Article

The International Journal of
Robotics Research
2016, Vol. 35(14) 1760–1778

DOI: 10.1177/0278364916679611
ijr.sagepub.com

Policy search for multi-robot coordination under uncertainty

Christopher Amato[1], George Konidaris[2], Ariel Anders[3], Gabriel Cruz[3], Jonathan P How[4] and Leslie P Kaelbling[3]

Abstract

We introduce a principled method for multi-robot coordination based on a general model (termed a MacDec-POMDP) of multi-robot cooperative planning in the presence of stochasticity, uncertain sensing, and communication limitations. A new MacDec-POMDP planning algorithm is presented that searches over policies represented as finite-state controllers, rather than the previous policy tree representation. Finite-state controllers can be much more concise than trees, are much easier to interpret, and can operate over an infinite horizon. The resulting policy search algorithm requires a substantially simpler simulator that models only the outcomes of executing a given set of motor controllers, not the details of the executions themselves and can solve significantly larger problems than existing MacDec-POMDP planners. We demonstrate significant performance improvements over previous methods and show that our method can be used for actual multi-robot systems through experiments on a cooperative multi-robot bartending domain.

Keywords

AI reasoning methods, autonomous agents, distributed robot systems

1. Introduction

In order to fulfill the potential of increasingly capable and affordable robot hardware, effective methods for controlling robot teams must be developed. Although many algorithms have been proposed for multi-robot problems, the vast majority are specialized methods engineered to match specific team or problem characteristics. Progress in more general settings requires the specification of a model class that captures the core challenges of controlling multi-robot teams in a generic fashion. Such general models—in particular, the Markov decision process (Puterman, 1994) and partially observable Markov decision process (Cassandra et al., 1994)—have led to significant progress in single-robot settings through standardized models that enable empirical comparisons between general planners that optimize a common metric.

Decentralized partially observable Markov decision processes (or *Dec-POMDPs* (Bernstein et al., 2002)) are the natural extension of such frameworks to the multi-robot case—modeling multi-agent coordination problems in the presence of stochasticity, uncertain sensing and action, and communication limitations. Unfortunately, Dec-POMDPs are exactly solvable only for very small problems. The search for tractable approximations led to the recent introduction of the *MacDec-POMDP* model (Amato et al., 2014). MacDec-POMDPs include temporally extended macro-actions that naturally model robot motor controllers that may require multiple time-steps to execute (e.g. navigating to a waypoint, lifting an object, or waiting for another robot) as opposed to low-level control inputs that must each last a fixed time interval. Planning then takes place at the level of selecting controllers to execute, rather than sequencing low-level motions, and MacDec-POMDP solution methods can scale up to reasonably realistic problems; for example, solving a multi-robot warehousing problem orders of magnitude larger than those solvable by previous methods (Amato et al., 2015). General-purpose planners based on MacDec-POMDPs have the potential to replace the abundance of ad-hoc multi-robot algorithms for specific task scenarios with a single precise and generic formulation of cooperative multi-robot problems that is powerful enough to include (and naturally combine) all existing cooperative scenarios.

[1]College of Computer and Information Science, Northeastern University, Boston
[2]Computer Science Department, Brown University, Providence
[3]CSAIL, MIT, Cambridge, USA
[4]LIDS, MIT, Cambridge, USA

Corresponding author:
Christopher Amato, 360 Huntington Ave, College of Computer and Information Science, Northeastern University, Boston, MA 02115.
Email: camato@ccs.neu.edu

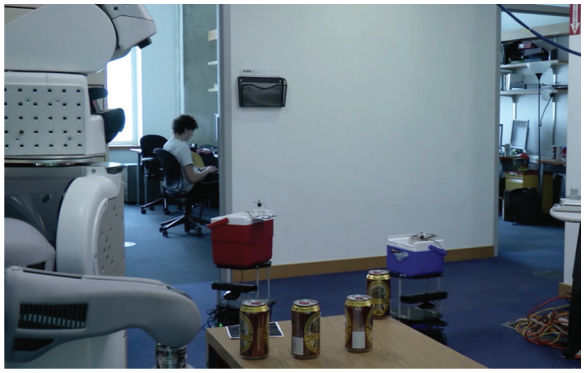

Fig. 1. The bartender and waiters domain which will be used in the experiments: Two TurtleBots and one PR2 must coordinate to deliver drinks as quickly as possible.

Unfortunately, existing MacDec-POMDP planners have two critical flaws. First, even though using macro-actions drastically increases the size of problems that can be solved, planning time still scales poorly with the horizon (i.e. plan length). Second, current methods assume that the underlying (primitive) problem is discrete, and that a complete low-level model of that problem is available. These difficulties significantly limit the applicability of existing MacDec-POMDP planners to robotics applications.

This paper introduces an extended model and a new MacDec-POMDP planning algorithm, which we call MacDec-POMDP heuristic search (MDHS). MDHS searches over policies represented as finite-state controllers, rather than the currently used policy trees. Finite-state controllers are often much more concise than trees, are easier to interpret, and can operate for an infinite horizon. Our model and MDHS only require a description of the problem at the macro-action level—at the level of modeling the outcome of executing given motor controllers, not the details of execution itself—substantially reducing the knowledge required for planning, and therefore the modeling effort required to apply the method to real-world robotics problems. We show that MDHS can solve significantly larger problems than existing MacDec-POMDP planners (and by extension, all existing Dec-POMDP planners), and demonstrate its application to a cooperative multi-robot bartending task, showing that MDHS can automatically optimize solutions to multi-robot problems from a high-level specification.

2. Motivating problem

As a motivating experimental domain we consider a heterogeneous multi-robot problem, shown in Figure 1. The robot team consists of a PR2 bartender and two TurtleBot waiters. There are is a bar area and three rooms in which people can order drinks from the waiters. Our goal is to bring drinks to the rooms with orders as efficiently as possible; since the robots cannot take orders until they visit a room, they must coordinate to service all three rooms quickly. We impose communication limitations so the robots cannot communicate unless they are in close range. As a result, the robots must make decisions based on their own sensor and communication information, reasoning about the status and behavior of the other robots. This is a challenging task with stochasticity in ordering, navigation, picking, and placing objects as well as partial observability in the orders and the location and status of the other robots.

We model this domain as a MacDec-POMDP and introduce a planning algorithm *capable of automatically generating controllers for the robots (in the form of finite-state machines) that collectively maximize team utility*. This problem involves aspects of communication, task allocation, and cooperative navigation—tasks for which specialized algorithms exists—but modeling it as a MacDec-POMDP allows us to automatically generate controllers that express and combine aspects of these behaviors—without specifying them in advance—while trading off their costs in a principled way.

3. Background

We first discuss Dec-POMDPs and then we present previous work on using macro-actions in Dec-POMDPs.

3.1. Dec-POMDPs

Our aim is to control a group of robots interacting with an environment in order to cooperatively solve a problem. At each time step t, each robot i must select an action to execute from its own set of (possibly real-valued and multivariate) available actions A_i, after which the robots collectively obtain a *single* reward, r^t. For example, consider a team consisting of a number of quadrotors and a number of ground robots, attempting to search an area for a particular object. At each time step, each robot must choose what to do (for the quadrotors: fly in a particular direction, at a particular height, and perhaps shine a spotlight; for the ground robots: drive in a particular direction with a specific camera angle) so as to most efficiently locate the object. The actions available to the robots may differ (because they are different robots, or because their immediate environment affords different actions), but they share a collective single reward (equivalently, cost or utility) function that expresses their joint goal and makes the problem cooperative.

We write the robot's collective action space as $A = A_1 \times A_2 \times \cdots \times A_{|I|}$ and we can construct a state space S such that the whole problem obeys the Markov property: it has a transition function $T(s^{t+1}|s^t, a^t)$ expressing the environmental dynamics and a reward function $R(s^t, a^t)$, and both depend only the state at times t and the collective action $a^t \in A$. Because we will assume that transition and reward functions are stationary with respect to time, we will often drop the time step superscript. Given such a problem, the goal of planning is to find a policy π mapping states to collective actions, so as to maximize the sum of rewards obtained

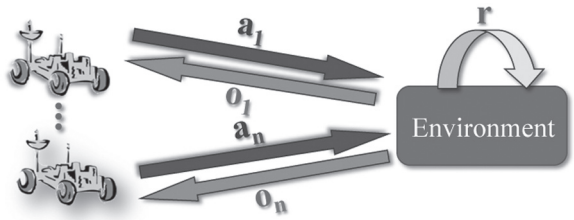

Fig. 2. Depiction of an n-agent Dec-POMDP with actions and observations for each agent, but a single joint reward.

over time. This formalization is known as a *multiagent Markov decision process* or MMDP (Boutilier, 1999), and it is intended to model multi-robot systems where actions are selected by a single centralized decision-maker, and where access to the Markov state space S is available (i.e. fully observable).

However, these two assumptions are often unrealistic in real multi-robot problems—the almost instantaneous communication required for centralized control and global state estimation is frequently impractical or impossible. In such cases, rather than constructing a single global state and making a single collective action-selection decision, we must instead find a *decentralized solution*, where each robot must act based on its own (often quite limited) history of observations about the world. To formalize each robot's limited view of the world we must have a model of how each robot's sensors react to possible states of the problem. We model this using an *observation function*, O, which maps the global state s to a distribution over each agent i's sensor space, Z_i. The goal of planning is now to find a *policy for each robot*—based only on its past observations—such that the resulting *joint policy* maximizes the expected sum of rewards.

This model is known as a *Decentralized Partially Observable Markov Decision Processes* (Dec-POMDP, depicted in Figure 2) and can be formally described by a tuple $\langle I, S, \{A_i\}, T, R, \{Z_i\}, O, h\rangle$, 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 $A = A_1 \times A_2 \times \cdots \times A_{|I|}$ the set of joint actions;
- 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 next state $s' \in S$ when the actions $a \in A$ are taken by the agents (i.e. $T(s, a, s') = \Pr(s'|a, s)$);
- R is a reward function: $R : S \times A \rightarrow \mathbb{R}$, the immediate reward for being in state $s \in S$ and taking the actions $a \in A$;
- Z_i is a finite set of observations for each agent, i, with $Z = Z_1 \times Z_2 \times \cdots \times Z_{|I|}$ the set of joint observations;
- O is an observation probability function: $O : Z \times A \times S \rightarrow [0, 1]$, the probability of seeing observations $o \in Z$ given actions $a \in A$ were taken which results in state $s' \in S$ (i.e. $O(o, a, s') = \Pr(o|s', a)$);
- h is the number of (possibly infinite) steps until termination, called the horizon.

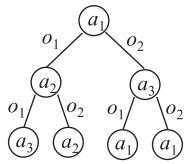

Fig. 3. Depiction of a single agent's policy tree with discrete observations listed as o_i and actions listed as a_i.

The model described above is very general. For example, it can describe multi-robot scenarios involving communication—the act of emitting a particular signal is modeled as an action, the presence of that signal in the environment is modeled as hidden state, and the receipt of that signal (perhaps with some delay, sensor noise, or limited range) is modeled using each agent's observation function. However, the model description itself does not tell each robot what a signal means, or how to react to any particular signal—or indeed an observation of any type—it might receive. This must be encoded in each robot's individual *control policy*, and it is the role of the planner to construct such a policy for each agent.

Formally, therefore, a solution to a Dec-POMDP is a *joint policy*—a set of policies, one for each agent. Because the full state is not directly observed, it is often beneficial for each agent to remember a history of its observations. A *local policy* for agent i is a mapping from local observation histories to actions, $H_i^O \rightarrow A_i$. Because the system state depends on the behavior of all agents, it is typically not possible to estimate the system state (i.e. calculate a belief state) from the history of a single agent, as is often done in POMDPs.

Since a policy is a function of history, rather than of a directly observed state (or a calculated belief state), it is typically represented explicitly. The most common representation is a policy tree (as seen in Figure 3), where the vertices indicate actions to execute and the edges indicate transitions conditioned on an observation (with the history represented as the current path in the tree).

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

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

which is the expected sum of rewards for the agents given the action prescribed by the policy at each step until the horizon is reached. In the finite-horizon case, the discount factor, γ, is typically 1. In the infinite-horizon case, the discount factor $\gamma \in [0, 1)$ is included to maintain a finite sum and $h = \infty$. An *optimal policy* beginning at state s is $\pi^*(s) = \mathrm{argmax}_{\pi} V^{\pi}(s)$.

When modeling multi-robot scenarios, it is often reasonable to separate planning and execution: execution must be decentralized, but we can often perform centralized planning prior to releasing the robot team into the environment. Dec-POMDP solution methods therefore typically assume that the set of policies is generated in a centralized manner, but executed in a decentralized manner based on each agent's histories.

Although Dec-POMDPs have been widely studied, optimal (and boundedly optimal) methods do not scale to large problems, while approximate methods 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). For additional details on these methods, we refer the reader to relevant surveys (Oliehoek and Amato, 2016; Amato et al., 2013; Oliehoek, 2012).

3.2. Macro-actions for Dec-POMDPs

As described above, Dec-POMDPs typically require synchronous decision-making: every agent chooses an action to execute, and then executes it in a single time step. When each agent is a robot, this restriction is problematic for two reasons. First, the actions available in many robot systems are controllers (e.g. for waypoint navigation, grasping an object, waiting for a signal), and planning consists of sequencing the execution of those controllers. Each controller will require different amounts of time to execute, so synchronous decision-making requires waiting until all robots have completed their controller execution (and achieved common knowledge of this fact). This is inefficient and may not even be possible in some cases (e.g. when controlling airplanes or underwater vehicles that cannot stay in place). Second, the planning complexity of a Dec-POMDP is doubly exponential in the horizon. A planner that reasons about all robots' possible policies at every time step will only ever be able to make very short plans.

The Dec-POMDP model was therefore recently extended to plan using temporally extended actions, or macro-actions (Amato et al., 2014) (hence the MacDec-POMDP model). The macro-actions are intended to model higher-level robot controllers that execute by choosing low-level actions (like actuating motors) and take several time steps to execute. The resulting formulation uses higher-level planning to compute near-optimal solutions for problems with significantly longer horizons by extending the MDP-based options framework (Sutton et al., 1999) to Dec-POMDPs by using macro-actions, m_i, that execute a policy in a low-level Dec-POMDP from states that satisfy its initial conditions, until a terminal condition is met.

Note that this extension is not straightforward in the multi-agent case due to the resulting asynchronous nature of decision-making. While decision-making in the single-agent case can take place when the agent terminates a macro-action, in the multi-agent case, decision-making needs to take place only for agents that terminate. Furthermore, we must keep track of the progress of all agents in executing their macro-actions to properly evaluate solutions and allow other agents to continue their macro-actions even if some agents terminate theirs.

To simplify evaluation and reasoning about completion times, previous work assumes policies over macro-actions can be executed in a lower-level Dec-POMDP. That is, a Dec-POMDP with macro-actions is defined as a Dec-POMDP where we also assume M_i represents a finite set of macro-actions for each robot, i, with $M = M_1 \times M_2 \times \cdots \times M_{|I|}$ the set of joint macro-actions (Amato et al., 2014). Macro-actions are valid in a particular initiation set ($\mathcal{I}$), which may depend on the underlying state of the system or some high-level observations, and continue until some terminal conditions (β) are met, which again may depend on the underlying system state or high-level observations. This models a robot controller that can only be executed under some conditions, but once executed continues to run until it has reached some goal (or decides that it has failed).

Policies for each robot, μ_i, can be defined for choosing macro-actions instead of primitive actions. For example, policy trees can defined with nodes now representing macro-actions and edges representing terminal conditions or high-level observations (labeled with β in Figure 4). If macro-action policies are built from primitive actions, we can evaluate the high-level policies in a way that is similar to other Dec-POMDP-based approaches. That is, because we assume the macro-actions and the lower-level Dec-POMDP are known, we can "unroll" the policies over macro-actions into policies over primitive actions. Therefore, macro-actions can be chosen asynchronously, but because robots are assumed to have synchronized clocks, the underlying primitive actions are executed in a synchronous manner. Specifically, 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. The joint policy and macro-action policies can then be evaluated as

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

The goal is to obtain a *hierarchically optimal policy*: $\mu^*(s) = \operatorname{argmax}_{\mu} V^{\mu}(s)$, which produces the highest expected value that can be obtained by sequencing the robots' given macro-actions.

Two Dec-POMDP algorithms have been extended to the MacDec-POMDP case (Amato et al., 2014), but other extensions are possible. These algorithms use dynamic programming to construct one policy tree for each robot starting from the leaves and moving up to the root node (with nodes as macro-actions and edges as terminal conditions or high-level observations). Because many of the synchronous

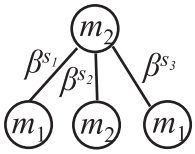

Fig. 4. Depiction of a robot's policy using macro-actions (m) and branching on terminal conditions (β).

decision-making assumptions are broken in the macro-action case, many aspects of the Dec-POMDP algorithms do not directly transfer. Nevertheless, the inspiration from these algorithms can be used to search through the space of possible tree-based policies with evaluation from equation (1). We use a different approach that does not rely on these previous Dec-POMDP or MacDec-POMDP methods and discuss the relevant details below.

4. Finite-state controllers for MacDec-POMDPs

A tree-based representation of a policy causes each robot to remember its entire history to determine its next action. In finite-horizon problems, the memory requirement is exponential in the horizon and for infinite-horizon problems a robot would need infinite memory. Clearly, this is not feasible. As an alternative, we introduce a finite-state controller representation that retains only finite memory and provide algorithms for generating these controllers.

Finite-state controllers (FSCs) can be used to represent policies in an elegant way since a robot can be conceptualized as a device that receives observations and produces actions. As shown in Figure 5, FSCs operate in a manner similar to policy trees. There is a designated initial node and after action selection at a node, the controller transitions to the next node depending on the resulting observation. This continues for an arbitrary number of steps in the problem. Nodes in a robot's controller represent internal states, which prescribe actions based on that robot's finite memory.

A set of controllers, one per robot, provides the joint policy of the robots. Finite-state controllers explicitly represent infinite-horizon policies, but can also be used (as a possibly more concise representation) for finite-horizon policies. FSCs are a widely used as solution representations for POMDPs and Dec-POMDPs (Amato et al., 2010; Bai et al., 2013; Bernstein et al., 2009; Kaelbling et al., 1998; Poupart and Boutilier, 2003; Szer and Charpillet, 2005; Wu et al., 2010a).

4.1. Mealy controllers

Two main types of controllers, Moore and Mealy, have been used for POMDP and Dec-POMDP solutions (both of which are shown in Figure 5). Moore controllers associate actions with nodes and Mealy controllers associate actions

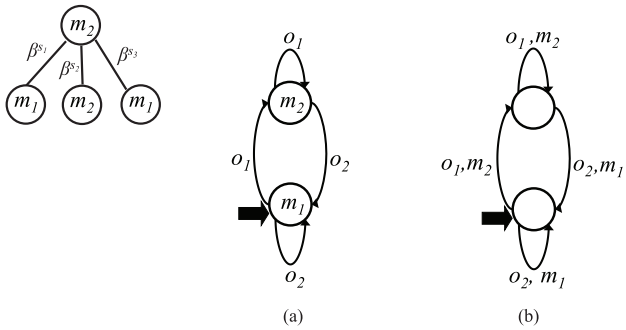

Fig. 5. A robot's (a) Moore and (b) Mealy finite-state controller with initial nodes designated with an arrow.

with controller transitions (i.e. nodes and observations). We use the Mealy representation.

A *Mealy controller* for robot i is a tuple $c_i = \langle Q_i, A_i, Z_i, \delta_i, \lambda_i, q_i^0 \rangle$:

- Q_i is the set of nodes;
- A_i and Z_i are the output and input alphabets (i.e. the action chosen and the observation seen);
- $\delta_i : Q_i \times Z_i \to Q_i$ is the node transition function;
- $\lambda_i : Q_i \times Z_i \to A_i$ is the output function for nodes $\neq q_i^0$ that associates output symbols with transitions;
- $\lambda_i^0 : Q_i \to A_i$ is the output function for node q_i^0;
- $q_i^0 \in Q_i$ is the initial node.

Because action selection depends on the observation as well as the current node, for the first node (where no observations have yet been seen), the action only depends on the node. For all other nodes, the action output depends on the node and observation with $\lambda_i(q_i, o_i)$.

Mealy controllers are a natural policy representation for MacDec-POMDPs because the initial conditions of macro-actions can be easily verified. That is, since the macro-action is chosen after an observation is seen, MacDec-POMDPs in which initial conditions depend solely on robot's local observations can be verified directly. As such, algorithms that use Mealy controllers can ensure that valid macro-action policies are generated for each robot.

For a set of Mealy controllers, c, when the initial state is s, the joint observation is o and the current nodes of c are q, the value is denoted $V_c(q, o, s)$ and satisfies

$$V_c(q, o, s) = R(s, \lambda(q, o)) + \gamma \sum_{s', o'} \Pr(s'|s, \lambda(q, o)) \Pr(o'|s', \lambda(q, o)) V_m(\delta(q, o), o', s')$$

where $\lambda(q, o) = \{\lambda_1(q_1, o_1), \ldots, \lambda_n(q_n, o_n)\}$ are the actions selected by each robot given the current node of its controller and the observation seen while $\delta(q, o) = \{\delta_1(q_1, o_1), \ldots, \delta_n(q_n, o_n)\}$ are the next nodes for each robot given that robot's current node and observation. Because the first nodes do not depend on observations, the value of the controllers c at b is $V_c(b) = \sum_s b(s) V_c(q_0, s)$,

where q_0 is the set of initial nodes for all robots and $b(s)$ represents the probability of being in state s. (The value can also be represented as $V_c(b) = \sum_s b(s)\, V_c(q_0, o^*, s)$, where $o*$ are dummy observations that are only received on the first step.)

4.2. Macro-action controllers

Representing policies in MacDec-POMDPs with the finite-state controllers discussed above is trivial since we can replace the primitive actions with macro-actions. The output function becomes $\lambda_i : Q_i \times Z_i \rightarrow M_i$ where Z_i are now the observations resulting from macro-actions and M_i are the macro-actions for robot i. Unfortunately, evaluation of these macro-action controllers is complicated by the fact that macro-actions may require different amounts of time. We could use the approach described above (in Section 3.2) to represent the policy over macro-action in terms of primitive actions, but this requires a full model of the underlying Dec-POMDP and the primitive-action representation of each macro-action. Because such information may be difficult or impossible to obtain (such as when macro-actions contain continuous actions and observations), we instead explicitly consider time until macro-action completion when performing evaluation.

To perform this evaluation, we can build on recent work for modeling decentralized partially observable *semi-Markov* decision processes (Dec-POSMDPs) (Omidshafiei et al., 2015). The Dec-POSMDP model explicitly considers actions with different durations, using a reward model that accumulates value until *any* robot terminates a (macro-) action and a transition model that considers how many time steps take place until termination. These lengths of time may be different based on the various and probabilistic termination times for different macro-actions from different initial conditions. The previous Dec-POSMDP model was defined for a specific class of problem where robots are mostly independent except for their effect on joint environmental states.[1]

We propose a more general Dec-POSMDP model, $\langle I, S, \{A_i\}, T, R, \{Z_i\}, O, h\rangle$, as follows:

- I is a finite set of robots;
- $S = S_{\text{Dec}} \times S_1^m \times S_2^m \times \cdots \times S_{|I|}^m$, which includes the world state (i.e. underlying Dec-POMDP state) and a state for each of the macro-actions that are currently being executed for each robot;
- $A_i = M_i$, where the actions are the macro-actions;
- T, the probability of transitioning to next state s' now also includes the number of discrete time steps until completion of any robot's macro-action as $\Pr(s', k|s, m)$, where k is this number of steps and m is the joint set of macro-actions being executed;
- $R(s, m)$, the reward function is the value until any robot terminates, $E\{r^t + \ldots + \gamma^{k-1} r^{t+k} | s, m, t\}$, starting at time t, which is defined more formally below;
- Z_i is now a finite set of high-level observations that are only observed after a robot's macro-action has been completed;
- O, the observation probability function, generates an observation for each robot based on the resulting state, s', and the macro-action that was executed, $\Pr(o|s', m)$;
- h, the horizon is the number of (low-level, not macro-action) steps until termination.

For concreteness, we discuss the case when an underlying Dec-POMDP is known and discrete time steps are used, but these assumptions are not required (as we discuss briefly below).

We formally define the reward model as

$$R(s, m) = \mathrm{E}\left[\sum_{t=0}^{t_{min}} \gamma^t R(a^t_{\text{Dec}}, s^t_{\text{Dec}}) \,|\, s, \pi_m\right] \tag{2}$$

where π_m is the joint macro-action policy (i.e. the policy of the macro-actions currently operating in the underlying Dec-POMDP) and t_{min} is the smallest number of time steps until any robot terminates ($\min_i \min_t \{s_{\text{Dec}} \in \beta_{m_i}\}$ starting at Dec-POMDP state s_{Dec} and macro-action states s_i^m and taking actions in the underlying Dec-POMDP a_{Dec}. Here, we use β_{m_i} to represent the termination set of macro-action m_i which we assume depends on states s_{Dec} (but it could also depend on observation histories, as discussed above). Note that macro-actions will often be partially completed, so s_i^m is needed to correctly calculate the remaining time steps.

If we have a model of the underlying Dec-POMDP, (represented as R_{Dec} and P_{Dec} for the underlying reward, transition and observation models with joint actions a and joint observations o), we can evaluate the macro-actions until at least one of them stops as

$$\begin{aligned} R(s, m) = & R(s_{\text{Dec}}, s_m, m) = \\ & R_{\text{Dec}}(s_{\text{Dec}}, \pi_m(s_m)) + \\ & \gamma \sum_{s', o_{\text{Dec}}} P_{\text{Dec}}(s'_{\text{Dec}} | s_{\text{Dec}}, \pi_m(s_m)) \cdot \\ & P_{\text{Dec}}(o_{\text{Dec}} | s'_{\text{Dec}}, \pi_m(s_m)) \cdot \\ & \prod_i \left[(1 - \mathbb{I}_{\beta_{m_i}}(s'_{\text{Dec}})) P(s'_{m_i} | s_{m_i}, o_{\text{Dec}_i}) \right] \cdot \\ & R(s'_{\text{Dec}}, s'_m, m) \end{aligned} \tag{3}$$

where $\mathbb{I}_{\beta_{m_i}}$ is an indicator variable that is 1 when s'_{Dec} is a terminal condition of m_i (or when a terminal condition has already been met for robot i) and $P(s'_{m_i} | s_{m_i}, o_i)$ represents the transition in the macro-action state of robot i based on the observation seen.

Similarly, we can calculate the transition probabilities $\Pr(s', k|s, m)$ that the macro-actions will execute until any other configuration is reached for a given amount of time k

if we have the underlying Dec-POMDP model as

$$\begin{aligned} P(s',k|s,m) = & P(s'_{\text{Dec}}, s'_m, k|s_{\text{Dec}}, s_m, m) = \\ & \sum_{s_m^{k-1}, s_{\text{Dec}}^{k-1}} P_{\text{Dec}}(s'_{\text{Dec}}|s_{\text{Dec}}^{k-1}, \pi_m(s_m^{k-1})) \cdot \\ & \sum_{o_{\text{Dec}}} P_{\text{Dec}}(o_{\text{Dec}}|s'_{\text{Dec}}, \pi_m(s_m^{k-1})) \cdot \\ & \prod_i [P(s_{m_i}^{k-1}|s_{m_i}, o_{\text{Dec}_i})] \cdot \\ & P(s_{\text{Dec}}^{k-1}, s_m^{k-1}, k-1|s_{\text{Dec}}, s_m, m) \end{aligned} \quad (4)$$

where s_{Dec}^{k-1} and s_m^{k-1} are states in the underlying Dec-POMDP and macro-actions (with s^{k-1} the combined state) after $k-1$ steps. That is, we can calculate the transition probability recursively based on the probabilities of the possible states after $k-1$ discrete time steps.

The observation function can be defined based on the macro-action that was taken and the resulting state s'. Because the resulting state includes the underlying Dec-POMDP state, these observations can depend on the other robots and other parts of the environment.

Using this model—in the case when the reward, transition and observation models are calculated as described above or assumed to be known—we can evaluate a joint policy of macro-actions, μ, using the following Bellman equation

$$V^{\mu}(s) = R(s,m) + \sum_k^{\infty} \gamma^k \sum_{s'} \Pr(s',k|s,m) \cdot \sum_{o'} \Pr(o'|s',m)\, V^{\mu}(s') \quad (5)$$

When the joint policy, μ, is represented as a set of Mealy controllers, the Bellman equation becomes

$$\begin{aligned} V^{\mu}(q,o,s) = & \\ & R(s,\lambda(q,o)) + \sum_k^{\infty} \gamma^k \sum_{s'} \Pr(s',k|s,\lambda(q,o)) \cdot \\ & \sum_{o'} \Pr(o'|s',\lambda(q,o))\, V^{\mu}(\delta(q,o'),o',s') \end{aligned} \quad (6)$$

Note that, in the Dec-POSMDP, observations are only generated for robots that complete their macro-actions. As such, the observation, o_i, and the current controller node, q_i, do not update until robot i terminates its macro-action execution. These equations can be evaluated by solving the corresponding set of equations or approximated using Monte Carlo methods (as described below).

When the underlying Dec-POMDP model is not known, the reward, transition and observation models can be defined explicitly in terms of the macro-actions. We assume the state-space of an underlying Dec-POMDP is known, but the full model (i.e. the reward, transition and observation functions) and the policies of the macro-actions do not need to be known. For instance, in our experiments, we determine the reward model by defining it over the states and macro-actions (e.g. positive reward for beer being delivered) and determine the transition and observation models by repeatedly executing macro-actions in the domain to determine their terminal states, times for completion and possible sensor and communication information.[2] In general, if the low-level Dec-POMDP model and macro-action policies are not known, the Dec-POSMDP model can be calculated directly (by using a simulator or the domain to estimate the high-level rewards, transitions and observations) or through other models of the dynamics. Also, note that while the model still includes the states of the Dec-POMDP, it does not include the Dec-POMDP actions and observations. As a result, these low-level quantities can be continuous and the low-level transition dynamics and observation model may be very complicated, but we need only consider the effects of the macro-actions in terms of the high-level transitions, observations and rewards.

In our experimental domain, S_i^m will be the amount of time that robot's macro-action has been executing (since this is sufficient information to determine how much more time will be required in that problem). More generally, S_i^m could correspond to a node in a lower-level finite-state controller or other relevant information for updating the states of the macro-actions.

4.3. Exploiting domain structure

The Bellman equation provides a formal framework for evaluating policies in MacDec-POMDPs directly if we have a model of the system. When a full model is not available, a simulator can also be used to perform Monte Carlo evaluation of a solution. This can be done by generating a number of *trajectories* that each produce a single return for a sampled sequence of states, observations and rewards over the number of steps in the problem.

Specifically, using the time steps from the underlying Dec-POMDP, the value of the kth trajectory that starts at state s_0 and uses policy μ is given by $V^{\mu,k}(s_0) = r_k^{\text{Dec}_0} + \ldots + \gamma^T r_k^{\text{Dec}_T}$, where $r_k^{\text{Dec}_t}$ is the reward from the underlying Dec-POMDP given on the tth step. The value after K trajectories is then averaged as: $\hat{V}^{\mu}(s_0) = \sum_{k=1}^{K} \frac{V^{\mu,k}(s_0)}{K}$. The simulator can often operate using the time steps from the underlying Dec-POMDP as the states may need to be updated at this frequency and the termination of each macro-action can be checked at this time. Nevertheless, we can also compute the value based on the time steps and rewards at the macro-action level with

$$V^{\mu,k}(s_0) = r_k^0 + \cdots + \gamma^{T-t_\tau} r_k^{\tau} \quad (7)$$

where r_k^t is now the reward in the Dec-POSMDP at the tth (of τ) macro-action step and t_τ is the number of (primitive) time steps taken by the last macro-action(s).[3] As the number of samples increases, the estimate of the policy's value will

Algorithm 1 Sample-based evaluation.

```
1: function SAMPLEEVAL(μ,s0,numSims,maxTime)
2:     totalReturn ← 0
3:     for sim 0 to numSims do
4:         s ← s0, q ← q0, o ← o*
5:         t ← 0, t^Ag ← 0, minTime ← 0
6:         minRobots ← null
7:         termConds ← null
8:         while t < maxTime do
9:             minTime ← ∞
10:            for all robots i ∈ I do
11:                for all β_i of λ_i(q_i, o_i) do
12:                    t ← SampleFromDist(s, λ_i(q_i, o_i), β_i)
13:                    if t − t_i^Ag = minTime then
14:                        minRobots ← minRobots ∪ i
15:                        termConds ← termConds ∪ β
16:                    else if t − t_i^Ag < minTime then
17:                        minRobots ← {i}
18:                        termConds ← {β}
19:                        minTime ← t − t_i^Ag
20:            t+ =minTime
21:            s′ ← sampledState(s, λ(q, o),termConds)
22:            r ← R(s′)
23:            for all robots i ∈minRobots do
24:                o_i ← sampleObs(s′,termConds)
25:                q_i ← δ(q_i, o_i)
26:                t_i^Ag = 0
27:            for all robots i ∉minRobots do
28:                t_i^Ag + =minTime
29:            totalReturn + = r
30:    return totalReturn/numSims
```

Algorithm 2 MacDec-POMDP heuristic search (MDHS).

```
1: function HEURSEARCH(s0,n)
2:     V ← V_init
3:     polSet ← ∅
4:     repeat
5:         θ ← selectBest(polSet)
6:         Θ′ ← expandNextStep(θ)
7:         for θ′ ∈ Θ′ do
8:             if isFulPol(θ’,n) then
9:                 v ← valueOf(θ’,s0)
10:                if v > V then
11:                    μ* ← θ
12:                    V ← v
13:                    prune(polSet,V)
14:            else
15:                v̄ ← valueUpperOf(θ’,s0)
16:                if v̄ > V then
17:                    polSet ← polSet ∪ θ′
18:        polSet ← polSet \ θ
19:    until polSet is empty
20:    return μ*
```

approach the true value (as shown for the POMDP (Thrun, 1999) and Dec-POMDP (Wu et al., 2010b) case).

Generating a full model or simulator in complex domains remains difficult, but many domains possess structure that allows efficient evaluation. For example, in the bartender domain, we perform a sample-based evaluation of policies using a high-level simulator. As mentioned above, this simulator uses state information for the macro-actions that consists of distributions for the completion time at each terminal condition given each possible initial condition. Having this timing information is a much less restrictive assumption than knowing the full policy of each macro-action. We also assume that the reward only depends on the state, and that observations only depend on the state and terminal condition of the macro-action. This simulator allows us to evaluate policies while keeping track of the relevant state information and execution of macro-actions. While, these assumptions allow for more efficient evaluation, our heuristic search algorithm does not require these assumptions and is general enough to solve any Dec-POSMDP in which we can generate candidate controllers and evaluate them.

Pseudocode for sample-based evaluation is given in Algorithm 1. The sum of the returns as well as the state, the current node and last observation of each robot, the current time in the system and the amount of time each robot has been executing its macro-action are initialized in lines 2, 4 and 5. The minimum time interval before termination of the next macro-action as well as the corresponding robots and terminal conditions are initialized on line 6. At each iteration, the simulator determines the set of robots which terminate their macro-actions in the least amount of time (in lines 8–18). The completion time of each robot's macro-action is sampled in SampleFromDist, and then adjusted based on the amount of time the macro-action has already been running t_i^{Ag}. The system time and state updates based on the termination of these completed macro-actions (in lines 19 and 20), the reward is added to the return in line 21 and the corresponding robots receive new observations and transition in their controllers (in lines 22–25). Robots that have not finished their macro-actions have their timers updated (in lines 26–27). The iterations continue until the system time reaches a limit (maxTime). This sample-based evaluation can calculate the value of policies in problems with very large (and continuous) state spaces using a small number of simulations.

5. Policy search

Policy evaluation is an important step, but we must also determine what policies each robot will use in the domain. Specifically, we propose to generate finite-state controllers using a heuristic search method that searches over the action selection and node transition parameters for each agent. The result is an optimized set of controllers, one for each

agent. Controller optimization methods have been able to generate high-quality controllers in the (primitive-action) Dec-POMDP case (Amato and Zilberstein, 2009; Szer and Charpillet, 2005), but such methods have yet to be developed for the macro-action case. Our new method, termed MacDec-POMDP heuristic search (or MDHS), integrates our sample-based evaluation and searches for a policy that is optimal with respect to a given controller size. MDHS constructs a search tree of possible controllers for each robot (i.e. possible action selection and node transition parameters at each node), and searches through this space of policies by fixing the parameters (of all robots) for one node at a time, using heuristic upper bound values to direct the search.

Pseudocode of MDHS is in Algorithm 2. A lower bound value, $\underline{V}$ is initialized with the value of the best known joint policy (e.g. a random or hand-coded policy) in line 2. An open list, *polSet*, which represents the set of partial policies that are available to be expanded is initialized to be the empty set in line 3. At each step, the partial joint policy (node in the search tree) with the highest estimated value is selected (using selectBest in line 5). This partial policy is then expanded in line 6, generating policies with the action selection and node transition parameters for an additional node specified (all children in the search tree). This set is called Θ'. As stated above, expanding a search node consists of adding new search nodes for each possible combination of action selection and node transition parameters for each agent for one more node in the controller (e.g. if the current search node has fixed parameters for one of the ten nodes in each agent's controller, the children will now fix the parameters for two of the ten nodes). Each policy in Θ' is examined in the loop beginning at line 7. If an expanded policy is fully specified (i.e. all controller nodes have action selection and node transition parameters specified), its value is compared with the value of the best known policy ($\underline{V}$), which is updated accordingly (allowing for pruning of policies with value less than the new $\underline{V}$). This procedure is shown in lines 8-13. If a policy is not fully specified, its upper bound is calculated and it is added to the candidate set for expansion as long as that bound is greater than the value of the current best policy (in lines 15–17). The partial policy that was expanded is removed from the candidate set (the open list) in line 18 and this process continues until the optimal policy (of size n) is found.

While this approach will generate a set of optimal controllers of a fixed size when it completes, it can also be stopped at any time to return the best solution found so far. In our simple implementation, we set the initial lower bound to be the value of a random policy and the upper bound as the highest-valued single trajectory (i.e. simulation in Algorithm 1) which uses random actions for controller nodes that have not been specified (rather than the expected value). These are relatively loose values, but performed well in our experiments. To more quickly generate candidate solutions before the search terminates, we also initiated the search with a set of random (rather than blank) controllers which had action selection and node transition parameters updated during the search.

5.1. Improving the heuristic search algorithm

A naive implementation of Algorithm 2 will not be very efficient. In particular, 'expandNextStep(θ)' on line 6 will create all possible children (i.e. search nodes) for each robot that define the different action selection and node transition parameters to one more node in the controller. That is, if all n robots have $|M_i|$ macro-actions, $|Z_i|$ observations and $|Q_i|$ nodes in the controller, $(|M_i|^{|Z_i|})^n$ $(|Q_i|^{|Z_i|})^n$ new search nodes get generated. The upper bound for each of these children must be calculated and those that have a higher value than the current best policy are added to the open list. This upper bound calculation is time consuming and a large number of search nodes will be added to the open list before a better solution can be found. Only certain macro-action actions are valid for the given initial conditions and only some observations are possible after taking a macro-action, so all the children do not need to be considered, but the number remains high.

As an alternative, we also consider an incremental version of MDHS. In the incremental version, 'expandNext Step(θ)' is broken up into 'expand NextStepAction(i, θ)' and 'expandNextStepTrans(i, θ)' for each robot i. That is, instead of adding parameters for action selection and node transitions for all robots, expansion is done for one robot at a time and separately for action selection and node transitions. Specifically, we loop through the robots to generate action selection parameters for each robot's next controller node and then loop through the robots again to generate transition parameters for those nodes. After each robot's expansion, many fewer search nodes are added to the open list when compared with the naive implementation ($|M_i|^{|Z_i|}$ for the action case and $|Q_i|^{|Z_i|}$ for the transition case). As a result, we may generate candidate solutions more quickly. As we generate these candidate solutions, the lower bound can be updated and we may never generate all the children that are considered in the naive implementation. Overall, this incremental algorithm can be expected to produce higher-quality solutions in a given amount of time and speed up convergence to an optimal solution.

In order to take full advantage of this incremental expansion, we need a heuristic that is able to consider the partially defined controller nodes that are generated. The upper bound heuristic listed above is very loose and will not change significantly when transitions are defined (due to the fact that transitions are often to controller nodes that have not yet been defined). Instead, we use an upper bound heuristic based on the cross-product MDP (Meuleau et al., 1999; Szer and Charpillet, 2005), which considers a centralized, fully observable solution for nodes that have not

been defined, but otherwise uses the actions and transitions defined by the controller at the given search node. Because a full (primitive) model is needed to generate the cross-product MDP, we use a centralized hand-coded mapping that assumes the states are fully observable (e.g. in the bartender domain the orders are observable by all robots without traveling to the respective rooms) for controller nodes that have not yet been defined. Specifically, in the bartender and waiter problem, the resulting upper bound policy has each waiter waiting in the kitchen until the bartender is ready, then getting a drink from the bartender, delivering the drink to the room with the oldest order and then returning to the kitchen to continue this cycle. Evaluation of this heuristic is done online as controller parameters are defined and the upper bound converges to the true value of the controller when all action selection and node transitions are specified.

6. Experiments

We perform comparisons with previous work on existing benchmark domains and demonstrate the effectiveness of our MDHS policy search in the bartender scenario. We compare only with MacDec-POMDP methods since our previous work showed that primitive-action Dec-POMDP methods cannot scale to problems of the size considered in this paper (due to the resulting increase in the action and observation space as well as the problem horizon) (Amato et al., 2014). In the first two problems, the simple version of our approach from Algorithm 2 is used and controller sizes are fixed to be 5 nodes. As an alternative, controller sizes could be generated from trajectories in the simulator (similar to previous methods (Amato and Zilberstein, 2009)) or learned (Liu et al., 2015). Experiments were run on a single core of a 2.2 GHz Intel i7 with a maximum of 8 GB of memory. The simulation experiments provide a quantitative analysis of the efficacy of the MacDec-POMDP planner, while the real world experiments show that our method can be used for actual multi-robot systems.

6.1. A benchmark problem

For comparison with previous methods, we consider robots navigating among movable obstacles (NAMO) (Stilman and Kuffner, 2005). In this problem, two robots must navigate to a goal location, but the paths to that location are blocked by some number of obstacles that require either a single or multiple robots to move. Therefore, the robots must reason about navigation and coordination choices in order to most efficiently move to the goal. This domain was designed as a finite-horizon problem (Amato et al., 2014) so we add a discount factor (of 0.9) for the infinite-horizon case.

Our previous tree-based MacDec-POMDP methods were designed for finite-horizon problems (Amato et al., 2014), but they can produce policies that have a high value in infinite-horizon problems by using a large planning horizon. In fact by using a horizon of 50, the optimal tree-based methods can produce a solution within 0.046 of the optimal (infinite-horizon) value in both instances of the problem we consider (due to discounting making additional value beyond horizon 50 below that number). As mentioned above, in these comparisons, we used our simple MDHS method with a random lower bound and the best single trajectory that was sampled as an upper bound. No other parameters are needed except for the desired controller size for each robot (which balances time and computational complexity).

Table 1. Values, times (in s) and policy sizes on NAMO benchmarks of size 5×5 and 25×25.

	MDHS controller		Tree	
	5×5	25×25	5×5	25×25
Value	−5.33	−9.91	−5.30	−9.87
Time	180	180	388	4959
Size	5	5	100^{49}	100^{44}

As seen in Table 1, MDHS ("MDHS Controller"), produces solutions that are near optimal (as given by the finite-horizon "Tree" method) in much less time and with a much more concise representation. The previous tree-based dynamic programming method can produce a (near) optimal solution, but requires a representation exponential in the problem horizon and must search through many more possible policies before generating a solution. It is important to note that while these domains have a large number of states (5000 in the 5×5 case and 3.125×10^6 in the 25×25 case), the number of macro-actions and observations is small (4 and 12 respectively). Furthermore, macro-actions are not possible for some situations (e.g. robots will not try to move an obstacle until they observe that they are next to one). As a result, the tree-based method is still able to solve this problem, but will not scale to larger action and observation spaces.

6.2. A small warehousing problem

A small warehousing problem has also been modeled and solved as a MacDec-POMDP (Amato et al., 2015). In this problem, a team of robots must find packages that may be in various depots and return them to a shipping location. Some packages must be pushed by multiple robots and some can be retrieved by a single robot. Furthermore, various communication assumptions were used such as no communication, limited communication (within a specified radius) and signaling through the use of a light. Previous solution methods (which are based on the policy tree methods (Amato et al., 2014)) were able to automatically generate a set of policies for a team of iRobot Creates, but were unable to exceed a problem horizon of 9. This lack of scalability of the tree-based methods is due to the larger problem size

Table 2. Values, times (in s) and policy sizes on three warehousing problems (with no and limited communication as well as signaling). The tree-based method is unable to solve these problems.

	MDHS controller			Tree		
	No Com	Com Limit	Signal	No Com	Com Limit	Signal
Value	11.38	12.41	14.89	–	–	–
Time	180	180	180	–	–	–
Size	5	5	5	–	–	–

(approximately 1.26×10^9 states, 6-11 macro-action and 36 observations).

As seen in Table 2, MDHS can produce concise solutions very quickly on these problems. Because of the limited horizon that can be achieved by the tree-based methods, it is not possible to generate a useful bound on the optimal solution for this problem. Nevertheless, our method is able to solve warehousing problems for any arbitrary horizon, while the tree-based methods could not.

As an additional comparison, we also evaluate the controllers generated by our MDHS algorithm for the same number of steps as the previous algorithm (9) without a discount factor. In this case, the previous tree-based method produced solutions with values of 1.16, 1.60, and 1.68 while our method produces solutions with values 1.12, 1.14 and 1.61 for the no communication, limited communication and signaling cases, respectively. Note that our solution was not optimized for this particular horizon, but it shows that both methods have similar solution quality when executed for only 9 steps.

6.3. Bartender and waiters problem

The bartender and waiters problem is a multi-robot problem modeled after waiters gathering drinks and delivering them to different rooms. The waiters can go to different rooms to find out about and deliver drink orders. The waiters can go to the bar to obtain drinks from the bartender. The bartender can serve at most one waiter at a time and the rooms can have at most one order at a time. Because there is only one type of beverage in our problem, the policy of the bartender is simply to always pick a beverage when it does not have one and serve the first waiter to request it. Any waiter can fulfill an order (even if that waiter does not have previous knowledge about the order). Drink orders are created stochastically: a new order will arise in a room with 1% probability at each (low-level) time step when one does not currently exist. The reward for delivering a drink is $100-(t_{\text{now}} - t_{\text{order}})/10$, where t_{now} is the current time step and t_{order} is the time step at which the order was created.

The domain consists of three types of macro-actions for the waiters, as shown in Table 3. These macro-actions consist of navigation decisions such as traveling to each of the different rooms or to the bar area as well as the ability to request a drink from the bartender.

Table 3. Macro-actions for the waiters.

ROOM_N	Go to room *n*, observe orders and deliver drinks.
BAR	Go to the bar and observe current status of the bartender.
GET_DRINK	Obtain a drink from the bartender.

Table 4. Observations for the waiters.

Variable	Values	Description
loc	{room_n, bar}	waiter's location
orders	{True, False}	drink order for current room
holding	{True, False}	waiter holding drink status
bartender	not_serving	not serving and not ready to serve
	ready_to_serve	not serving and is ready to serve
	serving_waiter	serving a drink
	no_obs	cannot observe bartender

The state variables each waiter can observe are shown in Table 4. These observations involve seeing a high-level indication of the location, whether there is an order (when a waiter is in a room), whether the waiter is holding a drink and some status information about the bartender (when the waiter is in the bar area). The details of how we implemented this domain on real robots are included in the next subsection.

To develop a simulator that is similar to the robot implementation, we estimated the macro-action times by measuring them in the actual domain over a number of trials (starting the macro-actions at possible initial conditions and executing until each possible terminal condition, generating probability distributions for the terminal conditions and times). The rewards and observations were defined as above (using partial, but not noisy observations). Additional details are also provided in the next subsection. There is a large amount of uncertainty in the problem in terms of the time required to complete a macro-action and outcomes such as receiving orders. We did not explicitly model failures in the navigation or PR2 picking/placing or noise in the observation model, but these could be easily modeled.

Our instance of the bartender and waiters problem consisted of one bartender and two waiters. The domain had four rooms: the bar and rooms 1–3. As mentioned above, rooms 1–3 could order at most one drink at a time and only one drink type was used. We had a total of 5 macro-actions since there is a macro-action for each room as well as one for requesting a drink. There were also 64 observations (from Table 4) and the underlying state space consists of the continuous locations of the TurtleBots, the status of the PR2 and discrete variables for orders in each room and

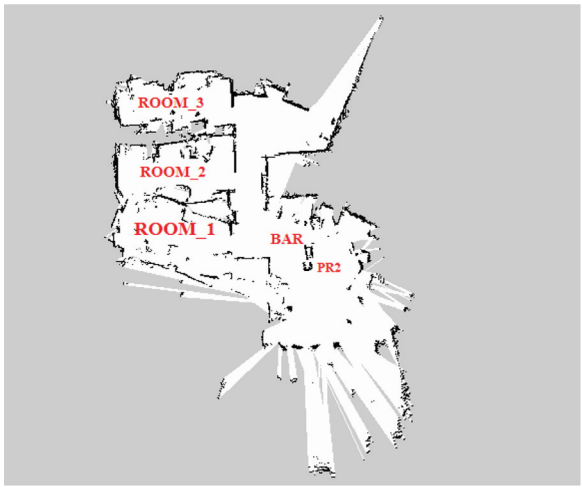

Fig. 6. Navigation map generated by the TurtleBots.

whether each TurtleBot is holding a beverage. No communication was used except between the bartender and waiters in the bar area.

Robot implementation

As shown in Figure 1, we used two TurtleBots (we call the blue one *Leonardo* and the red one *Raphael*) as waiters and the PR2 as a bartender. The TurtleBots had two types of macro-actions: navigation and obtaining a drink from the bartender.

The navigation actions were created using a map, shown in Figure 6, with the ROS TurtleBot_navigation package (Foote, 2015) and adding simple collision avoidance. For picking and placing drinks, we combined several ROS controllers for grasping and manipulation. The GET_DRINK macro-action implemented a queueing system to serve multiple TurtleBots in the order they arrived. Specifically, The PR2 always picks up a drink and waits for a TurtleBot to arrive to ask for it. To make sure multiple TurtleBots did not attempt to get a beverage at the same time we implemented a simple queue where TurtleBots would send a message to the PR2 to enter the queue. Then, the PR2 would send a message to the first TurtleBot in the queue when it was ready to place a drink. Once the TurtleBot left the PR2, the PR2 would pick up another drink and wait for the next TurtleBot. Each TurtleBot had a cooler for the PR2 to place drinks into. The cooler was identified with an AR tag in order to locate the TurteBot and place the drink.

For the observations, we used state action deduction and communication. That is, the GET_DRINK action was assumed to always succeed (but may require different amounts of time); the TurtleBot asserted it was holding a drink after this action. When the TurtleBot entered a room it would prompt the user to take the drink it was holding or to place an order. The user could give a boolean response by toggling a red button on top of the TurtleBot. After a user picked up the drink, the waiter observed not_holding until it completed the next GET_DRINK action. The location observations were set with the localization functionality of the TurtleBot_navigation stack. To obtain information about the bartender, the PR2 would broadcast its current state (serving, not_serving, or ready_to_serve). The TurtleBots were only able to listen to the message in the bar location.

Bartender and waiter problem results

Our MDHS planner automatically generated the bartender and waiter solution based on the macro-action definitions and our high-level problem description (discussed above). That is, because the simulator was created based on the domain, solutions could be generated using the simulator and executed in the actual domain. The solution is a set of Mealy controllers (one for each robot) that maps nodes (which can represent different histories) and observations to actions.

First, to easily examine the results, we generated policies with 1 and 2 nodes. Figures 7 and 10 show the Mealy controllers for the 1-node and 2-node case, respectively. Nodes are labeled with ellipses, observations as rectangles, and actions as diamonds. Given a node and observation, the Mealy controller shows the next node (using a solid line) and corresponding action (with a dashed line). For clarity, only transitions to different nodes are labeled in our diagrams. In the one node case, there are no transitions to new nodes; hence, the controllers are a reactive memoryless policy based on current observation.

To more clearly show the 1-node results, Figures 8(a) to 8(c) display parts of the generated policies. Analysis of the solution is naturally segmented into three phases: *bar, delivery,* and *ordering*, which correspond to (1) the waiter being located in the bar, (2) holding a drink, and (3) not holding a drink. As can be seen in Figures 7 and 8, the solution spread out the serving and delivery behaviors of the TurtleBots between the three rooms: Leonardo only visited rooms 1 and 3, whereas Raphael focused on rooms 2 and 1. Additionally, the TurtleBots' controllers selected the BAR macro-action even when drinks were not ordered. This allowed the TurtleBots to have drinks that were ready to deliver, even if they did not previously know about an order.

Figure 8(a) shows the macro-actions for each TurtleBot when it is located in the bar (after the TurtleBot executes the BAR macro-action that takes navigates it to the bar from any location or in the initial problem configuration). Once in the bar, the TurtleBot can observe the bartender's status. If the bartender is ready_to_serve, either agent will execute the GET_DRINK action. Following the GET_DRINK action, Raphael and Leonardo will execute ROOM_2 and ROOM_3 macro-actions, respectively. If the bartender is not ready_to_serve the waiter will execute ROOM_1 or ROOM_2 macro-actions, depending on the observation. The distance is farthest to ROOM_3 so it requires less

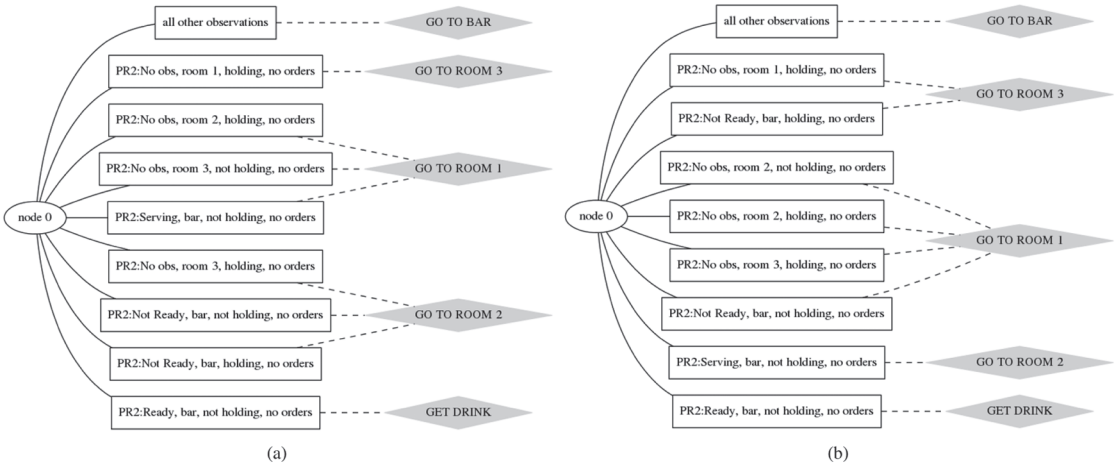

Fig. 7. 1-node controllers for the TurtleBots in the bartender and waiters problem with nodes as ellipses, observations as rectangles, and actions as diamonds: (a) *Leonardo*'s 1-node controller; (b) *Raphael*'s 1-node controller.

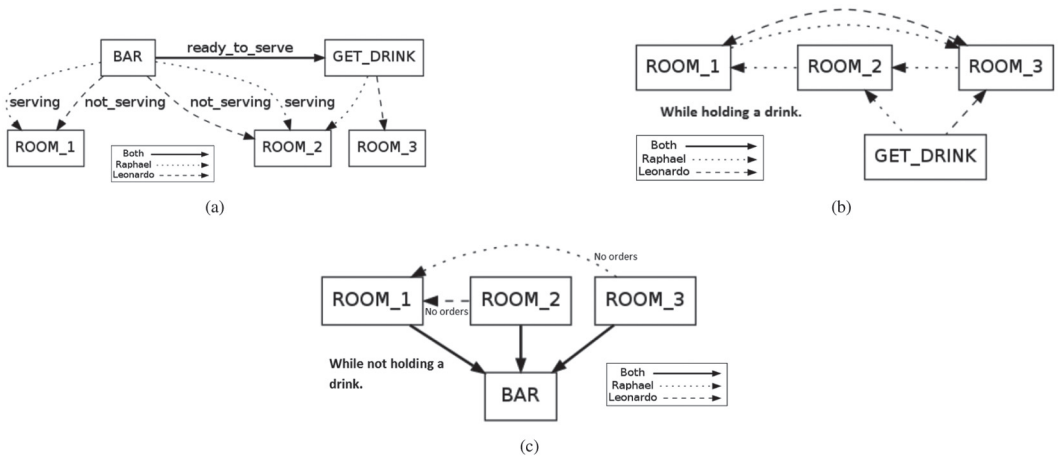

Fig. 8. Controller phases for each waiter: (a) *Bar* phase while located in the bar; (b) *Delivery* phase while holding a drink; (c) *Ordering* phase while not holding a drink.

time to visit the other rooms when the bartender is not ready_to_serve.

Once a TurtleBot is holding a drink, it is in the *delivery* phase. Figure 8(b) shows the sequence of macro-actions executed in this case. Raphael receives a drink from the bartender and tries to complete deliveries in the following order: ROOM_2, ROOM_1, ROOM_3. That is, it continues looping through all rooms while holding a drink. Leonardo executes the ROOM_3 macro-action after receiving a drink from the bartender. If the drink is not delivered then it chooses the ROOM_1 macro-action. It continues looping between ROOM_1 and ROOM_3 actions until a delivery is made.

After a TurtleBot has delivered a drink, it enters the *ordering* phase. Figure 8(c) shows the macro-action sequence for the case when the waiters are not holding any drinks. The dashed and dotted lines show the two cases when the waiters do not go to the bar. This happens when there is no order placed in rooms 2 and 3; the waiters go to the bar for all other observations. This behavior balances off having a drink ready for unknown orders and the time used to visit other rooms.

An example execution of our generated controllers (for the 1-node case) is shown in Figure 9. Initially, the TurtleBots start in the bar next to the PR2. The PR2 immediately starts picking up a drink (Figure 9(a)) and the two TurtleBots navigate to different rooms (Figure 9(b)). Then Leonardo returns to the kitchen and successfully receives a drink from the PR2 (Figure 9(c)). While holding the drink, Leonardo tries to make a delivery by going to room 2 (Figure 9(e)). There is no drink order in room 2, so Leonardo continues to room 1 and successfully delivers the drink to a thirsty graduate student (Figure 9(f)). While Leonardo is served by the PR2, Raphael goes to the bar and observes the PR2 is busy (Figure 9(d)). After observing the PR2 is serving, Raphael navigates to room 1 to collect drink orders (Figure 9(e)).

It is important to note that the 1-node controllers cannot contain a more complex solution that allows each waiter to choose what room to go to after receiving a drink from

the bartender depending on previous actions or observations (since no memory is used). This controller is an elegant solution given the constraint: Raphael serves room 2 then room 1, whereas Leonardo room serves room 3 followed by room 1. This resultant behavior shows cooperation between the two robots to efficiently cover the rooms.

As seen Figure 10, adding another node allows for more elegant and intricate solutions since the 2-node controller can keep track of more information. Because there are only two nodes and the solution is optimized to improve performance, not clarity, it is somewhat difficult to interpret the meaning of the different nodes. Nevertheless, the multiple nodes are used to remember actions taken and observations seen. To simplify the analysis, we can look at three different scenarios. Scenario 1 is receiving a drink from the bar and trying to deliver it when no orders are received. Scenario 2 is going to the bar when the bartender is always not ready. Scenario 3 is going to the bar when the bartender is always serving another agent.

The first case is the behavior after receiving a drink from the bar. We would expect both agents to cycle through all of the rooms to deliver a drink, with room 3 being the least frequently visited since it is far away. In the one node case, Leonardo goes through all rooms in the sequence: room 2, room 1, room 3 and Raphael visits room 3, then room 1. The 2-node case focuses more on the first two rooms for a higher expected return. Leonardo goes through room 3, room 2, room 1 and back to room 3, while Raphael visits room 1 then room 2 or room 2 then room 1 depending on which node it is currently in. By delivering to the first two rooms more frequently the agents are exploiting the fact that these rooms are closer and the multiple nodes allows this pattern to be more efficient (with a choice of room 1 or room 2).

The second case, is what happens when the TurtleBots go to the bar and the bartender is not ready. We would expect the TurtleBots to try to visit the bar again or go to a nearby room to check if the bar is ready as quickly as possible. In the one node case Leonardo visits room 2 and Raphael visits room 1. In the 2-node case, Leonardo always waiting in the bar until the bartender is ready. Raphael will either wait for the bartender until it is ready or go between room 1 and the bar depending on which node it is in. This waiting behavior seeks to get drinks as quickly as possible, while gaining order information only when deemed beneficial. Again, we see the 2-node solution is able to use memory to improve the solution.

The third case shows a clear difference between the one node and two node solutions. The TurtleBots have very different behavior for what to do after visiting the bar when the bartender is serving the other TurtleBot. Because the time needed for the bartender to complete serving is large, we see the TurtleBots visit further rooms to collect orders. This is in contrast to the 1-node case where the behavior is limited to visiting rooms 1 or 2.

We also examine the values of the 1-node, 2-node and larger 5-node controllers in the simulator. These values were computed by executing the controllers generated by MDHS in the simulator for 1000 (primitive) time-steps using 10,000 Monte Carlo simulations. The solution value was 1254 (an average of 13.95 drinks delivered) for the 1-node controllers, 1289 for the 2-node controllers (14.31 drinks delivered) and 1302 (14.64 drinks delivered) for the 5-node controller.[4] For comparison, a hand-coded controller that assigns one robot (Leonardo) to room 3 (since it is farthest from the kitchen) and one robot (Raphael) to rooms 1 and 2, produces a solution with value 851 (10.40 drinks delivered). More sophisticated hand-coded controllers are possible, but, in general, it is very difficult for a human to determine a good solution in complex problems such as this one.

Additional results comparing the simple and incremental versions of MDHS for the 1-node and 5-node case are seen in Figure 11. In the 1-node case (Figure 11(a)), both the simple and incremental versions produced high-quality solutions quickly, but the incremental version required much less time to produce and converge to the optimized solution. Results for the 5-node case (Figure 11(b)) are similar, but more time is required to search through parameters for the larger controller. Note that in the 5-node case, the graph is trimmed and does not show convergence to the final value of 1302. Here, the incremental version of MDHS is always able to produce a higher-valued policy with a given amount of time.

These results demonstrate that the MDHS planner is able to effectively generate a solution to a cooperative multi-robot problem, given a declarative MacDec-POMDP planner. Note that the same planner solved all these experimental problems based on a high-level domain description.

7. Related work

Other frameworks exist for multi-robot decision making. For instance, behavioral methods have been studied for performing task allocation over time with loosely-coupled (Parker, 1998) or tightly-coupled (Stroupe et al., 2004) tasks. These are heuristic in nature and make strong assumptions about the type of tasks that will be completed. Market-based approaches use traded value to establish an optimization framework for task allocation (Dias and Stentz, 2003; Gerkey and Matarić, 2004). These approaches have been used to solve real multi-robot problems (Capitán et al., 2013; Kalra et al., 2005), but are largely aimed at tasks where the robots can communicate through a bidding mechanism.

One important related class of methods is based on linear temporal logic (LTL) (Belta et al., 2007; Loizou and Kyriakopoulos, 2004) to specify behavior for a robot; reactive controllers that are guaranteed to satisfy the resulting specification are then derived. These methods are appropriate when the world dynamics can be effectively described non-probabilistically and when there is a useful characterization

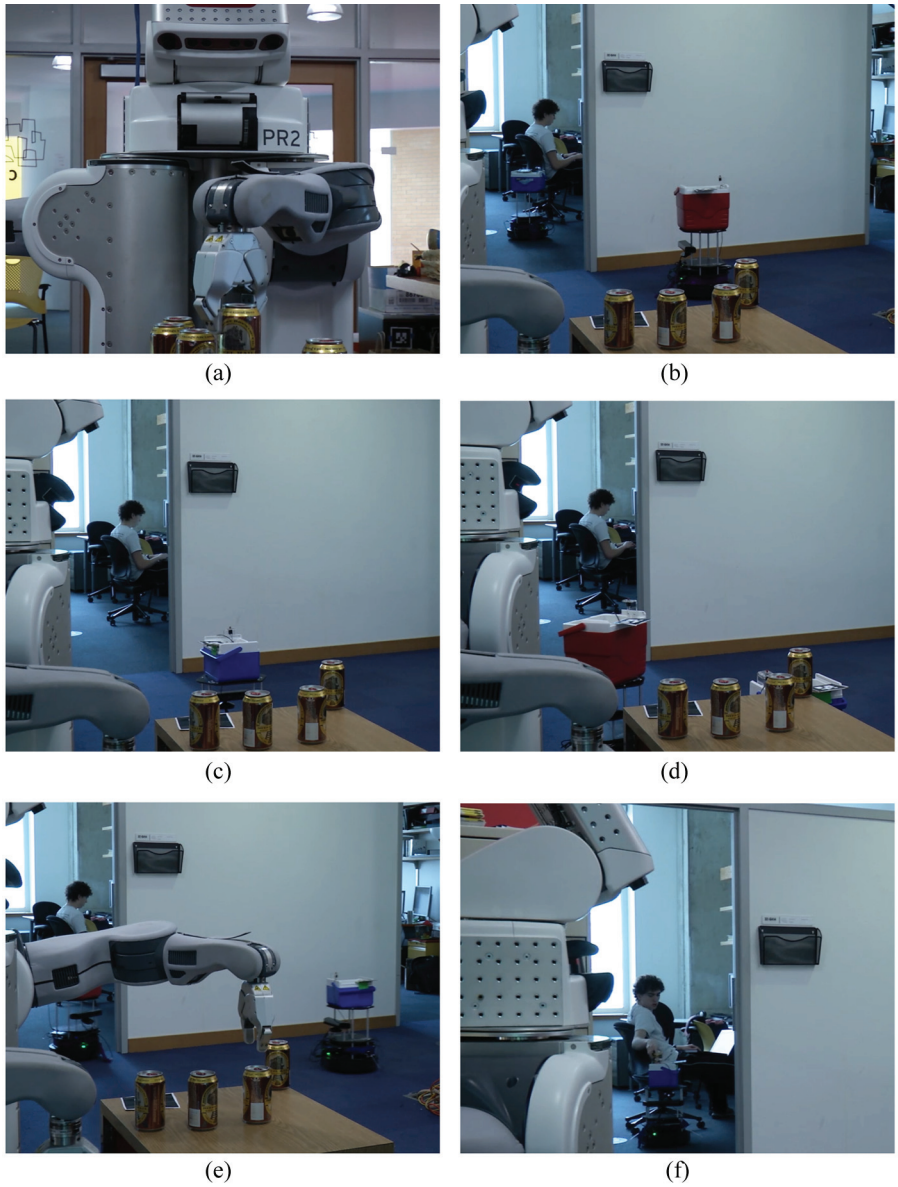

Fig. 9. Images from the bartender and waiter experiments: (a) PR2 picking up a drink; (b) TurtleBots go to first rooms; (c) Leonardo sees the PR2 ready and gets a drink; (d) Raphael sees the PR2 serving Leonardo; (e) TurtleBots go to rooms 1 and 2; (f) Leonardo delivers to room 1.

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. By contrast, our approach (probabilistically) models the *domain* and allows the planner to automatically optimize the robots' behavior.

There has been less work on scaling Dec-POMDPs to real robotics scenarios, Emery–Montemerlo et al. (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, and their system required synchronized execution by the robots. The introduction of MacDec-POMDP methods has largely eliminated these two concerns.

While several hierarchical approaches have been developed for multi-agent systems (Horling and Lesser, 2004), very few are applicable to multi-agent models based on MDPs and POMDPs. Ghavamzadeh et al. (2006) developed a multi-agent reinforcement learning approach with a given task hierarchy, but this work is limited to a multi-agent (fully-observable) SMDP model with communication, making it a subclass of a MacDec-POMDP. Other researchers have developed models similar to MacDec-POMDPs in the *centralized* multi-robot setting (Messias et al., 2013a,b).

Fig. 10. 2-node controllers for the TurtleBots in the bartender and waiters problem with nodes as ellipses, observations as rectangles, actions as diamonds and node transitions with solid lines (lack of a line represents transition back to the same node): (a) *Leonardo*'s 2-node controller; (b) *Raphael*'s 2-node controller.

8. Discussion

In this paper, we consider the case where macro-actions are given. This will often be the case in multi-robot domains as controllers typically exist for common tasks such as navigation, grasping and manipulation. Even if the individual performance of each controller is poor, by planning at the macro-action level, they may be able to be sequenced in way that effectively solves the problem. Nevertheless, controllers can also be generated from a high-level description. For example, related work has shown how a motion planner can be used to generate controllers along with distributions for completion times and terminal conditions given initial conditions (Omidshafiei et al., 2015). As a result, the probabilities for our model in equation (5) could be generated and a reward function can be defined (e.g. Omidshafiei et al. (2015) used a simple additive reward structure with individual rewards from the motion planner). The cited work requires a model of the low-level dynamics, but it can be continuous and complex. The high-level problem description consists of defining regions of interest that the motion planner will use as initial conditions and target as terminal conditions (i.e. regions of belief space that the robots should navigate to). Omidshafiei et al. (2015) assume the regions are given and each controller (i.e. motion plan) is independent, but alleviating these limitations is an area of future work.

Sometimes, it may not be possible to generate even a high-level model or simulator for a problem of interest. In these cases, the method in this paper cannot be used. One alternative is learning a solution (i.e. finite-state controllers for the robots) directly from data. Such a learning method has been explored where the data is given as a set of trajectories (in the form of macro-actions taken and observations seen over time for each agent as well as rewards received for the team) (Liu et al., 2016). Experiments show that controllers can be learned that outperform 'expert' controllers from a relatively small amount of data. The amount of data is insufficient to learn a model, but allows high-performing solutions to be learned. In this paper, we learn the timing distributions and terminal conditions for each macro-action

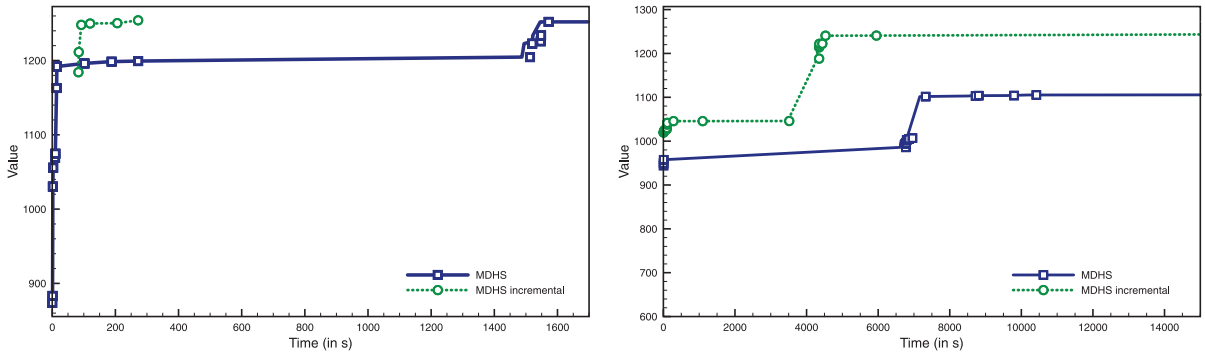

Fig. 11. Comparison of the simple and incremental versions of MDHS showing the value produced over time (in seconds). Note the 5-node controller graphs are cropped and do not show the final value of 1302: (a) one node; (b) five nodes.

separately (by executing them in the domain from various initial conditions), but the domain (or a sufficiently accurate simulator) may not be available and many macro-action executions may be necessary. Future work could examine issues such as how much data is needed to learn an accurate model and how robust the methods in this paper are to model errors.

9. Summary and conclusion

We have introduced an extended MacDec-POMDP model for representing cooperative multi-robot systems under uncertainty using a high-level problem description, and developed MDHS, a new MacDec-POMDP planning algorithm that searches over policies represented as finite-state controllers. While our previous work introduced macro-actions to Dec-POMDPs and showed that multi-robot problems could be represented and solved using them, the new model and planner are applicable to a much wider range of multi-robot problems, for two reasons.

First, we now require a much simpler simulator for the planning phase—one that models only the outcomes of motor controller execution, rather than the execution itself. Such a simulator is substantially easier to build for real robot problems. Second, MDHS can solve significantly larger problems than previous planners. For the bartenders and waiters problem, an accurate low-level simulator would have been hard to build; even if it had been built, generating a solution for the resulting problem would have been beyond the reach of existing planners. MDHS was able to *automatically generate controllers for a heterogeneous robot team that collectively maximized team utility, using only a high-level model of the task*. It is therefore a significant step forward in the development of general-purpose planners for cooperative multi-robot systems.

Acknowledgements

The research was completed while Chris Amato was in the Department of Computer Science at the University of New Hampshire and George Konidaris was in the Departments of Computer Science & Electrical and Computer Engineering at Duke University. We would also like to thank Wheeler Ruml for helpful discussions about heuristic search and Sammie Katt for assisting with finding errors in the paper.

Funding

The author(s) disclosed receipt of the following financial support for the research, authorship, and/or publication of this article: This work was supported by US Office of Naval Research under MURI program award #N000141110688, NSF award #1463945 and the ASD R&E under Air Force Contract #FA8721-05-C-0002. Opinions, interpretations, conclusions and recommendations are those of the author and are not necessarily endorsed by the United States Government.

Notes

1. Specifically, the previous Dec-POSMDP model considers a factored state which consists of the locations of each robot and an environment state. The transitions for the locations each robot are assumed to be independent of the other robots (i.e. no collisions) and the environmental state is assumed to be fully observable in particular locations. Therefore, the model used in this paper is the generalization of the previous one, allowing the algorithms in this paper to be applied in both cases (and the algorithms for the more specific Dec-POSMDP can be extended to this model).
2. Currently, human input is used to choose the macro-actions and abstract to sensor and communication information into discrete high-level observations, but removing this input is an area of future work.
3. For cases where the last macro-action does not terminate at exactly time step T, a partial reward may be generated by the simulator from the underlying Dec-POMDP or other reward model.
4. There is a standard error of approximately 1.3 in these calculations.

References

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

Amato C, Chowdhary G, Geramifard A, et al. (2013) Decentralized control of partially observable Markov decision processes. In: *Proceedings of the fifty-second IEEE conference on decision and control*, pp. 2398–2405.

Amato C, Konidaris GD, Cruz G, et al. (2015) Planning for decentralized control of multiple robots under uncertainty. In: *Proceedings of the international conference on robotics and automation*, pp. 1241–1248.

Amato C, Konidaris GD and Kaelbling LP (2014) Planning with macro-actions in decentralized POMDPs. In: *Proceedings of the international conference on autonomous agents and multiagent systems*, pp. 1273–1280.

Amato C and Zilberstein S (2009) Achieving goals in decentralized POMDPs. In: *Proceedings of the international conference on autonomous agents and multiagent systems*, pp. 593–600.

Bai H, Hsu D and Lee WS (2013) Integrated perception and planning in the continuous space: A POMDP approach. *International Journal of Robotics Research* 33: 1288–1302.

Belta C, Bicchi A, Egerstedt M, et al. (2007) Symbolic planning and control of robot motion. *Robotics & Automation Magazine, IEEE* 14(1): 61–70.

Bernstein DS, Amato C, Hansen EA, et al. (2009) Policy iteration for decentralized control of Markov decision processes. *Journal of Artificial Intelligence Research* 34: 89–132.

Bernstein DS, Givan R, Immerman N, et al. (2002) The complexity of decentralized control of Markov decision processes. *Mathematics of Operations Research* 27(4): 819–840.

Boutilier C (1999) Sequential optimality and coordination in multiagent systems. In: *Proceedings of the international joint conference on artificial intelligence*, pp. 478–485.

Capitán J, Spaan MTJ, Merino L, et al. (2013) Decentralized multi-robot cooperation with auctioned POMDPs. *International Journal of Robotics Research* 32(6): 650–671.

Cassandra AR, Kaelbling LP and Littman ML (1994) Acting optimally in partially observable stochastic domains. In: *Proceedings of the national conference on artificial intelligence*, pp. 1023–1028.

Dias MB and Stentz A (2003) A comparative study between centralized, market-based, and behavioral multirobot coordination approaches. In: *Proceedings of the IEEE/RSJ international conference on intelligent robots and systems*, volume 3, pp. 2279–2284.

Emery–Montemerlo R, Gordon G, Schneider J, et al. (2005) Game theoretic control for robot teams. In: *Proceedings of the international conference on robotics and automation*, pp. 1163–1169.

Foote T (2015) turtlebot_navigation - ros wiki. Available at: http://wiki.ros.org/turtlebot_navigation (accessed May 2015).

Gerkey BP and Matarić MJ (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 and Makar R (2006) Hierarchical multi-agent reinforcement learning. *Journal of Autonomous Agents and Multi-Agent Systems* 13(2): 197–229.

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

Kaelbling LP, Littman ML and Cassandra AR (1998) Planning and acting in partially observable stochastic domains. *Artificial Intelligence* 101: 1–45.

Kalra N, Ferguson D and Stentz A (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.

Liu M, Amato C, Anesta E, et al. (2016) Learning for decentralized control of multiagent systems in large partially observable stochastic environments. In: *Proceedings of the national conference on artificial intelligence*, pp. 2523–2529.

Liu M, Amato C, Liao X, et al. (2015) Stick-breaking policy learning in Dec-POMDPs. In: *Proceedings of the international joint conference on artificial intelligence*, pp. 2011–2017.

Loizou SG and Kyriakopoulos KJ (2004) Automatic synthesis of multi-agent motion tasks based on LTL specifications. In: *Proceedings of the forty-third IEEE conference on decision and control*, volume 1, pp.153–158. IEEE.

Messias JV, Spaan MTJ and Lima PU (2013a) GSMDPs for multi-robot sequential decision-making. In: *Proceedings of the twenty-seventh AAAI conference on artificial intelligence*, pp. 1408–1414.

Messias JV, Spaan MTJ and Lima PU (2013b) Multiagent POMDPs with asynchronous execution. In: *Proceedings of the international conference on autonomous agents and multi agent systems*, pp. 1273–1274.

Meuleau N, Kim KE, Kaelbling LP, et al. (1999) Solving POMDPs by searching the space of finite policies. In: *Proceedings of the conference on uncertainty in artificial intelligence*, pp. 417–426.

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

Oliehoek FA and Amato C (2016) *A Concise Introduction to Decentralized POMDPs*. Springer.

Omidshafiei S, Agha–mohammadi A, Amato C, et al. (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.

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

Poupart P and Boutilier C (2003) Bounded finite state controllers. *Advances in Neural Information Processing Systems* 16: 823–830.

Puterman ML (1994) *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley–Interscience.

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

Stroupe AW, Ravichandran R and 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*, volume 4, pp.4190–4197. IEEE.

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

Szer D and Charpillet F (2005) An optimal best-first search algorithm for solving infinite horizon DEC-POMDPs. In: *Proceedings of the European conference on machine learning*, pp.389–399.

Thrun S (1999) Monte carlo POMDPs. *Advances in Neural Information Processing Systems* 12: 1064–1070.

Wu F, Zilberstein S and 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 and Chen X (2010b) Rollout sampling policy iteration for decentralized POMDPs. In: *Proceedings of the conference on uncertainty in artificial intelligence*, pp. 666–673.