Using Conflict Resolution to Inform Decentralized Learning

Shanjun Cheng
Altisource
Greensboro, North Carolina
chengshanjun@gmail.com

Anita Raja
Department of Software and Information Systems
The University of North Carolina at Charlotte
Charlotte, NC
anraja@uncc.edu

Victor Lesser
Department of Computer Science
University of Massachusetts Amherst
Amherst, MA 01003
lesser@cs.umass.edu

ABSTRACT

Learning consistent policies in decentralized settings is often problematic. The agents have a myopic view of their neighboring states that could lead to inconsistent action choices. The fundamental question addressed in this work is how to determine and obtain the minimal overlapping context among decentralized decision makers required to make their decisions more consistent. Our approach is a two-phased learning process where agents first learn their policies offline within the context of a simplified environment where it is not necessary to know detailed context information about neighbors. These local policies are then applied in more complex "real" environments where it is expected that agents will encounter a much higher rate of inconsistencies (conflicts) with neighborhood actions. When conflicts are observed, agents switch to "special" states that augment local policy states with additional non-local state information and learn other actions to take in this specific situation. This results in action choices that are less likely to lead to conflicts. We evaluate our approach by addressing meta-level decisions in a complex multiagent weather tracking domain. Experimental results show that our approach achieves good performance on utility and conflict resolution by exploring only a small fraction of the whole search space.

Categories and Subject Descriptors

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

General Terms

Algorithms, Performance

Keywords

Agent Reasoning; Multiagent Learning; Self-organization

1. INTRODUCTION

Locally learned agent policies can lead to inconsistent agent actions in a multi agent context. While there are applications where the existence of inconsistent actions merely leads to lower utility gains for the multi agent system (MAS), we are specifically interested in applications where inconsistencies could be mission-critical in that they violate hard domain constraints like trying to use the same resource for two different activities or where there is a dramatic decrease in utility if the inconsistency is not addressed. This has led to more complex models [4,7–9] of action invocations that involve checking with other agents to ascertain that action choices are consistent before actual action execution. These detected inconsistencies in local policies can be resolved to varying degrees depending on the level of effort. For instance one simple way to resolve inconsistencies is to have one or more of the agents engaged in the conflict to voluntarily retract their action; other choices would involve more complex interactions among agents. We explore the question of what types of mechanisms can be introduced into decentralized agent learning so that the likelihood of inconsistent actions occurring is minimized. One way of solving this problem is for each agent to acquire and integrate more context information about neighbors into its local decision-making so that it is more likely for its action choice to be consistent with those of its neighbors. However including detailed information about neighbors expands the search space exponentially of each local agent. This slows down agent learning dramatically and also introduces additional communications.

The basic premise of this work is that when agents in a multiagent system (MAS) have more contextual information about the states of other agents in their environment, then their local policies tend to be more coordinated. Since the costs associated with obtaining such contextual information in large state spaces can be significant, we claim that it is beneficial to augment agents with the capability to **selectively** obtain context. Our approach involves a two-phased learning process where each agent first uses a policy gradient algorithm to learn offline policies assuming the context of a simplified environment wherein it is not necessary to know detailed information about its neighbors to get reasonably coordinated local policies. This effectively means enlarging the state space but using default values to represent the information about neighbors.

Agents then execute these learned local policies in online environments where the simplistic assumptions are no longer expected to be valid. When conflicts resulting from multiple neighboring agents applying their local policies are observed, the second learning phase where the agents use the same policy gradient algorithm to learn online policies is invoked. First a deterministic negotiation-based conflict resolution

Appears in: *Proceedings of the 12th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2013), Ito, Jonker, Gini, and Shehory (eds.), May, 6–10, 2013, Saint Paul, Minnesota, USA.*

process is used to resolve the conflicts. In more recent work [3], this conflict resolution process was implemented as a decentralized constraint-optimization process. If some conflicts remained unresolved, the agents dynamically augment the state space of the conflicting agents with new "special" states that include information about the actual context of the neighboring agents rather than the default context. From the new special state, the agent then explores actions that could potentially resolve the conflicts.

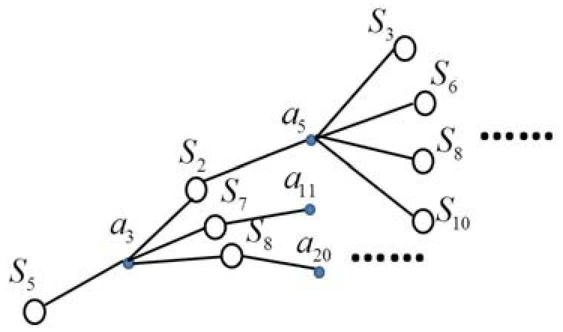

Figure 1: Agent *A's* partial search tree

We now describe an example scenario to further elucidate this idea of "learning only where learning is needed". Consider two neighboring agents A and B in a MAS that need to ensure that their actions are coordinated. Figure 1 describes agent *A's* partial search tree for environment *E1*. This tree is learned by agent A using a policy gradient algorithm in the initial offline learning phase where it assumes that agent B operates in environment *E1* as well. Once it's search space is expanded under these simplified assumptions, agent *A's* action policy is determined offline and then applied in a real online environment where the environmental assumptions about B may or may not be true.

Suppose at real-time, agents A and B are operating in the same (both agents encounter homogeneous weather patterns) environment *E1*. At time *T0*, agent A determines it is in state $S5$. It chooses the action a_3 prescribed by its offline policy for environment *E1* and deliberates about this action by identifying any conflicts that may arise with B's action choice. Agent A determines that there are no conflicts and so proceeds to execute action a_3.

Now consider a different scenario where agent B's environment is *E2* instead of *E1*. Agent A determines it is in state $S2$. It chooses action a_5 prescribed by its offline policy for environment *E1* and determines that there are conflicts with agent B while deliberating about this action. Agent A tries to resolve the conflict using a deterministic conflict resolution algorithm. Since it is unable to resolve the conflict, agent A then updates its search space with a new (special) state S_9 that has accurate information about agent *B's* environmental context. Agent A then expands the search space with state S_9 as root. It uses the policy gradient algorithm to learn that a_7 is the best action to take at state S_9. Figure 2 depicts this situation. This process repeats till coordination between he two agents is no longer needed.

We explore this two-phased learning technique as it relates to the development of decentralized meta-level control policies in an adaptive distributed sensor network for

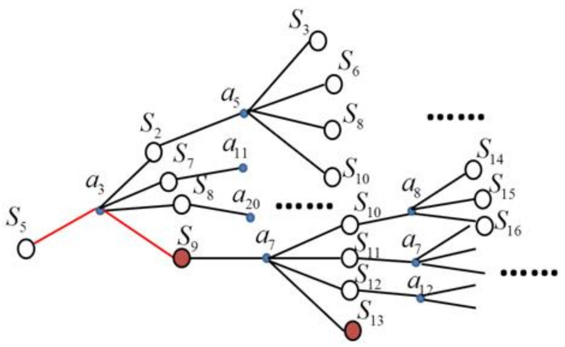

Figure 2: Agent *A's* search tree after expanding special state S_9

tracking weather phenomenon. The meta-level questions involve reorganization of the underlying agent network as well as adjusting the internal parameters of each agent when trying to maximize the performance of the agent network.

Our work benefits from the following assumptions: the number of special states that are added via online learning are limited and that the learned policy has a finite horizon; conflicts occur only with agents in the immediate neighborhood; agents can deliberate about taking the best action before executing the actions; and finally the policy construction process is time constrained and can use only up to 10% of the total deliberation time.

To summarize, the contributions of this work include equipping agents in a real-world application domain with the ability to learn only where learning is required while taking real-time constraints into account. It also introduces the use of deterministic conflict resolution strategies embedded in a decentralized reinforcement learning framework.

The rest of the paper is laid out as follows: Section 2 discusses the real-world tornado tracking application, NetRads and sets our work in the context of the state-of-the art. Section 3 is the description of our approach. Section 4 provides empirical results. Finally, Section 5 summarizes our work and outlines future directions.

2. BACKGROUND

2.1 Informed State Expansion

Wu and Durfee [12] present a solver that selectively unrolls the search space for single agents with large state spaces. It generates an (approximately) optimal policy while avoiding exhaustive enumeration of all possible states. In their work, the exploration of the state space that is likely to be reached by the optimal policy is emphasized. Alexander et al. [1] implemented a single agent meta-level control scheme to determine when the agent should stop unrolling in order to derive a partial policy while bounding the costs of state re-prioritization. Their approach collects performance profile information to make meta-level decisions on state expansion. Melo and Veloso [11] augment the action space with a pseudo-action that uses active perception to gather information from other agents to determine the correct local action. In our work described here, we unroll the state space of each agent in a multiagent system using a reinforcement learning that is informed by conflict resolution performance.

We re-prioritize the set of heuristics that guide the unrolling using actual online performance. The novelty of our work lies in the way it goes beyond smart space expansion by combining it effectively with a learning strategy.

2.2 Weather tracking application domain

Netrads [10, 15] is a network of adaptive radars controlled by a collection of Meteorological Command and Control (MCC) agents that determine for each radar where to scan based on emerging weather conditions. The NetRads radar is designed to quickly detect low-lying meteorological phenomena such as storm, rotation, reflectivity and velocity. The MCC agent can manage multiple radars simultaneously, where each radar belongs to exactly one MCC. The time allotted to the radar and its control systems for data gathering and analysis of tasks is known as a *heartbeat*. Each MCC agent has two choices of heartbeat: 60 seconds and 30 seconds. A shorter heartbeat allows the system to respond more rapidly to closely track the quickly evolving weather phenomena but with less fidelity.

Each radar has a scanning area represented by a circle and may have overlapping scanning areas with other radars. Two MCCs are *neighbors* if their radars share overlapping scanning areas. More generally, perception overlap and/or topologically features can be used to identify neighbors. MCCS have partial knowledge of their environment in that they observe the weather phenomenon and their neighbor actions with limited scope.

There are four types of weather phenomena, namely storms, rotations, velocity and reflectivity. A radar scanning *task* in the Netrads system is a weather phenomenon that has associated parameters such as position, radius, priority, radar elevation, etc. For example, storms occupy a much larger area than rotations and must be scanned at the lowest four radar elevations, but rotations must be scanned at the lowest six. The *utility* of a task from a single radar is the priority of the task multiplied by a factor meant to represent the quality of the data that would result from the scan. The priority of the task is specified by experts in the field (like meteorologists) while quality of the scan represents *how well* a particular portion of the atmosphere is sensed by a given radar configuration. For each task t_i, the utility is defined as:

$$u(t_i) = d(t_i) \times q(t_i) \qquad (1)$$

where $d(t_i)$ is determined by the priority set by the expert user for the weather pattern and $0 \leq d(t_i) \leq 1$; $q(t_i)$ is the function for the quality of scan for t_i and $q(t_i) : t_i \times (s_1, s_2, ..., s_n) \to r \in \Re$, where s_j denotes the scanning strategy of radar j.

Data Correlation is the level of interdependency between MCC agents and is in part based on the overlapping characteristics of potential scanning area of a radar; it is also based on where weather phenomena are occurring and the speed of their movements. The amount of inter-agent communication and coordination is generally proportional to the degree of data correlation.

Weather scenario (WS) is defined as the abstract type of general weather environments that NetRads is experiencing. For example, High Rotation Low Storm (HRLS) is one type of WS where the number of rotations is significantly larger than the number of storms in a series of heartbeats (e.g. lots of rotation phenomena move in followed by a few storm phenomena, and then followed by lots of rotation phenomena). The goal of NetRads is to maximize the sum of the utilities of the tasks scanned in each heartbeat.

2.3 Multiagent Meta-level Control in Netrads

Each MCC agent [10] is implemented with three deliberative-level phases within a heartbeat The phases are: *Data Processing*, *Local Optimization* and *Negotiation*. In prior work [5, 6], we introduced a *Multiagent Meta-Level Control (MMLC)* phase between the Data Processing and the Local Optimization phases. The role of the MMLC phase is to determine the meta-level policy to guide the deliberative-level actions in the *Local Optimization* and *Coordination* phases in order to optimize the data gathering process in the current heartbeat. There are two types of MMLC actions: (1) **Radar Reorganization** that involves potentially transferring the control of radars among different MCCs and (2) **Heartbeat Adaptation** that involves potentially modifying the heartbeat of MCCs. The motivation behind identifying these questions is that it is preferable that radars with large data correlation be allocated to the same MCC to reduce both the amount of communication and the time for coordination among MCCs. Moreover, adjusting the system heartbeat allows MCCs to adapt to changing weather conditions. A shorter heartbeat allows the system to respond more rapidly by closely tracking quickly evolving weather phenomena but with less resolution.

We modeled the MMLC problem as a global optimization problem where offline learning is used to determine the cost-to-go/reward function. This enables agents to adapt in knowledge-poor, partially observable environments. We constructed a restricted class of decentralized Markov-Decision Processes (DEC-MDPs) [2] with factored states with the ability to communicate and model interactions so that decisions made in one agent's meta-level DEC-MDP are coordinated with the meta-level DEC-MDPs of other agents.

Each agent's MMLC state has three features with 18, 19 and 4 possible values respectively, two of these features are vectors that contain the state information about neighbors. In a scenario with 30 NetRads agents with 9 neighbors each, each agent has to reason over approximately $18 \times 19^9 \times 4^9 \approx 1.5 \times 10^{18}$ possible states. The MMLC action is defined as an abstract representation of real action sets. A detailed plan is an instantiation of an MMLC action. For example, a detailed plan of the MMLC "Move less than 10% of the radars from MCC_1 to MCC_2" could be "Move Radar R_1 from MCC_1 to MCC_2". The goal of the work described in this paper is to learn MMLC policies that lead to not only locally appropriate actions but also actions that are likely to be consistent with the actions of other agents.

3. APPROACH

Conflicts among agent actions may occur when distributed MCCs simultaneously apply policies. Such conflicts, if left unresolved, have detrimental effects on the overall performance of Netrads. We define the following types of conflicts among agents' policies: Local Radar Conflicts (LRC) refer to load-balancing conflict situations in which meta-level action choices of MCC agents fail to efficiently balance the load of the multiagent system.; Shared Radar Conflicts (SRC) are constrained resource related conflicts that may arise when two or more agents attempt to move the same radar(s).; Inconsistent Heartbeat Conflicts (IHC) are network parameter related conflicts that occur when two neighboring agents have different heartbeats and have to communicate with each other during the *Negotiation* phase. SRCs generally affect the system performance the most and have the highest priority while the LRCs have the lowest priority to be resolved.

We now provide a high-level description of our approach called *Informed Unroll and Conflict Resolution Learning* (IU-CR-L). It is a decentralized algorithm that uses MMLC policies learned offline in homogenous weather environments to facilitate learning policies in real-time heterogeneous environments where different agents could encounter different weather patterns. Each agent running IU-CR-L specifically learns the following : (a) which states cause conflicts and when do these states need to be augmented with non-local information (based on on-line performance) creating what we call "special" states. (b) the priorities of the heuristics used to determine which part of the search space to expand. (c) policies for "special" states using a multiagent reinforcement learning algorithm (MARL).

In IU-CR-L, agents first learn offline local policies for a variety of weather scenarios based on the simplistic assumption that for each scenario, all agents experience the exact same weather phenomenon. These stochastic policies can cope with the uncertainty of observation and perform better than deterministic policies in partial observable environment. The *Scenario Library Module* of the agent stores the MDPs of each *WS* as well as the policies that are learned off-line. Since the offline agent learning does not capture the exact real-world environmental context of its neighboring agents in the local agent state, the state space is drastically reduced which in turn speeds up learning.

At real-time, each agent expands a small portion of its state space as described in Procedure 1 below. The agent then determines its initial states using observed information about its local feature values and current environment (weather scenario) as well as from communication with its neighbors about data correlation and heartbeat. By initially assuming a heterogenous weather pattern for the neighborhood, the agent chooses the action recommended by the appropriate offline learned policy and deliberates with neighboring agents to determine if the action results in conflicts. A negotiation-based conflict resolution algorithm is harnessed to deliberate and resolve these conflicts.

If during this deliberation there is no conflict with neighboring actions or the performance of conflict resolution is good, then the agent remains in this "normal state" S_i and executes the action prescribed by the offline policy. On the other hand, if conflicts continue to persist after the negotiation-based conflict resolution process, online learning is activated. This involves the agent first adding a "special state" S_i' to the local policy space. S_i contains an additional state vector that captures current non-local information obtained by communication with neighbors. If the special state S_i' has been previously expanded, its best action as prescribed by the previously learned policy is the new action choice. Otherwise, if the special state S_i' has not been expanded earlier, it is expanded as a sibling of the current state S_i. The policy space with this special state S_i' as root is expanded and the policy for this subtree is computed and augmented to the existing policy. To assist the agents in determining the most promising actions for overall performance improvement and expand the states that are most likely to be encountered. We define a set of heuristics (that are also part of conflict resolution negotiation strategy) and equip the agents to learn the priorities of the heuristics online. The above steps are then repeated until the problem horizon is reached. The agents use Policy Gradient Ascent with approximate policy prediction (PGA-APP) [13], a multiagent reinforcement learning algorithm, to learn both the offline policies for the initial state space as well as the online policies for the newly expanded parts of the tree. PGA-APP has mechanisms that allow estimation of the local policy gradient with respect to its neighbor's forecasted strategy without knowing the current strategy and the gradient of the neighbor.

3.1 The IU-CR-L Algorithm

Algorithm 1 Learning Algorithm IU-CR-L for MCC_i

```
1: Initialize empty $mdp$, $initState$, $PCR(\xi)$, $model$ and $\rho(t)$;
2: $openList \leftarrow \{initState\}$;
3: $mdp$ = **init-expand**($openList, model$);
4: Initialize policy $\pi$ as off-line optimal policy;
5: **repeat**
6:   Determine the $WS$ $MCC_i$ encounters;
7:   Communicate and observe the current state $s$;
8:   Consider and deliberate about action $a$ according to $\pi(s,a)$;
9:   $\{\psi^S(\xi), \psi^I(\xi), \psi^L(\xi)\} \leftarrow$ **compute-conflicts** $(\xi)$;
10:  $PCR(\xi) \leftarrow$ **dec-negotiation-alg** $(s, a, \xi)$;
11:  **if** $PCR(\xi) > \rho(t)$ **then**
12:    Execute action $a$;
13:    Update $\pi(s)$ using PGA-APP;
14:  **end if**
15:  **else**
16:    Determine the special state $s'$;
17:    **if** $s'$ is not expanded earlier **then**
18:      Add in $s'$ as a sibling state of $s$ in the $mdp$;
19:    **end if**
20:    Update the current state as $s'$;
21:    $a' \leftarrow$ **apply-heuristic** ();
22:    $a \leftarrow a'$
23:    $mdp$ = **informed-expand** $(s', a, model)$;
24:    Execute action $a$;
25:    Update $\pi(s')$ using PGA-APP;
26:  **end else**
27: **until** the process is terminated.
```

In IU-CRL-L, described in Algorithm 1, agents identify bad states by just "deliberating" about taking the best action at that state and predicting possible conflicts and not by actually "executing" the best action. The algorithm tries to identify a "good" state that reduces the number of conflicts and then executes the best action for that state. $\psi^S(\xi)$, $\psi^I(\xi)$ and $\psi^L(\xi)$ denote the number of SRC, IHC and LRC that exists in the neighborhood ξ respectively. While it may not be possible to deliberate about taking the best action in some domains, it is feasible in Netrads since it is regular practice for an MCC to coordinate with its neighbors and recognize whether or not the action it plans to take will result in a conflict.

init-expand($model$) , discussed in detail below, expands the initial MDP space S^{init} for MCC_i (line 3, Algorithm 1) and identifies the initial state s and corresponding best action a for the agent . It takes as input the *model* parameter consists of the learned policies stored in the Scenario Library and effectively helps determine which action has the highest probability and therefore should be expanded from state s. MCC_i deliberates about the action a to calculate the number of conflicts in its neighborhood ξ (line 7-9, Algorithm 1). **compute-conflicts**(ξ) computes the number of each type of conflicts in ξ. Then MCC_i uses a decentralized negotiation algorithm to resolve conflicts in ξ (line 10, Algorithm 1). $PCR(\xi)$ measures the performance of conflict resolution for neighborhood ξ. **dec-negotiation-alg**(ξ) is the decentralized negotiation algorithm that assigns mediators and uses a branch and bound algorithm [14] to deliberate about the actions to resolve conflicts from a partially global perspective.

If the performance of conflict resolution is good (denoted by $PCR(\xi) > \rho(t)$), MCC_i executes a and updates the policy $\pi(s)$ and the MDP space of MCC_i remains the same (line 11-14, Algorithm 1). [1] Otherwise, MCC_i searches for the special state s' among the siblings of s. To create s', an additional state vector that contains non-local context about the neighbors is added to state s. If s' has not been expanded earlier, we expand it as a sibling of s (line 17-19, Algorithm 1). The MDP space for MCC_i with s' as root is expanded by interleaving action expansion and negotiation (line 20-23, Algorithm 1).

apply-heuristic() is the process that uses heuristics to select an action for expansion. It applies the appropriate heuristic to select another action for conflict resolution. **informed-expand**$(s, a, model)$, also discussed below, is the procedure that does selective MDP expansion from state s and negotiation about action a to improve global performance. The procedure works as follows: MCC_i updates its action choice a by applying heuristics in sequence that are sorted from highest to lowest priority until the performance of conflict resolution is acceptable or time has run out on this cycle. The priority of each heuristic H_j reflects the effectiveness of H_j on conflict resolution in a specific environmental context and is learned implicitly using multiagent reinforcement learning. MCC_i expands action a and its subsequent search space if it has not been expanded earlier. At the end of the procedure, MCC_i executes action a and updates the policy $\pi(s')$ (line 24-25, Algorithm 1). PGA-APP learns the transition function for the *mdp* and computes the policy.

3.2 Initial MDP space

init-expand$(model)$, the procedure in IU-CR-L that expands the initial MDP space S^{init} for MCC_i is described in Procedure 1. Each agent obtains its S^{init} before online learning begins. It then uses the learned library of policies to determine the states and actions which are initially expanded, S^{init} is obtained as follows: MCC_i identifies its initial state s_0 based on its local state feature values and also the appropriate MDP state space and offline learned policy π (stored in the Scenario library) for the current weather scenario being encountered. MCC_i then expands the action $a = \pi(s)$ for state s_0 that has the highest probability

[1] $\rho(t)$ is a system defined threshold for conflict resolution performance that is a linear function of time t. It cannot be manipulated by agents. Overall system performance will be affected by the assigned value of $\rho(t)$.

Procedure 1 mdp = **init-expand**$(openList, model)$

```
1: repeat
2:   state ← dequeue (openList);
3:   action ← highest-prob-action (state, model);
4:   succs ← suc-states (state, action, model);
5:   for all succ ∈ succs do
6:     mdp ← update (state, action, succ, mdp);
7:     if succ is not a terminal state then
8:       if succ is explored by off-line learning then
9:         enqueue (succ, openList);
10:      end if
11:    end if
12:  end for
13: until openList is empty;
14: return mdp;
```

distribution in the policy . Every possible state s' resulting from this action is expanded and for s' similarly only the action that has the highest probability distribution in $\pi(s')$ is expanded. This process repeats until terminal states are reached.

This greedy expansion procedure guides the agent on what action to execute (though it may be suboptimal) for each state reached during execution. We set the depth of S^{init} to be 3. This is due to the fact that the horizon of the policies for the Netrads is three heartbeat periods. We defined this horizon manually after examining the behavior of the Netrads application in various scenarios. If the horizon is too short, it triggers MMLC too frequently which increases the cost of decision making and affects performance. On the other hand, a horizon that is too long may result in MMLC policies that are obsolete for the latter part of the horizon, given the dynamic nature of the environment.

3.3 Informed State Expansion

Procedure 2 mdp = **informed-expand**$(s, a, model)$

```
1: while termination condition is not met do
2:   PCR(ξ) ← dec-negotiation-alg (ξ);
3:   if PCR(ξ) > ρ(t) then
4:     if a is not expanded in the MDP space then
5:       mdp = partial-expand(s', a, model);
6:     end if
7:     update-heuristic-priority ();
8:   end if
9:   else
10:    a' ← apply-heuristic ();
11:    a ← a';
12:  end else
13: end while
```

Procedure 2 is the process that selectively expands the MDP space based on the performance of iterative conflict resolution when MCCs reach special states (line 23, Algorithm 1). **partial-expand**$(s, a, model)$ expands action a belonging to state s and the subsequent search space in a manner similar to **init-expand**$(s, a, model)$. When Procedure 2 starts, each agent checks the *termination condition* (line 1, Procedure 2) to decide whether the *MMLC* phase needs to be terminated or not. We set the following rules as the *termination condition* for the *MMLC* phase: 1) No conflict exists among the meta-level actions of each MCC; 2) The conflict resolution performance is good ($PCR(\xi) > \rho(t)$); and 3) The time limit for the *MMLC* phase is reached (this is 10% of deliberation time described as an assumption in Section 1).

We have formulated heuristics to expand appropriate actions (line 10, Procedure 2) and use the online learning to dynamically adjust the most promising direction that improves the overall performance. As discussed in Section 2.3, there are two types of MMLC actions in Netrads: **Radar Reorganization** and **Heartbeat Adaptation** The *Radar Reorganization* action is more complicated than the *Heartbeat Adaptation* action. It includes detailed actions that require deliberation of the specific type of radar movement as well as the direction of radar movement. 'Heavy Move' and 'Light Move' are two different types of radar movement. The former moves a large amount of radars to neighbors while the later moves few to decrease the load of the MCC. For example, one action for *Radar Reorganization* could be 'Heavy Move(MCC_1 to MCC_2)' which means MCC_1 applies a 'Heavy Move' and the radars are moved to MCC_2 (the direction of radar movement).. The *Heartbeat Adaptation* action has two action choices: 'Use 30 seconds heartbeat' and 'Use 60 seconds heartbeat'. Each heuristic is a preferred conflict resolution mechanism for a particular situation. The agents learn the priority of each heuristic online and use the heuristics to decide which action to take when a special state is encountered. We define three elementary heuristics that facilitate state space expansion by exploring a new action that has:

- H_1: a different **type** of radar movement from the current action choice. This heuristic helps in situations where the data correlation existing among overlapping areas is high.
- H_2: a different **direction** of radar movement from the current action choice. This heuristic helps in situations where the agent was planning to move its radars to already heavily loaded neighbors.
- H_3: a different **heartbeat** from the current heartbeat choice. This heuristic helps in situations where the agent's heartbeat differs from those of most of its neighbors.

H_4, H_5 and H_6 are the heuristics that enforce two of the above three elementary heuristics. For example, H_4 is the heuristic that the agent expands its MDP space by exploring a new action that has a different type as well as a different direction of radar movement from the current action choice; H_7 enforces H_1, H_2 and H_3. We use heuristic H_8 to denote that an agent does not change its current action.

Initially, there is no prior knowledge about which heuristic works best for MCC_i in a specific special state. For MCC_i, we define $\chi_{i,j}$ ($j = 1, 2, ..., 8$) as the priority that measures the effectiveness for applying heuristic H_j for MCC_i. **update-heuristic-priority**() updates the priority for each heuristic based on the actual performance on conflict resolution (line 7, Procedure 2):

$$\chi_{i,j} \leftarrow \frac{\chi_{i,j} \times N_{i,j}^{sum} + PCR(\xi)}{N_{i,j}^{sum} + 1} \quad (2)$$

where $N_{i,j}^{sum}$ is the total number of H_j that has been applied for MCC_i up to now.

4. EVALUATION

For the experiments reported here, we use a simulation environment of Netrads developed in the Farm simulator framework [10]. We evaluate the IU-CR-L algorithm on scenarios with 12 agents controlling 72 radars altogether; each scenario contains heterogenous *WSs* in different parts of the system based on the distribution of the tasks. A task represents a weather event and we are only concerned about *rotation* and *storm* tasks in the evaluation. The number of tasks varies from 80 to 200 for each scenario. There are nine types of possible *WSs* occurring in the system, and they are differentiated by the number of these two tasks. Each MCC has two choices of heartbeat: 30 seconds long or 60 seconds long.

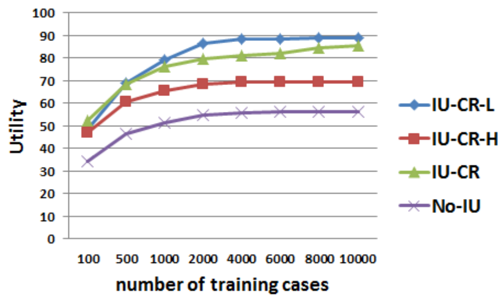

Figure 3: *Utility* of the four algorithms for 12 MCCs, for various number of training episodes.

We generate the training/test episodes by varying the following parameters: number and types of tasks; initial heartbeat for each MCC; the percentage of pinpointing tasks relative to all tasks (*PTaskRatio*). Pinpointing tasks generally require more agent coordination and often lead to higher total utilities. *Utility* measures the overall utility of a given configuration of radars for task scanning in a heartbeat. In the experiments, $\rho(t) = -0.2 \cdot t + 0.8$.

We compare the results of four approaches: *IU-CR-L*, *IU-CR-H*, *IU-CR* and *No-IU*. *IU-CR-L* is the learning approach discussed in Section 3 that iteratively expands the MDP search space, uses a decentralized negotiation algorithm to resolve conflicts, updates the priorities of heuristics and uses PGA-APP to update the policy. *IU-CR-H* uses the eight heuristics described in the previous section to expand the search space and learns their priorities online based on the conflict resolution performance . It does not use PGA-APP to update the policy. *IU-CR* is similar to *IU-CR-L* with the only difference being that it expands all the action choices instead of using heuristics to selectively expand the promising action choice. *No-IU* is the PGA-APP-based online learning approach that applies scenario appropriate policies learned offline and does no conflict resolution. It does not do any learning of special states but does continue to update its policies based on new experiences.

The comparison of the four approaches is to show (a) the effectiveness of using policies learned in simplified environments where possible; and (b) coordinating with other agents only when necessary via the smart expansion of the agent's state space. We empirically show that IU-CR-L learns useful policies for agents with a small amount of training episodes. Also, IU-CR-L achieves significantly better performance on utility and conflict resolution by expanding a small fraction (only 10% in the best cases) of the whole search space.

The Scenario library was populated with policies learned offline for a number of weather scenarios. These offline policies were based on 1000 training cases for each scenario. We then used 10000 training cases to learn the policies for the three online algorithms. We compared the performance of the resulting learned policies for different amounts of training. The results reported are the average values of 30 test episodes. In Figure 3, we observe that *IU-CR*, *IU-CR-H* and *IU-CR-L* perform significantly better than *No-IU* on *Utility* for all the training episodes. Conflict resolution helps to improve the overall performance on *Utility*.

We also observe that *IU-CR* performs better than *IU-CR-L* on *Utility* when 100 training episodes are used for online learning; this is because the heuristics with inaccurate priorities early in the learning stage may lead to incorrectly biased expansion of actions. As the number of training cases increases, the state space of each agent using *IU-CR-L* is more accurate about the neighbors' environmental states; and the actions that resolve conflicts have been frequently explored and executed. These two factors lead to improved performance of *IU-CR-L* with increased training. The results show that online PGA-APP is advantageous and it helps each agent to learn the scenario appropriate policy that improves the overall performance significantly.

Figure 4 shows that *IU-CR* and *IU-CR-L* achieve similar conflict resolution performance after 10000 training episodes. *IU-CR* reduces the number of unresolved conflicts by 78%, 76% and 68% for the 20%, 60% and 90% *PTaskRatio* cases respectively compared with *No-IU*; *IU-CR-H* reduces the number of unresolved conflicts by 53%, 56% and 54% while *IU-CR-L* reduces the number of unresolved conflicts by 75%, 71% and 74% for the three PTaskRatio cases respectively. *IU-CR-H* does not do as well in reducing conflicts because it does not have the advantage of using PGA-APP to update policies.

Table 1: Comparison results between *IU-CR* and *IU-CR-L* after 10000 training episodes.

Approach	# of states expanded	# of special states expanded	Expansion time (sec.)
IU-CR	382765303	1547	1.56
IU-CR-H	17432685	1084	0.084
IU-CR-L	12097539	862	0.084

Table 1 shows that for *IU-CR*, *IU-CR-H* and *IU-CR-L*, the special states added to the search space are limited. *IU-CR-H* and *IU-CR-L* expand significantly fewer states than *IU-CR* (95.5% and 96.8% less states respectively). Among these expanded states, only a small fraction ($< 0.01\%$) are visited during learning. Each learned policy associated with an encountered special state is iterated on average 437 times for 10000 training episodes.

In *IU-CR-L*, the heuristics help to balance the benefits of expanding more states and being selective in the direction of expansion. Although *IU-CR-L* learns fewer new special states compared to *IU-CR* and *IU-CR-H*, its policies perform better on *Utility* (see Figure 3). *IU-CR-L* suggests a new action for a special state 31.8% of the time; while *IU-CR-H* does so 13.5% of the time. In *IU-CR-L*, PGA-APP contributes to the learning of the expanded actions and thus leads to the scenario appropriate policies. *IU-CR* spends significantly more time (1.56 *secs* compared to 0.084 *secs*) on expansion of special states compared to *IU-CR-H* and *IU-CR-L*. This is because when *IU-CR* reaches a special state, it expands all the possible actions and expands the resulting search space of the actions which is time consuming.

Table 2: Comparison of effectiveness *IU-CR-L*'s state expansion with the increase of training episodes.

# of training episodes	# of states expanded	# of special states expanded	Average # of actions expanded at special states
100	790538	46	2.3
500	3012334	110	5.6
1000	3587659	137	5.1
4000	9347893	594	4.6
10000	12097539	862	3.4

Table 2 shows that the percentage of new states expanded using *IU-CR-L* is decreasing substantially with the increase of training episodes. During the earlier learning stage (from 100 to 500 training episodes), many new special states are encountered that results in a faster rate of growth of the search space. As learning progresses, the probability of expanding new special states decreases. *IU-CR-L* expands 281% and 19.1% more states when number of training episodes increases to 500 and 1000 respectively.

Since the dominant heuristics are more likely to be chosen in the earlier stages of learning; the average number of actions expanded at special states is low. When more training episodes are encountered, the number of heuristics that are applied increases because of the uncertainty of conflict resolution. So the average number of actions expanded also increases (5.6 compared to 2.3 in Table 2). As significant number of training episodes are encountered, the priority of each heuristic becomes more and more accurate. Consequently, the special states expanded later in the online learning apply the few dominant heuristics more often. For this reason, the average number of actions expanded at special states decreases (5.1, 4.6 and 3.4 respectively compared with 5.6 in Table 2). This shows that *IU-CR-L* learns the effectiveness of applying each heuristic.

5. CONCLUSIONS AND FUTURE WORK

In this work, we present a decentralized learning algorithm that harnesses off-line policies that are learned within the context of a simplified environment and selectively expands the search space acquiring on-line nonlocal state information in case of conflicts. When conflicts resulting from multiple neighboring agents applying their local policies are observed, agents switch to "special" states that augment local policy states with additional non-local state information and learn other actions to take in this specific situation. The

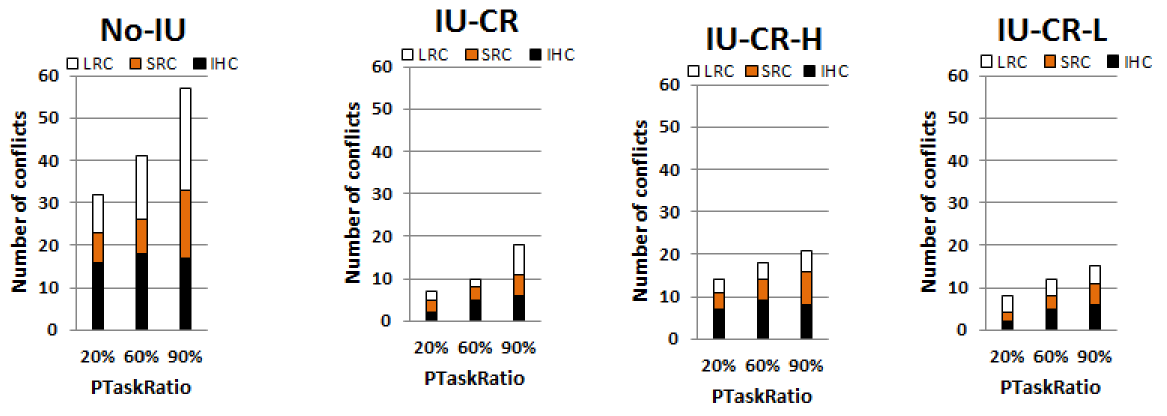

Figure 4: Number of conflicts (LRC, SRC and IHC) unresolved by the three algorithms for 12 MCCs, for *PTaskRatio* to be 20%, 60% and 90%.

agents are equipped with capabilities to learn on-line: (a) the trigger conditions for "special" states (b) priorities of heuristics and (c) policies for "special" states. The work contributes towards understanding the differences between tabula rasa learning and informed learning in rich knowledge based systems. Experimental results show that our approach achieves good performance on utility and conflict resolution by expanding only a small fraction of the whole search space.

6. ACKNOWLEDGEMENTS

We thank the reviewers for their helpful comments. This work is supported partially by the National Science Foundation (NSF) under Agreement No.IIS-1018067, No. IIS-1116078 and No. IIS-0964590. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect those of the National Science Foundation.

7. REFERENCES

[1] G. Alexander, A. Raja, and D. Musliner. Controlling deliberation in a markov decision process-based agent. In *Proc. of AAMAS*, pages 461–468, 2008.

[2] D. S. Bernstein, R. Givan, N. Immerman, and S. Zilberstein. The complexity of decentralized control of markov decision processes. *Math. Oper. Res.*, 27(4):819–840, 2002.

[3] S. Cheng. *Coordinating Decentralized Learning and Conflict Resolution Across agent Boundaries.* PhD thesis, University of North Carolina at Charlotte, 2012.

[4] S. Cheng, A. Raja, and V. Lesser. Multiagent Meta-level Control for a Network of Weather Radars. In *Proc. of International Conference on Intelligent Agent Technology (IAT)*, pages 157–164, 2010.

[5] S. Cheng, A. Raja, and V. Lesser. Multiagent meta-level control for radar coordination. *To appear in (WIAS) Web Intelligence and Agent Systems: An International Journal*, 2013.

[6] S. Cheng, A. Raja, and V. R. Lesser. Towards multiagent meta-level control. In *Proc. of AAAI*, pages 1925–1926, 2010.

[7] M. Ghavamzadeh and S. Mahadevan. Learning to communicate and act using hierarchical reinforcement learning. In *Proc. of AAMAS*, pages 1114–1121, 2004.

[8] C. Guestrin, D. Koller, and R. Parr. Multiagent planning with factored mdps. In *Proc. of NIPS*, pages 1523–1530, 2001.

[9] J. R. Kok and N. A. Vlassis. Collaborative multiagent reinforcement learning by payoff propagation. *Journal of Machine Learning Research*, 7:1789–1828, 2006.

[10] M. Krainin, B. An, and V. Lesser. An Application of Automated Negotiation to Distributed Task Allocation. In *Proc. of International Conference on Intelligent Agent Technology (IAT 2007)*, pages 138–145, November 2007.

[11] F. S. Melo and M. Veloso. Learning of coordination: exploiting sparse interactions in multiagent systems. In *Proceedings of AAMAS*, AAMAS '09, pages 773–780, Richland, SC, 2009.

[12] J. Wu and E. H. Durfee. Solving large TAEMS problems efficiently by selective exploration and decomposition. In *Proc. of AAMAS*, pages 291–298, 2007.

[13] C. Zhang and V. Lesser. Multi-Agent Learning with Policy Prediction. In *Proc, of AAAI*, pages 927–934, 2010.

[14] W. Zhang, G. Wang, Z. Xing, and L. Wittenburg. Distributed stochastic search and distributed breakout: properties, comparison and applications to constraint optimization problems in sensor networks. *Artif. Intell.*, 161(1-2):55–87, 2005.

[15] M. Zink, D. Westbrook, S. Abdallah, B. Horling, E. Lyons, V. Lakamraju, V. Manfredi, J. Kurose, and K. Hondl. Meteorological Command and Control: An End-to-end Architecture for a Hazardous Weather Detection Sensor Network. In *Proc. of the ACM Workshop on End-to-End, Sense-and-Respond Systems, Applications, and Services (EESR 05)*, pages 37–42, 2005.