Decentralized Acting and Planning Using Hierarchical Operational Models

Ruoxi Li, Sunandita Patra, Dana Nau
Dept. of Computer Science, and Institute for Systems Research
University of Maryland, College Park, MD 20742, USA
rli12314@cs.umd.edu, patras@umd.edu, nau@cs.umd.edu

Abstract

We describe Dec-RAE-UPOM, a system for decentralized multi-agent acting and planning in environments that are partially observable, nondeterministic, and dynamically changing. Dec-RAE-UPOM includes an acting component, Dec-RAE, and a planning component, UPOM. The acting component is similar to the RAE acting engine (Ghallab, Nau, and Traverso 2016), but incorporates changes that enable it to be used by autonomous agents working independently in a collaborative setting. Each agent runs a local copy of Dec-RAE and has its own set of hierarchical operational models that specify various ways to accomplish its designated tasks. Agents can communicate with each other to exchange information about their states, tasks, goals and plans in order to cooperatively succeed in missions. Communication is not always guaranteed or free, and agents need to reason about strategies to achieve optimal success and efficiency in missions under various constraints and with possibility of failures. To choose among alternative ways to accomplish tasks, our current implementation of Dec-RAE uses the UPOM planner (Patra et al. 2020). We also describe our work-in-progress on a new planner, D-UPOM, that incorporates some enhancements for planning in multi-agent environments.

Introduction

Recent work on the integration of acting and planning (Ghallab, Nau, and Traverso 2014; 2016) has advocated a hierarchical organization of an actor's deliberation functions, with online planning continually throughout the acting process. This view has led to the development of the RAE acting algorithm (Ghallab, Nau, and Traverso 2016), and to the development of three successively better planning-and-acting systems that use RAE as their acting component: APE (Patra et al. 2019b), RAE/RAEplan (Patra et al. 2019a), and RAE/UPOM (Patra et al. 2020).

To predict an action's outcome, most AI planning systems use an abstract *descriptive model* (e.g., a PDDL action definition). To perform an action, an acting system uses an *operational model* of the action – a piece of code telling the actor what to do. In systems that do both planning and acting, a key problem is the need for consistency between what the descriptive model predicts, and what the operational model says to do. The APE, RAE/RAEplan, and RAE/UPOM systems circumvent this problem by having each system's planner use the same operational model that RAE (the actor) uses. To predict the action's outcome, the planner runs the operational model in a simulated environment. This enables APE, RAE/RAEplan, and RAE/UPOM to operate effectively in environments that are partially observable, nondeterministic, and dynamically changing due to exogenous events.

A key limitation of the above work is that it is *single-agent* planning and acting. Although several of the above papers use test domains involving multiple robots, in each case the planning and acting is done by a single centralized system. In the current paper we describe our work on extending the approach to accommodate multiple agents that do their planning and acting in a decentralized fashion. Our contributions are as follows:

- We introduce Dec-RAE-UPOM, a decentralized multi-agent acting-and-planning engine that uses operational models like the ones used in RAE. It consists of two components, Dec-RAE and UPOM:
  - Dec-RAE, the acting component, is a generalization of RAE. Multiple agents can run Dec-RAE concurrently in a decentralized fashion, and can use it to perform actions and to communicate with each other.
  - UPOM, the planning component, is the same planner used in the RAE/UPOM system. UPOM uses a Monte-Carlo rollout technique based on the well-known UCT algorithm (Kocsis and Szepesvári 2006).
- We present experimental evaluations of Dec-RAE-UPOM in robot foraging problems. The results show that additional Monte-Carlo rollouts in the planning component improve the performance of the acting component in both single-agent and multi-agent settings. We also observe that communication enables coordination between agents thereby improves the performance of multi-agent foraging to a large extent.
- We describe our ongoing work on a decentralized version of UPOM, D-UPOM. UPOM in Dec-RAE does not support inter-agent plan coordination. But in D-UPOM, if an agent i needs to outsource a task τ to some other agent, i can ask

the other agents to predict how well they can accomplish τ, and outsource τ to the agent that can do the best job.

The rest of this paper is organized into sections on each of the following topics: (1) related work, (2) our formalism, (3) Dec-RAE-UPOM, (4) our experimental results, (5) our ongoing work on D-UPOM, and (6) limitations and opportunities for future work.

Related work

Hierarchically organized deliberation techniques such as HTN planning (Nau et al. 1999) and refinement acting (Ghallab, Nau, and Traverso 2016) are well-established in the AI planning literature. They have substantial advantages in working out interactions in more abstract plan spaces, thereby pruning away large portions of the more detailed search spaces (Durfee 2001).

Some AI planning researchers have been advocating a change in focus to a combination of planning and acting that incorporates 1) hierarchically organized deliberation, in which each action in a plan may be a task that may need further refinement and planning; and 2) continual planning and deliberation, in which the actor monitors, refines, extends, updates, changes and repairs its plans throughout the acting process, using both descriptive and operational models of actions (Ghallab, Nau, and Traverso 2016).

To predict an action's outcome, most AI planning systems use an abstract *descriptive model* (e.g., a precondition-and-effects model). To perform an action, an acting system must use a more-detailed *operational model* that tells what to do. In the RAE acting system (Ghallab, Nau, and Traverso 2016), these operational models are collections of *refinement methods*. A refinement method for a task t specifies *how to* perform t, i.e., it gives a procedure for accomplishing t by performing subtasks, commands and state variable assignments. The procedure may include any of the usual programming constructs: if-then-else, loops, etc. It recursively refines abstract activities into less abstract activities, ultimately producing commands to the execution platform. When several method instances are available for a task, RAE is capable of trying alternative methods in nondeterministic choices or making the choice using some heuristics.

Monte Carlo tree search (MCTS) is a promising approach for online planning because it efficiently searches over long planning horizons and is anytime (Browne et al. 2012). In treating the choice of child node to expand in the MCT as a multiarmed bandit problem, the UCT algorithm balances the tradeoff between exploration and exploitation, in order to find a near-optimal plan. The UCT-like UPOM planner (Patra et al. 2020) performs MCTS over a space of refinement trees generated using RAE's refinement methods, in order to find a near-optimal method for RAE to use for refining one of its tasks. RAE and UPOM thus constitute an integrated refinement acting-and-planning system in which both acting and planning use RAE's operational models.

Distributed problem solving is applied to a subfield of distributed artificial intelligence or multiagent systems that emphasizes on getting agents to work together well to solve problems that require collective effort (Durfee 2001). In Multi-Agent Planning, the planning process is either centralised (e.g., a master agent produces distributed plans for multiple slave agents to act upon), or decentralized (i.e., the planning process involves multiple agents) (Cardoso and Bordini 2017). Hierarchical deliberation has substantial advantages in working out interactions in more abstract plan spaces, thereby pruning away large portions of the more detailed search spaces (Durfee 2001).

The main steps of distributed hierarchical planning have been summarized in various work (Weerdt and Clement 2009; Cardoso and Bordini 2017; Durfee 2001): 1) global task (goal) refinement, decomposition of the global task into subtasks; 2) task allocation, use of task-sharing protocols to allocate tasks (goals); 3) coordination during planning, cooperative planning mechanisms that generates a globally optimal solution for the problem; 4) coordination during plan execution, mechanisms that carry out the solution, prevent conflicts, repair the plan and replan.

Dix et al. (2003) describe a formalism to integrate the HTN planning system SHOP (Nau et al. 1999) with the IMPACT multi-agent environment. While the formalism is a multi-agent system, the planning is carried out in a centralized fashion by a single agent, A-SHOP. HTN planning has been used for coordination in robot soccer (Obst and Boedecker 2006), where low-level primitive tasks are performed differently by agents depending on their roles in the team task, high-level tasks are expanded to subtasks in a centralized planner. Clement, Durfee, and Barrett (2007) developed an algorithm for hierarchical refinement planning and centralized plan co-ordination for actions with temporal extent. Summary information is derived from each high-level task in a plan hierarchy about all of its potential needs and effects under all of its potential refinements, then exchanged among agents. Coordination at abstract levels allows each agent to retain some local flexibility to refine its high-level task without jeopardizing coordination or triggering new rounds of renegotiation.

Among studies in decentralized hierarchical planning, market-based task-allocation has been applied to complex tasks that can be hierarchically decomposed (Zlot and Stentz 2006). A multi-agent model for plan synthesis that produces a global shared plan is based on unified HTN and POP approaches (Pellier and Fiorino 2007), where agents exchange proposals and counter-proposal to refine flaws. Planner9 (Magnenat, Voelkle, and Mondada 2009) is a HTN planner that considers different robots as computer clusters and distributes the planning of any task to any robot, thus takes advantage of all the available computational power using simple synchronization.

Our approach shares some similarities with reinforcement learning (RL) (Kaelbling, Littman, and Moore 1996; Sutton and Barto 1998; Geffner and Bonet 2013; Leonetti, Iocchi, and Stone 2016; Garnelo, Arulkumaran, and Shanahan 2016), since MCTS is also a typical technique in RL to increase sample efficiency. In Model-based RL, the model (e.g., system dynamics) is learned from real experience and gives rise to simulated experience. While in our work, the model which we use to construct the simulator, is given.

Decentralized partially-observable Markov decision pro-

cess (Dec-POMDP) is a framework for a team of collaborative agents to maximize a global reward based on local information. Each agent's individual policy maps from its action and observation histories to actions (Oliehoek 2012). Unfortunately, optimally solving Dec-POMDPs is NEXP-complete (Bernstein, Zilberstein, and Immerman 2013). In single-agent (i.e., MDP) domains, the options framework (Sutton, Precup, and Singh 1999) uses higher-level, temporally extended macro-actions (or options) to represent and solve problems, resulting in significant improvement in the performance. Amato et al. extend the framework to the multi-agent case by introducing a Dec-POMDP formulation with macro-actions modeled as options. It is an offline planner that generates a policy to select the best option on each state, while our approach is an planning and acting engine that selects the best refinement method for each task online.

To our knowledge, there is no prior work on decentralized refinement acting and (hierarchical) planning that uses operational models.

Formalism

Here we formalize a decentralized multi-agent refinement planning and acting domain with operational models as a tuple $\Sigma = (\mathcal{S}, \mathcal{I}, \mathcal{T}, \{\Omega_i\}, \{\mathcal{O}_i\}, \{\mathcal{B}_i\}, \{\mathcal{C}_i\}, \{\mathcal{P}_i\}, \{\mathcal{D}_i\}, \{\mathcal{R}_i\}, \{\mathcal{M}_i\})$, where

- $\mathcal{S}$ is a set of world states the agents may be in, where we represent each state s using a state variable formulation similar to the one in Ghallab, Nau, and Traverso (2016).
- $\mathcal{I}$ is a finite set of agents,
- $\mathcal{T}$ is a finite set of tasks and events the agents may have to deal with, where each task $\tau \in \mathcal{T}$ has 0 or more than 0 relevant methods in $\{\mathcal{M}_i\}$,
- Ω_i is a finite set of observations for each agent i,
- $\mathcal{O}_i$ is a set of observation probability functions for each agent, i, $\Omega \times \mathcal{C}_i \times \mathcal{S} \rightarrow [0, 1]$,
- $\mathcal{B}_i$ is a set of belief states that agent $i \in \mathcal{I}$ may have,
- $\mathcal{C}_i$ is a finite set of primitive actions (commands) that can be carried out by the actuators and sensors on agent i's execution platform,
- $\mathcal{P}_i$ is a set of state transition probabilities, $\mathcal{S} \times \mathcal{C}_i \times \mathcal{S} \rightarrow [0, 1]$,
- $\mathcal{D}_i$ is a set of time durations, $\mathcal{S} \times \mathcal{C}_i \times \mathcal{S} \rightarrow \mathbb{R}$,
- $\mathcal{R}_i$ is a finite set of reward (or cost) functions that are associated with entering some states, i.e., $\mathcal{S} \rightarrow \mathbb{R}$,
- $\mathcal{M}_i$ is the set of refinement methods, each of which specifies how agent i would perform a task or respond to a event $\tau \in \mathcal{T}$.

A refinement method is composed of 4 elements, where 1) $head$ specifies the name and parameters of the method, where the number of parameters could be arbitrary greater than that in the task which it is related to, 3) $agent$ is the agent subject that owns (or is responsible for) the method, 3) $tasks$ indicates the task that the method is capable of refining, 3) $body$ gives a procedure to accomplish a the task by performing subtasks, commands and state variable assignments, where the procedure may include any programming constructs (e.g., if-then-else, loops, etc.). We get a refinement method instance by assigning values to the free parameters of a method.

A refinement tree (Patra et al. 2020) is composed of 3 types of nodes: 1) a *disjunction* node is a task followed by its applicable method instances; 2) a *sequence* node is a method instance m followed by all the steps; and 3) a *sampling* node for an action a has the possible nondeterministic outcomes of a as its children.

Refinement planning under this formalism is essentially a tree search procedure over the space of refinement trees in order to find a near-optimal method to use for refining a task under the context at hand.

Let us illustrate this formalism by an example, where several robots (drones and roombas) forage for some target objects (e.g., dirt) in a initially known terrain. This domain includes but is not limited to

- a set of states S that gives the positions of agents and dirt
- a set of agents $I = \{d_1, r_1, r_2\}$, where d_1 is a drone, r_1 and r_2 are roombas,
- a set of refinement methods $\mathcal{M}_{r_1} \supseteq \{$m1-cleanSet$(s)$, m2-cleanSet$(s)$, m1-clean$(s)$, m1-broadcastGoal$(g)\}$ for agent r_1,
- a set of refinement methods $\mathcal{M}_{r_2} \supseteq \{$m3-cleanSet$(s, l)$, m2-clean$(s)\}$ for agent r_2,
- a set of refinement methods $\mathcal{M}_{d_1} \supseteq \{$m1-search$(a)$, m1-planTrajectory$(a)$, m1-flyTo$(l)\}$ for agent d_1,
- a set of commands $\mathcal{C}_{d_1} \supseteq \{$observe$(l)\}$ for agent d_1,
- a set of tasks $\mathcal{T} \supseteq \{$search(a), cleanSet(s), clean(l), planTrajectory(a), flyTo$(l)\}$.

m1-search(a)
  agent: d_1
  task: search(a)
  body: trajectory $\leftarrow$ do task planTrajectory(a)
    for l in trajectory:
      do task flyTo(l)
      execute command observe(l)
      if l has dirt:
        outsource task clean(l) to agent $i \in \{r_1, r_2\}$

m1-search(a) is a method for the drone d_1 to search area a along a trajectory, perform the command observe(l) to check if an intermediate location l is dirty before outsourcing a task to a roomba i to clean up the location. Suppose roomba r_1 has one method for the task clean(l), roomba r_1 has two method for the task clean(l), there would be 3 applicable method instances in total to the task that is being outsourced, as either r_1 or r_2 could be assigned to i.

```
m1-cleanSet(s)
  agent: r_1
   task: cleanSet(s)
  body: if s is ∅:
            return
        l ← closest l ∈ s
        do task broadcastGoal(l)
        do task clean(l)
        s ← s \ l
        do task cleanSet(s)
```

```
m2-cleanSet(s)
  agent: r_1
   task: cleanSet(s)
  body: if s is ∅:
            return
        l ← random l ∈ s
        do task clean(l)
        s ← s \ l
        do task cleanSet(s)
```

Task cleanSet(s) requires a set of locations s to be cleaned. Agent r_1 has two methods that can refine the task, m1-cleanSet(s) and m2-cleanSet(s). m1-cleanSet(s) is a $greedy$ method for roomba r_1 to clean the location set s, where roomba r_1 cleans the closest location in s recursively. m2-cleanSet(s) makes random choice of the first location to clean and cleans the rest of the set recursively.

```
m3-cleanSet(s, l):
  agent: r_2
   task: cleanSet(s)
  body: if s is ∅:
            return
        do task clean(l)
        s ← s \ l
        do task cleanSet(s)
```

Agent r_2 also has a $simple$ method m3-cleanSet(s, l) that refine task cleanSet(s), where l indicates the first location to clean. l's value is automatically assigned with some predefined rules (e.g, $l \in s$). In that case, the number of applicable method instances to task cleanSet(s) for agent r_2 is $|s|$.

Dec-RAE-UPOM

RAE (Ghallab, Nau, and Traverso 2016) is a refinement acting engine that uses a collection of hierarchical refinement methods with operational model to generate and traverse a refinement tree. UPOM (Patra et al. 2020) is a UCT-like Monte-Carlo tree search simulation procedure over the space of refinement trees in order to find a near-optimal method in RAE to accomplish the task under the context at hand. A decentralized version of RAE using UPOM, Dec-RAE-UPOM is a decentralized multi-agent planning and acting engine with UCT-like planning procedure using operational models that enables heterogeneous robots to cooperatively operate in a partially-observable, non-deterministic and dynamic environment with exogenous events and concurrent tasks. As is shown in Figure 1, each agent has its own RAE, UPOM, domain knowledge, execution platform, and internal state information.

Belief, desire and intention (Rao and Georgeff 2000) represents the information, motivational, and deliberative states of the agent. Respectively, we specify 4 types of communication messages that one agent can send to another: 1) $state$, the state information obtained from the agent's sensors, i.e., its belief state set about the world which is stored in a predefined data structure; 2) $goal$, a desire that has been adopted for active pursuit by the agent; 3) $task$, a desire that the agent needs other agents to accomplish (e.g. a subtask τ in agent A's method that needs to be accomplished by agent B); 4) $plan$, the refinement tree or the estimated reward λ of its root node that is associated with the process of refining a task ($plan$ communication is not yet supported and UPOM does not require communication).

Agent are built with both actuators and sensors to send and receive communication signals. Commands are given to agents to sense the communication network, send messages, or read messages. Received messages are buffered in memory waiting for the agent to read. We acknowledge the fact that communication is neither free, nor guaranteed to succeed. Therefore, each communication command is associated with a cost and a probability of success just like any other commands.

Dec-RAE agents, without using any planner, are able to reactively coordinate their actions through $state$, $goal$ and $task$ communication. With UPOM, each agent i can plan for the selection of method instances from $\mathcal{M}_i$ to resolve a tasks locally.

Experimental Evaluation

Multi-agent Foraging is one of the canonical testbeds for cooperative multi-agent systems, in which a collection of robots has to search and transport objects to specific locations (Zedadra et al. 2017). We developed a Vacuum World Simulator (Figure 2) where multiple $roomba$ agents cooperatively clean up a finite amount of $dirt$ objects scattered randomly within an $n \times n$ grid. Each $dirt$ object is associated with a value. Each $roomba$ agent has limited amount of time budget to carry out durative actions including moving forward, turning left, turning right, picking up a dirt right beneath it, and communicating with other agents. As a preliminary experiment, actions and observations are deterministic, communication between agents is free and guaranteed. The objective is for the $roomba$ team to maximize the score (i.e., the total value from collecting $dirt$ objects) within the limit of its time budget.

We tested and compared the performance of $roomba$ agents with several different types of decision strategies (Figure 3 & 4) denoted by following remarks: 1) $Greedy$, which means the agent has a $greedy$ method, i.e., m1-cleanSet(s), that repeatedly pursues the nearest target; 2) $Simple$, which means the agent has a $simple$ method, i.e., m1-cleanSet(s, l), to pursue a list of targets; 3) $UPOM$, which indicates that the agent has the same methods as a $simple$ agent, but uses UPOM to plan for the choice of the method instances; 4) n, the number of UCT rollouts that is configured in a $UPOM$ agent; 4) $Comm$, which indicates that goal communication is enabled by doing task broadcastGoal(g), where the agent broadcasts information

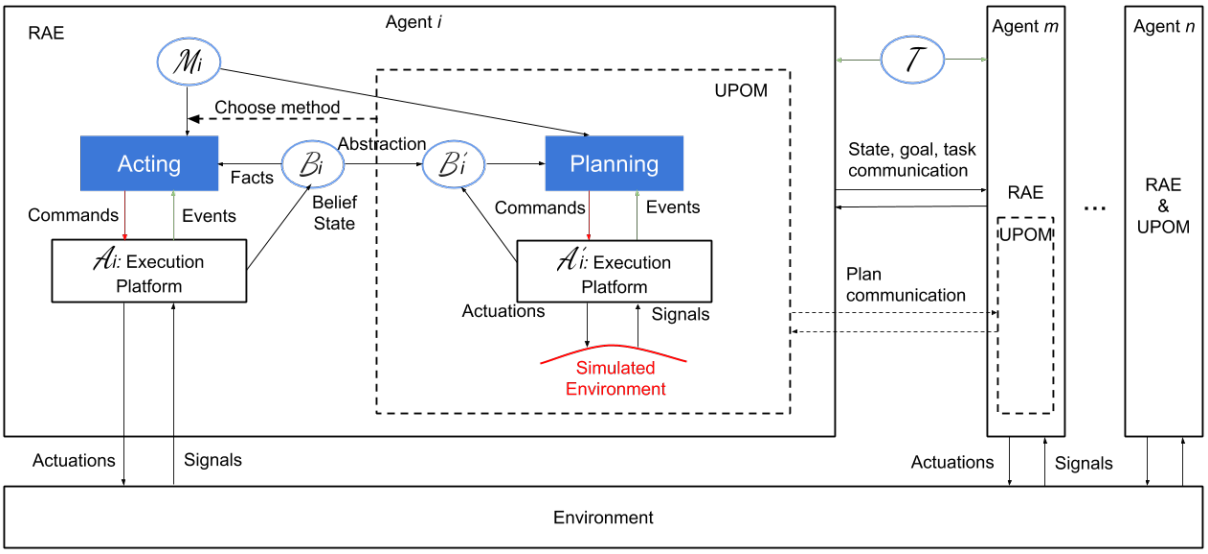

Figure 1: The system architecture of Dec-RAE-UPOM.

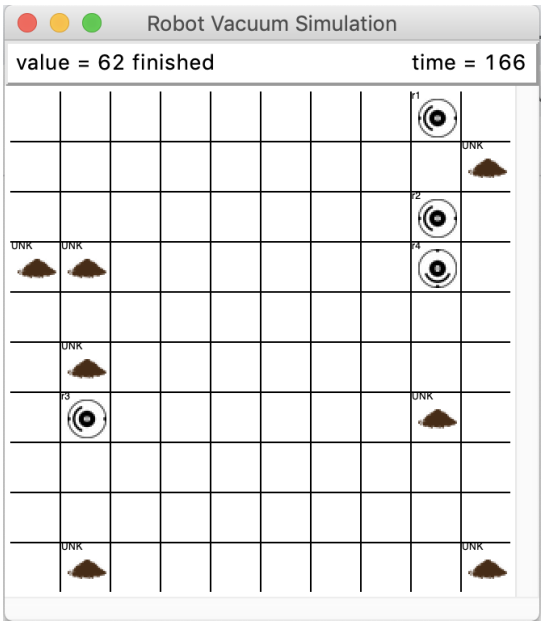

Figure 2: The Vacuum World Simulator.

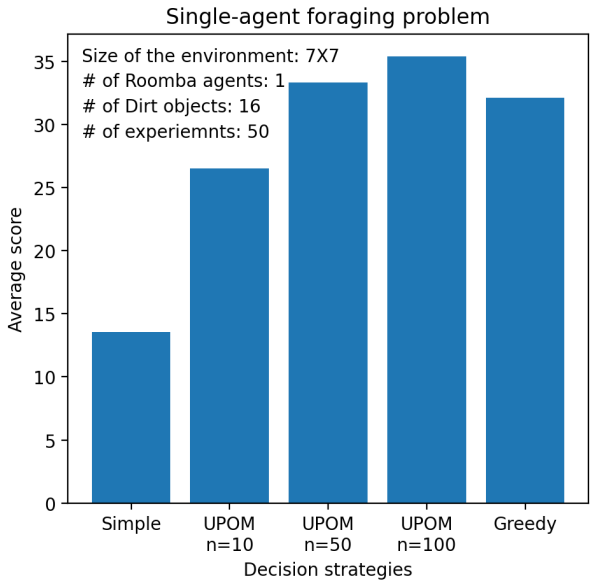

Figure 3: For each experiment, we deploy 1 $roomba$ agent along with 16 Dirt objects to a 7 $\times$ 7 grid. Each $roomba$ agent type's average score is obtained from solving 50 randomly generated problems.

about the target that it is actively pursuing. Within each experiment, all agents (if there are more than one agent) use the same strategy.

The single-agent experiments (Figure 3) show that a $UPOM$ agent performs much better than a $simple$ agent, since a reactive $simple$ agent would clean the set of locations in an arbitrary sequence, while the $UPOM$ agent tries to plan for the optimal sequence. The performance of a $UPOM$ agent further improves as the number of UCT rollouts increases, which surpasses a $greedy$ agent's performance with 50 rollouts. In multi-agent experiments (Figure 4), with goal communication enabled, agents would be aware of each other's goals, thus, are able to adjust their own goals accordingly to avoid duplication of efforts. We observe a 48.5% improvement in the $greedy$ agent team's performance with goal communication, compared to the performance without any communication. The $UPOM$ agent team with $comm$ performs slightly better than a $greedy$ agent team with $comm$, which is consistent to the result from the single-agent experiments.

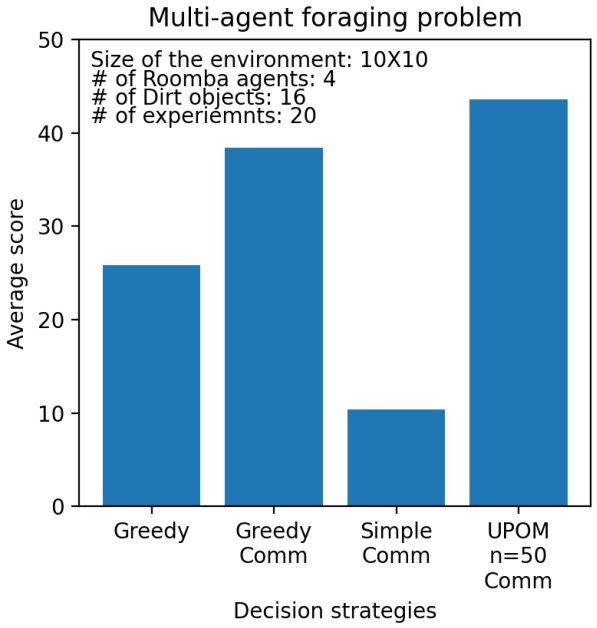

Figure 4: For each experiment, we deploy 4 $roomba$ agents along with 16 Dirt objects to a 10×10 grid. Each $roomba$ team's average score is obtained from solving 20 randomly generated problems.

D-UPOM

Each of the agents with Dec-RAE-UPOM has independent domain knowledge, execution platform, and state information. if an agent i needs to outsource a task τ to some other agent, without $plan$ communication, agent i has no idea how well they can accomplish τ. Therefore, agent i is only able to outsource the task to an agent that is selected randomly or based on agent i's subjective heuristics. In order to generate more optimal plans, agents need to coordinate with each other in the planning process and communicate their local plans with each other. Here we describe our work-in-progress on a decentralized version of UPOM, D-UPOM (Algorithm 1) that addresses the above issue. In D-UPOM, if an agent i needs to outsource a task τ to some other agent, i can ask other agents to predict how well they can accomplish τ, and outsource τ to the agent that can do the best job.

Firstly, we let $\gamma(s, c)$ be the set of states that may be reached after performing command c in state s. Applicable(b, τ) is the set of method instances applicable to τ in belief state b. The current context for an incoming external task τ is represented via a *refinement stack* σ which keeps track of how much further RAE has progressed in refining τ. next(σ, s) is the refinement stack resulting by performing $m[i]$ in state s, where $(\tau, m, i) = top(\sigma)$. Abstraction$(b)$ is the abstracted belief state that is used in D-UPOM's simulated environment. Responsible(m) is the agent subject that is responsible for executing method m. R(s) is the reward obtained from entering state s.

When an agent has to perform the task τ in a belief state b and a stack σ, it calls Select-Method $(b, \tau, \sigma, d_{max}, n_{ro})$

Select-Method$(b, \tau, \sigma, d_{max}, n_{ro})$:
1 $\tilde{m} \leftarrow \text{argmax}_{m \in \text{Applicable}(b,\tau)} h(\tau, m, b)$
$d \leftarrow 0$
2 **repeat**
  $d \leftarrow d + 1$
3 $b' \leftarrow$ Abstraction(b)
  for n_{ro} *times* **do**
    D-UPOM $(b', \text{push}((\tau, nil, nil), \sigma), d)$
  $\tilde{m} \leftarrow \text{argmax}_{m \in M} Q_{s,\sigma}(m)$
until $d = d_{max}$ *or* searching time is over
return $\tilde{m}$

D-UPOM(s, σ, d):
if $\sigma = \langle\rangle$ **then** return 0
$(\tau, m, i) \leftarrow \text{top}(\sigma)$
4 **if** $d = 0$ **then** return $h(\tau, m, s)$
if $m = nil$ *or* $m[i]$ *is a task* τ' **then**
  if $m = nil$ **then** $\tau' \leftarrow \tau$ # for the first task
  if $N_{s,\sigma}(\tau')$ *is not initialized yet* **then**
5 $M' \leftarrow \text{Applicable}(s, \tau')$
    if $M' = 0$ **then** return 0
    $N_{s,\sigma}(\tau') \leftarrow 0$
    for $m' \in M'$ **do**
      $N_{s,\sigma}(m') \leftarrow 0$
      $Q_{s,\sigma}(m') \leftarrow 0$
  $Untried_m \leftarrow \{m' \in M' | N_{s,\sigma}(m') = 0\}$
  if $Untried_m \neq \emptyset$ **then**
    $m_c \leftarrow$ random selection from $Untried_m$
  else $m_c \leftarrow \text{argmax}_{m \in M'} \phi(m, \tau')$
  $\sigma' \leftarrow \text{push}((\tau', m_c, 1), \text{next}(\sigma, s))$
  $a \leftarrow$ Responsible(m_c)
  if *a is self* **then** $\lambda \leftarrow$ D-UPOM$(s, \sigma', d-1)$
6 **else** $\lambda \leftarrow$ Request-Plan$(a, s, \sigma', d-1)$
7 $Q_{s,\sigma}(m_c) \leftarrow \frac{N_{s,\sigma}(m_c) \times Q_{s,\sigma}(m_c) + \lambda}{1 + N_{s,\sigma}(m_c)}$
  $N_{s,\sigma}(m_c) \leftarrow N_{s,\sigma}(m_c) + 1$
  return λ
if $m[i]$ *is an assignment* **then**
  $s' \leftarrow$ state s updated according to $m[i]$
  return D-UPOM$(s', \text{next}(\sigma, s'), d)$
if $m[i]$ *is a command* c **then**
  $s' \leftarrow \text{Sample}(s, c)$
  return $R(s') +$ D-UPOM$(s', \text{next}(\sigma, s'), d-1)$

Algorithm 1: D-UPOM and Select-Method.

(Algorithm 1) with two control parameters: n_{ro}, the number of rollouts, and d_{max}, the maximum rollout length (total number of sub-tasks and actions in a rollout). Select-Method performs an anytime progressive deepening loop calling D-UPOM n_{ro} times in a simulated environment based on the belief state b, until the rollout length reaches d_{max} or the search is interrupted. The selected method instance $\tilde{m}$ is initialized according to a heuristic h (line 1).

Just like UPOM, D-UPOM performs one UCT rollout recursively down the refinement tree until depth d is reached for stack σ. During the recursion, if another agent a is re-

sponsible for executing method m_c, one needs to request the agent to plan for m_c by calling Request-Plan$(a, s, \sigma', d-1)$ (line 6). Agent a is supposed to receive the request, use its own D-UPOM to perform one UCT rollout further down the refinement tree recursively, and send back the resulting estimated total reward λ from executing m_c. Request-Plan will return 0 if it times out.

D-UPOM naturally supports market-based task allocation: when the task τ' is potentially outsourced to different agents who are capable of accomplishing it, each agent plans for τ' using its methods and return the corresponding estimated rewards, the agent who has a method that obtains the highest reward will be chosen to accomplish τ'.

As a work-in-progress, D-UPOM has not yet been programmed or tested. We will further develop its theory, program and experiments in our future work.

Discussion

In this paper we have described Dec-RAE-UPOM, a system for decentralized multi-agent refinement planning and acting that uses operational models similar to the ones used in the RAE and RAE/UPOM systems. In our evaluations of Dec-RAE-UPOM's performance in robot foraging problems using the Vacuum World Simulator, the results show that the system's performance is improved by performing additional Monte-Carlo rollouts in UPOM, and that communication between agents also improves the performance. We have also described our work-in-progress on the D-UPOM planner, a decentralized version of UPOM that is expected to generate more optimal plans by exploiting $plan$ information obtained from other agents.

Multi-agent systems are usually affected by the combinatorial explosion of the state space due to increased amount of agents. A decentralized planner avoids this problem by making each agent plan locally and communicate their plans with each other in order to cooperate. However, functionalities such as action synchronization and conflict resolution between decentralized agents are not currently supported by our system.

The $plan$ communication in D-UPOM only delivers minimum information, i.e., the total reward obtained from sampling the method. In a similar planner, D-UCB in decentralized Monte Carlo Tree search (Dec-MCTS) (Best et al. 2018), agents exchange more explicit plan information back and forth so the planner in each agent is able to sample all the other agents' actions according to their plans. D-UPOM has the same potential in exchanging more plan information such as a explicit refinement tree and its associated Q values, which might help with plan merging, conflict resolution and action synchronization. Our formalism represents action durations, and we intend to reason about plan merging, conflict resolution and action synchronization as a temporal planning and scheduling problem in our future work.

In our experiments, communication commands have 0 cost (reward) and are guaranteed to succeed. We have not done enough investigations in cases where communication is not always guaranteed or free, and agents might need to proactively search for communication signals (e.g., by going to a high ground where there is higher chance to re-establish communication with others). A broader question that automotive agents needs to decide is who, when, how and what to communicate (Balch and Arkin 1970; Wei, Hindriks, and Jonker 2014). As an example, Dec-MCTS reasons about the value of communication messages to decide when and to whom each robot should communicate in order to minimize the communication cost. We hope our work could be developed to be more resilient and intelligent in terms of communication.

Since UPOM has been integrated with neural networks that learn to choose methods and approximate heuristics, we are also interested in extending learning to our future work.

Acknowledgements

This work has been supported in part by DARPA task order HR001119F0057, Lockheed Martin research agreement MRA17001006, NRL grant N00173191G001, and ONR grant N000142012257. The information in this paper does not necessarily reflect the position or policy of the funders, and no official endorsement should be inferred.

References

Amato, C.; Konidaris, G.; Kaelbling, L.; and How, J. 2019. Modeling and planning with macro-actions in decentralized pomdps. *Journal of Artificial Intelligence Research* 64:817–859.

Balch, T., and Arkin, R. 1970. Communication in reactive multiagent robotic systems. *Autonomous Robots* 1.

Bernstein, D. S.; Zilberstein, S.; and Immerman, N. 2013. The complexity of decentralized control of markov decision processes.

Best, G.; Forrai, M.; Mettu, R. R.; and Fitch, R. 2018. Planning-aware communication for decentralised multi-robot coordination. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 1050–1057.

Browne, C. B.; Powley, E.; Whitehouse, D.; Lucas, S. M.; Cowling, P. I.; Rohlfshagen, P.; Tavener, S.; Perez, D.; Samothrakis, S.; and Colton, S. 2012. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games* 4(1):1–43.

Cardoso, R., and Bordini, R. 2017. A multi-agent extension of a hierarchical task network planning formalism. *ADCAIJ* 6:5.

Clement, B.; Durfee, E.; and Barrett, A. 2007. Abstract reasoning for planning and coordination. *J. Artificial Intelligence Research (JAIR)* 28:453–515.

Dix, J.; Muñoz-Avila, H.; Nau, D. S.; and Zhang, L. 2003. Impacting shop: Putting an ai planner into a multi-agent environment. *Annals of Mathematics and Artificial Intelligence* 37(4):381–407.

Durfee, E. H. 2001. Distributed problem solving and planning. In Luck, M.; Mařík, V.; Štěpánková, O.; and Trappl, R., eds., *Multi-Agent Systems and Applications: European Agent Systems Summer School, EASSS 2001*, 118–149. Springer.

Garnelo, M.; Arulkumaran, K.; and Shanahan, M. 2016. Towards deep symbolic reinforcement learning. *CoRR* abs/1609.05518.

Geffner, H., and Bonet, B. 2013. *A Concise Introduction to Models and Methods for Automated Planning*. Morgan & Claypool.

Ghallab, M.; Nau, D.; and Traverso, P. 2014. The actors view of automated planning and acting: A position paper. *Artificial Intelligence* 208:1 – 17.

Ghallab, M.; Nau, D.; and Traverso, P. 2016. *Automated Planning and Acting*. Cambridge University Press.

Kaelbling, L. P.; Littman, M. L.; and Moore, A. W. 1996. Reinforcement learning: A survey. *JAIR* 4:237–285.

Kocsis, L., and Szepesvári, C. 2006. Bandit based monte-carlo planning. 282–293.

Leonetti, M.; Iocchi, L.; and Stone, P. 2016. A synthesis of automated planning and reinforcement learning for efficient, robust decision-making. 241:103–130.

Magnenat, S.; Voelkle, M.; and Mondada, F. 2009. Planner9, a HTN planner distributed on groups of miniature mobile robots.

Nau, D.; Cao, Y.; Lotem, A.; and Munoz-Avila, H. 1999. Shop: Simple hierarchical ordered planner. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'99, 968–973. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Obst, O., and Boedecker, J. 2006. Flexible coordination of multiagent team behavior using HTN planning. In Bredenfeld, A.; Jacoff, A.; Noda, I.; and Takahashi, Y., eds., *RoboCup 2005: Robot Soccer World Cup IX*, 521–528. Berlin, Heidelberg: Springer.

Oliehoek, F. A. 2012. *Decentralized POMDPs*. Berlin, Heidelberg: Springer Berlin Heidelberg. 471–503.

Patra, S.; Ghallab, M.; Nau, D.; and Traverso, P. 2019a. Acting and planning using operational models. *Proceedings of the AAAI Conference on Artificial Intelligence* 33:7691–7698.

Patra, S.; Ghallab, M.; Nau, D.; and Traverso, P. 2019b. APE: An acting and planning engine. *Advances in Cognitive Systems* 7.

Patra, S.; Mason, J.; Kumar, A.; Ghallab, M.; Traverso, P.; and Nau, D. 2020. Integrating acting, planning and learning in hierarchical operational models.

Pellier, D., and Fiorino, H. 2007. A unified framework based on HTN and POP approaches for multi-agent planning. In *2007 IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT'07)*, 285–288.

Rao, A., and Georgeff, M. 2000. Bdi agents: From theory to practice.

Sutton, R. S., and Barto, A. G. 1998. *Reinforcement learning - an introduction*. Adaptive computation and machine learning. MIT Press.

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

Weerdt, M., and Clement, B. 2009. Introduction to planning in multiagent systems. *Multiagent and Grid Systems* 5:345–355.

Wei, C.; Hindriks, K.; and Jonker, C. 2014. The role of communication in coordination protocols for cooperative robot teams. In *ICAART 2014 - Proceedings of the 6th International Conference on Agents and Artificial Intelligence*, volume 2.

Zedadra, O.; Jouandeau, N.; Seridi, H.; and Fortino, G. 2017. Multi-agent foraging: state-of-the-art and research challenges. *Complex Adaptive Systems Modeling* 5:1–24.

Zlot, R., and Stentz, A. 2006. Market-based multirobot coordination for complex tasks. *The International Journal of Robotics Research* 25(1):73–101.