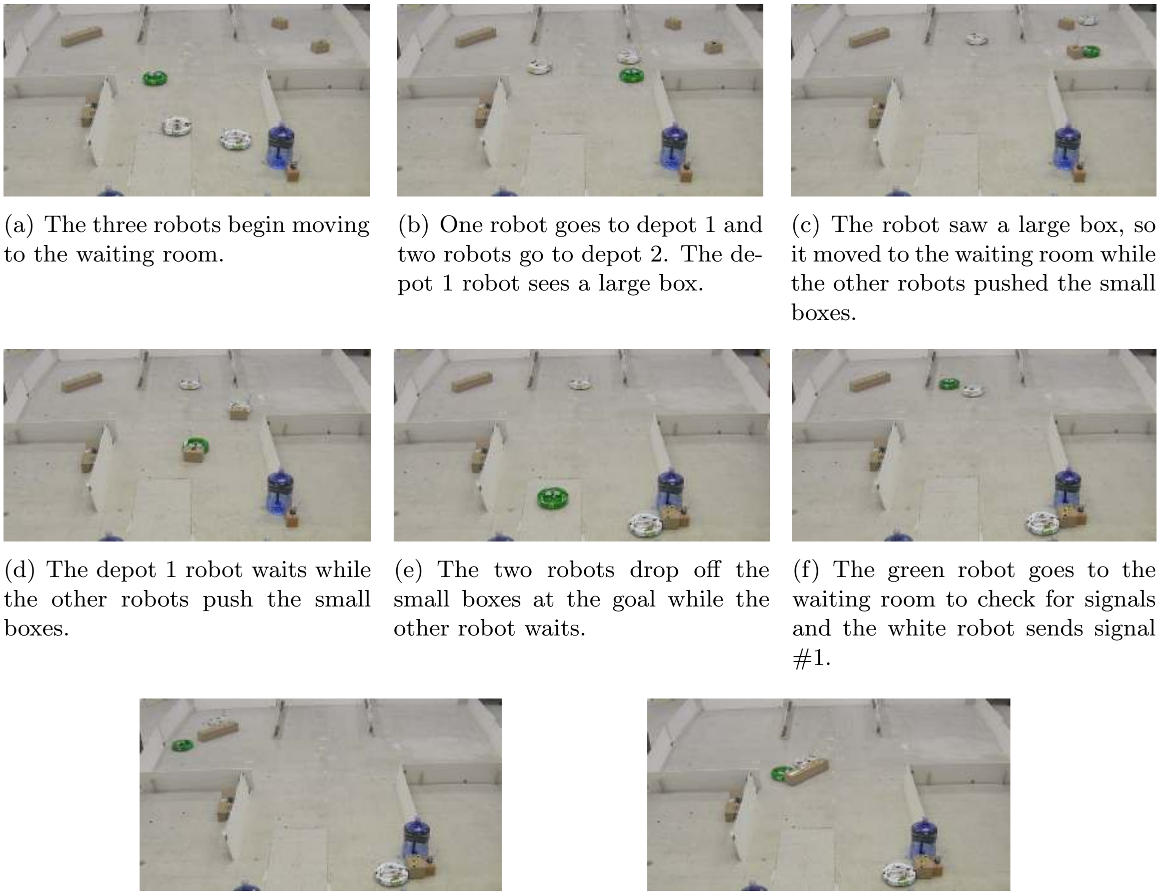


Figure 12: Path executed in policy trees for the no communication scenario by the white robot (left) and the green robot (right). Only macro-actions executed (nodes) and observations seen are shown. Observations are shown pictorially, with the box sizes (small as a square and large as a rectangle) and robots (white create) given along the corresponding edge.

there is only one robot in a depot and there is a small box to push, the robot will push the small box (Figures 11(c) and 11(d)). If the robot is in a depot with a large box and no other robots, it will stay in the depot, waiting for another robot to come and help push the box (Figure 11(d)). In this case, once the other robot is finished pushing the small box (Figure 11(e)), it goes back to the depots to check for other boxes or robots that need help (Figure 11(f)). When it sees another robot and the large box in the depot on the left (depot 1), it attempts to help push the large box (Figure 11(g)) and the two robots are successful pushing the large box to the goal (Figure 11(h)). The planner has *automatically derived a strategy for dynamic task allocation*—two robots go to each room, and then search for help needed after pushing any available boxes. This behavior was generated by an optimization process that considered the different costs of actions and the uncertainty involved (in the current step and into the future) and used those values to tailor the behavior to the particular problem instance.



(a) The three robots begin moving to the waiting room.

(b) One robot goes to depot 1 and two robots go to depot 2. The depot 1 robot sees a large box.

(c) The robot saw a large box, so it moved to the waiting room while the other robots pushed the small boxes.

(d) The depot 1 robot waits while the other robots push the small boxes.

(e) The two robots drop off the small boxes at the goal while the other robot waits.

(f) The green robot goes to the waiting room to check for signals and the white robot sends signal #1.

(g) Signal #1 is interpreted as a need for help in depot 1, so they move to depot 1 and push the large box.

(h) The two robots in depot 1 push the large box back to the goal.

Figure 13: Scenario 2 (limited communication).

6.2.3 Scenario 2: Local Communication

In scenario 2, robots can communicate when they are within one meter of each other. The macro-actions are the same as above, but we added ones to communicate and wait for communication. The resulting macro-action set is: `Go to depot 1`, `Go to depot 2`, `Go to the drop-off area`, `Pick up the small box`, `Pick up the large box`, `Drop off a box`, `Go to an area between the depots (the "waiting room")`, `Send signal #1`, `Send signal #2`, and `Wait in the waiting room for another robot`.

Here, we allow the robots to choose to go to a “waiting room” which is between the two depots. This permits the robots to possibly communicate or receive communications before committing to one of the depots. The waiting-room macro-action is applicable in any situation and terminates when the robot is between the waiting room walls. The depot macro-actions are now only applicable in the waiting room, while the drop-off, pick up and drop macro-actions remain the same. The wait macro-action is applicable in the waiting room and terminates when the robot observes another robot in the waiting room. The signaling macro-actions are applicable in the waiting room and are observable by other robots that are within approximately a meter of the signaling robot. The macro-observations are the same as in the previous scenario, but now include observations for the two signals. Note that *we do not specify how each communication signal should be interpreted, or when they should be sent.*

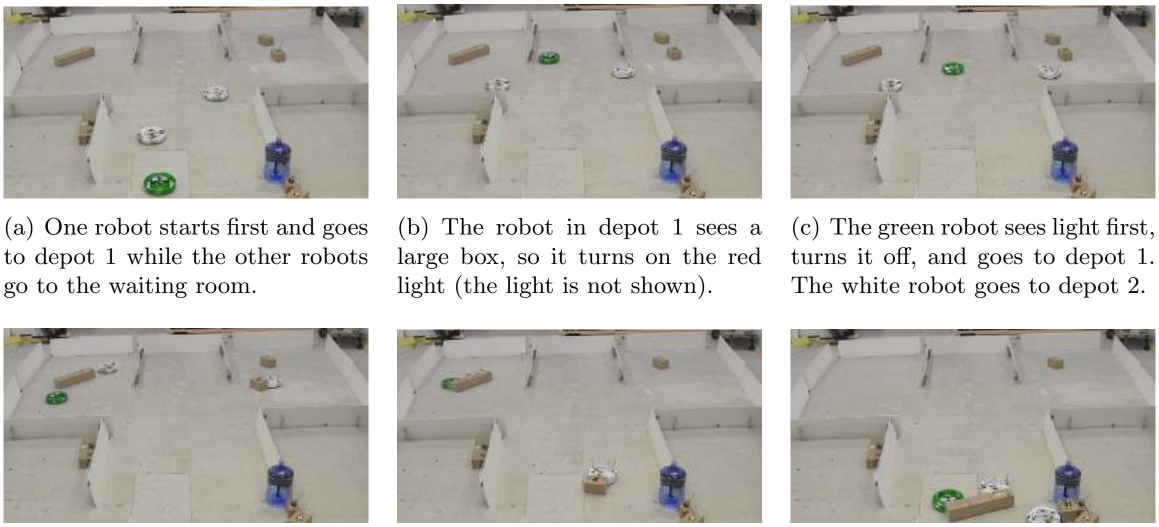
The results for this three-robot domain are shown in Figure 13. The robots go to the waiting room (Figure 13(a)) and then two of the robots go to depot 2 (the one on the right) and one robot goes to depot 1 (the one on the left) (Figure 13(b)). Because there are three robots, the choice for the third robot is random while one robot will always be assigned to each of the depots. Because there is only a large box to push in depot 1, the robot in this depot goes back to the waiting room to try to find another robot to help it push the box (Figure 13(c)). The robots in depot 2 see two small boxes and they choose to push these back to the goal (also Figure 13(d)). Once the small boxes are dropped off (Figure 13(e)) one of the robots returns to the waiting room and then is recruited by the other robot to push the large box back to the goal (Figures 13(f) and 13(g)). The robots then successfully push the large box back to the goal (Figure 13(h)). In this case, the planning process *determines how the signals should be used to perform communication.*

6.2.4 Scenario 3: Global Communication

In the last scenario, the robots can use signaling (rather than direct communication). In this case, there is a switch in each of the depots that can turn on a blue or red light. This light can be seen in the waiting room and there is another light switch in the waiting room that can turn off the light. (The light and switch were simulated in software and not incorporated in the physical domain.) The macro-actions were: `Go to depot 1`, `Go to depot 2`, `Go to the drop-off area`, `Pick up the small box`, `Pick up the large box`, `Drop off a box`, `Go to the "waiting room"`, `Turn on a blue light`, `Turn on a red light`, and `Turn off the light`.

The first seven macro-actions are the same as for the communication case except we relaxed the assumption that the robots had to go to the waiting room before going to the depots (making both the depot and waiting room macro-actions applicable anywhere). The macro-actions for turning the lights on are applicable in the depots and the macro-actions for turning the lights off are applicable in the waiting room. The macro-observations are the same as in the previous scenario, but the two signals are now the lights instead of the communication signals. While the lights were intended to signal requests for help in each of the depots, we did not assign a particular color to a particular depot. In fact, we did not assign them any meaning at all, allowing the planner to set them in any way that improves performance.

The results are shown in Figure 14. Because one robot started ahead of the others, it was able to go to depot 1 to sense the size of the boxes while the other robots go to the waiting room (Figure 14(a)). The robot in depot 1 turned on the light (red in this case, but not shown in the images) to signify that there is a large box and assistance is needed (Figure 14(b)). The green robot (the first other robot to the waiting room) sees this light, interprets it as a need for help in depot 1, and turns off the light (Figure 14(c)). The other



(a) One robot starts first and goes to depot 1 while the other robots go to the waiting room.

(b) The robot in depot 1 sees a large box, so it turns on the red light (the light is not shown).

(c) The green robot sees light first, turns it off, and goes to depot 1. The white robot goes to depot 2.

(d) Robots in depot 1 move to the large box, while the robot in depot 2 begins pushing the small box.

(e) Robots in depot 1 begin pushing the large box and the robot in depot 2 pushes a small box to the goal.

(f) The robots from depot 1 successfully push the large box to the goal.

Figure 14: Scenario 3 (signaling).

robot arrives in the waiting room, does not observe a light on and moves to depot 2 (also Figure 14(c)). The robot in depot 2 chooses to push a small box back to the goal and the green robot moves to depot 1 to help the other robot (Figure 14(d)). One robot then pushes the small box back to the goal while the two robots in depot 1 begin pushing the large box (Figure 14(e)). Finally, the two robots in depot 1 push the large box back to the goal (Figure 14(f)). This behavior is optimized based on the information given to the planner. *The semantics of all these signals as well as the movement and signaling decisions were decided on by the planning algorithm to maximize value.*

6.2.5 Simulation Results

We also evaluated the multi-robot experiments in the simulator to evaluate the difference in performance between option-based MBDP (O-MBDP) and option-based direct cross entropy policy search (O-DICE). Option-based dynamic programming is not scalable enough to solve the domains to the horizons considered. For O-MBDP, $maxTrees = 3$, which was chosen to balance solution quality and running time. For O-DICE, $Iter = 100$, $N = 10$, $N_b = 5$, and $\alpha = 0.1$, which were chosen based on suggestions from the original work (Oliehoek et al., 2008). A version of O-DICE was also implemented that, rather than maintaining sampling distributions for the whole tree, only maintains a single sampling distribution that is used at each node in the tree. This later version of O-DICE is referred to as O-DICE (1) and can be thought of as a biased form of Monte Carlo sampling.

As can be seen in Table 3, O-DICE outperforms O-MBDP in terms of both value and time. In all cases, versions of O-DICE are more scalable than O-MBDP, even though only 3 trees were used for O-MBDP. For problems in which both O-MBDP and O-DICE

No Communication

	O-MBDP(3)		**O-DICE (1)**		**O-DICE (full)**	
	value	time (s)	value	time (s)	value	time (s)
Horizon 7	0	10910	0	50	0	312
Horizon 8	0	27108	0	552	0	3748
Horizon 9	1.161	161454	1.169	601	1.158	5247
Horizon 10	–	–	2.163	618	2.159	6400
Horizon 11	–	–	3.033	699	3.120	10138

Communication

	O-MBDP(3)		**O-DICE (1)**		**O-DICE (full)**	
	value	time (s)	value	time (s)	value	time (s)
Horizon 7	0.225	49	0.221	46	0.217	207
Horizon 8	0.421	139	0.409	444	0.420	2403
Horizon 9	1.60	$*$	1.650	549	1.650	3715
Horizon 10	–	–	2.179	901	2.795	3838
Horizon 11	–	–	–	–	–	–

Signalling

	O-MBDP(3)		**O-DICE (1)**		**O-DICE (full)**	
	value	time (s)	value	time (s)	value	time (s)
Horizon 7	0.225	353	0.221	63	0.221	204
Horizon 8	0.421	16466	0.417	649	0.430	4011
Horizon 9	1.663	87288	1.691	659	1.694	7362
Horizon 10	–	–	2.392	682	2.782	7447
Horizon 11	–	–	3.756	763	3.964	10336

Table 3: Multi-robot warehouse simulation results for option-based MBDP (O-MBDP) and option-based direct cross entropy search (O-DICE) using parameters for full histories (full) or just a single value (1). Value and time in seconds is given with $-$ signifying the algorithm runs out of memory before generating any valid solution and $*$ signifying the algorithm runs out of memory before completion.

could produce solutions, the values were very similar, but the O-DICE methods required significantly less time. O-MBDP either runs out of memory (due to the large number of trees generated during a backup step) or takes a really long time to generate the $maxTrees$ trees. Using more efficient versions of MBDP (e.g., Wu et al., 2010a) should improve performance, but performance improvements could also be made to O-DICE. An extensive comparison has not been conducted between these algorithms even for primitive action Dec-POMDP domains, but we expect that performance will depend on the domain and parameters used (e.g., heuristics in MBDP). The full version of O-DICE was able to outperform the single parameter version of O-DICE in terms of value, but also required more time.

6.2.6 Infinite Horizon Comparisons

Unlike POMDPs, Dec-POMDP finite-horizon methods are typically not scalable enough to solve large or infinite-horizon problems. As a consequence, special-purpose infinite-horizon methods have been developed which typically use a finite-state controller policy representation instead of a policy tree. The finite-state controller allows memory to be bounded. As a consequence, finite-state controller-based methods are typically more scalable for large horizon problems, but perform poorly for smaller horizons.

Finite-state controllers, which condition action selection on an internal memory state, have been widely used in Dec-POMDPs (Bernstein et al., 2009; Amato, Bernstein, & Zilberstein, 2010a; Amato, Bonet, & Zilberstein, 2010b; Pajarinen & Peltonen, 2011; Wu et al., 2013; Kumar, Zilberstein, & Toussaint, 2015; Kumar, Mostafa, & Zilberstein, 2016). Finite-state controllers operate in the same way as policy trees in that there is a designated initial node and following action selection at that node, the controller transitions to the next node depending on the observation seen. This continues for the infinite steps of the problem. Finite-state controllers explicitly represent infinite-horizon policies, but can also be used for finite-horizon policies.

Recently, we and others have extended the ideas of macro-actions from this paper to use finite-state controller representations. In particular, heuristic search (Amato et al., 2017) and a DICE-based approach (Omidshafiei et al., 2017) have been explored. G-DICE (Omidshafiei et al., 2017) is the same as O-DICE except it is applied to the finite-state controller representation rather than the tree. The heuristic search method from (Amato et al., 2017) is similar to multi-agent A* approaches (Oliehoek et al., 2013; Szer, Charpillet, & Zilberstein, 2005; Oliehoek, Spaan, & Vlassis, 2008; Oliehoek, Whiteson, & Spaan, 2009), but again is applied to the finite-state controller representation rather than the tree.

It is worth noting that the key difference is the policy representation and the algorithms in this paper could be applied to finite-state controllers and many finite-state controller-based methods could be applied to trees. This paper introduces macro-actions in Dec-POMDPs and explores some initial algorithms for tree-based solutions; many future algorithms are now possible.

Nevertheless, for thoroughness of results, we provide the performance of the heuristic search method MDHS (Amato et al., 2017) on our benchmark problems. MDHS is an anytime algorithm, so it will continue to improve until the best parameters for the given controller size are found. For a fair comparison, we let it run for the same amount of time as the full version of O-DICE. We set the parameters in the same way as the previous work (Amato et al., 2017) (e.g., 5 controller nodes were used) and the initial lower bound was found from the best of 100 random controller parameterizations. Reporting results for all horizons of all domains becomes redundant, but the results we provide are representative of the other domains and horizon values.

As can be seen in Table 4, MDHS often achieves values that are similar to the O-DICE values, but sometimes significantly underperforms the other method. For instance, MDHS can only achieve 17% of the O-DICE value in the meeting in a grid problem with 4

Meeting in a Grid

	2 agents, hor=100		**2 agents, hor=200**		**4 agents, hor=20**	
	value	time (s)	value	% O-DICE	value	% O-DICE
	92.478	98%	192.407	99%	0.076	17%

NAMO

	size 10		**size 15**		**size 20**	
	value	% O-DICE	value	% O-DICE	value	% O-DICE
Horizon 10	−10	100%	−10	100%	−10	100%
Horizon 20	−18.533	88%	−20	100%	−20	100%
Horizon 30	−19.558	89%	−27.458	94%	−29.961	99%

Robot warehouse

	No Communication		**Communication**		**Signaling**	
	value	% O-DICE	value	% O-DICE	value	% O-DICE
Horizon 7	0	100%	0.205	94%	0.207	94%
Horizon 8	0	100%	0.393	94%	0.428	99%
Horizon 9	1.120	97%	1.138	69%	1.611	95%
Horizon 10	2.064	94%	1.167	42%	1.055	38%
Horizon 11	2.932	94%	−	−	3.807	96%

Table 4: Results for the controller-based MDHS method on our benchmark problems along with the performance relative of O-DICE (full).

agents, 38% of the O-DICE value is produced in the horizon 10 robot warehouse problem with signaling and 69% and 42% of the O-DICE value is produced in the horizon 9 and 10 warehouse problems with communication. The values for the NAMO problems are not particularly interesting as all policies have the same value until the horizon becomes significantly longer than the domain size (since the agent requires more steps to reach the goal as the domain size increase), but we still see that MDHS does not achieve the full O-DICE values for non-degenerate horizons.

In general, MDHS is more scalable in terms of the horizon (e.g., solving the horizon 11 robot warehouse problem with communication), but scalability depends on choosing a proper controller size to balance solution quality and computational efficiency. As a result, controller-based methods, such as MDHS, can return lower quality solutions on horizons that are solvable by the tree-based methods. MDHS will also require an intractable amount of time to improve solutions as the number of observations grows since it searches for assignments for all possible next observations in the controller (Omidshafiei et al., 2017). As is currently the case in (primitive) Dec-POMDPs, tree-based and controller-based algorithms both have their place in macro-action-based Dec-POMDPs. The performance of MDHS (or controller-based methods more generally) relative to tree-based methods is very problem and horizon dependent (as seen in our results). A general rule of thumb may be to use a tree-based method for finite-horizon problems that are solvable and to use controller-based (or other methods) otherwise.

7. Related Work

While many hierarchical approaches have been developed for multi-agent systems (Horling & Lesser, 2004), very few are applicable to multi-agent models based on MDPs and POMDPs. Perhaps the most similar approach is that of Ghavamzadeh et al. (Ghavamzadeh, Mahadevan, & Makar, 2006). This is a multi-agent reinforcement learning approach with a given task hierarchy where communication is used to coordinate actions at higher levels and agents are assumed to be independent at lower levels. This work was limited to a multi-agent MDP model with (potentially costly) communication, making the learning problem challenging, but the planning problem is simpler than the full Dec-POMDP case.

Other approaches have considered identifying and exploiting independence between agents to limit reasoning about coordination and improve scalability. Approaches include general assumptions about agent independence like transition independent Dec-MDPs (Becker, Zilberstein, Lesser, & Goldman, 2004b) and factored models such as ND-POMDPs (Nair et al., 2005) as well as methods that consider coordination based on 'events' or states. Events which may require or allow interaction have been explored in Dec-MDPs (Becker, Lesser, & Zilberstein, 2004a) and (centralized) multi-robot systems (Messias, Spaan, & Lima, 2013). Other methods have considered locations or states where interaction is needed to improve scalability in planning (Spaan & Melo, 2008; Velagapudi et al., 2011) and learning (Melo & Veloso, 2011).

The work on independence assumes agents are always independent or coordinate using a fixed factorization, making it less general than an option-based approach. The work on event and state-based coordination focuses on a different type of domain knowledge: knowledge of states where coordination takes place. While this type of knowledge may be available, it may be easier to obtain and utilize procedural knowledge. The domain may therefore be easier to specify using macro-actions with different properties (such as independence or tight coordination), allowing planning to determine the necessary states for coordination. Furthermore, this type of state information could be used to define options for reaching these coordination points. Lastly, macro-actions could possibly be used in conjunction with previous methods, further improving scalability.

As mentioned in the introduction, we do not target scalability with respect to the number of agents. Several such methods have been developed that make various assumptions about agent abilities and policies (e.g., Sonu, Chen, & Doshi, 2015; Varakantham, Adulyasak, & Jaillet, 2014; Velagapudi et al., 2011; Oliehoek et al., 2013; Nguyen, Kumar, & Lau, 2017a, 2017b). Macro-action-based methods could potentially be incorporated into these methods to again increase scalability in terms of both the number of agent as well as the horizon and other problem variables.

There are several frameworks for multi-robot decision making in complex domains. For instance, behavioral methods have been studied for performing task allocation over time with loosely-coupled (Parker, 1998) or tightly-coupled (Stroupe, Ravichandran, & Balch,

2004) tasks. These are heuristic in nature and make strong assumptions about the type of tasks that will be completed.

Linear temporal logic (LTL) has also been used to specify robot behavior (Belta, Bicchi, Egerstedt, Frazzoli, Klavins, & Pappas, 2007; Loizou & Kyriakopoulos, 2004); from this specification, reactive controllers that are guaranteed to satisfy the specification can be derived. These methods are appropriate when the world dynamics can be effectively described non-probabilistically and when there is a useful characterization of the robot's desired behavior in terms of a set of discrete constraints. When applied to multiple robots, it is necessary to give each robot its own behavior specification. In contrast, our approach (probabilistically) models the *domain* and allows the planner to automatically optimize the robots' behavior.

Market-based approaches use traded value to establish an optimization framework for task allocation (Dias & Stentz, 2003; Gerkey & Matarić, 2004). These approaches have been used to solve real multi-robot problems (Kalra, Ferguson, & Stentz, 2005), but are largely aimed to tasks where the robots can communicate through a bidding mechanism.

Emery-Montemerlo et al. (Emery-Montemerlo, Gordon, Schneider, & Thrun, 2005) introduced a (cooperative) game-theoretic formalization of multi-robot systems which resulted in solving a Dec-POMDP. An approximate forward search algorithm was used to generate solutions, but because a (relatively) low-level Dec-POMDP was used scalability was limited. Their system also required synchronized execution by the robots.

8. Discussion

We have considered local options in this paper, but our framework could support other types of options. For example, we could consider options in which the policy is local but the initiation and termination sets are not—for example, initiation and termination could depend on the agent's history, or other agent's states. Generalizing a local option in this way retains the advantages described here, because the decision about which option to execute already requires coordination but executing the option itself does not. We could also use options with history-based policies, or define multi-agent options that control a subset of agents to complete a task. In general, we expect that an option will be useful for planning when its execution allows us to temporarily ignore some aspect of the original problem. For example, the option might be defined in a smaller state space (allowing us to ignore the full complexity of the problem), or use only observable information (allowing us to ignore the partially observable aspect of the problem), or involve a single agent or a subset of agents communicating (allowing us to ignore the decentralized aspect of the problem).

We can gain additional benefits by exploiting known structure in the multi-agent problem. For instance, most controllers only depend on locally observable information and do not require coordination. For example, consider a controller that navigates to a waypoint. Only local information is required for navigation—the robot may detect other robots but their presence does not change its objective, and it simply moves around them—but choosing the target waypoint likely requires the planner to consider the locations and actions of all robots. Macro-actions with independent execution allow coordination decisions to be made only when necessary (i.e., when choosing macro-actions) rather than at every time step. Because MacDec-POMDPs are built on top of Dec-POMDPs, macro-action choice may depend on history, but during execution macro-actions may depend only on a single observation or on any number of steps of history, or even represent the actions of a set of robots. That is, macro-actions are very general and can be defined in such a way to take advantage of the knowledge available to the robots during execution.

We have so far assumed that the agent is given an appropriate set of macro-actions with which to plan. In all of our domains, there were quite natural choices for macro-actions and macro-observations (e.g., navigating to depots and observing that you are in a depot along with its contents), but such natural representations are not always present. Research on skill discovery (McGovern & Barto, 2001) has attempted to devise methods by which a single agent can instead acquire an appropriate set of options autonomously, through interaction with its (fully observable) environment. While some of these methods may be directly applicable, the characteristics of the partially observable, multi-agent case also offer new opportunities for skill discovery. For example, we may wish to synthesize skills that collapse uncertainty across multiple agents, perform coordinated multi-agent actions, communicate essential state information, or allow agents to synchronize and replan. Related work has begun to explore some of these topics (Omidshafiei et al., 2017, 2017a), but many open questions remain.

In terms of multi-robot domains, we demonstrated macro-action-based approaches on multiple other domains with limited sensing and communication. These other domains included a logistics (beer delivery) domain, where two robots must efficiently find out about and service beer orders in cooperation with a 'picker/bartender' robot, which can retrieve items (Amato, Konidaris, Anders, Cruz, How, & Kaelbling, 2015; Amato et al., 2017), a package delivery domain, where a group of aerial robots must retrieve and deliver packages from base locations to delivery locations while dealing with limited battery life (Omidshafiei, Agha-mohammadi, Amato, & How, 2015; Omidshafiei, Agha-mohammadi, Amato, Liu, How, & Vian, 2016; Omidshafiei et al., 2017a, 2017) as well as an adversarial domain in which a team of robots is playing capture the flag against another team of robots (Hoang, Xiao, Sivakumar, Amato, & How, 2018).

Also, our results have shown that the use of macro-actions can significantly improve scalability—for example, by allowing us to use larger grids with the same set of agents and obstacles in the NAMO problem (see Figure 8). However, in such cases—where the state space grows but the number of agents and significant interactions does not—we should in principle be able to deal with *any* size grid with no increase in computation time, because the size of the grid is irrelevant to the coordination aspects of the problem. This does not occur in the work presented here because we plan in the original state space; methods for constructing a more abstract *task-level* representation (Konidaris, Kaelbling, & Lozano-Perez, 2018) could provide further performance improvements.

It is also worth noting that our approach can incorporate state-of-the-art methods for solving more restricted scenarios as options. The widespread use of techniques for solving restricted robotics scenarios has led to a plethora of usable algorithms for specific problems, but no way to combine these in more complex scenarios. Our approach can build on the large amount of research in single and multi-robot systems that has gone into solving difficult problems such as navigation in a formation (Balch & Arkin, 1998), cooperative transport of an object (Kube & Bonabeau, 2000), coordination with signaling (Beckers, Holland, & Deneubourg, 1994) or communication under various limitations (Rekleitis, Lee-Shue, New,

& Choset, 2004). The solutions to these problems could be represented as macro-actions in our framework, building on existing research to solve even more complex multi-robot problems.

This paper focused on (sample-based) planning using macro-actions, but learning could also be used to generate policies over macro-actions. In particular, other work developed a method that learns policies using only high-level macro-action trajectories (macro-actions and macro-observations) (Liu, Amato, Anesta, Griffith, & How, 2016). As a result, the methods don't need any models and are applicable in cases where data is difficult or costly to obtain (e.g., human demonstrations, elaborate training exercises). Our experiments showed that the methods can also produce very high-quality solutions, even outperforming and improving upon hand-coded 'expert' solutions with a small amount of data. We also improved upon and tested these approaches in a multi-robot search and rescue problem (Liu, Sivakumar, Omidshafiei, Amato, & How, 2017). In general, using macro-actions with other multi-agent reinforcement learning methods (including the popular deep methods e.g., Foerster, Assael, de Freitas, & Whiteson, 2016; Omidshafiei, Pazis, Amato, How, & Vian, 2017b; Lowe, Wu, Tamar, Harb, Abbeel, & Mordatch, 2017; Rashid, Samvelyan, Schroeder, Farquhar, Foerster, & Whiteson, 2018; Palmer, Tuyls, Bloembergen, & Savani, 2018; Omidshafiei, Kim, Liu, Tesauro, Riemer, Amato, Campbell, & How, 2019) could be a promising way of improving performance, while allowing asynchronous action execution.

Finally, while this paper focused on dynamic programming (Hansen et al., 2004; Seuken & Zilberstein, 2007b) and direct policy search methods (Oliehoek et al., 2008), forward search methods (Oliehoek et al., 2013; Szer et al., 2005; Oliehoek et al., 2008, 2009; Dibangoye et al., 2016) are likely to perform well when using MacDec-POMDPs. When building up policies from the last step, as in dynamic programming, adding macro-actions to the beginning of a tree changes when the macro-actions deeper down the tree will be completed. In forward search methods, actions are added to the leaves of the tree, leaving the completion times for previous macro-actions in the policy (those at earlier heights) the same. We have not explored such search methods for MacDec-POMDPs, but they appear to be promising.

9. Conclusion

We presented a new formulation for representing decentralized decision-making problems under uncertainty using higher-level macro-actions (modeled as options), rather than primitive (single-step) actions. We called this framework the macro-action Dec-POMDP (MacDec-POMDP). Because our macro-action model is built on top of the Dec-POMDP framework, Dec-POMDP algorithms can be extended to solve problems with macro-actions while retaining agent coordination. We focused on local options, which allow us to reason about coordination only when deciding which option to execute. Our results have demonstrated that high-quality results can be achieved on current benchmarks, and that very large problems can be effectively modeled and solved this way. As such, our macro-action framework represents a promising approach for scaling multi-agent planning under uncertainty to real-world problem sizes.

We also have demonstrated that complex multi-robot domains can be solved with Dec-POMDP-based methods. The MacDec-POMDP model is expressive enough to capture multi-robot systems of interest, but also simple enough to be feasible to solve in practice. Our results show that a general purpose MacDec-POMDP planner can generate cooperative behavior for complex multi-robot domains with task allocation, direct communication, and signaling behavior emerging automatically as properties of the solution for the given problem model. Because all cooperative multi-robot problems can be modeled as Dec-POMDPs, MacDec-POMDPs represent a powerful tool for automatically trading-off various costs, such as time, resource usage and communication while considering uncertainty in the dynamics, sensors and other robot information. These approaches have great potential to lead to automated solution methods for general probabilistic multi-robot coordination problems with heterogeneous robots in complex, uncertain domains.

More generally, this work opens the door to many research questions about representing and solving multi-agent problems hierarchically. Promising avenues for future work include exploring different types of options, further work on reinforcement learning for either generating options or policies over options, and developing more scalable solution methods that exploit domain and hierarchical structure. One example of such structure would be the use of a factored reward function (Nair et al., 2005) which allows more efficient policy generation and evaluation.

Acknowledgements

We would like to thank Matthijs Spaan and Feng Wu for providing results as well as Ari Anders, Gabriel Cruz, and Christopher Maynor for their help with the robot experiments. Research supported in part by NSF project #1664923, ONR MURI project #N000141110688, DARPA YFA D15AP00104, AFOSR YIP FA9550-17-1-0124, and NIH R01MH109177.

References

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

Amato, C., Bonet, B., & Zilberstein, S. (2010b). Finite-state controllers based on Mealy machines for centralized and decentralized POMDPs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 1052–1058.

Amato, C., Chowdhary, G., Geramifard, A., Ure, N. K., & Kochenderfer, M. J. (2013). Decentralized control of partially observable Markov decision processes. In *Proceedings of the IEEE Conference on Decision and Control*, pp. 2398–2405.

Amato, C., Dibangoye, J. S., & Zilberstein, S. (2009). Incremental policy generation for finite-horizon DEC-POMDPs. In *Proceedings of the International Conference on Automated Planning and Scheduling*, pp. 2–9.

Amato, C., Konidaris, G. D., Anders, A., Cruz, G., How, J. P., & Kaelbling, L. P. (2015). Policy search for multi-robot coordination under uncertainty. In *Proceedings of the Robotics: Science and Systems Conference*.

Amato, C., Konidaris, G. D., Anders, A., Cruz, G., How, J. P., & Kaelbling, L. P. (2017). Policy search for multi-robot coordination under uncertainty. *The International Journal of Robotics Research*.

Aras, R., Dutech, A., & Charpillet, F. (2007). Mixed integer linear programming for exact finite-horizon planning in decentralized POMDPs. In *Proceedings of the International Conference on Automated Planning and Scheduling*, pp. 18–25.

Balch, T., & Arkin, R. C. (1998). Behavior-based formation control for multi-robot teams. *IEEE Transactions on Robotics and Automation*, *14*(6), 926–939.

Barto, A., & Mahadevan, S. (2003). Recent advances in hierarchical reinforcement learning. *Discrete Event Dynamic Systems*, *13*, 41–77.

Becker, R., Lesser, V., & Zilberstein, S. (2004a). Decentralized Markov Decision Processes with Event-Driven Interactions. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*, pp. 302–309.

Becker, R., Zilberstein, S., Lesser, V., & Goldman, C. V. (2004b). Solving transition-independent decentralized Markov decision processes. *Journal of Artificial Intelligence Research*, *22*, 423–455.

Beckers, R., Holland, O., & Deneubourg, J.-L. (1994). From local actions to global tasks: Stigmergy and collective robotics. In *Artificial life IV*, Vol. 181, p. 189.

Belta, C., Bicchi, A., Egerstedt, M., Frazzoli, E., Klavins, E., & Pappas, G. J. (2007). Symbolic planning and control of robot motion [grand challenges of robotics]. *Robotics & Automation Magazine, IEEE*, *14*(1), 61–70.

Bernstein, D. S., Amato, C., Hansen, E. A., & Zilberstein, S. (2009). Policy iteration for decentralized control of Markov decision processes. *Journal of Artificial Intelligence Research*, *34*, 89–132.

Bernstein, D. S., Givan, R., Immerman, N., & Zilberstein, S. (2002). The complexity of decentralized control of Markov decision processes. *Mathematics of Operations Research*, *27*(4), 819–840.

Bohren, J. (2010). SMACH. `http://wiki.ros.org/smach/`.

Boularias, A., & Chaib-draa, B. (2008). Exact dynamic programming for decentralized POMDPs with lossless policy compression. In *Proceedings of the International Conference on Automated Planning and Scheduling*.

Dias, M. B., & Stentz, A. T. (2003). A comparative study between centralized, market-based, and behavioral multirobot coordination approaches. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, Vol. 3, pp. 2279 – 2284.

Dibangoye, J. S., Amato, C., Buffet, O., & Charpillet, F. (2013). Optimally solving Dec-POMDPs as continuous-state MDPs. In *Proceedings of the International Joint Conference on Artificial Intelligence*.

Dibangoye, J. S., Amato, C., Buffet, O., & Charpillet, F. (2016). Optimally solving Dec-POMDPs as continuous-state MDPs. *Journal of Artificial Intelligence Research*, *55*, 443–497.

Dibangoye, J. S., Amato, C., Doniec, A., & Charpillet, F. (2013). Producing efficient error-bounded solutions for transition independent decentralized MDPs. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*.

Dietterich, T. G. (2000). Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research*, *13*, 227–303.

Emery-Montemerlo, R., Gordon, G., Schneider, J., & Thrun, S. (2005). Game theoretic control for robot teams. In *Proceedings of the International Conference on Robotics and Automation*, pp. 1163–1169.

Foerster, J., Assael, I. A., de Freitas, N., & Whiteson, S. (2016). Learning to communicate with deep multi-agent reinforcement learning. In *Advances in Neural Information Processing Systems*, pp. 2137–2145.

Gerkey, B. P., & Matarić, M. J. (2004). A formal analysis and taxonomy of task allocation in multi-robot systems. *International Journal of Robotics Research*, *23*(9), 939–954.

Ghavamzadeh, M., Mahadevan, S., & Makar, R. (2006). Hierarchical multi-agent reinforcement learning. *Journal of Autonomous Agents and Multi-Agent Systems*, *13*(2), 197–229.

Hansen, E. A., Bernstein, D. S., & Zilberstein, S. (2004). Dynamic programming for partially observable stochastic games. In *Proceedings of the National Conference on Artificial Intelligence*, pp. 709–715.

He, R., Brunskill, E., & Roy, N. (2011). Efficient planning under uncertainty with macro-actions. *Journal of Artificial Intelligence Research*, 523–570.

Hoang, T. N., Xiao, Y., Sivakumar, K., Amato, C., & How, J. (2018). Near-optimal adversarial policy switching for decentralized asynchronous multi-agent systems. In *Proceedings of the International Conference on Robotics and Automation*.

Horling, B., & Lesser, V. (2004). A survey of multi-agent organizational paradigms. *The Knowledge Engineering Review*, *19*(4), 281–316.

Kaelbling, L. P., Littman, M. L., & Cassandra, A. R. (1998). Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, *101*, 1–45.

Kalra, N., Ferguson, D., & Stentz, A. T. (2005). Hoplites: A market-based framework for planned tight coordination in multirobot teams. In *Proceedings of the International Conference on Robotics and Automation*, pp. 1170 – 1177.

Konidaris, G., & Barto, A. G. (2007). Building portable options: Skill transfer in reinforcement learning. *Proceedings of the International Joint Conference on Artificial Intelligence*, *7*, 895–900.

Konidaris, G., Kaelbling, L. P., & Lozano-Perez, T. (2018). From skills to symbols: Learning symbolic representations for abstract high-level planning. *Journal of Artificial Intelligence Research*, *61*, 215–289.

Konidaris, G. D., & Barto, A. G. (2009). Skill discovery in continuous reinforcement learning domains using skill chaining. In *Advances in Neural Information Processing Systems 22*, pp. 1015–1023.

Kube, C. R., & Bonabeau, E. (2000). Cooperative transport by ants and robots. *Robotics and Autonomous Systems*, *30*(1-2), 85–101.

Kumar, A., Mostafa, H., & Zilberstein, S. (2016). Dual formulations for optimizing dec-pomdp controllers. In *Proceedings of the International Conference on Automated Planning and Scheduling*.

Kumar, A., & Zilberstein, S. (2010). Point-based backup for decentralized POMDPs: complexity and new algorithms. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*, pp. 1315–1322.

Kumar, A., Zilberstein, S., & Toussaint, M. (2015). Probabilistic inference techniques for scalable multiagent decision making. *Journal of Artificial Intelligence Research*, *53*(1), 223–270.

Lim, Z., Sun, L., & Hsu, D. J. (2011). Monte Carlo value iteration with macro-actions. In *Advances in Neural Information Processing Systems*, pp. 1287–1295.

Liu, M., Amato, C., Anesta, E., Griffith, J. D., & How, J. P. (2016). Learning for decentralized control of multiagent systems in large partially observable stochastic environments. In *Proceedings of the AAAI Conference on Artificial Intelligence*.

Liu, M., Amato, C., Liao, X., Carin, L., & How, J. P. (2015). Stick-breaking policy learning in Dec-POMDPs. In *Proceedings of the International Joint Conference on Artificial Intelligence*.

Liu, M., Sivakumar, K., Omidshafiei, S., Amato, C., & How, J. P. (2017). Learning for multi-robot cooperation in partially observable stochastic environments with macro-actions. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 1853–1860.

Loizou, S. G., & Kyriakopoulos, K. J. (2004). Automatic synthesis of multi-agent motion tasks based on ltl specifications. In *Decision and Control, 2004. CDC. 43rd IEEE Conference on*, Vol. 1, pp. 153–158. IEEE.

Lowe, R., Wu, Y., Tamar, A., Harb, J., Abbeel, O. P., & Mordatch, I. (2017). Multi-agent actor-critic for mixed cooperative-competitive environments. In *Advances in Neural Information Processing Systems*, pp. 6379–6390.

McGovern, A., & Barto, A. G. (2001). Automatic discovery of subgoals in reinforcement learning using diverse density. In *Proceedings of the Eighteenth International Conference on Machine Learning*, pp. 361–368.

Melo, F., & Veloso, M. (2011). Decentralized MDPs with sparse interactions. *Artificial Intelligence*.

Messias, J. V., Spaan, M. T. J., & Lima, P. U. (2013). GSMDPs for multi-robot sequential decision-making. In *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence*, pp. 1408–1414.

Nair, R., Varakantham, P., Tambe, M., & Yokoo, M. (2005). Networked distributed POMDPs: a synthesis of distributed constraint optimization and POMDPs. In *Proceedings of the National Conference on Artificial Intelligence*.

Nguyen, D. T., Kumar, A., & Lau, H. C. (2017a). Collective multiagent sequential decision making under uncertainty. In *Proceedings of the AAAI Conference on Artificial Intelligence*.

Nguyen, D. T., Kumar, A., & Lau, H. C. (2017b). Policy gradient with value function approximation for collective multiagent planning. In *Advances in Neural Information Processing Systems*, pp. 4322–4332.

Oliehoek, F. A. (2012). Decentralized POMDPs. In Wiering, M., & van Otterlo, M. (Eds.), *Reinforcement Learning: State of the Art*, Vol. 12 of *Adaptation, Learning, and Optimization*, pp. 471–503. Springer Berlin Heidelberg.

Oliehoek, F. A., & Amato, C. (2016). *A Concise Introduction to Decentralized POMDPs*. Springer.

Oliehoek, F. A., Kooi, J. F., & Vlassis, N. (2008). The cross-entropy method for policy search in decentralized POMDPs. *Informatica*, *32*, 341–357.

Oliehoek, F. A., Spaan, M. T. J., Amato, C., & Whiteson, S. (2013). Incremental clustering and expansion for faster optimal planning in Dec-POMDPs. *Journal of Artificial Intelligence Research*, *46*, 449–509.

Oliehoek, F. A., Spaan, M. T. J., & Vlassis, N. (2008). Optimal and approximate Q-value functions for decentralized POMDPs. *Journal of Artificial Intelligence Research*, *32*, 289–353.

Oliehoek, F. A., Whiteson, S., & Spaan, M. T. J. (2009). Lossless clustering of histories in decentralized POMDPs. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*.

Oliehoek, F. A., Whiteson, S., & Spaan, M. T. J. (2013). Approximate solutions for factored Dec-POMDPs with many agents. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*.

Omidshafiei, S., Agha-mohammadi, A., Amato, C., & How, J. P. (2015). Decentralized control of partially observable Markov decision processes using belief space macro-actions. In *Proceedings of the International Conference on Robotics and Automation*, pp. 5962–5969.

Omidshafiei, S., Agha-mohammadi, A., Amato, C., & How, J. P. (2017). Decentralized control of multi-robot partially observable Markov decision processes using belief space macro-actions. *The International Journal of Robotics Research*.

Omidshafiei, S., Agha-mohammadi, A., Amato, C., Liu, S.-Y., How, J. P., & Vian, J. (2016). Graph-based cross entropy method for solving multi-robot decentralized POMDPs. In *Proceedings of the International Conference on Robotics and Automation*.

Omidshafiei, S., Kim, D.-K., Liu, M., Tesauro, G., Riemer, M., Amato, C., Campbell, M., & How, J. (2019). Learning to teach in cooperative multiagent reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*.

Omidshafiei, S., Liu, S.-Y., Everett, M., Lopez, B., Amato, C., Liu, M., How, J. P., & Vian., J. (2017a). Semantic-level decentralized multi-robot decision-making using probabilistic macro-observations. In *Proceedings of the International Conference on Robotics and Automation*.

Omidshafiei, S., Pazis, J., Amato, C., How, J. P., & Vian, J. (2017b). Deep decentralized multi-task multi-agent reinforcement learning under partial observability. In *Proceedings of the International Conference on Machine Learning*.

Pajarinen, J. K., & Peltonen, J. (2011). Periodic finite state controllers for efficient POMDP and DEC-POMDP planning. In Shawe-Taylor, J., Zemel, R., Bartlett, P., Pereira, F., & Weinberger, K. (Eds.), *Advances in Neural Information Processing Systems 24*, pp. 2636–2644.

Palmer, G., Tuyls, K., Bloembergen, D., & Savani, R. (2018). Lenient multi-agent deep reinforcement learning. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*, pp. 443–451.

Parker, L. E. (1998). ALLIANCE: An architecture for fault tolerant multirobot cooperation. *IEEE Transactions on Robotics and Automation*, *14*(2), 220–240.

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

Quigley, M., Conley, K., Gerkey, B. P., Faust, J., Foote, T., Leibs, J., Wheeler, R., & Ng, A. Y. (2009). ROS: an open-source robot operating system. In *ICRA Workshop on Open Source Software*.

Rashid, T., Samvelyan, M., Schroeder, C., Farquhar, G., Foerster, J., & Whiteson, S. (2018). QMIX: Monotonic value function factorisation for deep multi-agent reinforcement learning. In *Proceedings of the International Conference on Machine Learning*, pp. 4295–4304.

Rekleitis, I., Lee-Shue, V., New, A. P., & Choset, H. (2004). Limited communication, multi-robot team based coverage. In *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, Vol. 4, pp. 3462–3468. IEEE.

Seuken, S., & Zilberstein, S. (2007a). Improved memory-bounded dynamic programming for decentralized POMDPs. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, pp. 344–351.

Seuken, S., & Zilberstein, S. (2007b). Memory-bounded dynamic programming for DEC-POMDPs. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pp. 2009–2015.

Silver, D., & Ciosek, K. (2012). Compositional planning using optimal option models. In *Proceedings of the International Conference on Machine Learning*.

Sonu, E., Chen, Y., & Doshi, P. (2015). Individual planning in agent populations: Exploiting anonymity and frame-action hypergraphs. In *Proceedings of the International Conference on Automated Planning and Scheduling*.

Spaan, M. T. J., & Melo, F. S. (2008). Interaction-driven Markov games for decentralized multiagent planning under uncertainty. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*, pp. 525–532.

Stilman, M., & Kuffner, J. (2005). Navigation among movable obstacles: Real-time reasoning in complex environments. *International Journal on Humanoid Robotics*, *2*(4), 479–504.

Stone, P., Sutton, R. S., & Kuhlmann, G. (2005). Reinforcement learning for robocup soccer keepaway. *Adaptive Behavior*, *13*(3), 165–188.

Stroupe, A. W., Ravichandran, R., & Balch, T. (2004). Value-based action selection for exploration and dynamic target observation with robot teams. In *Proceedings of the International Conference on Robotics and Automation*, Vol. 4, pp. 4190–4197. IEEE.

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

Szer, D., Charpillet, F., & Zilberstein, S. (2005). MAA*: A heuristic search algorithm for solving decentralized POMDPs. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*.

Theocharous, G., & Kaelbling, L. P. (2003). Approximate planning in POMDPs with macro-actions. In *Advances in Neural Information Processing Systems*.

Thrun, S., Burgard, W., & Fox, D. (2005). *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press.

Varakantham, P., Adulyasak, Y., & Jaillet, P. (2014). Decentralized stochastic planning with anonymity in interactions. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 2505–2512.

Velagapudi, P., Varakantham, P. R., Sycara, K., & Scerri, P. (2011). Distributed model shaping for scaling to decentralized POMDPs with hundreds of agents. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*, pp. 955–962.

Wu, F., Zilberstein, S., & Chen, X. (2010a). Point-based policy generation for decentralized POMDPs. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*, pp. 1307–1314.

Wu, F., Zilberstein, S., & Chen, X. (2010b). Rollout sampling policy iteration for decentralized POMDPs. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, pp. 666–673.

Wu, F., Zilberstein, S., & Jennings, N. R. (2013). Monte-carlo expectation maximization for decentralized POMDPs. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pp. 397–403. AAAI Press.