Solving Multiagent Assignment Markov Decision Processes

Scott Proper
Oregon State University
Corvallis, OR 97331-3202, USA
proper@eecs.oregonstate.edu

Prasad Tadepalli
Oregon State University
Corvallis, OR 97331-3202, USA
tadepall@eecs.oregonstate.edu

ABSTRACT

We consider the setting of multiple collaborative agents trying to complete a set of tasks as assigned by a centralized controller. We propose a scalable method called "Assignment-based decomposition" which is based on decomposing the problem of action selection into an upper assignment level and a lower task execution level. The assignment problem is solved by search, while the task execution is solved through coordinated reinforcement learning. We show that this decomposition of the overall problem into two levels scales well and outperforms the state-of-the-art approaches including pure assignment-level search or pure coordinated reinforcement learning. We also show how this approach enables transfer learning from domains with few agents to domains with many agents.

Categories and Subject Descriptors

I.2.11 [**Distributed Artificial Intelligence**]: Multiagent systems

General Terms

Algorithms, Experimentation

Keywords

reinforcement learning, markov decision processes, assignment problem, coordination graphs

1. INTRODUCTION

Markov Decision Processes (MDPs) have proved to be useful and general models of optimal decision-making in stochastic environments. In this paper, we consider the setting of collaborative multiagent MDPs, which consist of multiple agents trying to optimize an objective. Multiagent formulations of MDPs are attractive because they naturally model distributed agents that make decisions based on limited information about other agents. In general, this class of problems is called a decentralized MDP (DEC-MDP), which is known to be NEXP-complete [3]. Even the best of DEC-MDP approaches have only been able to solve problems of about 100 states assuming some special structure [1]. On the other hand solving the multiagent system as a joint MDP over the joint state and action space is also computationally prohibitive and does not scale to a large number of agents.

Cite as: Solving Multiagent Assignment Markov Decision Processes, Scott Proper, Prasad Tadepalli, *Proc. of 8th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2009),* Decker, Sichman, Sierra and Castelfranchi (eds.), May, 10–15, 2009, Budapest, Hungary, pp. 681–688

We consider an assignment-based decomposition approach that is intermediate between the joint MDP approach and the independent agent approach. We assume a centralized controller that has relevant information about the states of all agents to assign tasks, and allocates tasks and resources to agents based on task-level value functions of agents. Once the tasks are assigned to agents, the lower-level actions of agents are decided by the task-level value functions until the tasks are reassigned by the central controller. We call such domains with multiple tasks and multiple agents Multiagent Assignment MDPs (MAMDPs). There exist many real world examples of MAMDP domains, such as fire and emergency response, vehicle routing and product delivery, and games such as real-time strategy games.

There have been attempts to decompose multi-agent MDPs in the literature. Meuleau et al. consider the notion of *weakly coupled MDPs*, where the overall MDP is divided into Markov task sets that do not share state space and can be solved completely independently after the tasks are allocated [10]. Dolgov et al. formulate a constrained MDP problem where the agents are independent except for constraints on the amounts of global resources they all share [2]. They solve this problem using a mixed integer linear program approach. Another approach is hierarchical multiagent reinforcement learning, where a value function that depends on the states and actions of all agents is learned at the top assignment level and a task-specific value function is learned at the lower level [9]. Our approach is similar, except that we use search at the higher level to allocate tasks. Our approach is much more space-efficient than a hierarchical approach, which requires a value function that takes space exponential in the number of agents at the root level.

As indicated by some of the above work, agents are not completely independent even after they are allocated tasks due to shared common resources. Coordination graphs, which are a form of conditional Markov random fields, have been used to model interactions between agents [5, 16]. Graph nodes represent agents and arcs between agent pairs represents potential interactions between their actions in influencing the reward or the outcome of the action. The long-term value of a joint action over all agents is approximated as a sum of a set of interaction terms, where each such term is based on the actions of a pair of agents and the global state. Bayesian network inference algorithms such as variable elimination and belief propagation have been adapted to finding

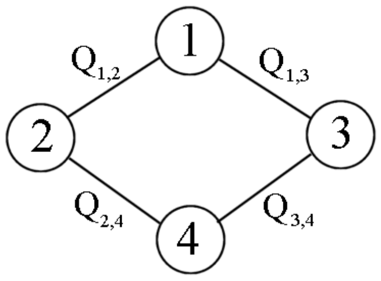

Figure 1: A possible coordination graph for a 4-agent domain. Q-values indicate an edge-based decomposition of the graph.

the best joint action that maximizes the total reward.

In many domains, coordination graphs change dynamically based on the state. The approaches based on coordination graphs are adapted to dynamic state-based coordination [6, 7]. For example, in the approach of [7], a set of rules dictate which agent should coordinate with whom, and the value of a state is based on the current coordination graph. We adopt their Max-Plus algorithm to solve coordination problems at the task execution level in this paper.

The main advantage of assignment-based decomposition is that problem complexity is divided between the assignment level and the task execution level. Complexity is reduced at the assignment level by ignoring the interactions between agents working on different tasks. Complexity is reduced at the task execution level by solving each task independently of others except for a limited set of interactions defined by coordination rules, e.g., to avoid collisions.

The combination of assignment-based decomposition and coordinated reinforcement learning has some advantages over using either one alone. First, consideration of local interactions such as collision avoidance can be delegated the task execution level, freeing the top level to focus on assignment decisions. Second, the coordination graph at the task execution level can take advantage of knowing the assignment when making coordination decisions. Third, since the lower level value functions are used in making the higher level assignment decisions, collision information is indirectly percolated to the assignment level.

We make three contributions in this paper. First, we describe an assignment-based decomposition approach to solve cooperative multiagent MDPs. Second, we show how to combine coordination graphs with a search-based assignment to achieve an integrated system that optimizes assignments and exhibits dynamic coordination. Third, we demonstrate empirical results in two domains that show that our approach scales better than existing approaches to multiagent coordination. We show this approach also allows transferring value functions directly from a domain with few agents to one with many.

2. MULTIAGENT ASSIGNMENT MARKOV DECISION PROCESSES

In this section, we first introduce Markov Decision Processes (MDPs), then generalize them to multiagent settings. An MDP is a tuple $\langle S, A, P, R\rangle$ where S is a finite set of states, A is a finite set of actions, P is a Markovian transition model that describes the probability $P(s'|s,a)$ of ending up in state s' when performing action a in state s, and $R : S \times A \rightarrow \mathbb{R}$ is a reward function that returns the reward $R(s,a)$ obtained after taking action a in state s. An agent's policy is defined as a mapping $\pi : S \rightarrow A$. The objective is to find an optimal policy π^* that maximizes the expected discounted future reward for each state s. We assume that the MDP has an infinite horizon, and that future rewards are discounted exponentially with a discount factor $\gamma \in [0, 1)$.

The optimal action-value function or Q-function gives the expected discounted future reward for any state s when executing action a and then following the optimal policy. The Q-function satisfies the following recurrence relation:

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s'} P(s'|s,a) \max_{a'} Q^*(s',a') \tag{1}$$

The optimal policy for a state s is the action $\arg\max_a Q^*(s,a)$ that maximizes the expected future discounted reward.

A *Multiagent Assignment MDP* extends the above framework to a set of n agents $G = \{g\}$ ($|G| = n$). Each agent g has its own set of local state S_g and actions A_g. We also define a set of tasks $T = \{t\}$, each associated with a set of state variables S_t that describe the task. The set of tasks (and corresponding state variables required to describe them) may vary between states. The joint action space is the Cartesian product of the actions of all n agents: $\mathbf{A} = A_1 \times A_2 \times ... \times A_n$. The joint state space is the Cartesian product of the states of all agents and all tasks. The reward is decomposed between all n agents, i.e., $R(s,a) = \sum_i^n R_i(\mathbf{s}, \mathbf{a})$, where $R_i(\mathbf{s}, \mathbf{a})$ is the agent-specific reward for state $\mathbf{s}$ and action $\mathbf{a}$.

$\beta : T \rightarrow G^k$ is an assignment of tasks to agents; here k indicates an upper bound on the number of agents that may be assigned to a particular task. $\beta(t)$ indicates the set of agents assigned to task t. We let $s_{\beta(t)}$ denote the joint states of all agents assigned to t, and $a_{\beta(t)}$ denote the joint actions of all agents assigned to task t. The total utility $Q(\mathbf{s}, \mathbf{a})$ depends on the states of all tasks and agents $\mathbf{s}$ and actions of all agents $\mathbf{a}$.

3. MULTIAGENT Q-LEARNING WITH COORDINATION GRAPHS

In this section, we examine the potential of using an existing technique to solve multiagent assignment MDPs. In a multiagent approach, the global Q-function $Q(\mathbf{s}, \mathbf{a})$ is approximated as a sum of agent-specific action-value functions: $Q(\mathbf{s}, \mathbf{a}) = \sum_i^n Q_i(\mathbf{s}_i, a_i)$ [8]. Further we approximate each agent-specific action-value as a function only of the agent's state s_i. A "selfish" agent-based version of multiagent Q-learning [13] updates each agent's Q-value independently using the update function:

$$\begin{aligned} &Q_i(\mathbf{s}_i, a_i) \leftarrow Q_i(\mathbf{s}_i, a_i) + \\ &\alpha\Big[R_i(\mathbf{s}, \mathbf{a}) + \gamma Q_i(\mathbf{s}'_i, a_i^*) - Q_i(\mathbf{s}_i, a_i)\Big] \end{aligned} \tag{2}$$

where $\alpha \in [0, 1]$ is the learning rate. The notation Q_i indicates only that the Q-value is agent-based. The parameters used to store the Q-function may either be unique to that agent or shared between all agents.

In most cases, each agent independently pursuing a policy to optimize its own Q_i will not optimize the total utility, since each agent's actions impact the state and the utility of others. Hence, collaborative agents need to coordinate. A

```
Initialize $Q(\mathbf{s}, \mathbf{a})$ optimistically
Initialize $\mathbf{s}$ to any starting state
**for** each step **do**
  Assign tasks $T$ to agents $M$ by finding $\arg\max_\beta \sum_t v_{\beta(t),t}$, where $v_{\mathbf{g},t} = \max_{a \in A_\mathbf{g}} Q(s_t, s_\mathbf{g}, a)$
  For each task $t$, choose actions $a_{\beta(t)}$ from $s_{\beta(t)}$ using $\epsilon$-greedy policy derived from $Q$
  Take action $\mathbf{a}$, observe rewards $\mathbf{r}$ and next state $\mathbf{s}'$
  For each task $t$, $Q(s_t, s_{\beta(t)}, a_{\beta(t)}) \leftarrow Q(s_t, s_{\beta(t)}, a_{\beta(t)}) + \alpha\Big[r_{\beta(t)} + \gamma \max_{a' \in A'_{\beta(t)}} Q(s'_t, s'_{\beta(t)}, a') - Q(s_t, s_{\beta(t)}, a_{\beta(t)})\Big]$
  $\mathbf{s} \leftarrow \mathbf{s}'$
**end for**
```

Figure 2: The assignment-based decompostion Q-learning algorithm.

coordination graph (see Figure 1) allows the agents to specify and model coordination requirements [4]. The presence of an edge in a coordination graph indicates that two agents should coordinate their action selection, for example, so as to avoid collisions. A coordination graph may be specified as part of the domain, or if the graph is *context specific* [6], a combination of rules provided with the domain. This set of rules determines whether an edge between any two vertices of the graph should exist, given the state.

As in [8] we use an edge-based decomposition of a context-specific coordination graph. The global Q-function for such a decomposition is approximated by a sum over all local Q-functions, each defined over an edge (i, j) of the graph:

$$Q(\mathbf{s}, \mathbf{a}) = \sum_{(i,j) \in E} Q_{ij}(\mathbf{s}_{ij}, a_i, a_j) \tag{3}$$

where $\mathbf{s}_{ij} \subseteq \mathbf{s}_i \cup \mathbf{s}_j$ is the subset of state variables relevant to agents i and j, and $(i, j) \in E$ describes a pair of neighboring nodes (i.e., agents). The optimal action for a coordination graph is given by $\arg\max_\mathbf{a} Q(\mathbf{s}, \mathbf{a})$. As with the agent-based Q-function, the notation Q_{ij} indicates only that the Q-value is edge-based, and parameters may or may not be shared between edges.

Coordination graphs are a powerful method of coordinating multiple agents, but they are ill-fitted for solving multi-agent assignment problems with arbitrary coordination constraints. We show a simple proof of this below. For simplicity we equate tasks and actions and assume that each action is relevant to a single task a or b:

PROPOSITION 3.1. *Arbitrary reward functions from the joint action space $A_1 \times \ldots \times A_n$ to $\{0, 1\}$ are not expressible using an edge-based decomposition over a coordination graph.*

Proof: Let $A_1 = \ldots = A_n = \{a, b\}$, hence there are 2^n joint actions. Each joint action may be mapped to 0 or 1, leading to 2^{2^n} possible functions. To represent these functions, we need at least 2^n bits. A coordination graph over n agents has at most $O(n^2)$ edges. Each edge has at most 4 constraints, one for each possible action pair. Thus, we have room for specifying only $O(n^2)$ values, which are not sufficient to represent 2^{2^n} possible functions. □

4. ASSIGNMENT-BASED DECOMPOSITION OF MAMDPS

Because coordination graphs cannot capture some of the coordination requirements that are needed in an MAMDP, we propose an alternate solution. Rather than using coordination graphs to make assignment decisions, we split the action selection step of the Q-learning algorithm into two levels: the upper assignment level, and the lower task execution level. At the assignment level, agents are assigned to tasks. Once the assignment decision is made, the lower level action that each agent should take to complete its assigned task is decided by Q-learning in a smaller state space. This two-level decision making process occurs each time-step of the reinforcement learning algorithm, taking advantage of the opportunistic reassignments.

At the assignment level, we ignore interactions between the agents assigned to different tasks. This action decomposition exponentially reduces the number of possible actions that need to be considered at the lowest level, at a cost of increasing the number of possible assignments that must be considered. Because each agent g need only consider its local state s_g and task-specific state s_t to come to a decision, this method can greatly reduce the number of parameters that are necessary to store. This reduction is possible because rather than storing separate value functions for each possible agent and task combination, we can share a single value function between multiple agent-task assignments.

We use $Q(s_t, s_{\beta(t)}, a_{\beta(t)})$ to denote the discounted total reward for a task t and set of agents $\beta(t)$ starting from a task state s_t, the joint state $s_{\beta(t)}$ and joint actions $a_{\beta(t)})$ of the agents in the team. We learn the Q-function for an assigned subset of agents using standard Q-learning approaches:

$$\begin{aligned} \mathrm{Q}(s_t, s_{\beta(t)}, a_{\beta(t)}) \leftarrow \mathrm{Q}(s_t, s_{\beta(t)}, a_{\beta(t)}) + \alpha\Big[r_{\beta(t)} + \\ \gamma \max_{a' \in A'_{\beta(t)}} \mathrm{Q}(s'_t, s'_{\beta(t)}, a') - \mathrm{Q}(s_t, s_{\beta(t)}, a_{\beta(t)})\Big] \end{aligned} \tag{4}$$

The assignment problem described is nontrivial – there are an exponential number of possible assignments in the number of agents. However, there is an opportunity to apply various search techniques to solve this problem, and we discuss several of these here.

Exhaustive search: One solution is to perform an exhaustive search over all possible assignments of agents to tasks. Each such assignment is given a weight, which is derived from the underlying value function, given by Equation 4. The value $v_{\mathbf{g},t}$ of a task t and set of agents $\mathbf{g}$ is thus the maximum value of all possible actions of those agents:

$$v_{\mathbf{g},t} = \max_{\mathbf{a} \in A_\mathbf{g}} Q(s_t, s_\mathbf{g}, \mathbf{a}) \tag{5}$$

We then exhaustively search for the mapping β that returns the maximum total value for all tasks $\max_\beta \sum_t v_{\beta(t),t}$. This method has the advantage that it is guaranteed to select the optimum assignment. However, for many agents, this search could become intractable. A faster approximate search technique might be necessary, and so we also considered two simple approximate search techniques.

Sequential greedy assignment: This search uses a simple method of greedily assigning agents to high-value tasks: for each task t we consider all sets of agents that might be assigned, and choose the set $\mathbf{g}$ that provides the maximum value $v_{\mathbf{g},t}$. We remove agents $\mathbf{g}$ from future consideration, and repeat until all tasks or agents have been assigned.

Swap-based hill climbing: This method uses the assignment at the previous step (or a random assignment for the first time this search occurs) as the starting point of a hill climbing search of the assignment space. At each step of the search, we consider all possible next states obtained by swapping a set of agents from one task with another set of the same size assigned to a different task. We then commit to the swap resulting in the most improvement, repeating until convergence.

5. ADVANTAGES OF ASSIGNMENT-BASED DECOMPOSITION

We analyze the time complexity of assignment-based decomposition as follows. The time required to perform an exhaustive search of the assignment space is the sum of the time required to pre-calculate $v_{\mathbf{g},t}$ values and the time required to perform the actual search. The time required to calculate a single $v_{\mathbf{g},t}$ value is $O(|A|^k)$, where $|A|$ is the number of actions a single agent may take, and k is an upper bound on the number of agents that may be assigned to a task. Therefore, the time required to pre-calculate all values of $v_{\mathbf{g},t}$ is $O(|A|^k|T|C^n_k)$ where C is the choice function, $|T|$ is the number of tasks, and n is the number of agents. An exhaustive search requires $O(n!/(k!)^{n/k})$ time, which is proportional to the number of ways to assign k agents each to n/k tasks. This is significantly reduced by both the sequential greedy search and swap-based hill climbing.

The advantage of assignment-based decomposition is much more apparent when we consider the space complexity of the value function. A value function over the entire state-action space would require $O(S_t^{|T|}S_a^n|A|^n)$ parameters, where S_t and S_a are the sizes of the state required to store local parameters for each task and agent respectively. Assignment-based decomposition uses considerably fewer parameters to store the task-based value function $Q(s_t, s_{\mathbf{g}}, \mathbf{a})$. Instead, we need space of only $O(S_tS_a^k|A|^k)$ parameters for each task.

A further advantage of the additive decomposition of the task execution level in Equation 3 is that each $Q_{i,j}$ function may share the same parameters. Generalizing, or transferring, that single shared value function to additional tasks and/or agents can be quite simple. In many cases, no additional learning is necessary. The same value function can often be used, for example, in domains with twice as many tasks and agents as the original domain. Only the size of the search space at the assignment level needs to grow.

6. COORDINATION GRAPHS AND MAMDPS

Assignment-based decomposition is sufficient coordination if the problem is completely decomposed after assignments have been made; however this is often not the case. The possibility remains of interference between agents assigned to different tasks. To handle such interactions, we define a coordination graph over agents acting on the task execution level. An edge should be placed between two agents when the actions of those agents might interfere, such as when a collision is possible or the two agents might need to share a common resource. Such coordination must be context-specific since agents are constantly changing states. Thus, it is necessary to combine the assignment decisions with context-specific coordination at the task execution level. To that end, we adapt some methods described in [7] and [8].

6.1 The Max-plus Algorithm

If we are to place multiple agents in a coordination graph, we must use an action selection algorithm that can take advantage of this structure. We wish to maximize the global payoff $\max_{\mathbf{a}} Q(\mathbf{s}, \mathbf{a})$, (where $Q(\mathbf{s}, \mathbf{a})$ is given by Equation 3). Initial work in coordination graphs suggested a variable elimination (VE) technique [5] to solve this problem, however work in [8] shows that VE techniques can be slow to solve large coordination graphs, require a lot of memory, and in addition can be quite complex to implement. Instead, [8] proposed the use of the *Max-plus* algorithm.

The Max-plus algorithm is a message-passing algorithm based on belief propagation for Bayesian networks [11, 15, 14]. Agents in Max-plus instead pass (normalized) values indicating the locally optimal payoff of each agent's actions along edges of the coordination graph. Max-plus finds the global payoff by having each agent i repeatedly sending messages μ_{ij} to its neighbors:

$$\mu'_{ij} = \max_{a_i}\Big\{Q_{ij}(\mathbf{s}_{ij}, a_i, a_j) + \sum_{k\in\Gamma(i)\setminus j} \mu_{ki}(a_i)\Big\} - c_{ij} \qquad (6)$$

where μ_{ki} is the incoming message, and μ'_{ij} is the outgoing message. All messages are in fact vectors over possible actions. $\Gamma(i)\setminus j$ represents all neighbors of i except j and c_{ij} is a normalization factor, calculated after the initial values μ'_{ij} have been found. Max-plus sets this to be the average over all values of the outgoing message: $c_{ij} = \frac{1}{|A_j|}\sum_{a_j} \mu'_{ij}(a_j)$. This prevents messages from exploding in value as multiple iterations of the algorithm proceed. Once messages have converged or a time limit has been reached, each agent chooses the action that maximizes $\arg\max_{a_i}\{Q_{ij}(\mathbf{s}_{ij}, a_i, a_j) + \sum_{j\in\Gamma(i)} \mu_{ji}(a_i)\}$ for that agent.

6.2 Dynamic Coordination

Although we use an edge-based decomposition (as in Section 2), it is often the case that rewards are received on a per-agent basis instead of a per-edge basis. Thus, we must compute local Q_i functions for each agent in the graph. Following [8] we do this by assuming each Q_{ij} contributes equally to each agent i and j of its edge:

$$Q_i(\mathbf{s}_i, a_i) = \frac{1}{2}\sum_{j\in\Gamma(i)} Q_{ij}(\mathbf{s}_{ij}, a_i, a_j) \qquad (7)$$

where $\Gamma(i)$ indicates the neighbors of agent i. The sum of all such Q_i functions equals Q in Equation 3. We assume each agent has at least one other neighbor in the coordination graph, i.e., the graph is connected. It is fairly straightforward to adapt these methods in cases where an agent does not need to coordinate with anyone.

```
Initialize $Q(\mathbf{s},\mathbf{a})$ optimistically
Initialize $\mathbf{s}$ to any starting state
**for** each step **do**
    Assign tasks $T$ to agents $M$ by finding $\arg\max_\beta \sum_t v_{\beta(t),t}$, where $v_{\mathbf{g},t} = \max_{a \in A_\mathbf{g}} \sum_{i,j \in \mathbf{g}} Q_{ij}(\mathbf{s}_{ij}, a_i, a_j)$
    Choose $\mathbf{a}$ from $\mathbf{s}$ using max-plus algorithm and $\epsilon$-greedy policy derived from $Q$
    Take action $\mathbf{a}$, observe rewards $\mathbf{r}$ and next state $\mathbf{s}'$
    Use rules given with domain to create coordination graph $G = (V, E)$ for state $\mathbf{s}'$
    Determine agent Q-functions $Q_i(\mathbf{s}_i, a_i)$ and $Q_i(\mathbf{s}'_i, a^*_i)$ for each agent $i$ using $Q_i(\mathbf{s}_i, a_i) = \frac{1}{2} \sum_{j \in \Gamma(i)} Q_{ij}(\mathbf{s}_{ij}, a_i, a_j)$
    For each edge $(i, j)$ of the coordination graph, update its Q-value using
    $Q_{ij}(\mathbf{s}_{ij}, a_i, a_j) \leftarrow Q_{ij}(\mathbf{s}_{ij}, a_i, a_j) + \alpha \sum_{k \in \{i,j\}} \frac{R_k(\mathbf{s},\mathbf{a}) + \gamma Q_k(\mathbf{s}'_k, a^*_k) - Q_k(\mathbf{s}_k, a_k)}{|\Gamma(k)|}$
    $\mathbf{s} \leftarrow \mathbf{s}'$
**end for**
```

Figure 3: The assignment-based decompostion Q-learning algorithm using coordination graphs.

Because our coordination graph is context-specific, to update the Q-function we must use an agent-based update. This is because the presence or absence of edges changes from state to state, so we cannot be assured that an edge that is present in the current time step was available in the last time step. To obtain the agent-based update equation for an edge-based decomposition, the agent-based update (Equation 2) is rewritten using Equation 7 to get:

$$Q_{ij}(\mathbf{s}_{ij}, a_i, a_j) \leftarrow Q_{ij}(\mathbf{s}_{ij}, a_i, a_j) + \alpha \sum_{k \in \{i,j\}} \frac{R_k(\mathbf{s},\mathbf{a}) + \gamma Q_k(\mathbf{s}'_k, a^*_k) - Q_k(\mathbf{s}_k, a_k)}{|\Gamma(k)|} \tag{8}$$

This update equation propagates the temporal-difference error from all edges including agents i and j to the local Q-function of each edge (i, j). This update is context-specific because it does not require the same edges to be present at each time step of the Q-learning algorithm; only that local Q_k functions can be computed for each vertex of the coordination graph, which is done using Equation 7. The notation Q_i and Q_{ij} indicates that the Q-values are agent-based or edge-based respectively. Q-function parameters are shared between agents and edges. The final Q-learning algorithm may be seen in Figure 3.

A complication arises during the assignment search step of Figure 2 when using coordination graphs. It is not possible to efficiently calculate the value of an assignment $v_{\mathbf{g},t}$ while still taking into account the contribution of edge-based Q-values Q_{ij} that occur between groups of agents assigned to different tasks. Hence, we approximate $v_{\mathbf{g},t}$ by only taking into account the local state and actions of its assigned agents, the state variables s_t, and ignoring inter-group edges of the graph. At the task execution level, we consider all interactions, but since the task assignment is fixed, the possible interactions are again limited.

7. EXPERIMENTAL RESULTS

We conducted experiments in two MAMDP domains: a simple product delivery and vehicle routing domain, and a cooperative multiagent predator-prey domain based on [7]. Our product delivery domain does not require coordination on the task execution level. It is a simple enough domain that flat and multiagent Q-learning results can be obtained (albeit requiring function approximation) for comparison to our approach using a two-level decomposition.

The multiagent predator-prey domain is a more complex domain. Standard Q-learning approaches do not work here. We also tested the use of coordination graphs, and several different assignment search techniques.

7.1 The Product Delivery Domain

The first domain we experimented with is a simple product delivery domain previously described in [12], shown in Figure 6. We assume a supplier of a single product that needs to be delivered to several shops from a warehouse using several trucks. The goal is to ensure that the stores remain supplied while minimizing truck movements. It takes one unit of time to go from any location to its adjacent location or to execute an unload action.

The shop inventory levels and truck load levels are discretized into 5 levels 0-4. States are factored, and state features include information about truck position, load, and shop inventories. Each truck has 9 actions available at each time step: unload 1, 2, 3, or 4 units, move in one of up to four directions, or wait. Trucks are loaded automatically upon reaching the depot. A small negative reward of -0.1 is given for truck move actions to reflect transportation costs.

Customer consumption is modeled at each shop by decreasing the inventory level by 1 unit with some probability, which independently varies from shop to shop. There is a penalty of -5 for stockouts, i.e., if a customer enters a store and finds the shelves empty. Thus, filling each store becomes a task for any of the multiple agents (trucks) to complete.

Our experiments involved four agents delivering goods to five shops. An assignment is therefore a mapping from shops to the trucks that will serve them. Each shop is assigned one truck, which may only unload at that shop. Thus, agents' actions cannot interfere with each other, and there is no need for coordination on the task execution level. Because not all shops can be delivered to, we add a "phantom truck" for the unassigned shop. This "agent" has no associated state features. Its existence allows the assignment step of the assignment-based decomposition to determine the appropriate penalty for not assigning a truck to any shop.

We conducted several experiments in this domain (see Figure 4). All results were averaged over 30 runs of 10^6 steps each. We tuned the learning rate α separately for each test, setting $\alpha = 0.1$ for the assignment-based decomposition test and $\alpha = 0.01$ for all others. We set the discount rate $\gamma = .9$, and used ϵ-greedy exploration with $\epsilon = .1$. Average reward was measured for 2,000 out of every 50,000 steps.

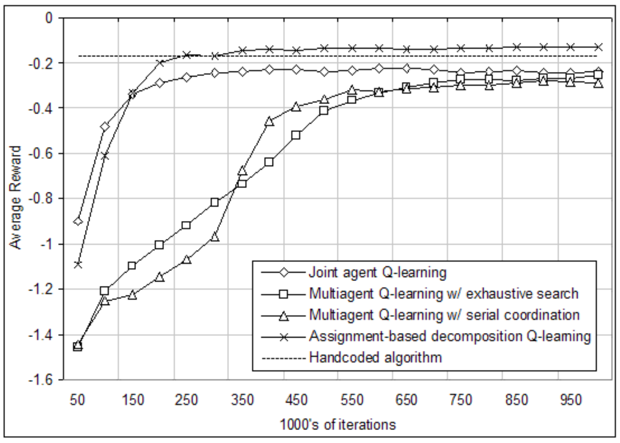

Figure 4: Comparison of various Q-learning approaches for the product delivery domain.

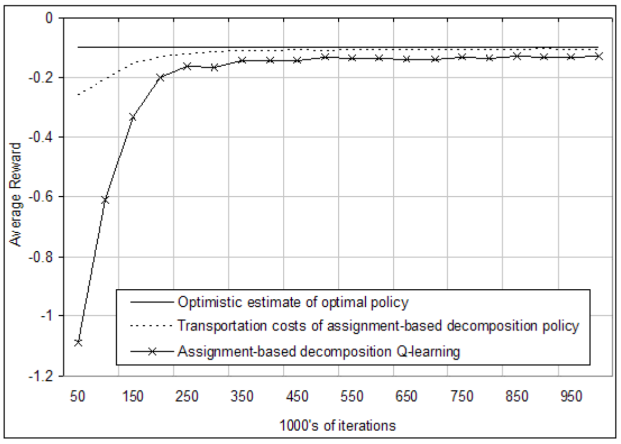

Figure 5: Examination of the optimality of policy found by assignment-based decomposition for product delivery domain.

Our assignment-based decomposition approach used an exhaustive search of possible assignments and no function approximation, so we required 11,250 parameters to store the value function $Q(s_t, s_{\beta(t)}, a_{\beta(t)})$ (5 shops, 5 shop inventory levels, 10 truck locations, 5 truck loads, and 9 possible actions per truck). Here s_t indicates state features about the assigned shop and its inventory, and $s_{\beta(t)}$ indicates features for truck position and load.

The joint and multiagent Q-learning approaches we used need too many parameters to keep the value function in a complete table. Thus, we used tabular linear function approximation [12], which sums over a set of terms, each derived from a table. Our multiagent Q-learning approaches sum over one term for each shop to obtain the value function. Each agent uses its own value function, so we used four times as many parameters as the assignment-based decomposition. Joint agent Q-learning sums over four times as many terms, additionally indexing with each truck, but required no additional parameters. Our handcoded approach worked similarly to the assignment based decomposition: for each truck-shop pair, a distance weight was calculated from the state features, then the assignment was made based on an exhaustive search over possible assignments, taking the assignment giving minimum total distance.

We tried two coordination methods for multiagent Q-learning: when selecting actions, we either exhaustively searched over all joint actions, or used a simple form of multiagent coordination called serial coordination, which greedily selects actions for agents one at a time, allowing each agent to know the actions selected by previous agents.

Assignment-based decomposition outperformed all other approaches, although our handcoded algorithm comes close. The multiagent Q-learning approaches performed the worst of these methods. In CPU time, both multiagent Q-learning with serial coordination and assignment-based decomposition approaches were much faster than those approaches using an exhaustive search of the action space (Table 1).

Table 1: Running times (in seconds), parameters required, and and terms summed over for five algorithms applied to the product delivery domain.

Algorithm	Time	Space	Terms
Joint agent Q-learning	142	45,000	20
Multiagent Q, exhaustive search	160	45,000	5
Multiagent Q, serial coordination	3	45,000	5
Assignment-based decomposition Q	3	11,250	1
Handcoded algorithm	3	N/A	N/A

We also examined the optimality of the policy found by assignment-based decomposition in the product delivery domain (Figure 5). The top line is an optimistic estimate of the optimal policy in this domain. We calculated this by multiplying the average number of customer visits per time step (1) by the transportation cost required to satisfy a single customer visit $(-.1)$ to get the average transportation cost/time step required to satisfy all customers $(-.1)$. This estimate is very optimistic because it ignores stockout costs, which are inevitable due to the stochastic nature of customer visits. Still, the average reward of the policy found by assignment-based decomposition is quite close to that of our estimate. We can take this analysis one step further: our estimate ignores stockout costs, we can similarly ignore the contribution of stockout events to the average reward of the policy found by assignment-based decomposition. The result is a graph of only the transportation costs incurred

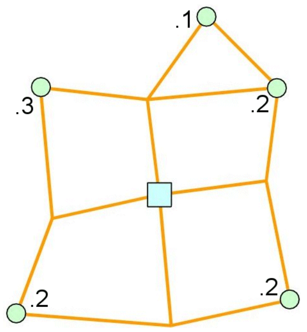

Figure 6: The product delivery domain, with depot (square) and five shops (circles). Numbers indicate probability of customer visit each time step.

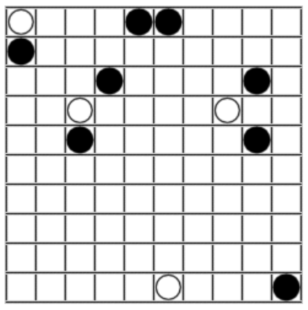

Figure 7: A possible state in an 8 vs. 4 toroidal grid predator-prey domain. All eight predators (black) are in a position to possibly capture all four prey (white).

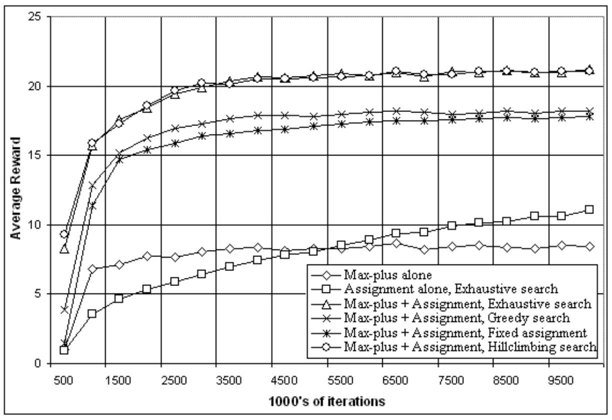

Figure 8: Comparison of action selection and search methods for the 4 vs 2 Predator-Prey domain.

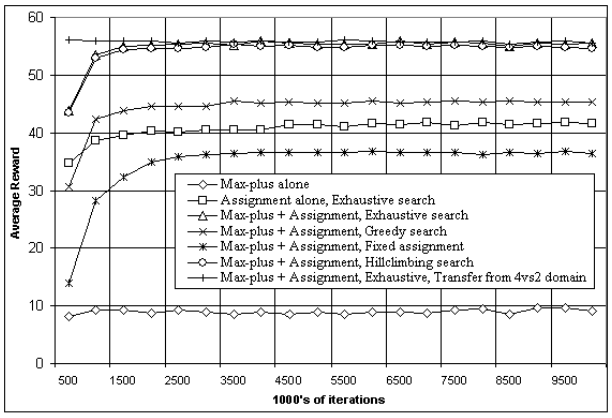

Figure 9: Comparison of action selection and search methods for the 8 vs 4 Predator-Prey domain.

by this policy, seen in Figure 5. From this we conclude that the policy found by assignment-based decomposition is very close to optimal if not the optimal in this domain.

7.2 Multiagent Predator-Prey Domain

The second domain we experimented in is a cooperative multiagent predator-prey domain based on work by [7]. That original domain required two agents (predators) to cooperate in order to capture a single prey. Agents move over a 10x10 toroidal grid world, and may move in four directions or stay in place. Prey move randomly to any empty square. Predators and prey move simultaneously, so predators must guess where the prey will be in the next time step. If predators collide, or if a predator enters the same space as the prey without an adjacent predator, the responsible predators are penalized and moved to a random empty square. The prey is captured (with a reward of 75) when one predator enters its square, and another predator is adjacent.

This domain exhibits two key differences to that of [7]: first, we increase the numbers of predators and prey from 2 vs. 1 to 4 vs. 2 or 8 vs. 4 (see Figure 7). Second, each time a prey is captured, it is randomly relocated somewhere else on the board and the simulation continues. Thus, our domain has an infinite horizon rather than being episodic.

There are several consequences of the increase in scale of our domain. Of course, the joint action and state spaces increase exponentially. More interesting is a need for predators to be assigned to prey such that exactly two predators are assigned to capture each, if the best average reward is to be found. Thus, this domain is an example of an MAMDP.

Once predators are assigned to prey, it is useful to coordinate the actions of predators on the task execution level to prevent collisions. Thus we introduce coordination graphs on the task execution level as described in Section 6. The existence of a top-level assignment provides several advantages, such as when defining the rules determining when agents should cooperate. We change only one of the coordination rules introduced in [7]. Predators should coordinate when either of two conditions hold:

- the Manhattan distance between them is less than or equal to two cells.
- both predators are assigned to the same prey.

The existence of predator assignments allows us to both create improved coordination rules on the task execution level, and reduces the number of state variables (i.e., prey) we are required to account for in the edge value function. The Q-value of each edge between predators cooperating to capture a prey need only be based on the positions of those predators relative to their assigned prey. The Q-values of each edge between predators cooperating only for collision avoidance need only be based on the positions of those two predators. The existence of these two kinds of edges does increase the number of parameters required to learn the value function, but far less than the exponential increase in parameters required to store a value function over two predators and two or more prey (without function approximation).

We conducted several experiments in the multiagent predator-prey domain (Figures 8 and 9). In these tests, we show results over 10^7 steps of our algorithms (Figures 2 and 3). Figure 8 shows the results for 4 predators vs. 2 prey, and Figure 9 shows the results for 8 predators vs. 4 prey. We compared the same set of search and coordination strategies in both domains. We set the learning rate $\alpha = 0.1$, discount rate $\gamma = .9$, and exploration rate $\epsilon = .2$. Average reward of the domain was measured for 2,000 steps out of every 500,000 steps. During test phases, ϵ was set to 0. Because the maximum reward receivable by two agents is 75, edge value functions were optimistically initialized to this value. We averaged over 30 runs to obtain the results for this paper.

We conducted six identical experiments for each domain: Max-plus action selection without an assignment-based decomposition (using sparse cooperative Q-learning as in [7]), assignment-based decomposition without using coordination graphs and using an exhaustive search of assignments (as in Figure 2), and assignment-based decomposition with max-plus action selection and four assignment methods: exhaustive search, sequential greedy assignment, swap-based hill climbing, and a fixed assignment (as in Figure 3). For the fixed assignment, we arbitrarily assigned pairs of predators to prey at the start of the run, then never reassigned them.

As may be seen from these results, Max-plus search alone performed poorly compared to the other techniques. This is because a coordination graph alone is unable to capture the coordination requirements of the predator-prey domain (see Proposition 3.1). Using assignment search alone results in a large increase in performance; this kind of search does capture some essential coordination requirements. However, this alone is also not enough: it is still possible for agents to interfere (collide) with each other after assignments have been made. This type of coordination is ideal for a coordination graph approach to solve as described in Section 6, as may be seen by the experiments combining assignment search with max-plus action search.

Of the various task assignment methods, fixed assignment and sequential greedy assignment did not perform well. Swap-based hill climbing performed almost identically to exhaustive search. This gives us hope that similar approximate search techniques can allow assignment-based decomposition to scale to a large number of agents.

We also experimented with transfer learning (Figure 9). Instead of initializing Q-values optimistically, we transferred parameters learned from the 4 vs. 2 to the 8 vs. 4 predator-prey domain. This is possible because both domains have the same number of parameters; as would any number of agents because the Q-functions are all based on 2 predators and 1 prey. We tested the resulting policy by turning off learning and using assignment-based decomposition with exhaustive assignment search and max-plus coordination. These results demonstrate that, thanks to assignment-based decomposition, a policy learned with few agents can scale successfully to many more agents.

8. DISCUSSION

We introduced Multiagent Assignment MDPs and gave a two-level decomposition method that is effective for this class of MDPs. This class of MDPs can capture many real-world domains such as vehicle routing and delivery, board and real-time strategy games, disaster response, fire fighting in a city, etc., where multiple agents and tasks are involved. We gave empirical results in two domains that demonstrate that the combination of assignment search at the top level and coordinated reinforcement learning at the task execution level is well-suited to solving such domains while either method alone is not sufficiently powerful.

Because a search over an exponential number of assignments is not scalable as the number of agents increases, we have also shown how several simple approximate search techniques perform effective assignment search. These results encourage the conclusion that assignment search is a practical approach for large cooperative multiagent domains. Future work includes scaling the approaches in this paper to work with much larger number of agents, tasks, and state variables and considering other kinds of interactions such as global resource constraints.

Future work in assignment-based decomposition could address adapting it to a decentralized domain. The Max-plus algorithm already can be decentralized [8], however assignment-based decomposition assumes a centralized controller. Adapting our algorithms to work in a DEC-MDP context could involve similar message-passing techniques to those used by the max-plus algorithm.

Acknowledgments

We gratefully acknowledge the support of the Defense Advanced Research Projects Agency under grant number FA8750-05-2-0249 and the National Science Foundation under grant number IIS-0329278.

9. REFERENCES

[1] R. Becker, S. Zilberstein, and C. V. Goldman. Solving transition independent decentralized markov decision processes. *JAIR*, 22:423–455, 2004.

[2] D. Dolgov and E. Durfee. Optimal resource allocation and policy formulation in loosely-coupled markov decision processes. In *AAMAS '04*, pages 315–324, June 2004.

[3] C. V. Goldman and S. Zilberstein. Decentralized control of control of cooperative systems: Categorization and complexity analysis. *JAIR*, 22:143–174, 2004.

[4] C. Guestrin, D. Koller, and R. Parr. Multiagent planning with factored MDPs. In T. G. Dietterich, S. Becker, and Z. Ghahramani, editors, *NIPS '01*, pages 1523–1530. MIT Press, 2001.

[5] C. Guestrin, M. Lagoudakis, and R. Parr. Coordinated reinforcement learning. In *ICML '02*, San Francisco, CA, July 2002. Morgan Kaufmann.

[6] C. Guestrin, S. Venkataraman, and D. Koller. Context specific multiagent coordination and planning with factored MDPs. In *AAAI '02*, pages 253–259, Edmonton, Canada, July 2002.

[7] J. R. Kok and N. A. Vlassis. Sparse Cooperative Q-learning. In *ICML '04*, pages 481–488, Banff, Canada, July 2004. ACM.

[8] J. R. Kok and N. A. Vlassis. Collaborative multiagent reinforcement learning by payoff propagation. *JMLR*, 7:1789–1828, 2006.

[9] R. Makar, S. Mahadevan, and M. Ghavamzadeh. Hierarchical multi-agent reinforcement learning. In *Proceedings of the 5th International Conference on Autonomous Agents*, pages 246–253, Montreal, Canada, 2001. ACM Press.

[10] N. Meuleau, M. Hauskrecht, K.-E. Kim, L. Peshkin, L. P. Kaelbling, T. Dean, and C. Boutilier. Solving very large weakly coupled markov decision processes. In *AAAI '98/IAAI '98*, pages 165–172, Menlo Park, CA, USA, 1998. AAAI Press.

[11] J. Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1988.

[12] S. Proper and P. Tadepalli. Scaling model-based average-reward reinforcement learning for product delivery. In *ECML '06*, pages 735–742, 2006.

[13] R. S. Sutton and A. G. Barto. *Reinforcement learning: an introduction.* MIT Press, 1998.

[14] M. Wainwright, T. Jaakkola, and A. Willsky. Tree consistency and bounds on the performance of the max-product algorithm and its generalizations. *Statistics and Computing*, 14(2):143–166, 2004.

[15] J. S. Yedidia, W. T. Freeman, and Y. Weiss. Understanding belief propagation and its generalizations. *Exploring artificial intelligence in the new millennium*, pages 239–269, 2003.

[16] X. Zhang, D. Aberdeen, and S. V. N. Vishwanathan. Conditional random fields for multi-agent reinforcement learning. In *ICML '07*, pages 1143–1150, New York, NY, USA, 2007. ACM.