Planning with Macro-Actions in Decentralized POMDPs

Christopher Amato, George D. Konidaris and Leslie P. Kaelbling
MIT CSAIL
Cambridge, MA 02139
{camato, gdk, lpk}@csail.mit.edu

ABSTRACT

Decentralized partially observable Markov decision processes (Dec-POMDPs) are general models for decentralized decision making under uncertainty. However, they typically model a problem at a low level of granularity, where each agent's actions are primitive operations lasting exactly one time step. We address the case where each agent has macro-actions: temporally extended actions which may require different amounts of time to execute. We model macro-actions as *options* in a factored Dec-POMDP model, focusing on options which depend only on information available to an individual agent while executing. This enables us to model systems where coordination decisions only occur at the level of deciding which macro-actions to execute, and the macro-actions themselves can then be executed to completion. The core technical difficulty when using options in a Dec-POMDP is that the options chosen by the agents no longer terminate at the same time. We present extensions of two leading Dec-POMDP algorithms for generating a policy with options and discuss the resulting form of optimality. Our results show that these algorithms retain agent coordination while allowing near-optimal solutions to be generated for significantly longer horizons and larger state-spaces than previous Dec-POMDP methods.

1. INTRODUCTION

Dec-POMDPs [8] are a very general model for sequential cooperative decision-making under uncertainty and partial observability. However, large problem instances remain intractable: some advances have been made in optimal algorithms [1, 2, 3, 9, 10, 23], but most approaches that scale well make very strong assumptions about the domain (e.g., assuming a large amount of independence between agents) [11, 20, 22] and/or have no guarantees about solution quality [24, 25, 31]. One reason for this difficulty is that actions are modeled as primitive (low-level) operations which last exactly one time step. The length of a single step can be adjusted (trading off solution quality for horizon length) but is always assumed to be the same for all agents. This allows synchronized action selection, but also requires reasoning about action selection and coordination at every time step.

Appears in: *Alessio Lomuscio, Paul Scerri, Ana Bazzan, and Michael Huhns (eds.), Proceedings of the 13th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2014), May 5-9, 2014, Paris, France.*

In single-agent domains, hierarchical approaches [4] using higher-level, temporally extended macro-actions have been explored as a more natural way to represent and solve problems, leading to significant performance improvements in planning [26, 30]. Our aim is to extend these methods to the multiagent case.

The primary technical challenge in using temporally extended actions for multiagent scenarios is that agent decision-making may no longer be synchronized: each agent's actions may complete execution at different times. In single-agent domains, an existing hierarchical reinforcement learning framework, the *options framework* [30], can be used to naturally model both the internal policies of each macro-action (or *option*) and the overall option selection policy. Extending these ideas to the multiagent case, we present a Dec-POMDP formulation where agents use macro-actions defined as options. To permit coordination, agents can use their shared option policies to reason about the progress of other agents and their impact on the world, allowing them to explicitly reason over what the other agents are doing and will do in the future.

The use of macro-actions in the multiagent case can incorporate the benefits of the single agent case such as simpler and more efficient modeling of real systems (e.g., robots with actions that execute predefined controllers) [29], more efficient planning [30], skill transfer [16], and skill-specific abstractions [12]. Additional benefits can be gained by exploiting known structure in the multiagent problem. For instance, in some cases macro-actions may only depend on locally observable information. One example is a robot navigating to a waypoint or target. Only local information is required for navigation, but choosing which waypoint or target to travel to likely requires coordinated decision-making. Macro-actions with independent execution allow coordination decisions to be made only when necessary (i.e., when choosing macro-actions) rather than at every time step. Because we build on top of Dec-POMDPs, macro-action choice may depend on history, but during execution macro-actions may be independent (as above), depend on any number of steps of history, may represent the actions of a set of agents, etc. Thus, our macro-action representation is a very general way to incorporate domain knowledge to improve efficiency in representation and solution. The benefit of a macro-action-based approach is that it enables high-level planning, potentially allowing near-optimal solutions for problems with significantly longer horizons and larger state-spaces than previous Dec-POMDP methods could solve.

We focus on the case where the agents are given *local* *options* which depend only on information locally observable to the agent during execution. We present a factored Dec-POMDP model for using such options, and introduce extensions of dynamic programming and memory-bounded dynamic programming algorithms for planning. Our results show that high-quality solutions can be found for a typical Dec-POMDP benchmark as well as large problems that traditional Dec-POMDP methods cannot solve: a four agent meeting-in-a-grid problem and a domain based on robots navigating among movable obstacles [28].

2. BACKGROUND

We first present a factored Dec-POMDP model and discuss options in the single-agent case.

2.1 Factored Dec-POMDPs

A Dec-POMDP [7] involves multiple agents that operate under uncertainty based on (possibly different) partial views of the world and unfolds over a finite or infinite sequence of steps. At each step, every agent chooses an action (in parallel) based purely on locally observable information, resulting in an immediate reward and an observation (which we will call a local state) being observed by each individual agent.

We consider the case where the state space is factored into k components, $S = S^1 \times \ldots \times S^k$. Each agent can fully observe some subset of these state factors, whose integer indices we include in set F_i. As such, we can define a set of *local states* for an agent $S_i = \times_j S^j$ for $j \in F_i$. A simple factorization could include factors for the location of each agent as well as the location (or condition) of several movable objects (such as boxes or other obstacles). Each agent could then observe its own location and the locations of some subset of objects. We could also include additional factors to provide partial or location-dependent information to the agents about box and other agent locations. Note that the observable state factors may overlap (e.g., when each agent can observe the location of the same object) and may not be exhaustive (i.e., when the problem is partially observable).

A *factored n-agent Dec-POMDP* is then specified by a tuple $(I, \{S^j\}, \{A_i\}, T, R, h)$, where:

- I is a finite set of agents,
- S^j is a finite set of values for each state factor with initial state value s_0^j, $S = \times_j S^j$ the set of joint states and S_i, the finite set of local states for each agent,
- 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 transition probability function $\Pr(s'|s, \vec{a})$ that specifies the probability of transitioning from state s to s' when actions $\vec{a}$ are taken by the agents,
- R is a reward function $R(s, \vec{a})$ specifing the immediate reward for being in state s and taking actions $\vec{a}$,
- and h is the *horizon*—the number of steps after which the problem terminates.

While this representation appears different from the traditional Dec-POMDP formulation, it can represent (almost) any Dec-POMDP problem. Specifically, the traditional formulation using observations can be represented in this factored way by considering $n + 1$ state factors: one for each agent's observation set and one for the (hidden) states of the Dec-POMDP. Each agent then has full observability of its associated observations, but not the other agent observations or the underlying system state. That is, the local states for each agent would just be their observations while the true state of the world is hidden.[1] Therefore, the approaches here are applicable to any Dec-POMDP domain.

Because the full state is not directly observed, it may be beneficial for each agent to remember a history of local states. Unlike in single-agent (or centralized) POMDPs, it is not typically possible to calculate a centralized estimate of the system state from the observation history of a single agent. A solution to a Dec-POMDP is then a *joint policy* — a set of policies, one for each agent in the problem. A *local policy* for an agent is a mapping from local observation histories to actions. The goal is to maximize the total cumulative reward, beginning at initial state s_0. The value of a joint policy, π, from state s is

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

which represents the expected value of the immediate reward for the set of agents summed for each step of the problem given the action prescribed by the policy until the horizon is reached. In the finite-horizon case, the discount factor, γ, is typically set to 1. An *optimal policy* beginning at state s is $\pi^*(s) = \mathrm{argmax}_\pi V^\pi(s)$.

Solving Dec-POMDPs is very computationally challenging (NEXP for optimal finite-horizon solutions [7]), but many problems may have structure that allows near optimal solutions to be found quickly. In the single-agent case, MDPs have benefited from temporally extended actions (options), allowing learning or planning to take place at a higher level.

2.2 Options in MDPs

In the single agent, fully observable setting an option is defined by a tuple:

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

consisting of a stochastic *termination condition* $\beta_m : S \to [0, 1]$, a (state-based) *initiation set* $\mathcal{I}_m \subset S$ and a stochastic *option policy* mapping states and actions to probabilities $\pi_m : S \times A \to [0, 1]$. The resulting problem is known as a *Semi-Markov Decision Process*, or SMDP [30]. Note that we can create an option for a single-step action a by defining $\pi_m(s, a) = \beta_m(s) = 1, \forall s$, and $\mathcal{I}_m = S$.

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

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

where $p(s'|s, m) = \sum_{k=0}^{\infty} p(s', k)\gamma^k$ with $p(s', k)$ representing the probability that option m will terminate in state s' after k steps.

Using this framework directly is not possible in a multiagent setting because some agents' options would terminate while others are still executing their options. That is, it is not clear how this Bellman equation can be directly extended to evaluate option choices for a subset of agents while the remaining agents continue execution.

[1]Extra observations may be needed to deal with choosing an action in the (fully observable) start state.

3. OPTIONS IN DEC-POMDPS

We describe an extension of single-agent options that allows temporally extended actions to be incorporated into multiagent planning. We focus on local options, though the framework could be extended to other types of options.

3.1 Local Options

A factored Dec-POMDP with local options is defined as a factored Dec-POMDP where we also assume M_i represents a finite set of options for each agent, i, with $M = \times_i M_i$ the set of joint options. A *local option* is defined by the tuple:

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

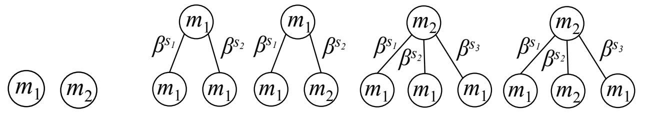
consisting of stochastic termination condition $\beta_{m_i} : S_i \to [0,1]$, initiation set $\mathcal{I}_{m_i} \subset S_i$ and option policy $\pi_{m_i} : S_i \times A_i \to [0,1]$. Note that this is the simplest case, where the policy, termination condition and initiation set depend only on local states. As we later discuss, option policies can also depend on an agent's history and initiation and terminal conditions can depend on histories or global states (e.g., also ending execution based on unobserved events).

Because it may be beneficial for agents to remember their histories when choosing an option to execute, we consider policies that remember option histories. We define an *option history* as $H_i = (s_i^0, m_i^1, \dots, s_i^{l-1}, m_i^l)$ which includes both the local states where an option was chosen and the selected options. While a history over primitive actions also provides the number of steps that have been executed in the problem (because it includes actions and observations at each step), an option history may require many more steps to execute than the number of options listed. We can also define a (stochastic) *local policy*, $\mu_i : H_i \times M_i \to [0,1]$ that depends on option histories. We then define a *joint policy* as μ which is a set of local policies (one for each agent).

Because option policies are constructed from primitive actions, we can evaluate policies with methods that are similar to other Dec-POMDP-based approaches. Given a joint policy, the primitive action at each step is given by the high level policy which chooses the option and the option policy which chooses the action. The joint policy and option policies can then be evaluated as: $V^\mu(s) = \mathrm{E}\left[\sum_{t=0}^{h-1} \gamma^t R(\vec{a}^t, s^t) | s, \pi, \mu\right]$. For example, we can evaluate a joint 2-agent policy μ which begins with options m_1 and m_2 at state s and executes for t steps as:

$$\begin{aligned}
&V_t^\mu(m_1, m_2, s) = \\
&\sum_{a_1,a_2} \pi_{m_1}(s_1, a_1)\pi_{m_2}(s_2, a_2)\Bigg[R(a_1, a_2, s) + \sum_{s_1',s_2'} T(s', a_1, a_2, s) \\
&\Bigg(\beta_{m_1}(s_1')\beta_{m_2}(s_2') \sum_{m_1',m_2'} \mu_1(s_1', m_1')\mu_2(s_2', m_2')V_{t-1}^\mu(s', m_1', m_2') \\
&\quad + \beta_{m_1}(s_1')\big(1 - \beta_{m_2}(s_2')\big) \sum_{m_1'} \mu_1(s_1', m_1')V_{t-1}^\mu(s', m_1', m_2) \\
&\quad + \big(1 - \beta_{m_1}(s_1')\big)\beta_{m_2}(s_2') \sum_{m_2'} \mu_2(s_2', m_2')V_{t-1}^\mu(s', m_1, m_2') \\
&\qquad + \big(1 - \beta_{m_1}(s_1')\big)\big(1 - \beta_{m_2}(s_2')\big)V_{t-1}^\mu(s', m_1, m_2)\Bigg)\Bigg],
\end{aligned}$$

where s_1, s_2 and s_1', s_2' are local states for agents 1 and 2 extracted from s and s' respectively. Note that agents' options may terminate at different times; the appropriate action is then chosen by the relevant agent's policy and evaluation continues. Because we are interested in a finite-horizon problem, we assume evaluation continues for h steps.



(a) Step 1 (b) Step 2 of DP

Figure 1: Policies for a single agent after one step of dynamic programming using options m_1 and m_2 where (deterministic) terminal states for options are represented as β^s.

3.2 Optimal Policies

We can define a *hierarchically optimal policy* $\mu^*(s) = \mathrm{argmax}_\mu V^\mu(s)$ which defines the highest-valued policy among those that use the given options. Because this policy may not include all possible history-dependent policies, it may have lower value than the optimal policy for the underlying Dec-POMDP (the *globally optimal policy*).[2]

A globally optimal policy can be guaranteed by including the primitive actions in the set of options for each agent. Because the same set of policies can be created from this primitive option set as would be created in the underlying Dec-POMDP, using this set of options retains the optimal policy. While including primitive actions in the option sets can allow a globally optimal policy to be retained, it typically makes little sense to do so. This is because option-based algorithms would now be reasoning about options at each time step and searching in the same space of policies as algorithms that solve the underlying Dec-POMDP directly.

4. ALGORITHMS

Because Dec-POMDP algorithms produce policies mapping agent histories to actions, they can be extended to consider options instead of primitive actions. We discuss how options can be incorporated into two such algorithms; extensions can also be made to other approaches.

In both cases, deterministic polices are generated which are represented as policy trees (as shown in Figure 1). A policy tree for each agent defines a policy that can be executed based on local information. The root node defines the option to choose in the known initial state, and options are specified for each legal terminal state of the root option (as seen in Figure 1(b)); this continues for the depth of the tree. Such a tree can be evaluated up to a desired horizon using the policy evaluation given above, which may not reach some nodes of the tree due to the differing execution times of some options.

4.1 Dynamic Programming

A simple exhaustive search method can be used to generate hierarchically optimal deterministic policies which utilize options. This algorithm is similar in concept to the dynamic programming algorithm used in Dec-POMDPs [14], but full

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

evaluation and pruning (removing dominated policies) are not used at each step. Instead we can exploit the structure of options to reduce the space of policies considered.

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

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

```
 1: function OptionDecDP($h$)
 2:     $t \leftarrow 0$
 3:     $someTooShort \leftarrow true$
 4:     $\mu_t \leftarrow \emptyset$
 5:     repeat
 6:         $\mu_{t+1} \leftarrow$ExhaustiveBackup($\mu_t$)
 7:         $someTooShort \leftarrow$TestPolicySetsLength($\mu_{t+1}$)
 8:         $t \leftarrow t+1$
 9:     until $someTooShort = false$
10:     Compute $V^{\mu_t}(s_0)$
11:     return $\mu_t$
12: end function
```

This algorithm will produce a hierarchically optimal deterministic policy because it constructs all legal deterministic option policies that are guaranteed to reach horizon h. This follows from the fact that options must last at least one step and all combinations of options are generated at each step until it can be guaranteed that additional backups will cause redundant policies to be generated. Our approach represents exhaustive search in the space of legal policies that reach a desired horizon. As such it is not a true dynamic programming algorithm, but additional ideas from dynamic programming for Dec-POMDPs [14] can be incorporated. For instance, we could prune policies based on value, but this would require evaluating all possible joint policies at every state after each backup. This evaluation would be very costly as the policy would be flattened after each backup and all combinations of flat policies would be evaluated for all states for all possible reachable horizons. Instead, the benefit of our approach is that only legal policies are generated using the initiation and terminal conditions for options. As seen in Figure 1(b), option m_1 has two possible terminal states while option m_2 has three. Furthermore, only option m_1 is applicable in local states s_1 and s_3. This structure limits the branching factor of the policy trees produced and thus the number of trees considered.

4.2 Memory-Bounded Dynamic Programming

Memory-bounded dynamic programming (MBDP) [25] can also be extended to use options as shown in Algorithm 2. Here, only a finite number of policy trees are retained (given by parameter $MaxTrees$) after each backup. After an exhaustive backup has been performed, a set of t-step trees is chosen by evaluating the trees at states that are generated by a heuristic policy (H_{pol} in the algorithm) that is executed for the first $h-t-1$ steps of the problem. A set of $MaxTrees$ states is generated and the highest valued trees for each state are kept. This process of exhaustive backups and retaining $MaxTrees$ trees continues, using shorter and shorter heuristic policies until the all combinations of the retained trees reach horizon h. Again, the set of trees with the highest value at the initial state is returned.

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

```
 1: function OptionMBDP($MaxTrees$,$h$,$H_{pol}$)
 2:     $t \leftarrow 0$
 3:     $someTooShort \leftarrow true$
 4:     $\mu_t \leftarrow \emptyset$
 5:     repeat
 6:         $\mu_{t+1} \leftarrow$ExhaustiveBackup($\mu_t$)
 7:         Compute $V^{\mu_{t+1}}$
 8:         $\hat{\mu}_{t+1} \leftarrow \emptyset$
 9:         for all $k \in MaxTrees$ do
10:             $s_k \leftarrow$ GenerateState($H_{pol}$,$h-t-1$)
11:             $\hat{\mu}_{t+1} \leftarrow \hat{\mu}_{t+1} \cup \arg\max_{\mu_{t+1}} V^{\mu_{t+1}}(s_k)$
12:         end for
13:         $t \leftarrow t+1$
14:         $\mu_{t+1} \leftarrow \hat{\mu}_{t+1}$
15:     until $someTooShort = false$
16:     return $\mu_t$
17: end function
```

This approach is potentially suboptiomal because a fixed number of trees are retained, and tree sets are optimized over states that are both assumed to be known and may never be reached. Nevertheless, since the number of policies retained at each step is bounded by $MaxTrees$, MBDP has time and space complexity linear in the horizon. As a result, MBDP (and its extensions [2, 18, 32]) have been shown to perform well in many large Dec-POMDPs. The option-based extension of MBDP uses the structure provided by the initiation and terminal conditions as in the dynamic programming approach in Algorithm 1, but does not have to produce all policies that will reach horizon h. Scalability can therefore be adjusted by reducing the $MaxTrees$ parameter (although solution quality may be reduced for smaller $MaxTrees$).

5. RELATED WORK

While many hierarchical approaches have been developed for multiagent systems [15], very few are applicable to multiagent models based on MDPs and POMDPs. Perhaps the most similar approach is that of Ghavamzadeh et al. [13]. This is a multiagent reinforcement learning approach with a given task hierarchy where communication is used to coordinate actions at higher levels and agents are assumed to be independent at lower levels. This work is limited to a multiagent SMDP model with (potentially costly) communication, making the learning problem challenging, but the planning problem is simpler than the full Dec-POMDP case.

Other approaches have considered identifying and exploiting independence between agents to limit reasoning about coordination. Approaches include general assumptions about agent independence like transition independent Dec-MDPs [6] and factored models such as ND-POMDPs [22] as well as methods that consider coordination based on "events" or states. Events which may require or allow interaction have been explored in Dec-MDPs [5] and (centralized) multi-robot systems [21]. Other methods have considered locations or states where interaction is needed to improve scalability in planning [27, 31] and learning [20].

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

6. EXPERIMENTS

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

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

The meeting-in-a-grid problem is an existing two-agent Dec-POMDP benchmark in which agents receive 0 reward unless they are both in one of two corners in a 3x3 grid [2]. Agents can move up, down, left, right or stay in place, but transitions are noisy, so an agent may move to an adjacent square rather than its desired location. Each agent has full observability of its own location, but cannot observe the other agent (even when they share the same grid square). We defined two options for each agent: each one moving the agent to one of the two goal corners. Options are valid in any (local) state and terminate when they reach the appropriate goal corner. An agent stays in a corner on a step by choosing the appropriate option again. It is clear that these options provide the important macro-actions for the agents and navigation is possible based on local information in this problem.

Results for this problem are split between Figure 2 and Table 1 because not all results are available for all algorithms. We compared against three leading approximate Dec-POMDP algorithms: MBDP with incremental policy

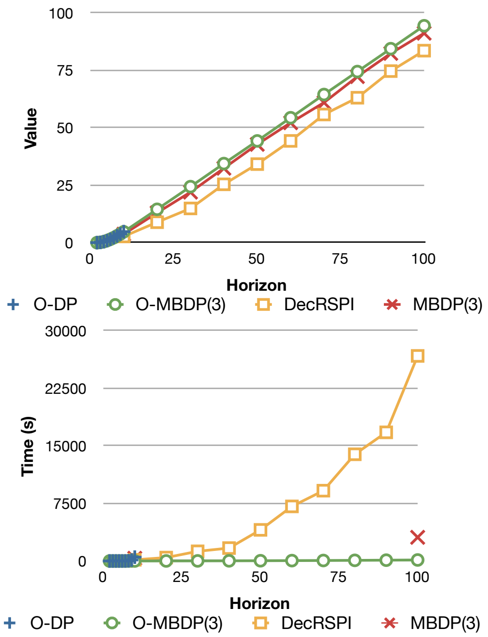


Figure 2: Value and time results for the meeting in a grid Dec-POMDP benchmark including leading Dec-POMDP approaches DecRSPI and MBDP as well as option-based DP and MBDP.

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

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

generation (MBDP-IPG) [2], rollout sampling policy iteration (DecRSPI) [33] and trial-based dynamic programming (TBDP) [34]. $MaxTrees = 3$ was used in both O-MBDP and MBDP-IPG (referred to as MBDP in the figure and table). Results for other algorithms are taken from the their respective publications. As such, results were generated on different machines, but the trends should remain the same. The top figure shows that all approaches achieve approximately the same value, but option-based DP (O-DP) cannot solve horizons longer than 10 without running out of memory. The bottom figure shows the time required for different horizons. All approaches run quickly for small horizons, but DecRSPI required an intractable amount of time as the horizon grows. The table shows time and value results for larger horizons. Again, all approaches achieve similar values, but O-MBDP is much faster than MBDP-IPG or TBDP.

Larger Grids with More Agents.

To test the scalability of these approaches, we consider growing the meeting-in-a-grid benchmark to a larger grid size and a larger number of agents. That is, agents still receive zero reward unless *all* agents are in one of the goal corners. The same options are used as in the 3x3 version of the problem. We generated results for several four-agent

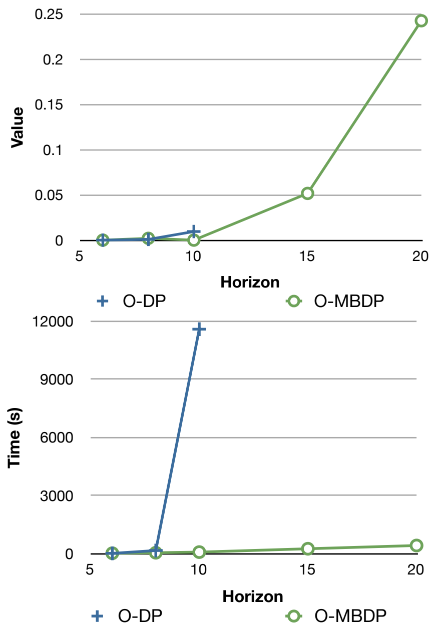


Figure 3: 4-agent meeting in a grid results showing (a) value and (b) running time on a 10×10 grid.

problems with random starting locations for each agent. We did not compare with current optimal or approximate Dec-POMDP methods because current implementations are not applicable to problems with more than two agents.

Results for option-based dynamic programming and MBDP on problems with a 10×10 grid are shown in Figure 3. Three trees were used for O-MBDP. It is worth noting that these are very large problems with 10^8 states. The dynamic programming method is able to solve problems with a long enough horizon to reach the goal (producing positive value), but higher horizons are not solvable. The MBDP-based approach is able to solve much larger horizons, requiring much less time than O-DP. O-MBDP is able to produce near-optimal values for horizons that are also solvable by O-DP, but results may be further from optimal as the horizon grows (as is often the case with MBDP-based approaches).

Two-Agent NAMO.

We also consider a two-agent version of the problem of robots navigating among movable obstacles [28]. Here, as shown in Figure 4, both agents are trying to reach a goal square (marked by G), but there are obstacles in the way. Each robot can move in four directions (up, down, left and right) or use a 'push' action to attempt to move a box to a specific location (diagonally down to the left for the large box and into the corner for both small boxes). The push action fails and the robot stays in place when the robot is not in front of the box. Robots can move the small boxes (b_1 and b_2) by themselves, but must move the larger box (b_3) together. Observations are an agent's own location (but not the location of the other agent) and whether the large box or the same numbered box has been moved (i.e., agent 1 can observe box 1 and agent 2 can observe box 2). There is noise in both navigation and in box movement: movement is successful with probably 0.9 and pushing the small and large boxes is successful with probably 0.9 and 0.8, respectively. To encourage the robots to reach the goal as quickly as possible, there is a negative reward (-1) when any agent is not in the goal square.

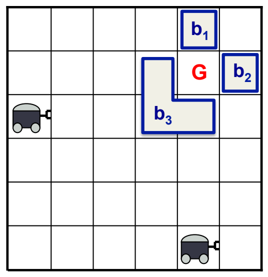


Figure 4: A 6x6 two-agent NAMO problem.

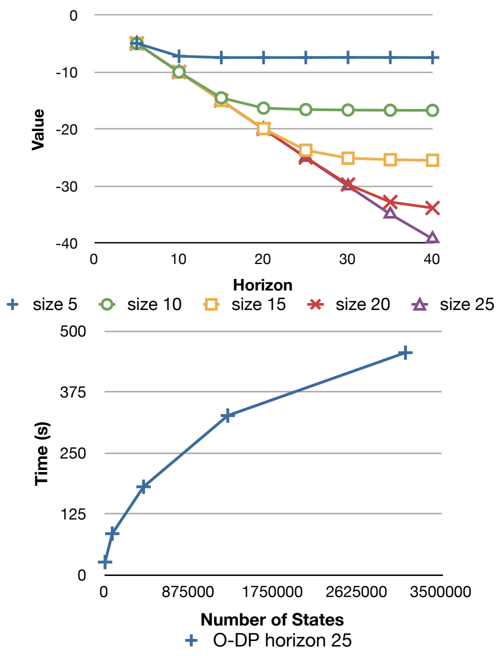


Figure 5: Value and time results for O-DP in the two-agent NAMO problem for various size grids (where size is the length of a single side)

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

Results for option-based dynamic programming are given in Figure 5. Here, O-DP performs very well on a range of different problem sizes and horizons. Because negative

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

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

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

A comparison with other Dec-POMDP algorithms (including O-MDBP) is shown in Table 2. For TBDP and GMAA-ICE* (a leading optimal Dec-POMDP algorithm) [23], the grid size was increased while at least horizon 4 could be solved and then the horizon was increased until it reached 100. Results for these algorithms were provided by personal communication with the authors and run on other machines, but the trends remain the same. For O-MBDP, 20 trees were used because smaller numbers resulted in poor performance, but parameters were not exhaustively evaluated. The results show that TBDP is able to solve the 4×4 problem, but runs out of memory when trying to solve any 5×5 problems. GMAA*-ICE can solve larger problem sizes, but runs out of memory for larger horizons. GMAA*-ICE scales better with the increased state space because it is able to exploit the factorization of the problem, but is limited to very small horizons because it is solving the underlying Dec-POMDP optimally. The inability for current approaches to solve these problems is not surprising given their size. In contrast, O-DP is able to solve the 25×25 problem which has over 3 million states states while O-MBDP solves the 50×50 problem that has has 50 million states. O-MBDP is able to solve even larger problems, but we did not analyze its performance beyond the 50×50 problem.

7. DISCUSSION

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

We have so far assumed that the agent is given an appropriate set of options with which to plan. Recent research on skill discovery [19] has attempted to devise methods by which a single agent can instead acquire an appropriate set of options autonomously, through interaction with its (fully observable) environment. While some of these methods may be directly applicable, the characteristics of the partially observable, multiagent case also offer new opportunities for skill discovery. For example, we may wish to synthesize skills that collapse uncertainty across multiple agents, perform coordinated multiagent actions, communicate essential state information, or allow agents to synchronize and replan.

Finally, our results have shown that the use of macro-actions can significantly improve scalability—for example, by allowing us to use larger grids with the same set of agents and obstacles in the NAMO problem (see Figure 4). However, in such cases—where the state space grows but the number of agents and significant interactions does not—we should in principle be able to deal with *any* size grid with no increase in computation time, because the size of the grid is irrelevant to the coordination aspects of the problem. This does not occur in the work presented here because we plan in the original state space; methods for constructing a more abstract *task-level* representation [17] could provide further performance improvements.

8. CONCLUSION

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

This work opens the door to many research questions about representing and solving multiagent problems hierarchically. Promising avenues for future work include exploring different types of options, acquiring various types of options from experience, and developing more scalable solution methods that exploit domain and hierarchical structure. One example of such structure would be the use of a factored reward function [22] which allows more efficient policy generation and evaluation.

9. ACKNOWLEDGEMENTS

We would like to thank Matthijs Spaan and Feng Wu for providing results. Research supported in part by AFOSR MURI project #FA9550-09-1-0538.

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

10. REFERENCES

[1] C. Amato, G. Chowdhary, A. Geramifard, N. K. Ure, and M. J. Kochenderfer. Decentralized control of partially observable Markov decision processes. In *Proceedings of the Fifty-Second IEEE Conference on Decision and Control*, 2013.

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

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

[4] A. Barto and S. Mahadevan. Recent advances in hierarchical reinforcement learning. *Discrete Event Dynamic Systems*, 13:41–77, 2003.

[5] R. Becker, V. Lesser, and S. Zilberstein. Decentralized Markov Decision Processes with Event-Driven Interactions. In *Proceedings of the Third International Conference on Autonomous Agents and Multiagent Systems*, pages 302–309, 2004.

[6] R. Becker, S. Zilberstein, V. Lesser, and C. V. Goldman. Solving transition-independent decentralized Markov decision processes. *Journal of Artificial Intelligence Research*, 22:423–455, 2004.

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

[8] D. S. Bernstein, S. Zilberstein, and N. Immerman. The complexity of decentralized control of Markov decision processes. In *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence*, pages 32–37, 2000.

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

[10] J. S. Dibangoye, C. Amato, O. Buffet, and F. Charpillet. Optimally solving Dec-POMDPs as continuous-state MDPs. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2013.

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

[12] T. Dietterich. Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research*, 13:227–303, 2000.

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

[14] E. A. Hansen, D. S. Bernstein, and S. Zilberstein. Dynamic programming for partially observable stochastic games. In *Proceedings of the Nineteenth National Conference on Artificial Intelligence*, pages 709–715, 2004.

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

[16] G. Konidaris and A. G. Barto. Building portable options: Skill transfer in reinforcement learning. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence*, pages 895–900, 2007.

[17] G. Konidaris, L. P. Kaelbling, and T. Lozano-Perez. Symbol acquisition for task-level planning. In *the AAAI 2013 Workshop on Learning Rich Representations from Low-Level Sensors*, 2013.

[18] A. Kumar and S. Zilberstein. Point-based backup for decentralized POMDPs: complexity and new algorithms. In *Proceedings of the Ninth International Conference on Autonomous Agents and Multiagent Systems*, pages 1315–1322, 2010.

[19] A. McGovern and A. Barto. Automatic discovery of subgoals in reinforcement learning using diverse density. In *Proceedings of the Eighteenth International Conference on Machine Learning*, pages 361–368, 2001.

[20] F. Melo and M. Veloso. Decentralized MDPs with sparse interactions. *Artificial Intelligence*, 2011.

[21] J. V. Messias, M. T. Spaan, and P. U. Lima. GSMDPs for multi-robot sequential decision-making. In *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence*, 2013.

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

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

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

[25] S. Seuken and S. Zilberstein. Memory-bounded dynamic programming for DEC-POMDPs. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence*, pages 2009–2015, 2007.

[26] D. Silver and K. Ciosek. Compositional planning using optimal option models. In *Proceedings of the Twenty-Ninth International Conference on Machine Learning*, 2012.

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

[28] 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.

[29] P. Stone, R. Sutton, and G. Kuhlmann. Reinforcement learning for robocup soccer keepaway. *Adaptive Behavior*, 13(3):165–188, 2005.

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

[31] P. Velagapudi, P. R. Varakantham, , K. Sycara, and P. Scerri. Distributed model shaping for scaling to decentralized POMDPs with hundreds of agents. In *Proceedings of the Tenth International Conference on Autonomous Agents and Multiagent Systems*, 2011.

[32] F. Wu, S. Zilberstein, and X. Chen. Point-based policy generation for decentralized POMDPs. In *Proceedings of the Ninth International Conference on Autonomous Agents and Multiagent Systems*, pages 1307–1314, 2010.

[33] F. Wu, S. Zilberstein, and X. Chen. Point-based policy generation for decentralized POMDPs. In *Proceedings of the Ninth International Conference on Autonomous Agents and Multiagent Systems*, pages 1307–1314, 2010.

[34] F. Wu, S. Zilberstein, and X. Chen. Rollout sampling policy iteration for decentralized POMDPs. In *Proceedings of the Twenty-Sixth Conference on Uncertainty in Artificial Intelligence*, 2010.