Policy Search for Multi-Robot Coordination under Uncertainty

Christopher Amato*, George Konidaris†, Ariel Anders‡, Gabriel Cruz‡,
Jonathan P. How§ and Leslie P. Kaelbling‡
*Department of Computer Science, University of New Hampshire, Durham, NH 03824
†Departments of Computer Science & Electrical and Computer Engineering, Duke University, Durham, NC 27708
‡CSAIL, MIT, Cambridge, MA 02139
§LIDS, MIT, Cambridge, MA 02139

***Abstract*—We introduce a principled method for multi-robot coordination based on a generic model (termed a MacDec-POMDP) of multi-robot cooperative planning in the presence of stochasticity, uncertain sensing and communication limitations. We present a new MacDec-POMDP planning algorithm that searches over policies represented as finite-state controllers, rather than the existing 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 to solve significantly larger problems than existing MacDec-POMDP planners. We demonstrate significantly improved performance over previous methods and application to a cooperative multi-robot bartending task, showing that our method can be used for actual multi-robot systems.**

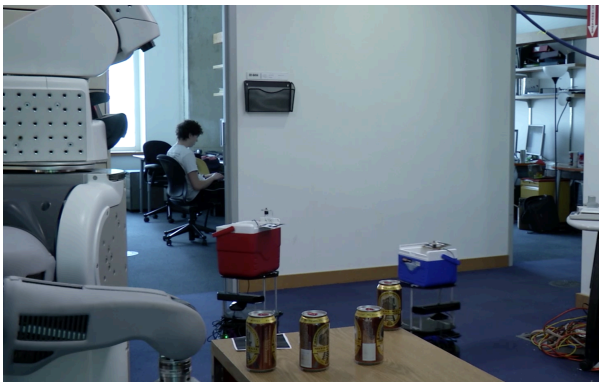


Fig. 1. The bartender and waiters domain

I. 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 [22] and partially observable Markov decision process [11]—have led to significant progress in single-robot settings through standardized models that enable empirical comparisons between very general planners that optimize a common metric.

Decentralized partially observable Markov decision processes (or *Dec-POMDPs* [8]) 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 [4], which includes temporally extended macro-actions that naturally model robot motor controllers which may require multiple time-steps to execute (e.g., navigating to a waypoint, lifting an object, or waiting for another robot). Planning now 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 robotics problems; for example, solving a multi-robot warehousing problem that was orders of magnitude larger than those solvable by previous methods [5]. 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.

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 still scales poorly with the horizon. Second, they assume that the underlying (primitive) problem is discrete, and that a low-level model of that problem is completely specified and available to all agents. These difficulties significantly limit the applicability of existing MacDec-POMDP planners.

This paper introduces a new MacDec-POMDP planning algorithm that 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. The resulting policy-search algorithms only require a model 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. We show that the new planner 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 our method can automatically optimize solutions to multi-robot problems from a high-level specification.

II. Motivating problem

As a motivating experimental domain we consider a heterogenous multi-robot problem, shown in Figure 1. The robot team consists of a PR2 bartender and two TurtleBot waiters. There are also three rooms in which people can order drinks from the waiters. Our goal is to bring drinks to the rooms with orders efficiently. 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 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.

III. Background

Decentralized Partially Observable Markov Decision Processes (Dec-POMDPs) generalize POMDPs to the multiagent, decentralized setting [8]. Multiple agents operate under uncertainty based on partial views of the world, with execution unfolding over a bounded or unbounded number of steps. At each step, every agent chooses an action (in parallel) based purely on locally observable information and then receives an observation. The agents share a single reward function based on the actions of all agents, making the problem cooperative, but their local views mean that execution is decentralized.

Formally, a Dec-POMDP is defined by 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 $A = \times_i A_i$ the set of joint actions; T is a state transition probability function, $T : S \times A \times S \to [0, 1]$, that specifies the probability of transitioning from state $s \in S$ to $s' \in S$ 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: $R : S \times A \to \mathbb{R}$, 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 $Z = \times_i Z_i$ the set of joint observations; O is an observation probability function: $O : Z \times A \times S \to [0, 1]$, the probability of seeing observations $o \in Z$ given actions $a \in A$ were taken which results in state $s' \in S$ (i.e., $O(o, a, s') = \Pr(o|a, s')$); and h is the number of (possibly infinite) steps until termination, called the horizon.

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, $H_i^O \to A_i$. 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 the POMDP case. 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, 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 set to 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 $\pi^*(s) = \mathrm{argmax}_\pi V^\pi(s)$. Dec-POMDP solution methods typically assume that the set of policies is generated in a centralized manner, but they are executed in a decentralized manner based on each agent's histories.

Although Dec-POMDPs have been widely studied [3, 18, 23], 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).

Macro-Actions for Dec-POMDPs: Dec-POMDPs typically require synchronous decision-making: every agent chooses an action to execute, and then executes it in a single time step. This restriction is problematic for two reasons. First, many systems use a set of controllers (e.g, for waypoint navigation, grasping an object, waiting for a signal), and planning consists of sequencing the execution of those controllers. These controllers often require different amounts of time to execute, so synchronous decision-making requires waiting until all agents have completed their controller execution (and achieved common knowledge of this fact). This is inefficient and may not even be possible in some domains (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 agents' possible policies at every time step will only ever be

able to make very short plans.

The Dec-POMDP model was recently extended to plan using macro-actions, or temporally extended actions [4] (hence the MacDec-POMDP model). This formulation uses higher-level planning to compute near-optimal solutions for problems with significantly longer horizons by extending the MDP-based options framework [26] to Dec-POMDPs by using macro-actions, m_i, that execute a policy in a low-level Dec-POMDP from states that satisfy a the initial conditions until some terminal condition is met. Policies for each agent, μ_i, can be defined for choosing macro-actions that depend on high-level observations. Because 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. 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: $V^{\mu}(s) = \mathrm{E}\left[\sum_{t=0}^{h-1} \gamma^t R(a^t, s^t) | s, \pi, \mu\right]$. The goal is to obtain a *hierarchically optimal policy*: $\mu^*(s) = \mathrm{argmax}_{\mu} V^{\mu}(s)$, which produces the highest expected value that can be obtained by sequencing the agents' given macro-actions.

Two Dec-POMDP algorithms have been extended to the MacDec-POMDP case [4], but other extensions are possible. The key difference is that nodes in a policy tree now select macro-actions (rather than primitive actions) and edges correspond to terminal conditions (or high-level observations).

IV. Finite-state controllers for MacDec-POMDPs

A tree-based representation of a policy requires the agent 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 an agent would need infinite memory. Clearly, this is not feasible. As an alternative, we introduce a finite-state controller representation and provide algorithms for generating these controllers.

Finite-state controllers (FSCs) can be used to represent policies 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 the infinite steps of the problem. Nodes in an agent's controller represent internal states, which prescribe actions based on that agent's finite memory.

A set of controllers, one per agent, provides the joint policy of the agents. FSCs are a widely used as solution representations for POMDPs and Dec-POMDPs [2, 6, 9, 15, 21, 27, 28].

A. Mealy Controllers

There are two main types of controllers, Moore and Mealy, that have been used for POMDP and Dec-POMDP solutions. Moore controllers associate actions with nodes and Mealy controllers associate actions with controller transitions (i.e., nodes and observations). We use the Mealy representation.

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

- 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 $\lambda_m(q, o)$.

Mealy controllers are a natural policy representation for MacDec-POMDPs because the initial conditions of macro-actions can be easily checked. That is, since the (macro-)action is chosen after an observation is seen, MacDec-POMDPs in which initial conditions depend solely on agent'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 agent in a simple manner.

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

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

where $\lambda(q) = \{\lambda_1(q_1, o_1), \ldots, \lambda_n(q_n, o_n)\}$ are the actions selected by each agent 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 agent given that agent's current node and observation. Because the first nodes do not depend on observations, the value of the controllers m at b is $V_m(b) = \sum_s b(s) V_m(q_0, s)$, where q_0 is the set of initial nodes for all agents.[1]

B. 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 $\lambda_i : Q_i \times Z_i \to M_i$ where Z_i are now the observations resulting from macro-actions and M_i are the macro-actions for agent i. Unfortunately, evaluation of these macro-action controllers is non-trivial due to the fact that macro-actions may require different amounts of time. Therefore, we must explicitly consider time when performing evaluation.

To perform this evaluation, we can build on recent work for modeling decentralized partially observable *semi-Markov* decision processes (Dec-POSMDPs) [19]. The Dec-POSMDP model explicitly considers actions with different durations, using a reward model that accumulates value until *any* agent

[1] The value can also be represented as $V_m(b) = \sum_s b(s) V_m(q_0, o^*, s)$, where $o*$ are dummy observations that are only received on the first step.

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 (e.g., when all agents are executing macro-actions that require a large amount of time to terminate the time for any agent terminating will be long).

We propose a more general Dec-POSMDP model as follows. We can model the state as $S \times_i S_i^m$, which includes the world state and a state of each of the agents' currently executing macro-actions. In our experimental domain, S_i^m will be the amount of time that agent's macro-action has been executing (since this is sufficient information to determine how much more time will be required in that problem), but S^m could correspond to a node in a lower-level finite-state controller or other relevant information. The actions are the macro-actions, $A_i = M_i$. The transition probability function, T, now includes the number of discrete time steps until completion as $\Pr(s', k|s, m)$, where k is this number of steps and m is the joint set of macro-actions being executed. The reward function, $R(s, m)$, is the value until any agent terminates, $E\{r_t + \ldots + \gamma^{k-1} r_{t+k} | s, m, t\}$, starting at time t. Z_i is now a finite set of high-level observations that are only observed after an agent's macro-action has been completed. The observation probability function, $\Pr(o|s', m)$, generates an observation for each agent based on the resulting state and the macro-action that was executed. The horizon h is the number of (low-level, not macro-action) steps until termination.

Using this model, we can evaluate a joint policy that is represented as a set of Mealy controllers, μ, using the following Bellman equation:

$$V^{\mu}(q, o, s) = R(s, \lambda(q, o)) + \sum_{k}^{\infty} \gamma^k \sum_{s'} \Pr(s', k|s, \lambda(q, o)) \times \sum_{o'} \Pr(o'|s', \lambda(q, o)) V^{\mu}(\delta(q, o'), o', s').$$

Note that, in the Dec-POSMDP, observations are only generated for agents that complete their macro-actions. As such, the observation, o_i, and the current controller node, q_i, do not update until agent i terminates its macro-action execution.

C. Exploiting domain structure

The Bellman equation provides a formal framework for evaluating policies in MacDec-POMDPs directly if we have a model (or simulator) of the system. It is often difficult to construct a full model in complex domains, but many domains possess structure that allows efficient evaluation.

For example, in our bartender domain, we perform a sample-based evaluation of policies using a high-level simulator. This simulator uses distributions for each macro-action describing the completion time at each terminal condition given each possible initial condition. Having this time information is a much less restrictive assumption than knowing the full policy of each macro-action. We will also assume that the reward only depends on the state, and the 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.

Algorithm 1 MacDec-POMDP Sample-Based Evaluation

```
function SAMPLEEVAL(μ,s_0,numSims,maxTime)
    totalReturn= 0
    for sim 0 to numSims do
        s = s_0, q = q^0, o = o*
        t = 0, t^Ag = 0⃗
        minTime=minAgents=termConds=null
        while t < maxTime do
            for all agents i ∈ I do
                for all terminal conditions β_i of λ_i(q_i) do
                    t ← SampleFromDist(s, λ_i(q_i), β_i)
                    if t − t_i^Ag = minTime then
                        minAgents ← minAgents ∪ i
                        termConds ←termConds ∪ β
                    else if t − t_i^Ag < minTime then
                        minAgents ← {i}
                        termConds ← {β}
            t+ =minTime
            s' ← sampledStateUpdate(s, λ(q),termConds)
            r ← R(s')
            for all agents i ∈minAgents do
                o_i ← sampleObs(s',termConds)
                q_i ← δ(q_i, o_i)
                t_i^Ag = 0
            for all agents i ∉minAgents do
                t_i^Ag+ =minTime
            totalReturn + = r
    return totalReturn/numSims
```

Pseudocode for sample-based evaluation is given in Algorithm 1. The simulator keeps track of the world state, the current node and last seen observation of each agent, the current time in the system and the amount of time each agent has been executing its macro-action. At each iteration, the simulator determines the set of agents which terminate their macro-actions in the least amount of time. The system state updates based on the termination of these completed macro-actions and the corresponding agents observe new observations and transition in their controllers. These iterations continue until the system time reaches a limit. This sample-based evaluation can calculate the value of policies in problems with very large state spaces using a small number of simulations.

V. POLICY SEARCH

Policy evaluation is an important step, but we must also determine the policies of each agent. We propose a heuristic search method that seeks to optimize the parameters of each agent's controller. Such methods have been able to generate high-quality controllers in the (primitive-action) Dec-POMDP case [1, 27]. 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. Our heuristic search method constructs a search tree of possible controllers for each

Algorithm 2 MacDec-POMDP Heuristic Search (MDHS)

function HEURSEARCH(s_0,n)
  $\underline{V} \leftarrow \underline{V}_{init}$
  $polSet \leftarrow \emptyset$
  repeat
    $\theta \leftarrow$selectBest($polSet$)
    $\Theta' \leftarrow$expandNextStep(θ)
    for $\theta' \in \Theta'$ **do**
      if isFulPol(θ',n) **then**
        $v \leftarrow$ valueOf(θ',s_0)
        if $v > \underline{V}$ **then**
          $\mu^* \leftarrow \theta$
          $\underline{V} \leftarrow v$
          prune($polSet$,$\underline{V}$)
      else
        $\bar{v} \leftarrow$ valueUpperOf(θ',s_0)
        if $\bar{v} > \underline{V}$ **then**
          $polSet \leftarrow polSet \cup \theta'$
    $polSet \leftarrow polSet \setminus \theta$
  until $polSet$ is empty
  return μ^*

agent, and searches through this space of policies by fixing the parameters (of all agents) for one node at a time, using heuristic upper bound values to direct the search.

Pseudocode of our policy search approach 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). An open list, $polSet$, which represents the set of partial policies that are available to be expanded is initialized to be the empty set. At each step, the partial joint policy (node in the search tree) with the highest estimated value is selected. Then, this partial policy is expanded, 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'. These expanded policies are examined to determine if they are fully specified. If they are, their value is compared with the value of the best known policy, which is updated accordingly (allowing for pruning of policies with value less than the new $\underline{V}$). 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. The partial policy that was expanded is removed from the candidate set (the open list) 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 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 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. In the future, we can explore tighter bounds.

	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}

TABLE I
VALUES, TIMES (IN S) AND POLICY SIZES ON NAMO BENCHMARKS OF SIZE 5×5 AND 25×25.

VI. EXPERIMENTS

We perform comparisons with previous work on existing benchmark domains and demonstrate the effectiveness of our MDHS policy search in the bartender scenario. For all comparisons, we report the best solution found by our method after a time cut-off of 180 seconds. In the first two problems, controller sizes are fixed to be 5 nodes, but in the future, sizes could be generated from trajectories in the simulator (similar to previous methods [1]). Experiments were run on a single core of a 2.2 GHz Intel i7 with a maximum of 8GB of memory. The benchmark and simulation experiments provide a rigorous quantitative analysis on the efficacy of the MacDec-POMDP planner, while the real world experiments show that our method can be used for actual multi-robot systems.

A. A benchmark problem

For comparison with previous methods, we consider robots navigating among movable obstacles [24]. This domain was designed as a finite-horizon problem [4] so we add a discount factor (of 0.9) for the infinite-horizon case. Previous MacDec-POMDP methods were only designed for finite-horizon problems [4], but can produce policies that have a high value in infinite-horizon problems by using a large planning horizon. We compare against an optimal (finite-horizon) method for horizon 50, which can produce a solution within 0.046 of the optimal (infinite-horizon) value in both instances of the problem we consider. As mentioned above, our MDHS method used 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 agent (which balances off time with computational complexity).

As seen in Table I, our method ("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.

B. A small warehousing problem

A small warehousing problem has also been modeled and solved as a MacDec-POMDP [5]. Previous solution methods (which are based on the policy tree methods [4]) 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.

	MDHS Controller		Tree	
	NoCom	ComLimit	NoCom	ComLimit
Value	11.38	12.41	-	-
Time	180	180	-	-
Size	5	5	-	-

TABLE II
VALUES, TIMES (IN S) AND POLICY SIZES ON TWO WAREHOUSING PROBLEMS (WITH NO AND LIMITED COMMUNICATION). THE TREE-BASED METHOD IS UNABLE TO SOLVE THESE PROBLEMS.

As seen in Table II, MDHS can produce concise solutions very quickly on these problems. A direct comparison with previous work is not possible (due to the existing method's lack of scalability to a large enough horizon and lack of bounds on the solution quality). Nevertheless, our method is able to solve these problems, while previous methods could not. We omit the results for the other (signaling) problem discussed in the previous paper, but the results are similar.

As an additional comparison, we also evaluate the controllers generated by our MDHS algorithm for the same number of steps that the previous algorithm used (9) without a discount factor. In this case, the previous tree based method produced solutions with values of 1.16 and 1.6 while our method produces solutions with values 1.12 and 1.14. 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.

C. 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. Any waiter can fulfill an order (even if that waiter did not have previous knowledge about the order). Drink orders are created stochastically: a new order will be created 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 $100 - (t_{now} - t_{order})/10$, 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 III. The state variables each waiter can observe are shown in Table IV.

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 each possible initial condition and executing until each possible terminal condition, generating a distribution for the results). 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, but these could be easily modeled.

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 III
MACRO-ACTIONS 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

TABLE IV
OBSERVATIONS FOR THE WAITERS.

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. 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. No communication was used except between the bartender and waiters in the bar area. There were also 64 observations (from Table IV) and the underlying state space consists of the continuous locations of the TurtleBots and the status of the PR2 (which we abstract into execution time in macro-actions) and discrete variables for orders in each room and whether each TurtleBot is holding a beverage.

D. Robot implementation

As shown in Figure 1, we used two TurtleBots (we will call the blue one *Leonardo* and the red one *Raphael*) as waiters and the PR2 as a bartender. The TurtleBots had 2 types of macro-actions: navigation and obtaining a drink from the bartender. The navigation actions were created using a given map, shown in Figure 2, with 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.

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.

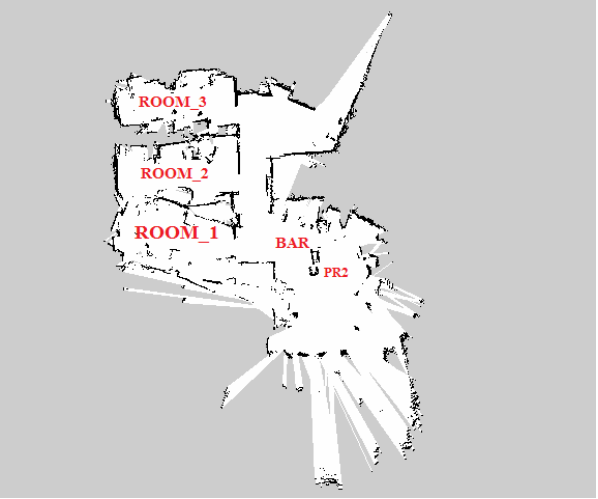


Fig. 2. Navigation map generated by the TurtleBots.

E. 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). The solution is a Mealy controller that maps nodes and observations to actions. To easily examine the results, we generated policies with 1 and 5 nodes. In the 1 node case, the action selections are memoryless and depend solely on the current state observation. For the 5 node case, the node transitions allow each robot to remember some relevant information.

Figures 3(a)–3(c) show parts of the generated 1-node 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. In general, 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 TurtleBot's 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 3(a) shows the macro-actions for each TurtleBot when it is located in the bar. BAR is a navigation macro-action that takes the TurtleBot to the bar from any location. 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 time visit the other rooms when the bartender is not ready_to_serve.

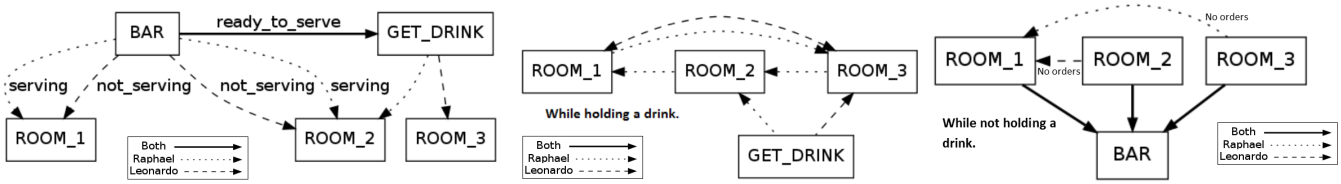
Once a TurtleBot is holding a drink, it is in the *delivery* phase. Figure 3(b) shows the sequence of macro-actions executed in this case. Raphael receives a drink from the bartender and tries to complete deliveries in the following order: ROOM_2, ROOM_1, ROOM_3. That is, it continues looping through 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.

It is important to note that the one-node controllers cannot contain a more complex solution that would let the waiters go to different rooms after receiving a drink. 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 clear cooperation between the two robots to efficiently cover the rooms.

After a TurtleBot has delivered a drink, it enters the *ordering* phase. Figure 3(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 4. Initially, the TurtleBots start in the bar next to the PR2. The PR2 immediately starts picking up a can (Figure 4(a)) and the two TurtleBots navigate to different rooms (Figure 4(b)). Then Leonardo successfully receives a drink from the PR2 (Figure 4(c)) and starts the *delivery* phase by going to room 2 (Figure 4(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 4(f)). While Leonardo is served by the PR2, Raphael goes to the bar and observes the PR2 is busy (Figure 4(d)). Since the PR2 is busy, Raphael continues the *ordering* phase by navigating to room 1 (Figure 4(e)).

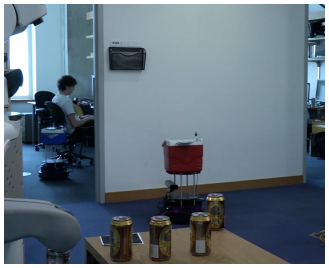
We also tested a 5-node controller on the robots. The solutions for the 1-node and 5-node cases deviate since the 5-node controller can keep track of more information. For example, the waiter can choose a different room based on information such as the other rooms the waiter has been in previously or whether orders have been placed in the other rooms recently. These solutions quickly become complex and non-obvious to specify by hand. Both controller sizes allow for high-quality solutions, but having 5 nodes permits improved cooperation. For instance, the 5-node solutions were able to select different rooms to visit after receiving a drink from the PR2 based on memory of orders in those locations as well as observing the other waiter. These improvements are seen in the simulator where solution value (over 50 macro-action steps or approximately 1000 low-level steps) was 1231 (an average of 13.98 drinks delivered) for the 1-node controller and 1296 (14.56 drinks delivered) for the 5-node controller. 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 917 (11.13 drinks delivered).



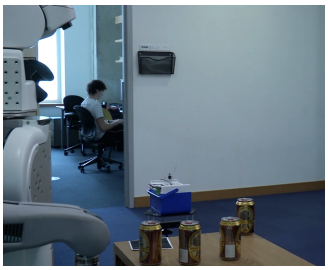
(a) *Bar* phase while located in the bar. (b) *Delivery* phase while holding a drink. (c) *Ordering* phase while not holding a drink.

Fig. 3. Controller phases for each waiter.

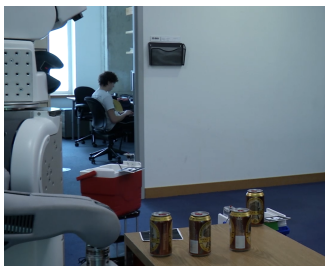
(a) PR2 picking up a drink.



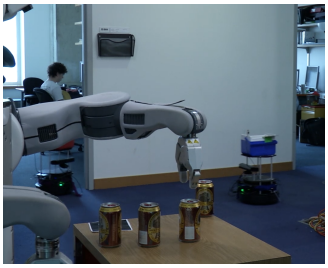
(b) TurtleBots go to first rooms.



(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.

Fig. 4. Images from the bartender and waiter experiments.

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.

VII. 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 [20] or tightly-coupled [25] 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 [12, 14]. These approaches have been used to solve real multi-robot problems [16, 10], 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) [7, 17] 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 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. [13] 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.

VIII. Summary and Conclusion

We introduced an extended MacDec-POMDP model for representing cooperative multi-robot systems under uncertainty using a high-level problem description. We also developed MDHS, a new MacDec-POMDP planning algorithm that searches over policies represented as finite-state controllers. In the bartender and waiters problem, MDHS was able to *automatically generate controllers for a heterogenous robot team that collectively maximized team utility, using only a high-level model of the task*. For this problem, an accurate low-level simulator would have been hard to build, and generating a solution is beyond the reach of existing planners. MDHS is therefore a significant step forward in the development of general-purpose planners for cooperative multi-robot systems.

Acknowledgments

Research 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.

References

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

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

[3] Christopher Amato, Girish Chowdhary, Alborz Geramifard, Nazim Kemal Ure, and Mykel J. Kochenderfer. Decentralized control of partially observable Markov decision processes. In *Proceedings of the Fifty-Second IEEE Conference on Decision and Control*, pages 2398–2405, 2013.

[4] Christopher Amato, George D. Konidaris, and Leslie P. Kaelbling. Planning with macro-actions in decentralized POMDPs. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*, pages 1273–1280, 2014.

[5] Christopher Amato, George D. Konidaris, Gabriel Cruz, Christopher A. Maynor, Jonathan P. How, and Leslie P. Kaelbling. Planning for decentralized control of multiple robots under uncertainty. In *Proceedings of the International Conference on Robotics and Automation*, 2015.

[6] Haoyu Bai, David Hsu, and Wee Sun Lee. Integrated perception and planning in the continuous space: A POMDP approach. *International Journal of Robotics Research*, 33:1288–1302, 2013.

[7] Calin Belta, Antonio Bicchi, Magnus Egerstedt, Emilio Frazzoli, Eric Klavins, and George J Pappas. Symbolic planning and control of robot motion. *Robotics & Automation Magazine, IEEE*, 14(1):61–70, 2007.

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

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

[10] Jesús Capitán, Matthijs T. J. Spaan, Luis Merino, and Aníbal Ollero. Decentralized multi-robot cooperation with auctioned POMDPs. *International Journal of Robotics Research*, 32(6):650–671, 2013.

[11] Anthony R. Cassandra, Leslie Pack Kaelbling, and Michael L. Littman. Acting optimally in partially observable stochastic domains. In *Proceedings of the National Conference on Artificial Intelligence*, 1994.

[12] M. Bernardine Dias and Anthony Stentz. 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, pages 2279 – 2284, October 2003.

[13] Rosemary Emery-Montemerlo, Geoff Gordon, Jeff Schneider, and Sebastian Thrun. Game theoretic control for robot teams. In *Proceedings of the International Conference on Robotics and Automation*, pages 1163–1169, April 2005.

[14] Brian P. Gerkey and Maja J. Matarić. A formal analysis and taxonomy of task allocation in multi-robot systems. *International Journal of Robotics Research*, 23(9):939–954, 2004.

[15] Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101:1–45, 1998.

[16] Nidhi Kalra, David Ferguson, and Anthony Stentz. Hoplites: A market-based framework for planned tight coordination in multirobot teams. In *Proceedings of the International Conference on Robotics and Automation*, pages 1170 – 1177, April 2005.

[17] Savvas G. Loizou and Kostas J. Kyriakopoulos. 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, pages 153–158. IEEE, 2004.

[18] Frans A. Oliehoek. Decentralized POMDPs. In Marco Wiering and Martijn van Otterlo, editors, *Reinforcement Learning: State of the Art*, volume 12 of *Adaptation, Learning, and Optimization*, pages 471–503. Springer Berlin Heidelberg, 2012.

[19] Shayegan Omidshafiei, Ali-akbar Agha-mohammadi, Christopher Amato, and Jonathan P. How. Decentralized control of partially observable Markov decision processes using belief space macro-actions. In *Proceedings of the International Conference on Robotics and Automation*, 2015.

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

[21] Pascal Poupart and Craig Boutilier. Bounded finite state controllers. In *Advances in Neural Information Processing Systems*, 16, pages 823–830. 2003.

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

[23] Sven Seuken and Shlomo Zilberstein. Formal models and algorithms for decentralized control of multiple agents. *Journal of Autonomous Agents and Multi-Agent Systems*, 17(2):190–250, 2008.

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

[25] Ashley W Stroupe, Ramprasad Ravichandran, and Tucker Balch. 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, pages 4190–4197. IEEE, 2004.

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

[27] Daniel Szer and François Charpillet. An optimal best-first search algorithm for solving infinite horizon DEC-POMDPs. In *Proceedings of the European Conference on Machine Learning*, pages 389–399, 2005.

[28] Feng Wu, Shlomo Zilberstein, and Xiaoping Chen. Point-based policy generation for decentralized POMDPs. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*, pages 1307–1314, 2010.