Multiagent Meta-level Control for Radar Coordination

Shanjun Cheng [a], Anita Raja [a], Victor Lesser [b]
[a] *Department of Software and Information Systems, The University of North Carolina at Charlotte*
Charlotte, NC, 28223
E-mail: {scheng6,anraja}@uncc.edu
[b] *Department of Computer Science, University of Massachusetts Amherst*
Amherst, MA, 01003
E-mail: {lesser}@cs.umass.edu

Abstract. It is crucial for embedded systems to adapt to the dynamics of open environments. This adaptation process becomes especially challenging in the context of multiagent systems. In this paper, we argue that multiagent meta-level control is an effective way to determine when this adaptation process should be done and how much effort should be invested in adaptation as opposed to continuing with the current action plan. We use a reinforcement learning based local optimization algorithm within each agent to learn multiagent meta-level control agent policies in a decentralized fashion. These policies will allow each agent to adapt to changes in environmental conditions while reorganizing the underlying multiagent network when needed. We then augment the agent with a heuristic rule-based algorithm that uses information provided by the reinforcement learning algorithm in order to resolve conflicts among agent policies from a local perspective at both learning and execution stages. We evaluate this mechanism in the context of a multiagent tornado tracking application called NetRads. Empirical results show that adaptive multiagent meta-level control significantly improves the performance of the tornado tracking network for a variety of weather scenarios.

Keywords: Multiagent Systems, Meta-level Control, DEC-MDPs, Multiagent Reinforcement Learning, Conflict Resolution

1. Introduction

Cooperative multiagent systems (MAS) are finding applications in a wide variety of domains, including sensor networks, robotics, collaborative decision making systems and distributed control. A cooperative MAS consists of a group of autonomous agents that interact with one another in order to optimize a global performance measure. These agents operate in an iterative three-step closed loop [35]: receiving sensory data from the environment, performing internal computations on the data, and responding by performing actions that affect the environment either using effectors or via communication with other agents. Two levels of control are associated with this loop: deliberative and meta-level control [19]. Each agent has a lower control level called deliberative control (or object level, see Fig. 1.), which involves the agent making decisions about what domain-level problem solving to perform in the current context and how to coordinate with other agents to complete tasks requiring joint effort. Each agent also has a higher control level that is **meta-level control** (see Fig. 1.), which involves the agent making decisions about deliberation control itself including whether to deliberate, how many resources to dedicate to this deliberation, and what specific deliberative control to perform in the current context. In the context of MAS, the meta-level component of each agent should have a multiagent policy that coordinates its deliberation with other agents to account for what could happen as deliberation (and execution) plays out. Fig. 2. describes the interaction among the meta-level control components of multiple agents.

Meta-level control in complex agent-based settings was explored in previous work [2] [3] [33] [34] [41]

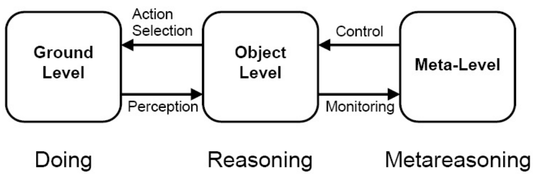


Fig. 1. Duality in reasoning and acting [19].

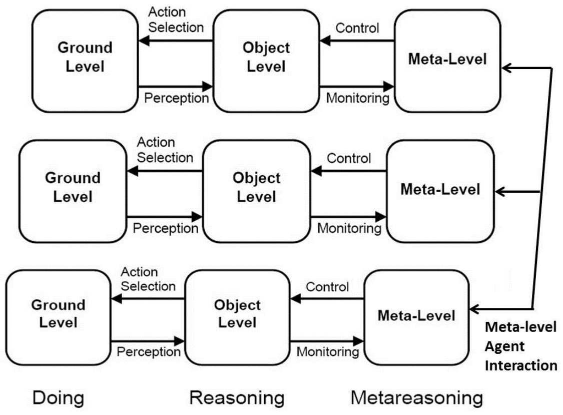


Fig. 2. Meta-level reasoning among multiple agents [19].

where a sophisticated architecture that could reason about alternative methods for computation was developed. We build on the earlier work and open a new vein of inquiry by addressing issues of scalability, partial information, and complex interactions across agent boundaries. Consider for instance a scenario where two agents A_1 and A_2 are negotiating about when A_1 can complete task T_1 that enables A_2's task T_2. This negotiation involves an iterative process of proposals and counter-proposals where at each stage A_2 generates a commitment request to A_1, A_1 performs local optimization computations (scheduling) to evaluate commitment requests; this process repeats until A_1 and A_2 arrive at a mutually acceptable commitment. The multiagent meta-level control decision would be to ensure that A_1 completes its local optimization in an acceptable amount of time so that A_2 can choose alternate methods in case the commitment is not possible. In setting up a negotiation, the meta-level control should establish when negotiation results will be available. This involves defining important parameters of the negotiation including the maximum number of cycles of proposal/counter-proposal, which in part involves trading off time allocated to local optimization versus the number of cycles of negotiation [33]. We model **multiagent meta-level control** (*MMLC*) as the process that facilitates agents to have a decentralized meta-level multiagent policy, where the progression of what deliberations the agents should do, and when, is choreographed carefully and includes branches to account for what could happen as deliberation plays out.

The key contributions of this work are:

1. A multiagent meta-level control approach that approximates decentralized partially observable Markov decision process (DEC-POMDP) with a stochastic decentralized Markov Decision Processes (DEC-MDPs) at the meta-level so that it can efficiently support agent interactions and reorganize the underlying network when needed.
2. Coordinate DEC-MDPs that use multiagent reinforcement learning to learn multiagent meta-level polices in a decentralized fashion and incorporate heuristic rules to resolve conflicts among agent polices locally at both learning and execution stages.
3. Efficiently decrease the exploration costs of DEC-MDPs by constructing abstract classes of scenarios/ states/actions where instances within a class have similar features. The use of meta-level actions that abstract real actions at the meta-level is especially novel.
4. Leverage the significance of shared tasks in our domain and the use of a factored reward function for the reinforcement learning to capture value of tasks from a partially global perspective instead of a local perspective.
5. Evaluation results show that our approach significantly improves the performance in the context of a multiagent tornado tracking application.

1.1. The NetRads Application Domain

In this work, we will focus on NetRads, a real-world application that will need meta-level control. NetRads [26] [46] 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. It has a dynamically evolving environment (different types of weather phenomena are occurring unexpectedly) and agents have only limited partial views of the whole system. The NetRads radar is designed to quickly detect low-lying meteorological phenomena such as tornadoes. 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*. The MCC agent is implemented with 3 deliberative-level phases in a heartbeat. The phases are: *Data Processing*, *Local Optimization* and *Negotiation*. Fig. 3. is one heartbeat with these phases. In *Data Processing*, each MCC gathers moment data from the radars and runs detection algorithms on the weather data. The results of this analysis lead to a set of weather-scanning tasks of interest for the next radar scanning cycle. In *Local Optimization*, the MCC determines the best set of scans for the radars associated with it to maximize the sum of the utilities[1] associated with its selected weather scanning tasks. In *Negotiation*, the MCC communicates with its neighboring MCCs to modify their local optimization based on the need for radars from multiple MCCs to be coordinated to accomplish some joint tasks and to avoid redundant scanning of the same area. The goal of NetRads is to maximize the overall utility of a given configuration of radars (refer to [27] for more details).

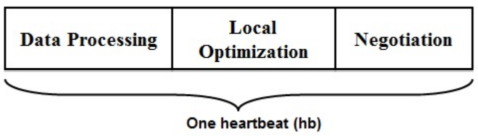


Fig. 3. Heartbeat with three deliberative-level phases.

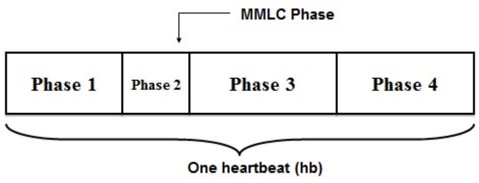


Fig. 4. Adding MMLC phase in the heartbeat.

In NetRads, an example domain action would be a radar scan of a weather task (more details will be discussed in Section 3.1). The deliberative action at each heartbeat would be the MCC spending some initial time in processing the radar data obtained during the last heartbeat, then performing a local optimization to determine the radar scanning strategies of the radars under its control, followed by negotiation rounds of alternating communication and recomputation of the local configuration. We augment the MCC agent with a new meta-level phase called *MMLC* to address the following problems:

P.1 How to re-organize the sub-nets of radars under each MCC to minimize the time required for MCC agents to negotiate with their neighboring agents?

P.2 How to adjust the system heartbeat to adapt to changing weather conditions so as to approximately balance system responses to emerging weather phenomenon and the accuracy of radar scanning strategies?

Each heartbeat is now split up into four phases (as Fig. 4. shown) containing both deliberative-level actions (Phase 1: *Data Processing*, Phase 3: *Local Optimization* and Phase 4: *Negotiation* are exactly the same as in [26]) and we insert a new phase, Phase 2: *MMLC*, which involves a meta-level decision process that is coordinated with neighboring MCCs.

In this work, we define two types of meta-level actions (more details will be discussed in Section 3.1.3) to address problems **P.1** and **P.2** defined above: (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 *MMLC* phase implements meta-level actions that handle the coordination of MCC agents and guide the deliberative-level actions in *Local Optimization* and *Negotiation*. The *MMLC* phase in each MCC is restricted to a constrained time period ($\leq 10\%$ of the whole heartbeat) to ensure enough time for the *Local Optimization* and *Negotiation* phases.

1.2. Summary of Our Approach

We view the MMLC problem as a decentralized coordination problem and describe a multiagent meta-level control approach that uses decentralized stochastic Markov decision processes (DEC-MDPs) [7] to implement the meta-level control of each agent. In NetRads, the agents are assumed to have only partial observability of the environment and thus additional uncertainty is introduced into the decision process. For that reason we use a stochastic [9] [32] approach where a policy specifies a probability distribution over potential action rather than one action to take.

It would be appropriate to model the MMLC problem in Netrads as a decentralized partially observable

[1]We define *utility* as a function of priority of tasks and quality of data. More details are provided in Section 3.1.1.

Markov decision process (DEC-POMDP) [7]. DEC-POMDPs lie in the NEXP-Complete complexity class. To make the problem tractable, we approximate the DEC-POMDP in NetRads with a stochastic[2] DEC-MDP model; a DEC-MDP is a DEC-POMDP with joint full observability [8]. The joint full observability of the DEC-MDP reduces the interagent communication significantly. However, it is known that solving a DEC-MDP is also a NEXP-Complete problem since it uses a global reward function that maximizes the overall reward in the system. We address this issue by approximating the solution to the DEC-MDP by using a factored reward function to define the Nash Equilibrium [38] instead of the global reward function. More details are provided in Section 3.2.

Multiagent reinforcement learning (MARL) is an attractive approach for agents to obtain effective coordination policies without explicitly building the complete decision models. We use a MARL [39] algorithm, *Policy Gradient Ascent with approximate policy prediction (PGA-APP)* [44] to learn stochastic policies for the meta-level DEC-MDPs belonging to individual agents offline. We do this offline learning in a very controlled way as discussed in Section 3.3. It should be noted that in [44], the PGA-APP algorithm is used to learn the stochastic policies at the deliberative level and in our work we use it at the meta level.

During learning, we decrease the complexity of the real state by using abstract features in the meta-level state (described in Section 3.1). We also use meta-level action (described in Section 3.1) that is abstracted for computational reasons. We categorize the real-world weather scenarios into different classes by considering how they influence the system performance and learn policies separately for each class through controlled experimentation to speed up learning. The learning of each agent uses both information that is locally observed and received from local interactions with its neighboring agents to improve performance.

During execution, the *MMLC* is triggered at each heartbeat. This is reasonable since the computation and communication cost of *MMLC* is acceptable in relation to the expected utility gained. The meta-level policies learned for each agent could potentially be optimal policies from a local perspective. However, applying the meta-level policies may cause conflicts (discussed in Section 4.1) between neighboring agents at both the learning and execution stages. For instance, a conflict occurs when both MCC agents want control of the same radar since a radar only can be controlled by one agent. Such conflicts if left unresolved have detrimental influences on the overall performance. In this work, we resolve the conflicts through a heuristic rule-based approach (discussed in Section 4.2) that uses pre-defined rules based on character of the conflict. We empirically show that this type of distributed meta-level control gives a performance advantage over an approach that resolves conflicts in a much more ad-hoc manner.

The rest of the paper is organized as follows: We investigate the MMLC problem in the context of the Netrads tornado tracking application. We use an example to identify and explain the meta-level control problems that need to be addressed in this application and motivate desired solutions to the problems. In Section 3, we formalize the MMLC problem and describe the DEC-MDP and multiagent reinforcement learning algorithm we use to determine local meta-level policies. In Section 4, we explore the conflicts that could occur among the decentralized local agent policies and describe how we augment the agent with algorithms to coordinate the meta-level control decisions when conflicts occur. In Section 5, we present our empirical evaluation and discussion of our results. Section 6 is a discussion of related work and Section 7 is a discussion of our conclusions and future work.

2. Motivating Example

At the highest level, the question we plan to address in NetRads is the following: *"How does the meta-level control component of each MCC agent learn decentralized policies so that it can efficiently support agent interactions with other MCC agents and reorganize the underlying network when needed?"* Specifically in NetRads, this involves addressing the following problems:

P.1 What triggers a radar to be handed off to another MCC and how do we determine which MCC to hand off the radar to?

P.2 How to assign different heartbeats to sub networks of MCCs in order to adapt to changing weather conditions?

The intuition behind identifying these specific meta-level problems is that it is preferable that radars with large *data correlation* be allocated to the same MC-

[2]We learn stochastic policies since they can cope with the uncertainty of observations to a certain degree and perform better than deterministic policies in partially observable environments.

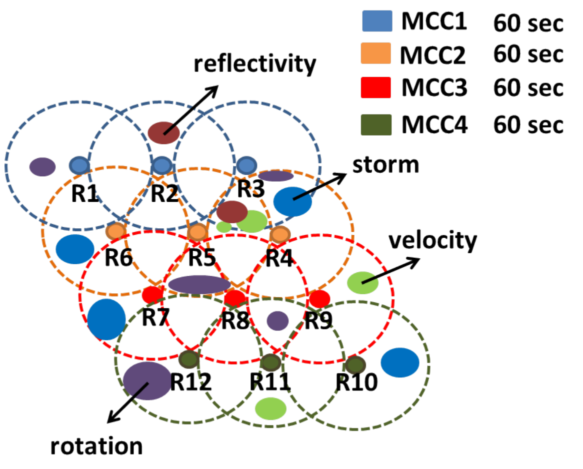


Fig. 5. Example MCC-Radar configuration in NetRads (each MCC controls three radars).

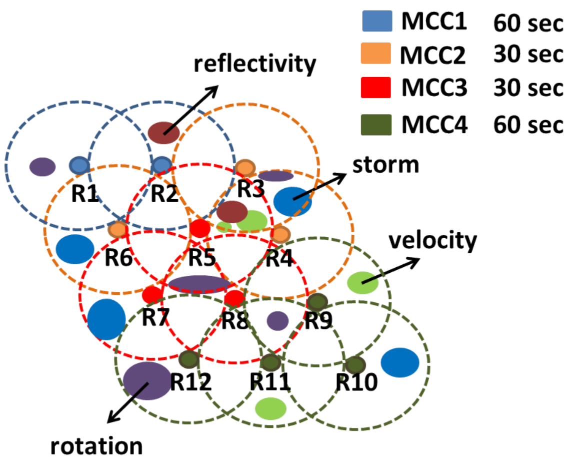


Fig. 7. Example Heartbeat adaptation.

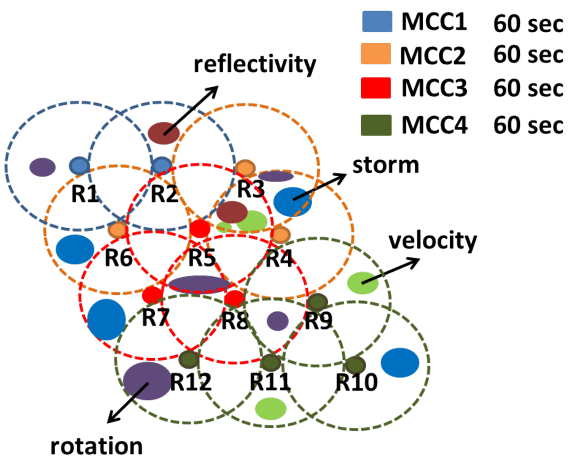


Fig. 6. Example Radar reorganization.

C. *Data Correlation* occurs when radars belonging to different MCCs share data of the same weather phenomenon. The data correlation 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. Allocating such data-correlated radars to the same M-CC potentially reduces the amount of communication and the time for negotiation among MCCs. Moreover, adjusting the system heartbeat allows MCCs to adapt to changing weather conditions. For example, if many scanning tasks occur in a certain region, meta-control may decide to use a shorter heartbeat to allow the system to respond more rapidly to closely track the quickly evolving weather phenomena which is especially important with tornados. In this work, a single heartbeat of MCC is set[3] to be 30 seconds (shorter) or 60 seconds (longer). This decision would also involve reorganizing the MCC neighborhoods so that there are clusters of MCCs with each cluster having a different heartbeat depending on the type and frequency of tasks that the cluster has to handle.

Fig. 5. is the example Netrads configuration, where each MCC controls three radars (e.g., MCC_2 controls radars $\{R_4, R_5, R_6\}$). Each radar has a scanning area represented by a circle and may have overlapping s-canning areas with other radars. Two MCCs are *neighbors* if the radars share overlapping scanning areas (In Fig. 5., MCC_1 and MCC_2 are neighbors while MCC_1 and MCC_3 are not). Tasks are defined as four types of weather phenomena (distinguished by d-ifferent colors in Fig. 5.): storm, rotation, reflectivity and velocity. We observe that R_3 currently has a large data correlation with R_4 and R_5, then reallocating it from MCC_1 to MCC_2 improves the performance. Suppose MCC_1, MCC_2 and MCC_3 execute the following solution on radar reorganization: " Move R_3 to MCC_2", "Move R_5 to MCC_3" and "Move R_9 to MCC_4" . Fig. 6. is the resulting NetRads configuration. In Fig. 6., we observe that many scanning tasks are occurring currently in the common boundary between MCC_2 and MCC_3, it is preferable for these two MCCs to use a shorter heartbeat (30 sec-

[3]There could be potentially many (more than two) heartbeat options and none of the mechanisms used in the paper are restricted to only two heartbeat options.

onds) to respond rapidly to the changing environments. MCC_2 and MCC_3 execute the solution on heartbeat adaptation: " Use the 30 second heartbeat". Fig. 7. is the resulting NetRads configuration. By making these meta-level changes in radar reorganization and heartbeat adaptation, NetRads saves on the communication cost and the time spent on negotiation among MCC-s. These changes also enhance the system's ability to quickly track dynamic weather phenomena.

3. Solution Approach

In this Section, we describe the formal framework we have developed for MMLC within the context of the NetRads tornado tracking application. We first define some key concepts that lay the foundations for the role of state and action abstraction in this work; we then formulate the control flow in the MMLC component for each agent. We then formalize the meta-level control problem handled by MMLC as a DEC-MDP and use a decentralized reinforcement learning algorithm to learn local optimal policies.

3.1. Formal Framework for MMLC

Like [33], we define the real state of the agent as the state that has the detailed information related to the agent's decision making and execution. It accounts for every task which has to be reasoned about by the agent; the execution characteristics of each of these tasks; and information about the environment such as types of tasks (defined later in this Section) arriving at the agent and frequency of arrival of tasks. Consequently the real agent state is continuous and complex. This leads to a combinatorial explosion in the real state space for meta-level control even for simple scenarios. The complexity of the real state can be handled by defining an abstract representation of the state that captures the important qualitative state information relevant to the meta-level control decision making process. We call this abstraction of the real state the **meta-level state**. The real meta-level action set of an agent is defined as the complete set of actions that each MCC agent needs to execute in order for instance to reorganize the control of radars among MCC agents. However, for the **meta-level actions** of radar reorganization, we use an abstract representation of the real action sets that capture the similar qualitative action information relevant to the meta-level control decision making process. We use an abstract representation because the action space for radar reorganization is very large if actions use, for example, "Move Radars $\{l_1, ...l_k\}$ from MCC_i to MCC_j" where $\{l_1, ...l_k\}$ is a subset of all the radars controlled by MCC_i with the length of k. The use of such a detailed action space increases exponentially the time to learn effective MDP policies. More details about how we construct the meta-level states and actions will be discussed later in Sections 3.1.2 and 3.1.3.

3.1.1. Key Terms

Prior to describing the meta-level states/actions, we define several key terms used in the rest of this paper:

Task: In NetRads, each radar scanning *task* in the system has a position, a velocity, a radius, a priority, a preferred scanning mode, and a type. *Tasks* may be one of a few different types: *storm*, *rotation*, *reflectivity* or *velocity*. Each of these types has its own distributions for the characteristics described above. Tasks may be either *pinpointing* or *non-pinpointing* (described below). The *utility* of a task is determined by the priority of the task and a factor meant to represent the quality of the data that would result from the scan (the priority is specified by experts in the field e.g. meteorologists). For each task t_i, the utility is defined as:

$$u(t_i) = d(t_i) \times q(t_i) \tag{1}$$

where $d(t_i)$ is determined by the priority of the requesting user or 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.

Pinpointing and Non-Pinpointing Task: *Pinpointing* tasks are those that contribute to a significant utility gain by scanning the associated volume of space with multiple radars belonging to the same or different MCCs at once. The utility gained from scanning a pinpointing task increases with the number of radars scanning the task up to a point; whereas, the utility for a *non-Pinpointing* task is the maximum of the utilities from the individual radars.

Degree of Data Correlation: *Degree of data correlation* captures how much data correlation MCC_i has with its neighbor(s). It is defined as $\langle D_1, D_2, ..., D_n\rangle$, in which n is the total number of MCC_i's neighbors, $D_j \in \{High, Medium, Low\}^4, j = 1, 2, ..., n$.

[4]We abstract the *degree of data correlation* into three buckets to decrease the total number of explored states in the MDPs. In our evaluation, the average number of radars each MCC controls is low

When radars belonging to different MCCs share data (especially data about shared pinpointing tasks), the communication among these MCCs would increase and thus there is more interdependency. In this paper, we assume the value to be $High$ if the percentage of pinpointing tasks between two MCCs is equal or more than 70%[5]; the value to be Low if the percentage of pinpointing tasks between two MCCs is equal or less than 30%; otherwise it is set to $Medium$.

Neighborhood Scenario: Each *neighborhood scenario* is a qualitative abstraction that captures the characteristics of a class of real scenarios that are similar in structure and policy. For each of these *neighborhood scenarios*, we will learn the best meta-level actions to perform, for instance, whether or not to adjust the heartbeat; and whether or not to move a radar to a neighbor. We define a set NS_i which consists of the *neighborhood scenarios* MCC_i might encounter based on the data correlation degrees it has with its neighbors. $NS_i = \langle f_1, f_2, ..., f_N \rangle$ where N is the total number of neighbors of the MCC. $f_j (j = 1, 2...N)$ denotes the jth neighbor's information that consists of its current heartbeat and the number of its current radars involved in the data correlation with MCC_i. $f_j (j = 1, 2...N)$ is defined as (V_j^{hb}, V_j^{radar}), in which $V_j^{hb} \in \{30seconds, 60seconds\}$ and $V_j^{radar} \in \{0, 1, many\}$[6]. "$many$" means more than one radar is involved in the data correlation. We use the qualitative value "$many$" to simplify the description of MCC's relation with its neighbors to reduce the number of different feature sets. In Fig. 5., suppose MCC_1 has a 30 seconds heartbeat and MCC_3 has a 60 seconds heartbeat. From the view of MCC_2, it is in $NS_2 = \langle (30seconds, 1), (60seconds, many) \rangle$ which means that MCC_2 has two neighbors (MCC_1 and MCC_3), MCC_1 has the 30 seconds heartbeat and 1 radar involved in the data correlation with MCC_2. MCC_3 has the 60 seconds heartbeat and "$many$" radars involved in the data correlation with MCC_2.

(≤ 5), thus making the variation of data correlation between MCCs very small. So categorizing the degree of data correlation into three buckets is able to roughly capture the difference. Setting more buckets is prone to result in more redundant states.

[5]We set the values (70% and 30%) manually to categorize the three levels of degree of data correlation. In the future, we plan to automatically learn these values.

[6]In NetRads, the average number of radars each MCC controls is low (≤ 5) and the probability that only one radar is involved in the data correlation between two MCC agents is much higher than the others. So we categorize V_j^{radar} into the three buckets to abstract and differentiate V_j^{radar} without the state space blowing up.

Weather Scenario: We will learn different meta-level actions based on the type of general weather scenarios that NetRads is experiencing. In the domain of NetRads, we evaluate the performance of our meta-level control in three different classes of *weather scenarios*: *High Rotation Low Storm* (HRLS), *Low Rotation High Storm* (LRHS), and *Medium Rotation Medium Storm* (MRMS). We choose to evaluate performance based on these three general classes since each class stresses the system in different ways. These are abstractions of real-world weather scenarios. The number of real-world weather scenarios is enormous, and learning based on the details of the scenarios is impossible from a practical perspective. In this work, HRLS denotes the weather scenario in which the number of rotations is significantly larger than the number of storms in a series of heartbeats (e.g. a significant number of rotation phenomena occur followed by a few storm phenomena, and then followed again by a significant number of rotation phenomena). LRHS is the weather scenario in which the number of storms is significantly larger than the number of rotations in a series of heartbeats. MRMS denotes the weather scenario in which the number of storms approximately equals that of rotations. Storms and Rotations have different distributions for the characteristics so that radars should adopt different scanning strategies. Even though this classification of the real world weather scenarios is very coarse, we have found that this level of detail is sufficient to generate meta-level policies that can improve the system performance significantly.

Communication Schedule: Each MCC has a communication schedule that is a result of negotiation with its neighboring MCCs with a similar heartbeat rate. In Section 4.2, we adapt the communication schedule of MCCs to resolve conflicts of heartbeats.

Learning Stage: This describes the agent's offline learning process when it adapts its behavior to improve performance.

Execution Stage: This describes the agent's real-time execution process when it chooses and implements the appropriate policy.

Factored DEC-MDP: A factored, n-agent DEC-MDP [6] is a DEC-MDP such that the world state can be factored into $n+1$ components, $\mathcal{S} = S_0 \times S_1 \times \ldots \times S_n$. The factorization separates the features of the world state that belong to one agent from those of the others and from the external features. S_0 refers to external features, which are parts of the state that the agents may observe and be affected by but do not affect themselves, such as weather or time. S_i refers to the set of state features for agent i, which is the part of the world state that an agent observes and affects.

3.1.2. Meta-level State

We have defined three features F_0, F_1 and F_2 that sufficiently capture the meta-level state information critical to the meta-level decision making process as an abstract representation at each MCC based on our previous discussion. We will use the motivating example from Section 2 to illustrate these features.

Feature F_0 contains *Information about Self.* Specifically it consists of MCC_i's current heartbeat and the number of MCC_i's current radars involved in the data correlation with its neighboring MCCs. It is defined as (V_i^{hb}, V_i^{radar}), in which $V_i^{hb} \in \{30\ seconds, 60\ seconds\}$ and $V_i^{radar} \in \{0, 1, many\}$. "$many$" means there are more than one radar involved in the data correlation. As discussed earlier, this helps determine abstractions of the states and actions of MDPs. In the example from Section 2 (Fig. 5.), suppose MCC_2 has a 30 seconds heartbeat and it has two radars (R_4 and R_5) involved in the data correlation with its neighboring MCCs. MCC_2 has the feature $F_0 = (30seconds, many)$ in its meta-level state.

Feature F_1 contains *Information about Neighbor(s).* It is the *Neighborhood Scenario* NS_i defined earlier in Section 3.1.1. MCC_i gets a good view of its neighborhood by introducing communication among neighboring MCCs during both the policy learning and execution stages. MCC_i communicates with its neighbors to formalize the feature F_1 during the policy learning and execution stages to determine its current meta-level state.

Feature F_2 is the *Degree of Data Correlation* as defined earlier in Section 3.1.1. Determining F_2 also involves communication between MCC_i and its neighbors.

In the Fig. 5. example, MCC_2 has the initial state: s^0, in which $F_0 = (30seconds, many)$, $F_1 = \langle (30seconds, 1), (60seconds, many) \rangle$ and $F_2 = \langle High, High \rangle$. Thus, the total number of meta-level states in this example is 2646 (F_0, F_1 and F_2 has 6, 49 and 9 domain values respectively; $6 \times 49 \times 9 = 2646$).

3.1.3. Meta-level Action

As described in Section 1.1, there are two types of meta-level actions: radar reorganization and heartbeat adaptation. The meta-level actions for heartbeat adaptation do not need to be abstracted, since there are only two action choices: "Use 30 seconds heartbeat"and "Use 60 seconds heartbeat". The action space of radar reorganization is very large if we use real actions, and when the action space blows up, the complexity of solving the associated DEC-MDP increases exponentially. The meta-level actions for radar reorganization are abstracted to two qualitative modes[7]. The two modes are: ***Heavy Move*** and ***Light Move***. Abstracting meta-level actions for radar reorganization in this way substantially reduces the number of explored states in the MDP. For example, suppose each MCC_i supervises x radars and has y neighbors. Without abstracting meta-level actions, each radar of MCC_i has $y+1$ possible handoff choices (to be handed off to one of MCC_i's neighbors or stay under MCC_i). The total number of possible action sets for the x radars is $(y+1)^x$ which leads to $(y+1)^x$ exploring states in the MDP. Using abstracted meta-level actions, for each neighbor of MCC_i, MCC_i has 3 possible choices $\{\phi$, *Heavy Move*, *Light Move*$\}$. The total number of possible action sets in this case is 3^y. In the domain of NetRads, the number of radars each MCC supervises can be large. 3^y is substantially smaller than $(y+1)^x$ in most cases, especially in the case that x is large[8]. Suppose MCC_i has high data correlation with its neighbors which then leads to taking the meta-level action *Heavy Move* of MCC_i. This meta-level action is implemented as a series of detailed actions that "Move radars to neighboring MCC agents until data correlation degree between MCC_i and its neighbors changes to Low"; *Light Move* of MCC_i is defined as "Move less than 20% of MCC_i's radars to its neighbors until data correlation degree between MCC_i and its neighbors changes to Low". The *meta-level action* of radar reorganization of MCC_i is defined as: $Mode(MCC_i\ to\ MCC_j)$, which means "Move radars from MCC_i to MCC_j using the qualitative mode $Mode$". In Fig. 5., one action for MCC_2 could be "$LightMove(MCC_2\ to\ MCC_1)$ & $LightMove(MCC_2\ to\ MCC_3)$".

When a certain agent takes an action, the "action" that is really implemented is a detailed plan. A **detailed plan** is an instantiation of an meta-level action. Each meta-level action for radar reorganization could have different detailed plans associated with. For example, in Fig. 5., "$HeavyMove(MCC_3\ to\ MCC_2)$" has the detailed plans such as: "Move R_7 and R_8 to

[7]In this work, we use only two qualitative modes since the average number of radars each MCC controls is low. Setting more modes is prone to result in more redundant states.

[8]In the case that $x = 8$ and $y = 3$, using abstracted meta-level actions reduces the number of explored states in the MDP by 99.9%. ($(3+1)^8 = 65536$, $3^3 = 27$)

MCC_2"and "Move R_7, R_8 and R_9 to MCC_2". The heuristic rule-based algorithm that changes the detailed plans of meta-level actions to resolve conflicts will be described in Section 4.

3.1.4. *Meta-level Control Flow*

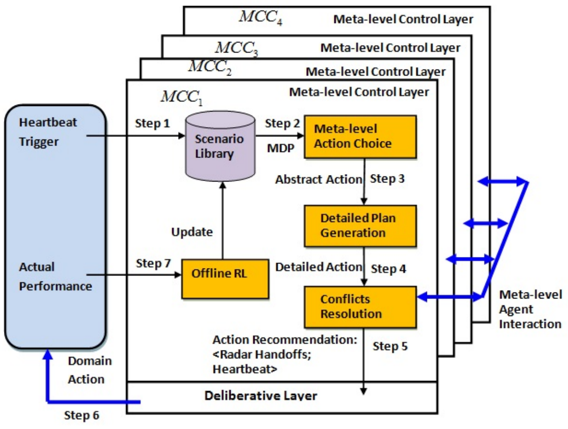


Fig. 8. Control flow in *MMLC* of each MCC. Each MCC has a meta-level control layer that includes: the *Offline RL Module*, the *Meta-level Action Choice Module*, the *Detailed Plan Generation Module* and the *Conflicts Resolution Module*.

Fig. 8. describes the control flow within each MCC. The *Scenario Library Module* stores the MDPs of each *weather scenario* as well as the meta-level policies. More details on how the module is constructed will be discussed in Section 3.2. It is also worthwhile to note that we do not include in the meta-level state a factored component indicating the local neighborhood weather pattern. In this work, each MCC is assumed to know the class of *weather scenario* it currently is experiencing. Consider the motivating example in Section 2, suppose when MMLC is triggered by the heartbeat (Step 1 in Fig. 8.), the *weather scenario* is HRLS. Each MCC chooses the MDP for HRLS and applies its policy according to its current meta-level state (Step 2 in Fig. 8.). The policies for each *weather scenario* are learned offline which is the role of the *Offline RL Module* (discussed in Section 3.3). The *Meta-level Action Choice Module* chooses the appropriate meta-level action based on current policy, depending on the specific weather scenario that the agent is in. For example, MCC_2 chooses its meta-level action ("Light Move (MCC_2 to MCC_3)"[9], "Use shorter heartbeat") based on using the HRLS weather scenario MDP policies (Step 3 in Fig. 8.). The *Detailed Plan Generation Module* maps the meta-level action to a detailed plan associated with this action which includes radar/MCC reconfiguration and heartbeat adaptation. This module uses a greedy search to compute the first detailed plan. This may cause the conflicts among detailed plans of the meta-level actions of neighboring MCCs. The *Conflicts Resolution Module* resolves the conflicts (the heuristic rule-based algorithm that is used will be discussed in Section 4). Suppose MCC_2 chooses the detailed plan of its meta-level action: "Move R_4 to MCC_3" and "Use 30 seconds heartbeat"[10] (Step 4 in Fig. 8.). As a result of conflicts resolution, MCC_2 changes its detailed plan to "Move R_5 to MCC_3" and "Use 30 seconds heartbeat" (Step 5 in Fig. 8.) and executes it. At runtime, when the *MMLC* phase is triggered at every heartbeat (more details will be discussed in Section 3.3), each MCC agent adopts the scenario-appropriate policy, resolves conflicts and executes the detailed plan of its meta-level action.

3.2. *DEC-MDP Formalization*

We frame the decentralized meta-level control in NetRads as a stochastic, factored DEC-MDP which is a DEC-MDP where the policy for each agent can be stochastic. A DEC-MDP [8] is an extension of MDP, where the outcome of an action can potentially depend on the state of all the other agents and their actions. The main difference between other models such as MMDP [10] and DEC-POMDP [8] concerns the observability assumption: MMDP uses full observation of the global state; DEC-POMDP uses only partial observation. We map the multiagent meta-level control problem (n agents in the system) to a factored DEC-MDP model in the following way. The model is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$, where

- $\mathcal{S} = S_1 \times S_2 \times \ldots \times S_n$ is a finite set of factored world states, where S_i is the state space of agent i. In NetRads, the local state of each MCC agent is the *meta-level state* as defined in Section 3.1.2 and it is not directly observed by the agent but computed as a result of communication among neighboring agents and detailed local state information of the agents.

[9]"Light Move"and "Heavy Move"are the two modes of meta-level actions as defined in Section 3.1.3.

[10]This is one of the meta-level actions for heartbeat adaptation as defined in Section 3.1.3.

- $\mathcal{A} = A_1 \times A_2 \ldots \times A_n$ is a finite set of joint actions, where A_i is the action set for agent i. In NetRads, the action of each MCC agent is the *meta-level action* that includes radar reorganization and heartbeat adaptation as defined in Section 3.1.3.
- $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathcal{R}$ is the transition function. $T(s' \mid s, a)$ is the probability of transiting to the next state s' after a joint action $a \in A$ is taken by agents in state s.
- $\mathcal{R} = \{R_1, R_2, \ldots, R_n\}$ is a set of factored reward functions. $R_i : S \times A \to \mathcal{R}$ provides agent i with an individual reward $r_i \in R_i(s, a)$ for taking action a in state s. In NetRads, the reward $R_i(s, a)$ for MCC_i represents the average of *utilities* (as defined in Section 3.1.1) of all tasks in the coverage areas of MCC_i and its neighbors. $R_i(s, a) = \frac{1}{N}\sum_{k=1}^{N} u(t_k)$, where $\{t_1, ..., t_N\}$ is the set of all tasks in the coverage areas of MCC_i and its neighbors. $R_i(s, a)$ is computed as a result of communication among neighboring agents.

We make this a stochastic DEC-MDP by defining a solution as a stochastic policy for each agent. A *stochastic policy* of an agent i is denoted by $\pi_i(s) \in PD(A_i)$, where $PD(A_i)$ is the set of probability distributions over actions A_i; s is the abstract meta-level state for agent i. Stochastic policies can cope with the uncertainty of observation and perform better than deterministic policies in partial observable environment. As described in Section 3.1.4, the *Scenario Library Module* stores the MDPs of each *weather scenario* as well as the meta-level policies. We argue that the learning is speeded up by categorizing different weather scenarios and learning policies for each MDP of each weather scenario. We take into consideration that learned policies of the overlapping states in individual factored MDPs may not converge, the policies stored in the *Scenario Library Module* are averaged polices learned from the individual factored MDPs. It should be noted that the policies for the overlapping state for different *weather scenario* could be different which makes sense for us to learn polices for different *weather scenarios*.

In Netrads, the global reward is the sum of the utilities of all the tasks completed by all the MCCs. The sum of the local rewards does not always reflect the global reward since the existence of overlapping and shared tasks could lead to redundant accounting of task utilities. So the reward function ($R_i(s, a)$) for the learning phase has a partially global component that accounts for the utility of tasks completed by the neighborhood agents. The environment of NetRads is dynamic and the number of tasks is changing rapidly, so we use average of utilities (instead of sum of utilities) of the tasks to reflect the radar scanning performance. During the learning stage, each agent communicates with its neighbors to compute its reward from a neighborhood perspective (Step 7 in Fig. 8.). This partially global reward function is a better reflection of real global reward compared to a purely local reward because when the partially global reward increases (or decreases), the real global reward will also increase (or decrease) correspondingly.

The decentralized learning process in NetRads is thus made much easier by the reduced amount of communication facilitated by the DEC-MDP model and the factored partially global reward function that moves the Nash equilibrium approximate solution of the DEC-MDP closer to the global reward which is actual solution to the stochastic DEC-MDP.

3.3. Policy Learning Using PGA-APP

Learning is a key component of MAS, which allows an agent to adapt to the dynamics of other agents and the environment and improves the agent performance or the system performance (for cooperative MAS). Effective learning algorithms are a key component to develop policies in cooperative MAS. However, due to the non-stationary environment (just as our NetRads domain) where multiple interacting agents are learning simultaneously, single-agent reinforcement learning techniques are not guaranteed to converge in multiagent settings. *Multiagent Reinforcement Learning (MARL)* is a common approach for solving multiagent decision making problems that allows agents to dynamically adapt to changes in the environment, while requiring minimum domain knowledge.

Several MARL algorithms have been proposed and studied [5] [11] [18] [22], all of which have some theoretical results of convergence in general-sum games. A common assumption of these algorithms is that an agent (or player) knows its own payoff matrix. To guarantee convergence, each algorithm has it own additional assumptions, such as requiring an agent to know a Nash Equilibrium and the strategy of the other players [5] [11] [18], or observe what actions other agents executed and what rewards they received [18] [22]. For practical applications, these assumptions are very constraining and unlikely to hold, and, instead, an agent can only observe the immediate reward after selecting and performing an action.

Zhang and Lesser [44] proposed a practical MARL algorithm, called *Policy Gradient Ascent with approximate policy prediction (PGA-APP)*, that exploits the idea of policy prediction. PGA-APP only requires an agent to observe its reward when choosing a given action. PGA-APP empirically converges faster and in a wider variety of situations than other state-of-the-art MARL algorithms. We describe the PGA-APP algorithm which is implemented in the *Offline RL Module*. In this work, offline learning is preferred for two reasons. The first is to control the scenarios in the simulation system. While the real NetRads environment could have varying weather scenarios in play at any point in time, we make a simplifying assumption in our simulation work that the entire radar network sees only one specific scenario at any point in time. This ensures that multiple agents simultaneously contribute to a particular weather scenario's DEC-MDP policy, thereby speeding up the learning process. Further more, we can control communication in offline learning since our reward function is targeted for policy generation and not necessarily for execution. We plan to relax the single scenario assumption in the future as discussed in Section 7 (on future work). The learning component will learn stochastic policies based on a Nash Equilibrium solution concept for radar reorganization and heartbeat adaptation.

We map the PGA-APP algorithm to the domain of NetRads to learn the MMLC policies offline (the *Offline RL Module* in Fig. 8.). PGA-APP uses Q-learning to learn the expected value of each action in each state to estimate the partial derivative with respect to the current strategies (line 5, Algorithm 1). The value function $Q(s, a)$ stores the reward MCC_i expects if it executes action a at state s. The stochastic policy $\pi(s, a)$ stores the probability that MCC_i will execute action a at state s. The actions here are *meta-level actions* and the states are *meta-level states* as defined in Sections 3.1.3 and 3.1.2. We use the $\epsilon - Greedy$ [39] exploration scheme to pick actions for learning (line 4, Algorithm 1). The $\epsilon - Greedy$ exploration scheme balances the exploration with exploitation by selecting both greedy actions and random actions during the learning stage. As shown by Line 5 in Algorithm 1, Q-learning only uses the immediate reward to update the expected value. The reward r is the partially global reward defined in Section 3.2. We are thus introducing communication among neighboring agents in the learning stage to better model the reward function in the DEC-MDP. In the online execution of the policy, we do not need to compute this immediate reward, thereby avoiding any extra communication among MCC agents[11]. We also introduce communication among MCC's neighbors in the learning and execution stages to calculate some features of the meta-level states. With the value function Q and current policy π, PGA-APP then can calculate the partial derivative, as shown by Line 8, Algorithm 1 [44]. As shown in Line 9, Algorithm 1, PGA-APP approximates the second component by the term $-\gamma|\widehat{\delta}(s, a)|\pi(s, a)$. When $MCCs$' strategies converge to a Nash equilibrium, this approximation derivative will be zero and will not cause the agents to deviate from the equilibrium. The negative sign of this approximation term is intended to adjust the policy with the derivative prediction length and increase the convergence speed.

Algorithm 1 Zhang & Lesser's PGA-APP Algorithm

```
1:  Let $\theta$ and $\eta$ be the learning rates, $\xi$ be the discount
    factor, $\gamma$ be the derivative prediction length;
2:  Initialize value function $Q$ and policy $\pi$;
3:  **repeat**
4:    Select an action $a$ in current state $s$ according to policy
      $\pi(s, a)$ with suitable exploration;
5:    Observing reward $r$ and next state $s'$, update
      $Q(s, a) \leftarrow (1 - \theta)Q(s, a) + \theta(r + \xi \max_{a'} Q(s', a'))$;
6:    Average reward $V(s) \leftarrow \Sigma_{a \in A}\pi(s, a)Q(s, a)$;
7:    **foreach** *action* $a \in A$ **do**
8:      **if** $\pi(s, a) = 1$ **then** $\widehat{\delta}(s, a) \leftarrow Q(s, a) - V(s)$
        **else** $\widehat{\delta}(s, a) \leftarrow (Q(s, a) - V(s))/(1 - \pi(s, a))$;
9:      $\delta(s, a) \leftarrow \widehat{\delta}(s, a) - \gamma|\widehat{\delta}(s, a)|\pi(s, a)$;
10:     $\pi(s, a) \leftarrow \pi(s, a) + \eta\delta(s, a)$;
11:   **end**
12:   $\pi(s) \leftarrow \prod_{\Delta}[\pi(s)]$;
13: **until** *the process is terminated*;
```

NetRads is designed to quickly detect low-lying meteorological phenomena, so time is a critical concern. A heartbeat consists of four phases, it is important that the *MMLC* takes negligible amount of time so that there is enough time for the complex operations of *Local Optimization* and *Negotiation* phases. During the learning stage, the *MMLC* is triggered at every heartbeat which means the meta-level actions of the MCC-

[11]This is very little considering that meta-level control is operating at a 30 or 60 second heartbeat and there is already much more communication occurring as a result of negotiation.

s are changed at every heartbeat. This is acceptable in our work, since the offline learning takes very little time by exploring the state space that is abstracted and computing the partially global reward using neighborhood communication. Our evaluation results show that triggering *MMLC* at every heartbeat during learning helps improve the overall performance. A long period trigger for the *MMLC* during learning will make the meta-level policy obsolete due to dynamic nature of the environment (the weather phenomena are changing quickly and dynamically that results in the changes of the current meta-level states of the MCCs).

During the execution stage, the *MMLC* is also triggered at every heartbeat. This is acceptable since the cost of *MMLC* is negligible compared with the expected utility gained. Communication with neighbors is also used for MCCs to calculate some features of the meta-level state at this stage. Each MCC then chooses the proper policy and applies the appropriate detailed plan (it is generated after conflict resolution) based on its meta-level state. We argue that it is worthwhile running the conflict resolution algorithm in the *Conflicts Resolution Module* as long as the expected utility gained outweighs the cost of running the algorithm at the execution stage. After the two deliberative-level phases (*Local Optimization* and *Negotiation*) are completed, domain actions of radar scanning are implemented based on the set of tasks (Step 6 in Fig. 8.).

In the next Section, we investigate the conflicts that could occur when agents apply PGA-APP locally and present a heuristic rule-based algorithm to resolve conflicts between two agents during both the learning and execution stages in order to compute the approximately optimal meta-level actions.

4. Conflict Resolution

The policies learned using the PGA-APP algorithm for each MDP could be optimal policies for each agent from a local perspective. However, as mentioned earlier, the chosen meta-level actions could cause conflicts between agents when the associated detailed plans are generated by the *Detailed Plan Generation Module*. As will be discussed in the experimental Section, resolving such conflicts in an intelligent manner improves overall system performance. We first define the types of conflicts that may happen in this case. We present a heuristic rule-based algorithm that uses predefined rules to locally resolve the conflicts by changing the detailed plans of the meta-level actions among agents at both learning and execution stages.

4.1. Conflict Type

We define the following types of conflicts among agents' detailed plans associated with meta-level actions:

a) **Local Radar Conflicts (LRC)** refer to situations in which meta-level action choices of MCC agents fail to efficiently balance the load of the multiagent system. Consider the situation (Fig. 5.) where both MCC_2 and MCC_4 decide to move radars to MCC_3. It is acceptable for MCC_3 to receive radars from either MCC_2 or MCC_4, but receiving radars from both will result in a very high[12] load for MCC_3 (MCC_3 controlling too many radars could increase the time and messages for local negotiation). This is a *LRC* between MCC_2 and MCC_4. A LRC is recognized in the following way: Neighboring MCCs exchange messages to inform each other about their current detailed plans. If MCC_i finds that its neighbor MCC_j has the detailed plan of moving radars to the same agent MCC_k, it sends a message to tell MCC_k the number of radars it plans to move to MCC_k. Meanwhile, MCC_j does the same type of communication with MCC_k as MCC_i does. MCC_k receives messages from MCC_i and MCC_j, adds the potential number of radars to the current number of radars associated with. If the sum exceeds the threshold for high load, MCC_k will notify MCC_i and MCC_j that a LRC exists between them.

b) **Shared Radar Conflicts (SRC)** are inconsistencies that may arise when two or more agents attempt to move the same radar(s). In Fig. 5., a *SRC* occurs when MCC_1 and MCC_3 both require the control of the same radar belonging to MCC_2. A SRC is recognized in the following way: Neighboring MCCs exchange messages to inform each other about their current detailed plans. If MCC_i finds that its neighbor MCC_j has the detailed plan that competes for the same radar, it recognizes this as a SRC. Meanwhile, MCC_j also recognizes the SRC in a similar fashion.

c) **Inconsistent Heartbeat Conflicts (IHC)** occur when two neighboring agents have different heartbeats and have to communicate with each other dur-

[12]The threshold for a high load of the MCC varies according to different simulation scenarios. For example, in a simulation scenario that each MCC has an average number of 5 radars associated with, the threshold is set to be 8 radars under one MCC.

ing the *Negotiation* phase. The MCCs are assumed to communicate with neighboring MCCs that have the same heartbeats during the *Negotiation* phase. Suppose in Fig. 5., MCC_1 decides to use the shorter heartbeat (30 seconds) and MCC_2 decides to use the longer one (60 seconds). It is not possible for MCC_1 to communicate with MCC_2 at the end of every 30 seconds' heartbeat as defined in Section 1.1. An IHC is recognized when MCC_i finds the neighboring agent MCC_j tries to use a different heartbeat by exchanging messages.

4.2. *The Heuristic Rule-based Algorithm*

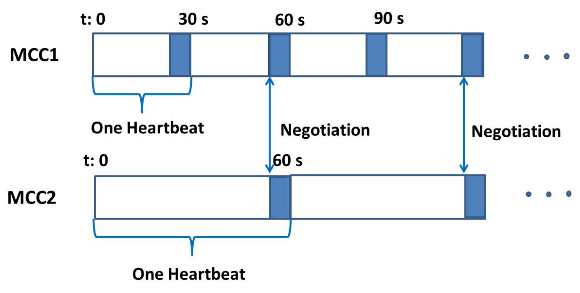


Fig. 9. Heartbeat Adaptation.

Algorithm 2 is the heuristic rule-based algorithm for MCC_i. This algorithm uses the following pre-defined heuristic rules to solve the three types of conflicts:

a) **Resolution Rule for LRC:** To resolve a *LRC* between two MCCs, the two MCCs exchange messages describing the degree of data correlation, the MCC with larger data correlation first applies its current detailed plan, the other MCC stochastically chooses another of the candidate detailed plans that will not result in a *LRC* and applies it. It should be noted that the candidate detailed plan only involves changing of radar reorganization, the current heartbeat should remain the same since the change of heartbeat may incur new IHC conflicts. If no such detailed plan is found, the other MCC aborts its current detailed plan. If both MCCs have the same amount of data correlation, the MCC is to first apply its detailed plan by a pre-defined ordering. The detailed plan of the MCC that has larger data correlation is always more valuable to implement since this MCC has more information about *pinpointing tasks* in its region. Pinpointing tasks are generally more valuable since they affect multiple agents.

b) **Resolution Rule for SRC:** To resolve a *SRC* between two MCCs, the two MCCs exchange messages describing the current number of radars associated with, the MCC with fewer radars receives the shared radar(s). If both MCCs have the same number of radars, the MCC is chosen to receive the shared radar(s) by a pre-defined ordering. The more radars one MCC controls, the more time and messages it spends for local negotiation that decreases the utility for radar scanning. It is preferred to assign the shared radar(s) to a MCC with the lower load.

c) **Resolution Rule for IHC:** To resolve an *IHC*, let the MCC with shorter heartbeat adapt its communication schedule to the MCC with longer heartbeat (as Fig. 9. shown, MCC_1 communicates with MCC_2 every two heartbeats). In this work, we are not resolving IHC by changing heartbeats of MCCs.

Algorithm 2 The Heuristic Rule-based Algorithm

```
1: for each neighboring $MCC_j$ ( $j > i$ )do
2:     $MCC_i$ send message to $MCC_j$ describing $a_{i,DP}$
3:     $MCC_j$ check $a_{j,DP}$ and $a_{i,DP}$
4:     if $Con\_Check(a_{j,DP}, a_{i,DP}) = true$ then
5:         $MCC_j$ apply rules (defined in Section 4.2) to resolve the conflict
6:         $a_{j,DP} \leftarrow a'_{j,DP}$
7:         $a_{i,DP} \leftarrow a'_{i,DP}$
8:         $MCC_j$ send message to $MCC_i$ describing new $a_{i,DP}$
```

We now describe the heuristic rule-based algorithm presented in Algorithm 2. Let $a_{i,DP}$ and $a_{j,DP}$ be the current detailed plans for MCC_i and MCC_j respectively; let $a'_{i,DP}$ and $a'_{j,DP}$ be the new detailed plans after conflict resolution for MCC_i and MCC_j respectively; let $Con\_Check(a_{j,DP}, a_{i,DP})$ be the function that returns whether any type of conflict (*LRC*, *SRC* or *IHC*) exists between $a_{j,DP}$ and $a_{i,DP}$. This function uses the mechanism described in Section 4.1 to recognize the conflicts. Each MCC_i initializes the conflict resolution process by sending messages to its neighboring[13] MCCs (Line 1-2, Algorithm 2). Each neighboring MCC then applies the heuristic rules to resolve

[13]We only consider the subsequent ($j > i$) neighboring MCCs in order to avoid resolving the conflict between two MCCs repeatedly.

the conflict (Line 5, Algorithm 2) if any type of conflict exists and updates the new detailed plan for MCC_i and itself (Line 6-7, Algorithm 2).

In this algorithm, each conflict is resolved by replacing the original detailed plan with a new one. However, each updated detailed plan does not take into account the influence it has on other neighboring agents and may introduce new conflicts with other agents. For example, in Fig. 5., the conflict between MCC_2 and MCC_3 is resolved by changing the detailed plan of MCC_2 and this change may result in a new conflict between MCC_2 and its another neighbor MCC_1. Thus, while this localized algorithm has low overhead, it provides no guarantee that all conflicts in the system will be resolved.

In the next Section, we will evaluate the impact of MMLC in the performance of NetRads. We first generate meta-level heuristics manually to show meta-level control is useful and then show that our learning algorithm allows the network of NetRads to dynamically adjust to changing weather phenomena. At the end, we show that our heuristic rule-based algorithm helps resolve the conflicts among agent's meta-level actions.

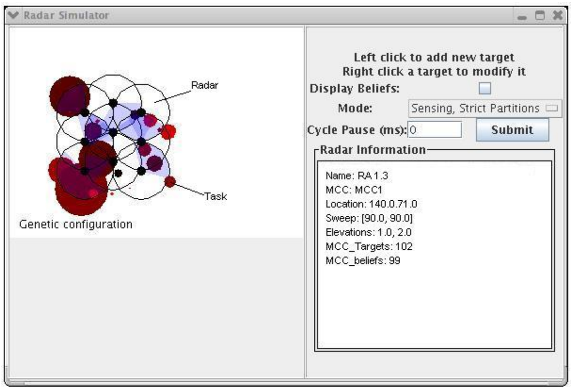
5. Empirical Evaluation

We use the simulator of the NetRads radar system [26] to evaluate our algorithm. In this simulator, radars are clustered based on location, and each cluster of radars has a single MCC. Each MCC has a feature repository where it stores information regarding tasks in its spacial region, and each task represents a weather event. The simulator additionally contains a function that abstractly simulates the mapping from physical events and scans of the radars to what the MCC eventually sees as the result of those scans. MCCs discover and track the movement of the weather events through this process.

Tasks are created at a MCC based on radar moment data that has been just received. Tasks can be either *pinpointing* or *non-pinpointing*.

5.1. Experiment Setup

For the experiments reported here, we use the simulation setup where there are 3, 12 and 30 MCCs (agents). This is the setup used by Krainin et. al [26]. Fig. 10. is the snapshot of the radar simulator for a particular real-time scenario. In Fig. 10., each hollow circle represents a radar and each filled circle represents a task (we are only concerned about *rotation* and *storm* tasks in the evaluation.). The Radar Information Panel (Fig. 10.) provides information about a particular radar including its name, its MCC supervisor, its physical location in the plane coordinate system, the angle range it sweeps, the target task it scans and the belief value of the negotiation algorithm in Phase 4: *Negotiation*. We test the results for three different types of *weather scenarios*: *High Rotation Low Storm* (HRLS), *Low Rotation High Storm* (LRHS), and *Medium Rotation Medium Storm* (MRMS). There are a total of 80 tasks in each *weather scenario*. HRLS contains 60 rotation tasks and 20 storm tasks; LRHS contains 60 storm tasks and 20 rotation tasks; MRMS contains 40 storm tasks and 40 rotation tasks.



Mode: Sensing, Strict Partitions
Cycle Pause (ms): 0

Name: RA1.3
MCC: MCC1
Location: 140.0.71.0
Sweep: [90.0, 90.0]
Elevations: 1.0, 2.0
MCC_Targets: 102
MCC_beliefs: 99

Fig. 10. Snapshot of Radar Simulator.

We generate the training/test cases by varying such parameters as number of MCCs, number and types of tasks, initial heartbeat for each MCC, *percentPinpointing* and etc. *percentPinpointing* is defined as the percentage of pinpointing tasks relative to all tasks in a specific training/test case. We vary *percentPinpointing* to evaluate the performance on different numbers of pinpointing tasks. We also scale up the number of tasks in training/test cases. *Utility* and *Negotiation Time* are the parameters that are used to analyze performance. *Utility* is defined as the overall utility of a give configuration of radars where two sets of factors contribute to the utility [27]. The first set of factors is concerned with *how well* a particular portion of the atmosphere is sensed by the given radar configuration. The second set of factors is concerned with *how important* the scanned sectors are to the end users. *Negotiation Time* denotes the average time (seconds) that MCCs spend in *Negotiation* (Phase 4). If a MCC chooses to spend less time in *Negotiation*, then the remaining amount of time in the fixed heartbeat is allocated to *Data Processing* and *Local Optimization*. This is an instance of MMLC determines what resources to allocate to different deliberative actions.

For the experiments, we have the following assumptions:

1. All the MCCs are in the same type of *weather scenario* that is set for the simulation.
2. All the MCCs have the same number of radars associated and same heartbeats (all are 30 seconds or 60 seconds) initially.

As described in Section 3.3, the real NetRads environment could have varying weather scenarios in play at any point in time. The first assumption is made to ensure that multiple agents simultaneously contribute to a particular weather scenario's DEC-MDP policy, thereby speeding up the learning process. We plan to relax the single scenario assumption in future work as discussed in Section 7. The second assumption is made to ensure fairness in comparison to make the results more interpretable.

We compare the results of four algorithms: *No-MLC*, *Adaptive Heuristic Heartbeat (AHH)*, *PGA-APP* and *PGA-APP-CR*.

– *No-MLC* is the algorithm that with no explicit or implicit meta-level control (It has all the phases except *MMLC* in a heartbeat).
– *AHH* is the algorithm where we incorporate hand-generated heuristics in meta-level control to adaptively change the heartbeat of each MCC. The rules are simple: For each MCC_i, at the end of *Data Processing* (Phase 1), if there are more rotation phenomena in the region of MCC_i, MCC_i increases the heartbeat for its next period, otherwise, MCC_i decreases the heartbeat for its next period (longer heartbeat is better for rotations due to the need for more scanned elevations, and shorter heartbeat is better for storms). The heuristics also help to address the radar reorganization problem. Assigning the same heartbeat to the neighboring MCCs with overlapping region results in better communication/negotiation in the *Negotiation* phase to help reducing the amount of data correlation in the next heartbeat period which has some of the same effect as handing off radars.
– *PGA-APP* augments MCCs with meta-level control based on offline RL (PGA-APP) to adjust the system heartbeat and re-organize the subnets of radars to adapt to changing weather conditions.
– *PGA-APP-CR* (*PGA-APP* with *conflict resolution*) is the algorithm where we incorporate heuristic rules to resolve local conflicts.

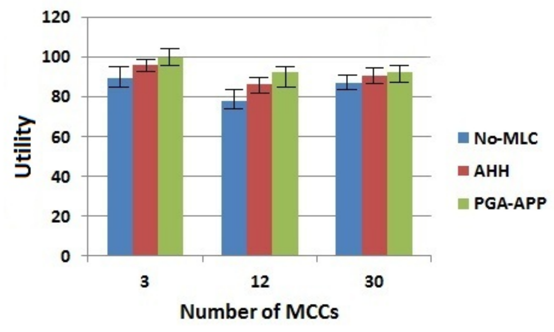
For the *MMLC* phase, we used 1000 training cases and each has a long sequence of training data to learn the policies for each abstract scenario offline.

Learning parameters (defined in line 1, Algorithm 1) will affect the convergence of PGA-APP. For non-competitive problems (e.g., NetRads), with too large a γ (derivative prediction length), MCC_i may not predict its neighbor's strategy correctly. Then the gradient based on the wrong neighboring MCC' strategy deviates too much from that of the current strategy, and MCC_i adjusts its strategy in the wrong direction. In the experiments, PGA-APP used prediction length $\gamma = 0.2$. With higher learning rates θ and η, PGA-APP learns a policy faster at the early stage, but the policy may oscillate at late stages [44]. Properly decaying θ and η makes PGA-APP converge better. PGA-APP uses value-learning rate $\theta = 0.8$ and policy-learning rate $\eta = 1/(1000 + t)$, where t is the current number of iterations. We ran 30 test cases for each of the four algorithms described above for the three different weather scenarios (defined in Section 3.1.1).

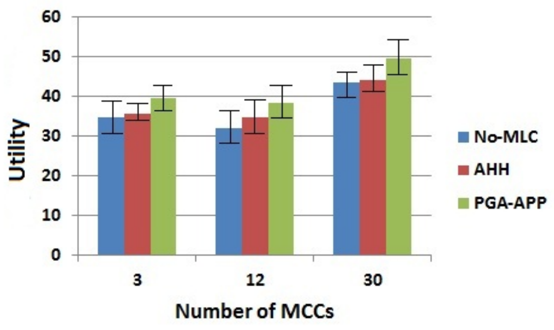
In the experimental evaluation, we first compare *PGA-APP* with *No-MLC* and *AHH*. We show that adaptive multiagent meta-level control significantly improves the performance for a variety of scenarios. Then we compare *PGA-APP-CR* with *PGA-APP* to show the effects of applying rules to resolve conflicts between agents.

5.2. *Performance of* PGA-APP

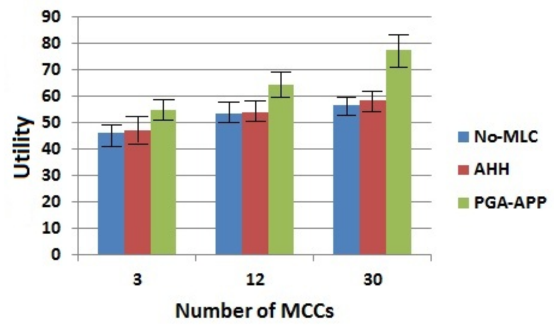
We ran test cases for each *weather scenario* consisting of 3, 12 and 30 MCCs with 9, 60 and 150 radars respectively. The number of tasks is 80 for all test cases and *percentPinpointing* is set to 60%. Fig. 11. shows the performance of *No-MLC*, *AHH* and *PGA-APP* on *Utility* for a variety of scenarios. *AHH* performs significantly better than *No-MLC* on *Utility* in all comparisons (Fig. 11.(a)). The improvement is 7%, 11% and 4% for networks with 3, 12 and 30 MCCs respectively. This shows the effectiveness of adding meta-level control to agent reasoning in HRLS scenarios. According to the hand-generated rules in *AHH*, the three MCCs would all set their heartbeat to 60 seconds for HRLS. The three MCCs would then have more time on *Local Optimization* and *Negotiation* so that the final configurations of scanning tasks for the next heartbeat peri-



(a) **HRLS Scenarios**



(b) **LRHS Scenarios**



(c) **MRMS Scenarios**

Fig. 11. *Utility* of *No-MLC*, *AHH* and *PGA-APP* in different weather scenarios for number of MCCs to be 3, 12 and 30.

od would be more optimized. This results in larger *Utility*. In HRLS scenarios, *PGA-APP* performs significantly better than *No-MLC* and a little better than *AHH*. p values from t-tests are 0.0074, 0.037 and 0.016 for the 3, 12 and 30 MCCs cases respectively compared with *No-MLC*. The minor difference in performance between *PGA-APP* and *AHH* on HRLS scenarios leads to the speculation that the heartbeat is a critical factor for rotations compared with radar reorganization.

In both LRHS and MRMS scenarios (Fig. 11.(b) and Fig. 11.(c)), *AHH* performs a little better than *No-MLC*. The improvement is 1%, 1% and 3% for the 3, 12 and 30 MCCs cases respectively. *PGA-APP* performs significantly better than *No-MLC*. In LRHS scenarios, p values are 0.023, 0.0095 and 0.0071 for the three cases respectively while in MRMS scenarios, the p values are 0.008, 0.029 and 0.035 respectively. *PGA-APP* performs significantly better than *AHH*. In LRHS scenarios, p values are 0.0086, 0.0013 and 0.0028 for the 3, 12 and 30 MCCs cases respectively; in MRMS scenarios, the p values are 0.0074, 0.0082 and 0.034 respectively. We observe that the 30 seconds heartbeat is not an important factor in LRHS scenarios (*AHH* increases small amount of *Utility*.). In *PGA-APP*, each MCC adopts the policy appropriate to its *neighborhood scenario*. The meta-level action for radar reorganization that allocates radars with large data correlation to the same MCC helps reduce the time for negotiation between MCCs which would increase the time for *Local Optimization*. As discussed in Section 5.1, MMLC is able to leverage that in different situations. In certain situations (e.g., there are many internal tasks compared to boundary tasks) it is better to do a good job in local optimization and allocate fewer cycles to negotiation while in other situations more cycles for negotiation would be better (e.g., many pinpointing tasks exist in boundary regions between MCCs). *PGA-APP* performs significantly better on learning policies to control when and which radars should be moved.

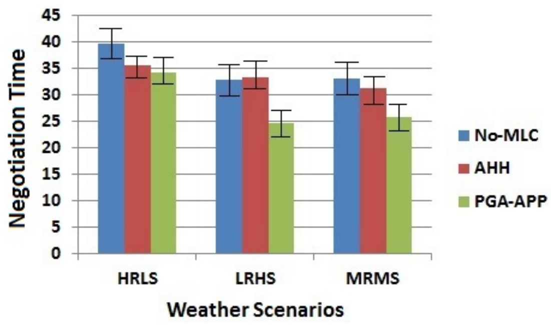


Fig. 12. *Negotiation Time* of *No-MLC*, *AHH* and *PGA-APP* in different weather scenarios.

In Fig. 12., *PGA-APP* performs significantly better than *No-MLC* on *Negotiation Time* for each *weather scenario* with p values of 0.0028, 0.033 and 0.0058 respectively. *PGA-APP* uses the least time in the *Negotiation* phase and achieves the highest *Utility* in each *weather scenario*. This shows that adaptive meta-level

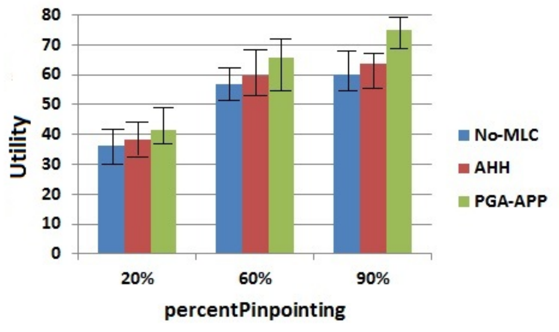


Fig. 13. *Utility* of *No-MLC*, *AHH* and *PGA-APP*, for *percentPinpointing* to be 20%, 60% and 90%.

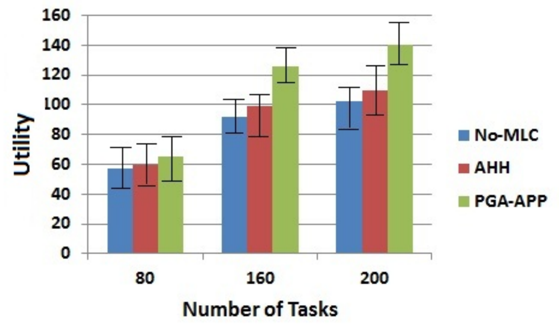


Fig. 14. *Utility* of *No-MLC*, *AHH* and *PGA-APP*, for number of tasks to be 80, 160 and 200.

control allows for effective use of the heartbeat i.e. by ensuring that meta-level control parameters are coordinated so that negotiations converge quickly, more time can be spent on *Data Processing* and *Local Optimization*. *AHH* does not perform better than *No-MLC* on all *weather scenarios* since *AHH* is not as adaptive as *PGA-APP* in dynamic conditions. It spends 1% more *Negotiation Time* than *No-MLC* in LRHS scenarios.

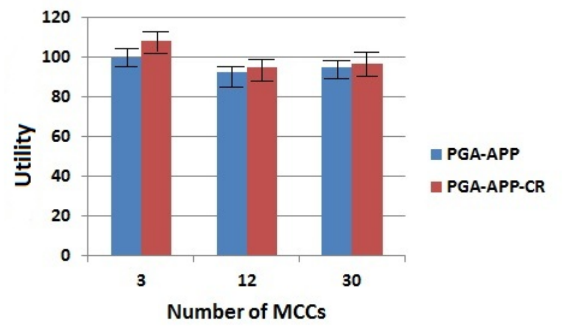
We varied *percentPinpointing* (setting it to 20%, 60% and 90%) and ran test cases on all the three *weather scenarios*. In Fig. 13., we note that *Utility* increases with the increase of the percentage of pinpointing tasks to all tasks for *No-MLC*, *AHH* and *PGA-APP*. More pinpointing tasks occurring in the boundary regions between MCCs would increase the utilities for scanning pinpointing tasks to increase *Utility* of all the scanning tasks. In all *percentPinpointing* settings, *AHH* performs better than *No-MLC* (The increase is 5%, 5% and 6% for the 20%, 60% and 90% *percentPinpointing* cases respectively compared with *No-MLC*) and *PGA-APP* achieves the best performance (The increase is 9%, 10% and 17% for the three cases respectively compared with *AHH*).

In Fig. 14., we scaled up the number of total tasks to 160 and 200 and compared the performance with that of 80 tasks (*percentPinpointing* is fixed at 60%). *Utility* increases substantially with the increase of number of tasks for all three methods. This is mainly because the increase of pinpointing tasks results in large increase of utility. *PGA-APP* performs significantly better than *No-MLC* on *Utility* with p values of 0.0046, 0.023 and 0.0076 for networks with 80, 160 and 200 tasks respectively. *PGA-APP* also performs significantly better than *AHH* on *Utility* with p values of 0.049, 0.00063 and 0.0035 for the three cases respectively. Thus, these results show that in situations where activities in different MCCs need to be coordinated, meta-level reorganization of control responsibilities is an effective tool.

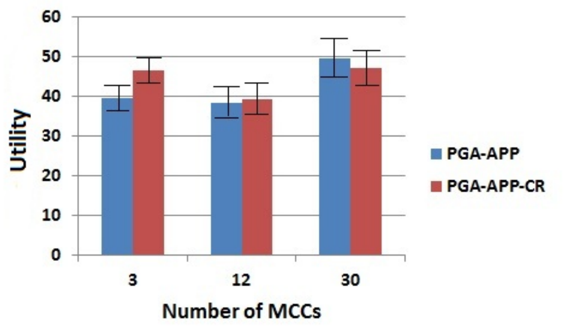
5.3. Resolving Conflicts with Heuristic Rules

In *PGA-APP*, the SRC and LRC conflicts are resolved implicitly based on the order of MCCs. For example, a SRC between MCC_1 and MCC_3 is resolved by assigning the shared radar to MCC_1 since it has a lower index. A LRC is resolved in *PGA-APP* that one MCC with lower index takes its action and the other aborts its action. When an IHC occurs, the MCC will only communicate to the neighbors that have the same heartbeat. In *PGA-APP-CR*, such conflicts are resolved in an explicit way. In the following part, we empirically show that resolving conflicts explicitly improves overall system performance.

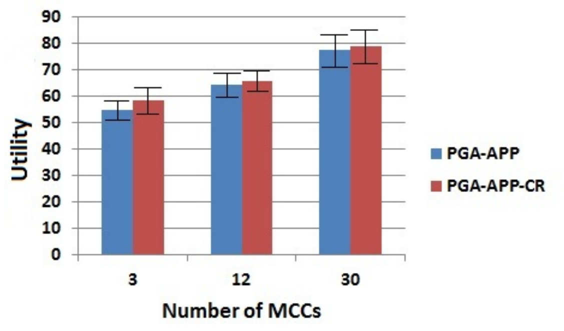
We re-ran the test cases for Fig. 11. to compare *PGA-APP-CR* and *PGA-APP*. Fig. 15. shows the performance on *Utility*. Compared with *PGA-APP*, *PGA-APP-CR* achieves the highest increase with respect to *Utility* when the number of MCCs is 3 (*PGA-APP-CR* increases 9%, 17% and 7% respectively for the three weather scenarios). When the number of MCCs increases from 3 to 12 in each weather scenario, the improvement of *PGA-APP-CR* on *Utility* decreases. The increase on *Utility* is only 2%, 2% and 3% for the three weather scenarios respectively. When the number of MCCs increases, each MCC has more dependency with other MCCs and the probability of having conflicting actions with neighboring MCCs increases significantly. On the other hand, resolving conflicts locally between two MCCs using heuristic rules in *PGA-APP-CR* in this situation will more likely introduce additional conflicts. Combining these factors, we can ex-



(a) **HRLS Scenarios**



(b) **LRHS Scenarios**



(c) **MRMS Scenarios**

Fig. 15. *Utility* of *PGA-APP* and *PGA-APP-CR* in different weather scenarios for number of MCCs to be 3, 12 and 30.

plain why the performance goes down with the scaling of MCCs. In the LRHS scenarios, *PGA-APP-CR* performs 5% worse than *PGA-APP* with respect to *Utility* when the number of MCCs goes up to 30. When the utility lost by bringing in new conflicts outweighs the expected utility gained by locally resolved conflicts, *Utility* would decrease. *PGA-APP-CR* performs well on *Utility* in small problems (3 agents in NetRads). Due to its mostly localized view, *PGA-APP-CR* is not expected to perform well as the number of MCCs scales up and the dependencies among agents increase significantly.

Category	Average number of conflicts before conflict resolution	Average number of conflicts after conflict resolution
$0 \sim 5$	3 (LRC: 1; SRC: 1; IHC: 1)	0
$5 \sim 10$	7 (LRC: 2; SRC: 3; IHC: 2)	2 (LRC: 1; SRC: 1; IHC: 0)
> 10	18 (LRC: 8; SRC: 7; IHC: 3)	10 (LRC: 6; SRC: 4; IHC: 0)

Table 1

Results showing the average number of conflicts before and after conflict resolution in three categories.

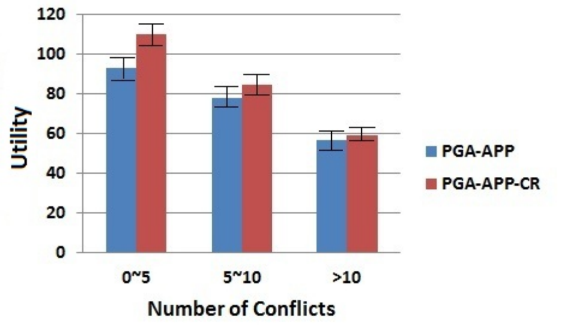


Fig. 16. *Utility* of *PGA-APP-CR* and *PGA-APP*, number of conflicts varies.

We ran test cases on all the three *weather scenarios* based on the number of conflicts in each case (the number of MCCs is 12, the number of tasks is 160). In Fig. 16., we observe that *PGA-APP-CR* performs better (18%, 8% and 4% respectively) than *PGA-APP* with respect to *Utility* in each bucket. The rules to resolve local conflicts work quite well with few conflicts. When the number of conflicts increases, the effect of rules fades. This is mainly because the rules are defined to resolve local conflicts between two agents, not capable to prevent new conflicts among subsets of agents when applying the rules. Resolving local conflicts from a local perspective is not sufficient to find globally optimum solution. Table 1 shows the conflict resolution performance in the three different categories as well as Fig. 16. All the IHC conflicts are resolved in the three categories because the two MCCs adapt their communication schedule without changing either of their heartbeats. In simple cases when there are few conflicts (the first category in table 1), *PGA-APP-CR* is capable of resolving all of them. When the number of conflicts increases, the heuristic rules are not capable of eliminating all the conflicts in the problem. In table 1, we note that when there are more than 10 simultaneous conflicts in the network, the heuristic rules can resolve only about 44% of the conflicts. Let us consider the motivating example in Section 2 (Fig. 5.). Suppose there is one conflict between MCC_1 and MCC_2 and another conflict between MCC_2 and MCC_3. The heuristic rules help to resolve these two conflicts locally by changing the actions of MCC_1 and MCC_3 separately. There is a possibility that a conflict exists in the newly changed actions of MCC_1 and MCC_3 and this could not be detected in our current approach. For example, MCC_1 decides to take the control of R_5 after resolving the local conflict between MCC_2 and itself. At the same time, MCC_3 also decides to take the control of R_5 after resolving the local conflict between MCC_2 and itself. Thus, a SRC occurs between MCC_1 and MCC_3.

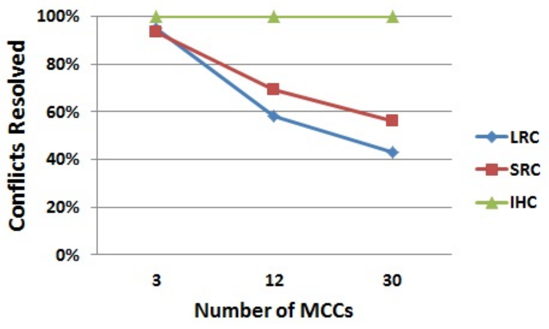


Fig. 17. Percentage of conflicts resolved in different weather scenarios for number of MCCs to be 3, 12 and 30.

Fig. 17 shows the results of conflict resolution using *PGA-APP-CR* with number of MCCs increasing. As mentioned earlier, all the IHC conflicts are successfully resolved by adapting communication schedules between two MCCs. In the cases that there are only 3 MCCs, both LRC and SRC conflicts are resolved very efficiently. 95% of LRC conflicts and 93.5% of SRC conflicts are resolved in this situation. When the number of MCCs increases, there are more dependencies among different agents which makes the conflict resolution more complicated because of the shared tasks in the overlapping areas, especially the pinpointing tasks that need significant coordination among MCCs. In such complicated situations, the performance of *PGA-APP-CR* on resolving LRC and SRC conflicts decreases significantly. We observe that the percentage of conflicts resolved for LRC drops below 50% when the number of MCCs reaches 30.

The results of our initial evaluation show that PGA-APP performs significantly better when learning meta-level policies for radar reorganization (**P.1**) and heartbeat adaptation (**P.2**). The results also show that the heuristic rule-based algorithm helps resolve some conflicts and improve the overall utility.

6. Related Work

In this Section, we will discuss the current state of the art as it relates to this research. We will discuss research in bounded rationality, multiagent deliberation and multiagent meta-level control, stochastic models, multiagent reinforcement learning algorithms and NetRads application domain and compare it to our work.

6.1. Bounded Rationality, Multiagent Deliberation and Multiagent Meta-level Control

In complex environments, autonomous systems generally require the ability to reason about resource allocation to computation at any point in time. The basic idea of bounded rationality arises in the work of Simon with his definition of procedural rationality [37]. He argues that people find satisfactory solutions to problems rather than optimal solutions because people do not have unlimited processing power. Simon's work has addressed the implications of bounded rationality in the areas of psychology, economics and artificial intelligence. In the area of agent design, he has considered how the nature of environment can determine how simple an agent's control algorithm can be and still produce rational behavior. Russell, et al. [36] cast the problem of creating resource-bounded rational agents as a search for the best program that an agent can execute. In searching the space of programs, the agents, called bounded-optimal agents, can be optimal for a given class of programs or they can approach optimal performance with learning, given a limited class of possible programs.

Recent research efforts in distributed control share the goal of applying agent technology to intelligent network management and data harvesting. A multiagent system (MAS) allows for the distribution of knowledge, data, and resources among individual a-

	Raja and Lesser's work	Our work
Application domain	Meta-level control is used in a task allocation domain where meta-level control supports decisions on when to accept, delay, or reject a new task.	Meta-level control is used in a more complex application of real-world system to support agent interaction and network reorganization.
Abstraction	Abstract agent states and scenarios are implemented for meta-level control policies learning.	The agent states/scenarios as well as the agent actions are abstracted which helps bound the size of the state/policy set making the problem tractable.
Single/ Multi agent(s)	Meta-level control is implemented in an individual agent and in multiple agents.	Multiagent meta-level control is implemented to guide the deliberative actions, because the application domain is time-critical and distributing the meta-level control among agents help save the time on learning and executing policies.
Stochastic model	The problem is modeled as a MDP to learn deterministic policies.	The problem is formalized as a DEC-MDP because each agent has local observability of the global state and the actions of agents interacted. The polices learned are stochastic polices.
Frequency of meta-level control	The need for meta-level control is triggered at each event.	Meta-level control is triggered at each heartbeat. We will also use methods to adaptively trigger meta-level control.

Table 2

Comparison between Raja and Lesser's work and our work.

gents and its modularity supports the development and maintenance of complex highly reliable systems [42]. Several distributed control algorithms based on multi-agent systems have been proposed to solve centralized control problems efficiently and proved to converge to the global optimal solution [29] [28] [31]. In our work, the agents learn and apply policies at the meta-level which is time constrained. We do not use these distributed control algorithms to learn the policies due to high communication overhead and long convergence time.

Meta-level control is the ability of an agent to optimize its long-term performance by choosing and sequencing its deliberation and execution actions appropriately. To the best of our knowledge, Raja and Lesser [33] were the first to explore the issue of meta-level control in complex agents situated in social and dynamic environments. A meta-level agent architecture for bounded-rational agents which supports alternative approaches for deliberative computation was described and experimental results showed the benefits of using this architecture. We compare their work and ours in Table 2.

Cox and Raja [19] [41] presented in plain language and simple diagrams a brief description of a model of metareasoning that mirrors the action-selection and perception cycle in first-order reasoning. In our work, meta-level control is used in a more complex application involving a real-world system to support agent interaction and network reorganization. The meta-level states and actions are abstracted that helps bound the size of policy set and make the problem tractable. Kennedy [23] introduces distributed meta-management where a single agent has multiple meta-levels (metareasoning methods) that monitor each other and the same object level. This requires choreographing the meta-levels, albeit within the same agent. To our knowledge, there is very little work done in the area of exploring the coordination of meta-level control parameters across agents that I exploit in this work. Carlin and Zilberstein [14] considered a decentralized monitoring problem that allows the use of metareasoning to monitor the progress of the anytime algorithms, where the meta-level actions are known to each agent (having 4 options). Our work has a similar basis but handles an application where the set of meta-level actions is more complicated and exponential in size.

6.2. Stochastic Models

A stochastic model [35] is a tool for estimating probability distributions of potential outcomes by allowing for random variation in one or more inputs over time. NetRads is taken as a stochastic model, since the environment is partially observable and each agent has uncertainty of observations. In [30], Peshkin et al. describe *partially observable identical payoff stochastic game* (POIPSG), the interaction of a set of agents with a Markov environment in which they all receive the same payoffs and the agents do not have the identity observation function. The POIPSG model does not work for NetRads, since the reward function of a MCC agent is implemented as a function of its neighbors that directly relates to system performance.

The *multiagent MDP* (MMDP) [10] is a straightforward extension of the MDP to multiple agents by factoring the action space into actions for each of the agents. The disadvantage of the MMDP and related approaches is that it makes an important assumption that renders it inappropriate for many multiagent systems. This is that each agent has the same (complete) world view. Many multiagent systems have each of the agents observing a different part of the environment. The MMDP model does not work for NetRads, because: 1) In NetRads, each MCC has its local observability of the global state. 2) Communication in meta-level control(it is time-critical) is very expensive, if we assume that each MCC can communicate its observation with other MCCs, the cost is too huge for *MMLC* phase to survive. The performance of *MMLC* phase will influence that of deliberative phases.

In [8], decentralized and centralized control problems are studied with a view of computational complexity. They consider two different models of decentralized control of MDPs. One is a generalization of a partially-observable Markov decision process (POMDP), which they call a decentralized partially-observable Markov decision process (DEC-POMDP). In a DEC-POMDP, the process is controlled by multiple distributed agents, each with possibly different information about the state. It does not scale well with large number of agents. The other is a generalization of an MDP, which is called a decentralized Markov decision process (DEC-MDP). A DEC-MDP is a DEC-POMDP with the property of joint full observability. We map the multiagent meta-level control problem in NetRads to a DEC-MDP, because it satisfies the joint full observability that the observations made by the agents fully determine the current state. In [43], the authors view communication as a way of expanding an agent's partial view by exchanging local information not observed by other agents. In NetRads, we introduce the neighborhood communication to access relevant information in the neighborhood.

6.3. Multiagent Reinforcement Learning

Multiagent Reinforcement Learning (*MARL*) [13] is a common approach for solving multiagent decision making problems. It allows agents to dynamically adapt to changes in the environment and keep stability of the agents' learning dynamics, while requiring minimum domain knowledge. Previous techniques of *MARL* have the problem of not converging in the worst case. Bowling [11] uses a variable learning rate to overcome this shortcoming. Bowling presents the Win or Learn Fast heuristic (WoLF) that makes a rational algorithm convergent in a two-agents, two-actions game. WoLF has its own assumption that requires an agent to know a Nash Equilibrium and the strategy of the other players. For practical application like NetRads, this assumption is not hold. The MCC agent can only observe the immediate reward after selecting and performing an local action. Abdallah & Lesser's *Weighted Policy Learner* (WPL) algorithm [1] is a variant of the WoLF [12] algorithm for multiagent meta-level control. The main characteristic of the WoLF algorithm is its ability to change the learning rate to encourage convergence in a multiagent RL scenario. In early work, WPL was used to learn the meta-level control policies for NetRads [15] [17] [16]. However since then, we found that another learning approach provided better results. Zhang and Lesser [44] presented a new gradient ascent algorithm with policy prediction, called *Policy Gradient Ascent with approximate policy prediction (PGA-APP)*, that outperforms WPL in learning results. PGA-APP guarantees that an agent can estimate its policy gradient with respect to the opponent's forecasted strategy without knowing the current strategy and the gradient of the opponent.

Zhang and Lesser [45] used the networked distributed POMDP (ND-POMDP) framework to model cooperative multiagent decision making. They presented a scalable learning approach that synthesizes multiagent reinforcement learning and distributed constraint optimization. They grouped agents that have interactions among them and constructed interaction hypergraph to model this relationship. In NetRads, the joint actions of agents may change the network configuration from time to time and a fixed interaction hypergraph is not able to model this.

In the RL literature, temporal abstraction and hierarchical control are used to combat the curse of dimensionality in a principled way. Sutton et al. [40] extended the usual notion of action to include options - closed loop policies for taking action over a period of time. They showed that options and primitive actions can be used interchangeably in both planning and learning methods. Ghavamzadeh & Mahadevan [20] presented a hierarchical RL framework that studied how lower level policies over subtasks or primitive actions can themselves be composed into higher level polices.

These works emphasize the importance and advantages of abstraction in RL. Our work in meta-level control is different from these works. We use meta-level state/action as an abstract representation of the state/action that captures the similar qualitative information relevant to the meta-level control decision making process. The use of options is inappropriate for NetRads, since the environment is highly dynamic and a policy needs to be applied at each heartbeat to closely track the emerging weather phenomena.

Guestrin et al. [21] showed factored value functions allowed agents to find globally optimal joint action using one single MDP. We use smaller MDPs to learn local policies and then coordinate them, thus avoiding the explosion of the size of policies when the number of agents scales up. Kok & Vlassis [25] used a framework that exploited the dependencies between agents to decompose the global payoff function. In their work, the global reward was factored as a sum of local rewards; while in NetRads, we use a partially global view that reflects shared overlapping tasks to model the individual reward for each agent.

6.4. State of the Art Research on NetRads

Krainin et al. [26] proposed a distributed negotiation mechanism in NetRads that improves the overall system utility with a significantly reduced computational load. An et al. [4] extended this work to introduce the concept of “goal" to model end-users’ preferences over multiple heartbeats and casts the complex sensing resource allocation problem as a continuous time optimization problem. An et al. uses a genetic algorithm to generate optimal scanning strategies of each single MCC and a distributed negotiation model to coordinate multiple MCCs’ scanning strategies over multiple heartbeats. Meta-level control was not introduced in either of their work. We use MARL algorithm to learn the meta-level policies for DEC-MDP. In [24], the authors used an approximate distributed optimization approach to coordinate radars for real-time weather sensing. The approach performed efficiently in terms of resource utilization and communication for most scenario settings in comparison to a negotiation-based algorithm specifically designed for this domain structure. Like Krainin et al., they are concerned about optimizing the deliberative actions (radar scanning), not meta-level actions (such as heartbeat adaptation and radar re-organization) and our approach could be easily built on their distributed optimization algorithm.

7. Conclusion and Future Work

In this paper, we describe a MMLC model that coordinates DEC-MDPs at the meta-level and implements a multiagent version of a RL-based algorithm, called PGA-APP, to learn the policies of the individual MDPs. Previous work in the domain of NetRads [26] showed that a decentralized technique at the deliberation-level with a low number of required optimizations improved tasked allocation in this time-constrained domain. Our hypothesis in this paper is that MMLC that reasons about the deliberative-level approach and coordinates the deliberation across agents leads to improvement in performance.

MMLC allows each agent to carefully choreograph the progression of what deliberations agents should do and when. It also makes agents account for what could happen as deliberation plays out. In our approach, policies for each *weather scenario* are learned offline and each agent adopts the policy appropriate to its meta-level state at runtime. We efficiently decrease the exploration costs of DEC-MDPs by constructing abstract classes of scenarios/states/actions where instances within a class have similar features. The reward in our learning algorithm captures the utility from a partially global perspective to measure the correlation among agents. We use heuristic rules to guide the learning and resolve conflicts among agent policies. Empirical evaluation shows that multiagent meta-level control is an efficient way as the problem scales (up to at least 30 agents) to adjust system parameters and reorganize the network with the goal of improving performance in the context of a multiagent tornado tracking application. Our model can be applied to other domains such as meeting scheduling and sensor networks where two agents with different views of policies for negotiation need to be reconciled.

While our reinforcement learning-based approach described in this paper provides good policies for each agent from a local perspective, the heuristic rule-based approach to conflict resolution provides no guarantee of optimality at the global level. We plan to investigate the use of decentralized constraint-based coordination algorithms to guarantee coordination at the global level in our future work. Currently we resolve the conflict of heartbeat between agents by adapting their communication schedule. In the future, we will investigate conflict resolution by changing heartbeats of a subset of the agents. We will extend our work to learn the strategies that adaptively balance the frequency of MMLC trigger in situations where the cost of MMLC increases substantially. As mentioned earlier we make a simplifying assumption that the entire NetRads system simultaneously experiences a single weather scenario in order to speed up learning. In future work we will extend this work to more complicated environments where different MCCs experience and track different weather scenarios simultaneously. We plan to study how the policies learned under simplified environmental assumptions apply to real-time heterogeneous weather environments. We will investigate when heterogeneous weather scenarios lead to more conflicts and whether MMLC will result in the same level of global performance improvement.

8. Acknowledgements

We thank Chongjie Zhang for his help on framing the ideas in this work and the three reviewers for their insightful comments. This work is supported partially by the National Science Foundation (NSF) under Agreement No.IIS-1018067 and No.IIS-0964590. It is also supported partially by the Engineering Research Centers Program of the NSF under Agreement No.EEC-0313747. 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.

References

[1] S. Abdallah and V. Lesser. Learning the Task Allocation Game. In *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multi-Agent Systems*, pages 850–857, Hakodate, Japan, 2006. ACM Press.

[2] G. Alexander, A. Raja, E. Durfee, and D. Musliner. Design paradigms for meta-control in multi-agent systems. In *Proceedings of AAMAS 2007 Workshop on Metareasoning in Agent-based Systems*, pages 92–103, Hawaii, 2007.

[3] G. Alexander, A. Raja, and D. Musliner. Controlling deliberation in a markov decision process-based agent. In *Proceedings of the Seventh International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS08)*, pages 461–468, Estoril, Portugal, 2008.

[4] B. An, V. Lesser, D. Westbrook, and M. Zink. Agent-mediated Multi-step Optimization for Resource Allocation in Distributed Sensor Networks. In Y. Tumer and S. Sonenberg, editors, *In Proceedings of 10th Int. Conf. on Autonomous Agents and Multiagent Systems Ű Innovative Applications Track (AAMAS 2011)*, Taipei, Taiwan, May 2011. IFAAMAS.

[5] B. Banerjee and J. Peng. Generalized multiagent learning with performance bound. *Autonomous Agents and Multi-Agent Systems*, 15(3):281–312, 2007.

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

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

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

[9] D. P. Bertsekas and J. N. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, Belmont, MA, 1996.

[10] C. Boutilier. Sequential optimality and coordination in multiagent systems. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-99)*, pages 478–485, Stockholm, 1999.

[11] M. Bowling and M. Veloso. Multiagent learning using a variable learning rate. *Artificial Intelligence*, 136:215–250, 2002.

[12] M. Bowling and M. Veloso. Scalable Learning in Stochastic Games. In *Proceedings of AAAI 2002 Workshop on Game Theoretic and Decision Theoretic Agents*, July 2002.

[13] L. Busoniu, R. Babuska, and B. D. Schutter. A comprehensive survey of multiagent reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part C*, 38(2):156–172, 2008.

[14] A. Carlin and S. Zilberstein. Decentralized monitoring of distributed anytime algorithms. In *Proceedings of the tenth International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2011)*, pages 157–164, Taipei, Taiwan, 2011.

[15] S. Cheng, A. Raja, and V. Lesser. Multiagent Meta-level Control for a Network of Weather Radars. In *Proceedings of 2010 IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT-2010)*, pages 157–164, Toronto, Canada, 2010.

[16] S. Cheng, A. Raja, and V. R. Lesser. Towards multiagent meta-level control. In *Proceedings of AAAI, Student Abstract and Poster Program*, pages 1925–1926, Atlanta.

[17] S. Cheng, A. Raja, and V. R. Lesser. Multiagent meta-level control for predicting meteorological phenomena. In *Proceedings of AAAI-2010 Workshop on Metareasoning in Robust Social Systems*, pages 6–13, Atlanta, GA, 2010.

[18] V. Conitzer and T. Sandholm. Awesome: A general multiagent learning algorithm that converges in self-play and learns a best response against stationary opponents. *Machine Learning*, 67(1-2):23–43, 2007.

[19] M. Cox and A. Raja. Metareasoning: A Manifesto. In *Proceedings of AAAI 2008 Workshop on Metareasoning: Thinking about Thinking*, pages 1–4, Chicago,IL, July 2008.

[20] M. Ghavamzadeh and S. Mahadevan. Learning to communicate and act using hierarchical reinforcement learning. In *Proceedings of Autonomous Agents and Multi-Agent Systems*, pages 1114–1121, 2004.

[21] C. Guestrin, D. Koller, and R. Parr. Multiagent planning with factored mdps. In *Proceedings of Neural Information Processing Systems*, pages 1523–1530, 2001.

[22] J. Hu and M. P. Wellman. Nash q-learning for general-sum stochastic games. *Journal of Machine Learning Research*, 4:1039–1069, 2003.

[23] C. Kennedy. Decentralised metacognition in context-aware

autonomic systems: Some key challenges. In *Proceedings of Workshops at the Twenty-Fourth AAAI Conference on Artificial Intelligence*, 2010.

[24] Y. Kim, M. Krainin, and V. R. Lesser. Effective variants of the max-sum algorithm for radar coordination and scheduling. In *Proceedings of 2011 IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT-2011)*, pages 357–364, Lyon, France, 2011.

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

[26] M. Krainin, B. An, and V. Lesser. An Application of Automated Negotiation to Distributed Task Allocation. In *Proceedings of 2007 IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT 2007)*, pages 138–145, Fremont, California, November 2007. IEEE Computer Society Press.

[27] J. Kurose, E. Lyons, D. McLaughlin, D. Pepyne, B. Philips, D. Westbrook, and M. Zink. An end-user-responsive sensor network architecture for hazardous weather detection, prediction and response. In *Proceedings of the Second Asian Internet Engineering Conference, AINTEC*, pages 1–15, 2006.

[28] R. Mailler and V. Lesser. Solving distributed constraint optimization problems using cooperative mediation. In *Proceedings of Third International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2004)*, volume 1, pages 438–445, 2004.

[29] P. Modi, P. Scerri, W.-M. Shen, and M. Tambe. Distributed sensor networks: A multiagent perspective. In *Kluwer Academic Publishers*, 2003.

[30] L. Peshkin, K.-E. Kim, N. Meuleau, and L. P. Kaelbling. Learning to cooperate via policy search. In *Proceedings of Uncertainty in Artificial Intelligence*, pages 489–496, 2000.

[31] A. Petcu and B. Faltings. S-DPOP: Superstabilizing, fault-containing multiagent combinatorial optimization. In *Proceedings of the National Conference on Artificial Intelligence (AAAI-05)*, pages 449–454, Pittsburgh, Pennsylvania, July 2005.

[32] M. L. Puterman. *Markov decision processes: Discrete stochastic dynamic programming*. John Wiley Sons, Inc., 2005.

[33] A. Raja and V. Lesser. A Framework for Meta-level Control in Multi-Agent Systems. *Autonomous Agents and Multi-Agent Systems*, 15(2):147–196, October 2007.

[34] A. Raja and V. Lesser. Coordinating agent's meta-level control. In *Proceedings of AAAI 2008 Workshop on Metareasoning: Thinking about Thinking*, pages 106–112, Chicago, IL, 2008.

[35] S. J. Russel and P. Norvig. *Artificial Intelligence A Modern Approach*. Pearson Education, 2006.

[36] S. J. Russell, D. Subramanian, and R. Parr. Provably bounded optimal agents. In *Proceedings of International Joint Conference on Artificial Intelligence*, pages 338–345, 1993.

[37] H. A. Simon. *Models of Bounded Rationality*. The MIT Press, Cambridge, Massachusetts, 1982.

[38] S. P. Singh, M. J. Kearns, and Y. Mansour. Nash convergence of gradient dynamics in general-sum games. In *Proceedings of Uncertainty in Artificial Intelligence*, pages 541–548, 2000.

[39] R. S. Sutton and A. G. Barto. *Reinforcement Learning*. MIT Press, 1998.

[40] R. S. Sutton, D. Precup, and S. P. Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1-2):181–211, 1999.

[41] M. T.Cox and A. Raja. Chapter: Metareasoning: An introduction. In *Metareasoning: Thinking about thinking*. MIT Press, 2011.

[42] M. Wooldridge. An introduction to multiagent systems. In *John Wiley & Sons, Ltd.,*, 2002.

[43] P. Xuan and V. R. Lesser. Multi-agent policies: from centralized ones to decentralized ones. In *Proceedings of Autonomous Agents and Multi-Agent Systems*, pages 1098–1105, 2002.

[44] C. Zhang and V. Lesser. Multi-Agent Learning with Policy Prediction. In *Proceedings of the 24th National Conference on Artificial Intelligence (AAAI)*, Atlanta, GA, USA, 2010.

[45] C. Zhang and V. R. Lesser. Coordinated multi-agent reinforcement learning in networked distributed pomdps. In *Proceedings of the 25th National Conference on Argificial Intelligence (AAAI)*, 2011.

[46] 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 *Proceedings of the ACM Workshop on End-to-End, Sense-and-Respond Systems, Applications, and Services (EESR 05)*, pages 37–42, Seattle, WA, 2005.